ANALYSIS OF MEMORY ORGANIZATION FOR DYNAMIC NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

An increasing number of neural memory networks have been developed, leading to the need for a systematic approach to analyze and compare their underlying memory structures. Thus, in this paper, we first create a framework for memory organization and then compare four popular dynamic models: vanilla recurrent neural network, long short term memory, neural stack and neural RAM. This analysis helps to open the dynamic neural networks' black box from the memory usage prospective. Accordingly, a taxonomy for these networks and their variants is proposed and proved using a unifying architecture. With the taxonomy, both network architectures and learning tasks are classified into four classes. And a one-to-one mapping is built between them to help practitioners select the appropriate architecture. To exemplify each task type, four synthetic tasks with different memory requirements are developed. Moreover, we use two natural language processing applications to evaluate the methodology in a realistic setting.

1 INTRODUCTION

Recurrent neural networks have been extensively studied and enjoy their success in a lot of sequence learning problems. Elman and Jordan propose the first classic version of recurrent network (RNN) which introduces memory by adding a feedback from the hidden layer to itself for sequence recognition Elman (1990)Jordan (1986). They are often referred to as vanilla RNN (vRNN) nowadays. Although vRNN is theoretically Turing complete if well-trained Doya (1993), it's usually ineffective when the sequence is long.

Many dynamic neural networks have emerged recently to improve the vRNN architecture. (Recurrent, dynamic, and memory neural network are used interchangeably in this paper.) Some of them adopt internal memory; some adopt external memory; some adopt logic gates; while others adopt an attention mechanism. As expected, all of them have advantages for some specific tasks, but it's hard to decide which one is optimal for a new task unless we have a clear understanding of the functions of all memory networks' components. Intuitively, we all know that if the network possesses more components, it can make use of more information, but what kind of the extra information they are using and how useful this extra information is, are still not fully understood in the current literatures. Thus, the major goal of this paper is to open the recurrent neural networks' black box from the memory usage prospective. We illustrate the role and importance of memory by first principles, which is indispensable to continue developing better memory architectures, and that can also help debug these networks. At least in this respect we think that the message of this paper is clear and important for the neural network community. A secondary goal is to summarize all these popular models in a systematic manner and employ the knowledge gained from the different characteristics of these memory structures to help users select the type of memory network given the type of problem. We do so by proposing a taxonomy and connecting models' relative expressive power to the memory requirement of different tasks.

2 RELATED WORK

Among the abundance of recurrent network papers, very few focus on understanding and analysis. Omlin & Giles (1996) discussed how vRNN behaves like deterministic finite-state automata. Gers & Schmidhuber (2001)Rodriguez (2001)Schmidhuber et al. (2002) compared LSTM Hochreiter &

Schmidhuber (1997) and vRNN's performance on some context-free/sensitive language. Collins et al. (2016) studied capacity of recurrent nets and how difficult they are to train. Karpathy et al. (2015) visualized long-term interactions and representations learned by recurrent networks. Greff et al. (2017) empirically studied the importance of various computational components of LSTM. Jozefowicz et al. (2015) evaluated a variety of recurrent neural network architectures and tried to find the best one. Chung et al. (2014) evaluated GRU Cho et al. (2014) compared to LSTMs. These works usually study the performance of networks based on the output error, our work focuses more on how these networks encoded information in order to solve a problem. What's more, these works are based on the basic RNNs and gated recurrent networks such as LSTM and GRU, while our work includes generalized recurrent networks such as neural stack Sun (1993)Joulin & Mikolov (2015), neural Turing machine Graves et al. (2014), etc., which completes and updates the literature.

3 Memory Structure Analysis

In this section, we will analyze memory structures of four popular recurrent neural networks: vRNN, LSTM, neural stack and neural RAM. Attention is paid to how their underlying memory organizations lead to different features and expressive power.

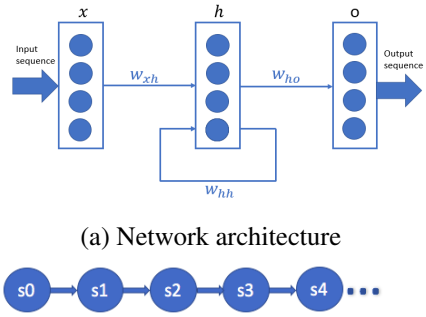
3.1 vRNN

The vRNN network Jordan (1986) is composed of three layers: input, hidden recurrent and output layer. Besides all the feed forward connections, there is a feedback connection from the hidden layer to itself. The architecture of it is shown in Fig.1(a). The dynamics of the hidden layer can be written as,

$$\mathbf{h}_t = f(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{hh}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{b}_h), \tag{1}$$

$$\mathbf{o}_t = f(\mathbf{w}_{ho}^{\mathrm{T}}\mathbf{h}_t + \mathbf{b}_o), \tag{2}$$

where $\mathbf{x}_t$, $\mathbf{h}_t$ and $\mathbf{o}_t$ are the input, hidden state and output vector at time t. We use $\mathbf{w}$ and $\mathbf{b}$ to represent weight and bias of corresponding sizes in this paper. $f(x)$ is the nonlinear activation function.



(a) Network architecture

(b) Memory visualization

Figure 1: vRNN

vRNN induces memory by encoding the past information in its hidden state units $\mathbf{h}_t$. Thus, the memory of vRNN is called state memory or internal memory. Fig.1(b) shows the state transition diagram of vRNN, where $s0$, $s1$, ..., $s4$ represent the state at time t_0, t_1, ..., t_4 respectively. The arrows show the variables' dependency relationship. Here state $s1$ is decided only by $s0$, $s2$ is decided only by $s1$ and so on. (All the memory visualization figures in this paper ignore the current input.) As the number of hidden units is limited in practice, there is always a compromise between memory depth and memory resolution in the vRNN De Vries & Principe (1992). For long memory depths sequences, vRNN needs a very large number of hidden units to achieve an acceptable accuracy. If the sequences are composed by symbols or discrete numbers, this can also be understood from Markov transition model prospective. To be specific, vRNN tries to learn a first order Markov transition model (with transition probability 1) where the current state is decided only by current input and the state at one previous step. Thus for first order Markov sequences, since the state space is not very large (the number of state is less than the size of input symbols' alphabet), vRNN always performs well. However, for higher order Markov sequences or sequences that don't have Markov property, vRNN still tries its best to build a first order Markov state model, which will result in a very large state space(it has to combine several old states into a new state). The compromise between memory depth and memory resolution (which are related to the number and temporal resolution of the states) would make vRNN not suitable for these kinds of sequences.

3.2 LSTM

LSTM was proposed to deal with the vanishing gradient problem of vRNN. In this section, we will analyze how LSTM provides more flexibility from the memory usage prospective. Different from vRNN, in the classical LSTM as shown in Fig.2(a), the feedback connection of hidden layer has to go through an external memory $\mathbf{m}_t$,

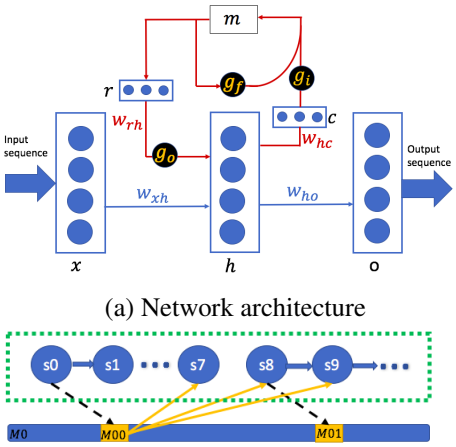
$$\mathbf{c}_t = f(\mathbf{w}_{hc}^{\mathrm{T}}\mathbf{h}_t + \mathbf{b}_c), \tag{3}$$

$$\mathbf{m}_t = g_{i,t}\mathbf{c}_t + g_{f,t}\mathbf{m}_{t-1}, \tag{4}$$

$$\mathbf{r}_t = \mathbf{m}_t, \tag{5}$$

$$\mathbf{h}_t = f(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}} g_{o,t}\mathbf{r}_{t-1} + \mathbf{b}_h), \tag{6}$$

where $\mathbf{h}_t$ (or $\mathbf{c}_t$) is the state of the network. The external memory $\mathbf{m}_t$ is a combination of $\mathbf{m}_{t-1}$ and current state $\mathbf{c}_t$. If $g_{i,t} = 0$ and $g_{f,t} = 1$ for several successive time steps, the content saved in the external memory $\mathbf{m}_t$ would be the long term memory of the system.



(a) Network architecture

(b) Memory visualization

Figure 2: LSTM: A blue belt named $M0$ represents the external memory: at t_1, memory $M00$ is generated and stored, at time t_9, $M00$ is updated to $M01$. The black dash arrows represent the effect of the current state on the external memory. The state index is also the time index.

This external memory $\mathbf{m}_t$ adds more flexibility to the state transition diagram. As shown in Fig.2(b), the current state s_t (represented by hidden state $\mathbf{h}_t$) depends on either the previous one state s_{t-1} or the external memory $\mathbf{m}_{t-1}$ (if forget gate $g_{f,t} = 0$, input gate $g_{i,t} = 1$, s_t depends on s_{t-1}; if $g_{f,t} = 1$, $g_{i,t} = 0$, s_t depends on $\mathbf{m}_{t-1}$; if $0 < g_{f,t} < 1$, $0 < g_{i,t} < 1$, s_t depends on both $\mathbf{m}_{t-1}$ and s_{t-1}. Calculation details of these gates are in Appendix A.1). For example, $s1$ depends on one previous state $s0$ illustrated by the blue arrows, $s7$, $s8$ depend on the long term memory $M00$ illustrated by the yellow arrows, $s9$ depends on both the previous state $s8$ and the long term memory $M00$. The introduced external memory circumvents the compromise between the memory depth versus memory resolution that is always present in the state memory in vRNN. For instance, for a 10th order binary Markov sequence whose state dependence relationship is $s_t = f(s_{t-1}, s_{t-10})$, vRNN has to learn a state space with 2^{10} state (it has to combine 10 states into a new state), however, LSTM only needs to learn a state model with 2 states and an external memory storing the state information 10 steps before. By constructing this short path between long term memory and current state, LSTM works much better than vRNN for sequences that skip intermediate values of time dependencies.

Although LSTM is more effective than vRNN, we have to know its limitations. For example, if there is no skip in time dependence, i.e., $s_t = f(s_{t-1}, s_{t-2}, ..., s_{t-10})$, LSTM and vRNN have the same expressive power. This also tell us the argument that "LSTM is always better than vRNN" is not correct. Another drawback of LSTM is its transient storage of the long term memory. In other words, if the long term memory is updated, its old value is erased. For example, in Fig.2(b), at time t_9, when M00 is updated to M01, M00 is erased. Thus, the future states don't have access to memory $M00$ any more. According to this property, this architecture is extremely useful when the previous states don't need to be addressed again after they are updated.

3.3 Neural Stack

Neural stack refers to neural networks using a stack as its external memory. The stack is controlled by either a feedforward network or a vRNN. One property of stack is that only the topmost content of the stack can be read or written. Writing to the stack is implemented by three operations: push, adding an element to the top of the stack; pop, removing the topmost element of the stack; no-operation, keeping the stack unchanged.

The diagram for the neural stack network is shown in Fig.3(a)(Here, we use the architecture in Joulin & Mikolov (2015)). Elements in the stack would be updated as follows,

$$\mathbf{s}_t(i) = \begin{cases} d_t^{push}\mathbf{c} + d_t^{pop}\mathbf{s}_{t-1}(1) + d_t^{no-op}\mathbf{s}_{t-1}(0), & if\ i = 0, \\ d_t^{push}\mathbf{s}_{t-1}(i-1) + d_t^{pop}\mathbf{s}_{t-1}(i+1) + d_t^{no-op}\mathbf{s}_{t-1}(i), & otherwise, \end{cases} \tag{7}$$

$\mathbf{s}_t(i)$ is the content of the stack at time t in position i. $s_t(0)$ is the topmost content at time t, $\mathbf{c}$ is the candidate content to be pushed onto the stack, d_t^{push}, d_t^{pop} and d_t^{no-op}

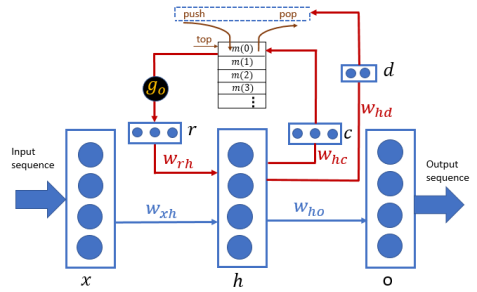
are push, pop and no-operation signals. In order to train the network with BPTT, all operations have to be implemented by continuous functions over a continuous domain. The calculation details of the stack contents and corresponding operators are in Appendix A.2. Since the recurrence is introduced by the stack memory, the dynamics of the model are,

$$\mathbf{h}_t = g(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}}\mathbf{r}_t + \mathbf{b}_h), \tag{8}$$

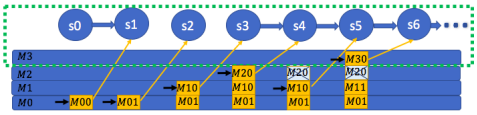
where $\mathbf{r}_t$ is the read vector at time t,

$$\mathbf{r}_t = g_o\mathbf{s}_t(0). \tag{9}$$

Although the architecture of neural stack looks very different from vRNN and LSTM. There are some underlying similarities between them from the memory organization prospective. Fig.3(b) shows the memory space for the neural stack. Different from LSTM, neural stack can store more than one useful content in its external memory bank. For example, at time t_0, $M00$ is saved in memory belt $M0$, at time t_2, $M10$ is saved in belt $M1$. A black arrow on the left of the memory content is used to point the top of stack at each time step. All these contents can be addressed when they are needed. For example, $M10$ is used again at time t_4 after popping out $M20$ in belt $M2$. With this external memory, all the useful information of the input is retained. Different from the state memory, the content of past is not altered, it is stored in its original form or the transformation form. As the content and the operations on the past are separate, we can efficiently select the useful content from this structured memory other than using the mixture of all the content before.



(a) Network architecture



(b) Memory visualization

Figure 3: Neural stack: the network first saves state $M00$ in belt $M0$ and updates it to $M01$. At time t_2, instead of replacing $M01$ with a new state $M10$, a new belt $M1$ is created to save $M10$. In this way, both $M01$ and $M10$ are kept. Similarly, at time t_5, $M30$ is saved in another belt $M3$. In time t_5, the content in the stack is $M01$, $M11$, $M30$ and $M30$ is the topmost element.

LSTM can be seen as a special case of the neural stack. In the neural stack, if all the contents in the stack below the topmost element will never be addressed again, only one memory belt is enough. In this case, neural stack degrades to LSTM as shown in Fig.3(b) with a green dash box. The stack operators (push, pop, no-op) in neural stack have the same function as the input and forget gates in LSTM: deciding how to revise the memory contents. The problem for LSTM is that the previous memories are erased after they are updated, which also happens continuously with the vRNN state. Hence both learning models have difficulties to accomplish some simple memorization tasks such as reversing a sequence. However, the external memory bank in neural stack can help to solve this problem by online storing and extracting more than one content.

Although the neural stack can go back to the previous memory, it has two constraints. Firstly, it can not jump to any memory position, the previous memory should be addressed and updated sequentially. For example, as shown in the second line in Fig.3(b), if we want to go back to the memory in the belt $M1$, we have to pass memory in belt $M2$ first. Secondly, after the memory content is popped out of the stack, it will be forgotten. For example, at time t_4, memory in belt $M2$ is popped out, so in the future time steps, content in belt $M2$ can not be accessed and updated any more.

From the state transition analysis above we can draw the conclusion that, for the tasks where the previous memory needs to be addressed sequentially (first in last out), the stack neural network is our first choice.

3.4 Neural RAM

Recently, some dynamic neural networks with an external random access memory have been studied. In these networks, all the contents in the memory bank can be randomly accessed. Neural Turing machine Graves et al. (2014) (NTM) is one example. Its network architecture is shown in Fig.4(a).

The challenge of this network is that all the memory addresses are discrete in nature. In order to learn read and write addresses by error backpropagation, they have to be extended to continuous domain.

A solution to this difficulty is to read from and write to all the memory slots with different strengths. These strengths can also be explained as the probabilities of each slot to be read from and written to. To be specific, the reading vector at time step t is,

$$\mathbf{r}_t = \sum_{i=0}^{M-1} w_t^r(i)\mathbf{m}_t(i), \tag{10}$$

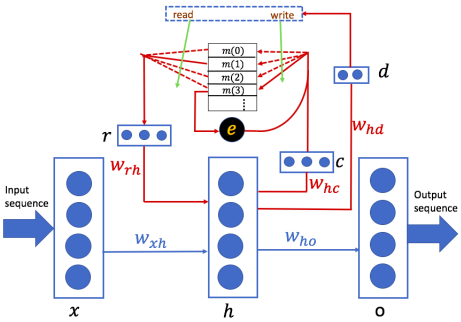
$\mathbf{m}$ is the memory bank with M memory slots, $w_t^r(i)$ is the normalized reading weight for ith slot at time t which satisfying, $\sum_i w_t^r(i) = 1, 0 \leq w_t^r(i) \leq 1$. In the writing process, each memory slot is updated as,

$$\mathbf{c}_t = f(\mathbf{w}_{hc}^{\mathrm{T}}\mathbf{h}_t + \mathbf{b}_c), \tag{11}$$

$$\mathbf{m}_t(i) = w_t^w(i)\mathbf{c}_t(i) + e_t(i)\mathbf{m}_{t-1}(i), \forall i \tag{12}$$

here $w_t^r(i)$ is the writing weight and $e_t(i)$ is the erasing weight for memory slot i at time t. The calculation details of these weights are in Appendix A.3. The dynamics of the hidden layer are,

$$\mathbf{h}_t = f(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}}\mathbf{r}_{t-1} + \mathbf{b}_h), \tag{13}$$



(a) Network architecture

(b) Memory visualization

Figure 4: Neural RAM

Fig.4(b) shows its memory structure. The RAM network can be seen as an improvement of the neural stack in the sense that all the contents in the memory bank can be read from and written to multiple times. And there is no requirement for the order of storing, updating and accessing memory elements. For example, in Fig.4(b), at time t_0, memory $M00$ is stored in belt $M0$, at time t_1, system control can directly jump to belt $M2$ to store $M20$. What's more, the reading and writing slots can be different. For example, at t_1, the network writes to belt $M2$ and reads the content in $M0$. The black arrows on the left of the contents in external memory represent the reading contents. This neural RAM network can degrade to neural stack if the memory accessing order is restricted. Similarly, it can degrade to LSTM if only one memory belt is used. From the analysis above, it is not hard to see that neural RAM is the most powerful network among all the models d

4 A Memory Network Taxonomy

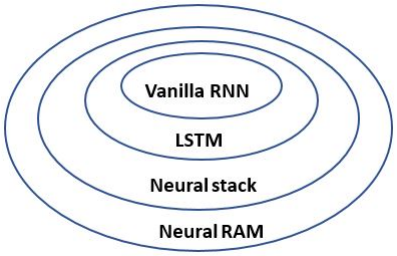


Figure 5: Memory Network Taxonomy

From the analysis in the above section we can draw a conclusion that, the innovation of LSTM versus the vRNN is the incorporation of an external memory and three gates to balance the external memory and internal memory; the innovation of neural stack is to extend one external memory to several external memories and to propose a method to visit the memory slots in a certain order; the innovation of neural RAM is to remove the constraint of the memory slots visiting order, which allows any memory slot to be visited at any time and any number of times. The different memory organizations make these networks have different expressive power. In this section, a taxonomy of recurrent neural network is proposed to classify all these popular models into four classes ordered by a rigorous inclusion relationship, as shown in Fig.5, i.e., vRNN⊆ LSTM⊆neural stack⊆neural RAM. Some classes are named after a typical model. For example, vRNN class also includes IRNN Le et al. (2015), highway network Srivastava et al. (2015), LSTM class also includes GRU Cho et al. (2014), peephole network Gers & Schmidhuber (2000). Neural stack class includes the architecture in Sun (1993); Sun et al. (2017); Joulin & Mikolov (2015); Grefenstette et al. (2015); Neural RAM class includes NTM Graves et al. (2014), DNC Graves et al. (2016) ,enhanced LSTM Graves (2013) and Weston et al. (2014), etc. The classification of these four types of networks are based on the their memory characteristics, i.e., internal memory; one external memory slot; external memory slots with a restricted visiting order; external memory slots without restricted visiting order. For instance, LSTM and GRU belong to the same class since both of them have one external memory slot, though their gate calculations are different.

Table 1: Mapping from network types to task types

networks	memory requirements of tasks
vRNN	only state memory, memory is forced to be used all the time
LSTM	state memory and memory of a single external event
Neural stack	memory of multiple events, information of each event should be used sequentially, only one event is accessible at each time step
Neural RAM	memory of multiple events, all are accessible at each time step, no restriction on how many times they are used

In the following subsections, we will first prove the inclusion relationship mathematically and then show how to link the property of different memory structures to the memory requirement of different tasks, which can help practitioners select the most parsimonious model for a specific task.

4.1 Inclusion Relationship Derivations

Theorem 1. *Neural RAM can be degraded to a neural stack if,*

i) all the reading weights except that for the topmost memory slot are set to zeros, $w_t^r(i) = 0, if\ i \neq 0$*;*

ii) only the writing weight for the topmost memory slot is learned, all others are copied from it, $w_t^r(i) = w_t^r(0), if\ i \neq 0$*;*

iii) in the writing process, instead of learning all the contents to be written to the stack as in Eq.(11), only the content of M0 is learned as, $\mathbf{c}_t(0) = t(\mathbf{w}_{hc}^{'\mathrm{T}}\mathbf{h}_{t-1} + b_c) + \gamma\mathbf{m}_{t-1}(1)$*, all others are calculated as,* $\mathbf{c}_t(i) = \mathbf{m}_{t-1}(i-1) + \gamma\mathbf{m}_{t-1}(i+1),\ if\ i \neq 0$*.;*

iv) only the writing and erasing weights for the topmost element are learned, all others are just a copy of the topmost's values, $w_t^r(i) = w_t^r(0)$, $e_t(i) = e_t(0)$.

Theorem 2. *The neural stack can be degraded to the LSTM if the pop signal is zero,* $d_t^{pop} = 0$*. The* d_t^{push} *in neural stack works as the input gate in LSTM, and* d_t^{no-op} *in neural stack works as the forget gate in LSTM.*

Theorem 3. *The LSTM is degraded to the vRNN if, i) all three gates are set as constants,* $g_o = 0$*,* $g_i = 1$ *and* $g_f = 0$*; ii) weight* $\mathbf{w}_{hc}$ *and bias* $\mathbf{b}_c$ *are set as constants* $\mathbf{w}_{hc} = \mathbf{I}$*,* $\mathbf{b}_c = \mathbf{0}$*; iii) the activation function* $t(x)$ *is set as linear activation function* $t_1(x) = x$*.*

Proof. All the proofs are in Appendix B. □

4.2 Mapping from network types to task types

It is not hard to see that the proposed taxonomy resembles the hierarchical organization of automata: vRNN⇔Finite state machine, neural stack⇔Deterministic pushdown automaton, neural RAM⇔Turing machine. Hence, if our task is sequence recognition or classification, the recognizable sequences for each network can be illustrated by the Chomsky hierarchy. However, these networks can do some more sequence learning tasks such as prediction. In this case, sequences do not need to satisfy the restrict grammars. For example, there is no need for the input sequences to always start from a start state and go back to an accepted state. Hence, in order to make our taxonomy fit into these more general sequences, we divide all the sequence learning tasks into four classes according to their memory requirements, as summarized in Table 1. This mapping can help practitioners select the most parsimonious architecture (we can always go for the most powerful model, but it needs more resources to train) for all sequence learning tasks if they know the memory requirement. In order to exemplify each task type, four tasks employing synthetic symbol sequences are developed: counting, counting with interference, reversing and repeat copying. We will analyze the memory requirements of them one by one.

Counting For the counting task, the input sequences are composed of a's, b's and c's. The output sequence is trying to count the number of a's. For instance, when the input sequence is $aaabcaa$, the output sequence would be 1233345. For this kind of sequences, a state variable is needed to remember the number of a's. Once receiving an a, there is a state transition. In this problem, the

state space is not very large. A first order Markov state model is more than enough to describe it. Hence, as long as the network has one feedback loop, the counting task can be completed. "Task can be completed" in this paper means the output error is almost zero.

Counting with interference For the counting with interference task, the input sequences are the same as the counting task. We still want to count the number of a, but if encountering b or c, the output should also be b or c. For example, if the input is $aabbaca$, the output sequence is $12bb3c4$. For this kind of problem, an external memory cache is required, because when b or c is encountered, the hidden layer's output (internal memory) will be over-written. If we want to recall the number of a's, this value needs to be stored in an external memory for future use, and an input gate will be needed to keep the external memory unaffected when inputting b and c. ($g_i = 1$ when input a and $g_i = 0$ when input b, c.) Thus LSTM, neural stack and neural RAM are capable of solving this problem. However, in vRNN, since the only memory is the state memory and the output is forced to be a function of this state memory, the interference of b and c would make vRNN unable to accomplish this task.

Reversing The third task is sequence reversing. For example, if the input sequence is $abacde\delta xxxxxx$, the output sequence should be $xxxxxxxedcaba$. δ is the delimiter symbol and x means any symbol. When encountering δ in the input sequence, no matter what the following symbols are, the output would be the input symbols before δ in a reverse order. For this task, all the useful past information should be stored and then retrieved in a reverse order. Hence, the memory should have the ability to store more than one element and the reading order is related to the writing order. Since vRNN does not have any memory bank and LSTM's memory is forgotten after it is updated, these two networks fail for this task. On the other hand, both neural stack and neural RAM can store more than one content and the task satisfies the "first in last out" principle, thus they can solve this task.

Repeat copying The hardest task is repeat copying, by which we mean the output sequence is several times repeated version of the input sequences. For example, if the input sequence is $adbc\varepsilon xxxxxxxxxxxxxx$, the output should be $xxxxxxxadbcadbcadbc$. That is, when encountering the repeating number symbol ε, the output will be the previous input sequence for ε times. For this kind of task, not only more than one past content need be stored, they should be retrieved more than one time, here the number is 3. Since all the saved information in neural stack is forgotten after being popped out, it is unable to learn the task. Thus, neural RAM is the only network that can handle this task.

This classification of tasks is very meaningful since it can guide the users in the right direction. If we select the wrong type of network, there will be an error and/or speed penalty no matter how we adjust the hyper parameters. As shown in our experiment part, for sequence reversing (which belongs to the third type of task), neural stack and neural RAM with 6 hidden neurons will converge to near zero error, but for vRNN and LSTM, even if we set the number of hidden neurons to 1000, their output will always fluctuate around a non-zero value.

5 EXPERIMENTS

In order to illustrate the impact of different memory organizations, we test the performance of the four networks on the synthetic tasks described in last section. We also use them to visualize how each network encodes information in order to solve a problem in AppendixD. Then, we used two natural language processing applications to elaborate how to employ the knowledge gained from the different characteristic of the memory structures to help users select the right type of network. The details about parameters settings are in AppendixC.

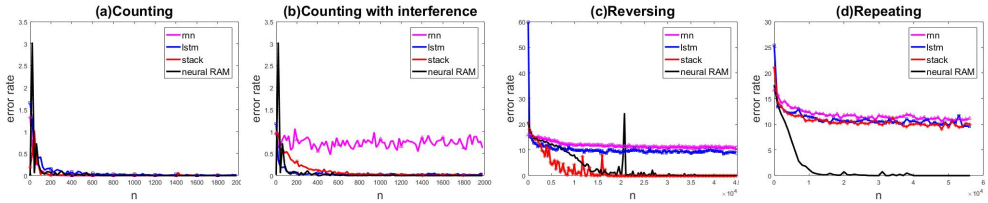


Figure 6: Learning curve for four synthetic tasks

Table 2: Average error for movie review

	vRNN	LSTM	neural Stack	neural RAM
error rate	31±5	**19±2.5**	23±10	20±9

Table 3: Average error for three tasks from bAbI tasks

Task	vRNN	LSTM	neural Stack	neural RAM
task 1	52±1.5	28.4±1.5	41±2.0	**9.0±12.6**
task 2	79±2.5	56.0±1.5	75±6	**39.2±20.5**
task 3	85±2.5	51.3±1.4	78±6.4	**39.6±16.4**

Learning curves for the four tasks using different networks are shown in Fig.6. The performance is measured in MSE for first two tasks and output entropy for the other two tasks. We use the same number of units in all these architectures for a fair comparison. From the result we can observe that for counting, all the four networks can achieve an almost zero error; for counting with interference, all the networks except for vRNN can complete the task; for sequence reversing, neural stack and neural RAM are the suitable networks; and for repeat copying, neural RAM is the only network to solving the problem. We also tried some different parameter settings, for instance, setting the number of hidden units from 5 to 1000, the performances are same as Fig.6 except for a different non-zero error value when the network is not capable to accomplish the task.

5.2 Natural language processing

For synthetic problems it was clear cut to design problems that exemplify the expressive power of the different memory networks. For real world problems this task is more complex because sometimes it is hard to pin point the memory requirements or the problem may be a blend of classes. In this case all the networks may solve the problem to a certain extent. Hence, we will illustrate how to select the minimum network resources to accomplish the task with relatively better performance in this section.

Sentiment Analysis The first experiment is a sentiment analysis which will infer the emotional tone of the text as negative or positive. In order to judge the emotional tone of the text at the end, an external memory whose value would be affected by some key words is useful. And since the goal is to classify the emotional tone as either 1 or 0, the specific contents of the text are not very important here so there is no need to store all of them. Hence, a network with an external memory slot should perform better than the one without. But the memory bank which can store multiple contents does not show more advantages here. We test these networks' performance on lmdb movie review dataset Pennington et al. (2014). The results are in Table 2, which shows that vRNN performs worst. LSTM, neural stack, neural RAM have similar performances. Thus, our analysis is verified.

Question Answering Then we investigate the performance of these four networks on three question answering tasks from bAbI datasetWeston et al. (2015). The target is to give an answer after reading a little story followed by a question. For example, the story is, "Mary got the milk there. John moved to the bedroom. Sandra went back to the kitchen. Mary travelled to the hallway." And the question is, "Where is the milk?". The machine is expected to give the answer "hallway". For this problem, in order to give the right answer, the machine should memorize the facts that Mary got the milk and travelled to the hallway. What's more, since the machine doesn't know the question when reading the story, it has to store all the potential useful facts. Thus an external memory bank whose whichever content can be visited is useful here. According to our memory capability analysis, the neural RAM should perform the best here. From the results in Table 3, we can see that neural RAM indeed achieves the best performance.

6 Conclusion

In this paper, we analyze the memory structure for several recurrent networks and propose a taxonomy of them. We use four synthetic tasks and two natural language processing problems to illustrate utility of the taxonomy. Although we showed differences in performance in the experiments, it is too early to say that we presented all the tools to select parsimoniously the memory architecture for a given application. Because the user has to analyze the requirements of the application, which may not be trivial, more work is needed to create rules of thumb to help practitioners.

REFERENCES

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

Bert De Vries and Jose C Principe. The gamma model-a new neural model for temporal processing. *Neural networks*, 5(4):565–576, 1992.

Kenji Doya. Universality of fully connected recurrent neural networks. 1993.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pp. 189–194. IEEE, 2000.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

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. In *Advances in Neural Information Processing Systems*, pp. 1828–1836, 2015.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2017.

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

Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 531–546. Hillsdale, NJ: Erlbaum, 1986.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pp. 190–198, 2015.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2342–2350, 2015.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Christian W Omlin and C Lee Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM (JACM)*, 43(6):937–972, 1996.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Paul Rodriguez. Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural computation*, 13(9):2093–2118, 2001.

Jürgen Schmidhuber, F Gers, and Douglas Eck. Learning nonregular languages: A comparison of simple recurrent networks and lstm. *Neural Computation*, 14(9):2039–2041, 2002.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Guo-Zheng Sun. Learning context-free grammar with enhanced neural network pushdown automaton. In *Grammatical Inference: Theory, Applications and Alternatives, IEE Colloquium on*, pp. P6–1. IET, 1993.

GZ Sun, C Lee Giles, HH Chen, and YC Lee. The neural network pushdown automaton: Model, stack and learning simulations. *arXiv preprint arXiv:1711.05738*, 2017.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014. URL http://arxiv.org/abs/1410.3916.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

A NETWORK ARCHITECTURE COMPONENTS

A.1 LSTM

The three gates are in LSTM are calculated as follows:

$$g_{i,t} = s(\mathbf{w}_{hg_i}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{w}_{xg_i}^{\mathrm{T}}\mathbf{x}_t + \mathbf{b}_{g_i}), \tag{14}$$

$$g_{f,t} = s(\mathbf{w}_{hg_f}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{w}_{xg_f}^{\mathrm{T}}\mathbf{x}_t + \mathbf{b}_{g_f}), \tag{15}$$

$$g_{o,t} = s(\mathbf{w}_{hg_o}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{w}_{xg_o}^{\mathrm{T}}\mathbf{x}_t + \mathbf{b}_{g_o}). \tag{16}$$

where $\mathbf{w}_{hg_i}$, $\mathbf{w}_{hg_f}$, $\mathbf{w}_{hg_o}$ are $K_h \times 1$ weights , $\mathbf{w}_{hg_i}$, $\mathbf{w}_{hg_f}$, $\mathbf{w}_{hg_o}$ are $K_i \times 1$ weights and b_{g_i}, b_{g_i} and b_{g_i} are bias. These three gates give flexibility to operate on memories.

A.2 NEURAL STACK

Neural network interacts with the stack memory by d_t^{push}, d_t^{pop}, d_t^{no-op}, $\mathbf{c}_t$ and $\mathbf{r}_t$. According to Sun (1993) the domain of the operations is relaxed to any real value in $[0, 1]$. This extension adds a continuous amplitude dimension to the operations. For example, if the push signal $d_{push} = 1$, the current vector will be pushed into the stack as it is, if $d_{push} = 0.8$, the current vector is first multiplied by 0.8 and then pushed into the stack.

d_t^{push}, d_t^{pop}, d_t^{no-op} and $\mathbf{c}_t$ are decided by the hidden layer outputs and the corresponding weights,

$$\mathbf{d} = [d_t^{push}, d_t^{pop}, d_t^{no-op}]^{\mathrm{T}} = s(\mathbf{w}_{hd}^{\mathrm{T}}\mathbf{h}_t + \mathbf{b}_{op}),$$

where $\mathbf{w}_{hd}$ is the $K_h \times 3$ weights and $\mathbf{b}_{op}$ is the 3×1 bias.

$$\mathbf{c}_t = g(\mathbf{w}_{hc}^{\mathrm{T}}\mathbf{h}_t + \mathbf{b}_c),$$

where $\mathbf{w}_{hc}$ is the $K_h \times N$ weights and $\mathbf{b}_{op}$ is the $N \times 1$ bias. Here we assume all the elements saved in the stack are $N \times 1$ vectors.

A.3 NEURAL RAM

In the neural RAM, the read weighting $w_t^r(i)$ is learned as,

$$\boldsymbol{w}_t^r = f(\mathbf{w}_{ha}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{b}_r), \tag{17}$$

$\mathbf{w}_{ha}$ is the $K_h \times M$ weight and $\mathbf{b}_a$ is $M \times 1$ bias, $\boldsymbol{w}_t^r = [w_t^r(0), w^r(1), ..., w^r(M-1)]^{\mathrm{T}}$ The nonlinear activation function f is usually set as softmax function. The write weighting is learned as,

$$\boldsymbol{w}_t^w = s(\mathbf{w}_{hb}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{w}_{xb}^{\mathrm{T}}\mathbf{x}_t + \mathbf{b}_w). \tag{18}$$

where $\mathbf{w}_{hb}$ is $K_h \times M$ weight,$\mathbf{w}_{xb}$ are $K_i \times M$ weight, $\mathbf{b}_b$ is $M \times 1$ bias.

And the erase weighting is learned as,

$$\mathbf{e}_t = s(\mathbf{w}_{he}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{w}_{xe}^{\mathrm{T}}\mathbf{x}_t + \mathbf{b}_e), \tag{19}$$

where $\mathbf{e}_t = [e_t(1),\, e_t(2),\, ...,\, e_t(M)]^{\mathrm{T}}$, $\mathbf{w}_{he}$ is $K_h \times M$ weights, $\mathbf{w}_{xe}$ is $K_i \times M$ weight, and $\mathbf{b}_e$ are $M \times 1$ bias. In practice, instead of learning the read and write head from scratch, some methods were proposed to simplify the learning process. For example in Neural Turing machine Graves et al. (2014), $e_t(i)$ is coupled with write weight $w_t^w(i)$, $e_t(i) = 1 - w_t^w(i)$. And reading weight $\boldsymbol{w}_t^r$ and writing weight $\boldsymbol{w}_t^w$ are obtained by content-addressing and location-addressing mechanisms. The content-addressing mechanism gives the weights $w_t^r(i)$ (or $w_t^w(i)$) by checking the similarity of the key $\mathbf{d}$ with all the contents in the memory, the normalized version is,

$$w_t^r(i) = \frac{exp(\alpha K[\mathbf{d}, \mathbf{m}_t(i)])}{\sum_j (exp(\alpha K[\mathbf{d}, \mathbf{m}_t(j)]))},$$

where α is the parameter to control the precision of the focus, K is a similarity measure. Then, the weights will be further adapted by the location-addressing mechanism. For example, the weights obtained by content addressing can firstly blend with the previous weight and then shifted for several steps,

$$w_t^r(i) = g_t w_{t-1}^r(i) + (1 - g_t) w_t^r(i), \tag{20}$$

$$w_t^r(i) = w_t^r([i - n]_M). \tag{21}$$

g_t is the gate to balance the previous weight and current weight, n is the shifting steps, $[i - n]_M$ means the circular shift for M entities. Since the shifting operation is not differentiable, the method in Graves et al. (2014) should be utilized as an approximation.

Another example is Graves et al. (2016) which improves the performance even more. To be specific, for reading, a matrix to remember the order of memory locations they are written to can be introduced. With this matrix, the read weight is a combination of the content-lookup and the iterations through the memory location in the order they are written to. And for writing, a usage vector is introduced, which guides the network to write more likely to the unused memory. With this modification, the neural RAM gets flexibility similar to working memory of human cognition which makes it more suitable to intelligent prediction. With these modifications, the training time for the neural RAM is also reduced.

B THEOREM PROOFS

B.1 PROOF OF THEOREM 1

Neural RAM is more powerful than neural stack because it has access to all the contents in the memory bank. If we restrict the read and write vector, neural RAM is degraded to neural stack. To be specific, for the read head $\boldsymbol{w}_t^r$, all the read weights except the topmost are set to zeros,

$$w_t^r(i) = \begin{cases} 0, & if\, i \neq 0 \\ t(\mathbf{w}_{ha}^{'\mathrm{T}}\mathbf{h}_{t-1} + b_a), & if\, i = 0 \end{cases}, \tag{22}$$

here $\mathbf{w}'_{ha}$ is $K_h \times 1$ vector and b_a is the scalar. Eq.(22) is a special case of Eq.(17).

$$\mathbf{c}_t(0) = t(\mathbf{w}_{hc}^{'\mathrm{T}}\mathbf{h}_{t-1} + b_c) + \gamma \mathbf{m}_{t-1}(1), \tag{23}$$

all others are calculated as,

$$\mathbf{c}_t(i) = \mathbf{m}_{t-1}(i-1) + \gamma \mathbf{m}_{t-1}(i+1),\ if\ i \neq 0. \tag{24}$$

And in the writing process, Eq.(23), Eq.(24) can be seen as a special case of Eq.(11) because $\mathbf{h}_{t-1}$ in (11) is a function of $\mathbf{m}_{t-1}$. Substitute (23)(24) into the memory update equation of neural RAM (12), we get,

$$\begin{aligned}
\mathbf{m}_t(i) &= w_t^w(i)\mathbf{c}_t(i) + e_t(i)\mathbf{m}_{t-1}(i) \\
&= \begin{cases} w_t^w(i)t(\mathbf{w}_{hc}^{'\mathrm{T}}\mathbf{h}_{t-1} + b_c) + \\ \gamma w_t^w(i)\mathbf{m}_{t-1}(1) + e_t(i)\mathbf{m}_{t-1}(i), & if i = 0 \\ w_t^w(i)\mathbf{m}_{t-1}(i-1) + \\ \gamma w_t^w(i)\mathbf{m}_{t-1}(i+1)\mathbf{c}_t(i) + e_t(i)\mathbf{m}_{t-1}(i), & otherwise \end{cases} \\
&= \begin{cases} w_t^w(i)t(\mathbf{w}_{hc}^{'\mathrm{T}}\mathbf{h}_{t-1} + b_c) + \\ \gamma w_t^w(i)\mathbf{m}_{t-1}(1) + e_t(i)\mathbf{m}_{t-1}(i), & if i = 0 \\ w_t^w(i)\mathbf{m}_{t-1}(i-1) + \\ \gamma w_t^w(i)\mathbf{m}_{t-1}(i+1)\mathbf{c}_t(i) + e_t(i)\mathbf{m}_{t-1}(i), & otherwise \end{cases}
\end{aligned} \tag{25}$$

Finally since Eq.(1) and Eq.(1) can be seen as a special case of Eq.(18) and Eq.(19), the neural stack can be treated as a special case of neural RAM. Compared the memory writing operation of neural stack (7) and neural RAM (25), we can see that, $w_t^w(0)$, $\gamma w_t^w(0)$ and $e_t(0)$ works as the push, pop and no-operate operations respectively. Compared the memory reading operation of neural stack and neural RAM, we can see that we can see that, $w_t^r(0)$ in neural RAM (22) works as the output gate in neural stack (9).

B.2 Proof of Theorem 2

The dynamic of LSTM is,

$$\mathbf{h}_t = f(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}} g_{o,t}\mathbf{r}_t + \mathbf{b}_h), \tag{26}$$

And the dynamic of neural stack is,

$$\mathbf{h}_t = g(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}}\mathbf{r}_t + \mathbf{b}_h), \tag{27}$$

According to Eq.(26) and Eq.(27), the dynamics of the neural stack have similar form as LSTM except for the reading vector, i.e., the reading vector is $\mathbf{r}_t = g_o\mathbf{s}_t(0)$. If we set the pop signal as zero, $d_t^{pop} = 0$, and no operation on the stack contents except for the topmost elements is available, then

$$\mathbf{s}_t(0) = d_t^{push}\mathbf{c} + d_t^{no-op}\mathbf{s}_{t-1}(0),$$

$$\mathbf{s}_t(i) = 0,\ if\ i \neq 0.$$

Since the d_t^{push}, d_t^{no-op} are calculated in the same way the input gate $g_{i,t}$ and forget gate $g_{f,t}$ are calculated in LSTM as shown in Eq.(14) to Eq.(15), the read vector would be,

$$\begin{aligned}
\mathbf{r}_t &= g_o\mathbf{s}_t(0) \\
&= g_o(d_t^{push}\mathbf{c} + d_t^{no-op}\mathbf{s}_{t-1}(0))
\end{aligned}$$

In this manner, this is exactly how the LSTM organizes its memory,

$$\mathbf{r}_t = g_o(g_{i,t}\mathbf{c} + g_{f,t}\mathbf{s}_{t-1}(0))$$

d_t^{push}can be seen as the input gate and d_t^{no-op} can be seen as the forget gate. Hence, it is proved that LSTM can be seen as a special case of neural stack.

B.3 Proof of Theorem 3

Compared to vRNN, LSTM introduces an external memory and the gate operation mechanism. So if we set the output gate $g_o = 0$, input gate $g_i = 1$ and the forget gate $g_f = 0$ instead of learning from

the sequences, the dynamics of LSTM is degraded to vRNN as follows,

$$\begin{aligned}
\mathbf{h}_t &= t(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}} g_o \mathbf{r}_t + \mathbf{b}_h) && (28)\\
&= t[\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}}\mathbf{r}_r + \mathbf{b}_h] && (29)\\
&= t[\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}}\mathbf{m}_t + \mathbf{b}_h]\\
&= t[\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}}(g_i\mathbf{c}_t + g_f\mathbf{m}_{t-1}) + \mathbf{b}_h]\\
&= t(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}}\mathbf{c}_t + \mathbf{b}_h) && (30)\\
&= t[\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}} t_1(\mathbf{w}_{hc}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{b}_c) + \mathbf{b}_h] && (31)\\
&= t(\mathbf{w}_{xh}^{\mathrm{T}}\mathbf{x}_t + \mathbf{w}_{rh}^{\mathrm{T}}\mathbf{h}_{t-1} + \mathbf{b}_h) && (32)
\end{aligned}$$

Here (29) is due to $g_o = 0$, (30) is due to $g_i = 1$ and $g_f = 0$, (32) is because the weight $\mathbf{w}_{hc}$ and bias $\mathbf{b}_c$ are set as constants and the activation function $t(x)$ is set as linear activation function,

$$\mathbf{w}_{hc} = \mathbf{I},$$

$$\mathbf{b}_c = \mathbf{0},$$

$$t_1(x) = x.$$

Since Eq.(28) is the dynamic of LSTM and Eq.(32) is the dynamic of vRNN, the argument that vRNN is a special case of LSTM is proved.

C EXPERIMENT PARAMETERS SETTING

C.1 COUNTING AND COUNTING WITH INTERFERENCE

In the experiment, the activation function is Relu in vRNN. In LSTM, the external memory's content is initialized as zero. In the neural stack, the push, pop and no-op operations are initialized as random numbers with mean 0 and variance 1. At first, there is only one content in the stack which is initialized as zero. The depth of the stack can increase to any number as required. In neural RAM, memory depth is set as $M = 3$. In LSTM, neural stack and neural RAM, memory width is set as $N = 3$, the nonlinear activation functions for all the gates are sigmoid functions and others are tanh. The number of input neurons, hidden neurons and output neurons are 3. All the weights are initialized as random numbers with mean 0 and variance 1, all the bias are initialized as 0.1. For counting task, the model is trained with the synthetic sequences up to length 20. When the input is a, the first elements in the output vector would add one, otherwise, the output vector is unchanged. After encoding, the input and output vectors are,

time step	input sequence	output sequence
1	[1 0 0]	[1 0 0]
2	[1 0 0]	[2 0 0]
3	[0 1 0]	[2 0 0]
4	[0 1 0]	[2 0 0]
5	[1 0 0]	[3 0 0]
6	[0 0 1]	[3 0 0]
7	[1 0 0]	[4 0 0]

For counting with interference task, after encoding, the input and output vectors are,

time step	input sequence	output sequence
1	[1 0 0]	[1 0 0]
2	[1 0 0]	[2 0 0]
3	[0 1 0]	[0 1 0]
4	[0 1 0]	[0 1 0]
5	[1 0 0]	[3 0 0]
6	[0 0 1]	[0 0 1]
7	[1 0 0]	[4 0 0]

C.2 REVERSING AND REPEAT COPYING

Some network settings are different from the first two experiments. In vRNN, the activation function in the hidden layer is sigmoid function since we use entropy instead of mean square error as the cost function. In neural RAM, the word size and memory depth are set as 16. The length of read and write vectors are also set as 16. The number of input neurons, hidden neurons and output neurons are 6, 64, 6. The model is trained with sequences up to length 20. In repeat copying experiment, the training sequences are composed of a starting symbol ϵ, some symbols in set $\{a, b, c, d, e\}$ followed by a repeating number symbol δ and some random symbols. ϵ, a, b, c, d, e are one-hot encoded with on value 1 and off value 0; δ is encoded with on value n and off value 0, n is the repeating number.

C.3 SENTIMENT ANALYSIS

In our experiments, the number of input neurons, hidden neurons and output neurons are 50, 64, 2 for all the four network architectures. After encoding all the words into vectors, they are fed into the network one by one. Here we use a pretrained model: GloVe Pennington et al. (2014) to create our word vector. The matrix contains 400,000 word vectors, each with a dimensionality of 50. The matrix is created in a way that words having similar definitions or context reside in the relatively same position in the vector space. The decision of the tone of the paragraph will be made at the end of the paragraph. The output is [1, 0] for the positive text and [0, 1] for the negative text. The dataset adopted here is the lmdb movie review data**?** which has 12500 positive reviews and 12500 negative reviews. Here we use 11500 reviews for training and 1000 data for testing. In our experiments, the nonlinear activation functions for all the gates are sigmoid. The activation functions at the output layer are sigmoid and others are tanh. In neural RAM, the word size and memory depth are set as 64. The number of read and write head are 4 and 1. The results are average of 20 runs with random initializations.

C.4 QUESTION ANSWERING

For each task, we use the 10,000 questions to train and report the error rates on the test set. The experimental settings for LSTM and neural RAM are the same as Graves et al. (2016) and the results for these two networks are from Graves et al. (2016). In vRNN and neural stack, the nonlinear activation functions for all the gates are sigmoid. The activation functions at the output layer are sigmoid and others are tanh. The number of input neurons, hidden neurons and output neurons are 150, 64, 150. The memory width for neural stack is 64.

D MEMORY WORKING PROCESS VISUALIZATION FOR THE SYNTHETIC TASKS

D.1 COUNTING

Fig.7 to Fig.10 show the details of the memory contents after the models are well-trained. They are tested on a input sequence $bbacacbababababcc$. The little checks on the left side of each image mark the memories which affect the output results. In Fig.7, when a is received, the first element of the hidden layer is increased by 1. In Fig.8, when a is received, the first element of $M0$ is decreased by around 0.15. As long as there is at least one element ($h0$ in vRNN and the first element of $M0$ in LSTM) in the memory learning the pattern, after multiplying with the weight vector, the output of the network can give the expected values. Fig.9 shows how the neural stack uses its memory. Although the neural stack has the potential to use unbounded number of stack contents, it only uses the topmost content ($M0$) here, i.e. the push and no-operation cooperate to learn the pattern. The contents pushed into $M1$ to $M5$ are never used again. Fig.10 shows the memory contents of the neural RAM and the corresponding operations of it. From Fig.10(a), we can see that $M0$, $M1$ and $M2$ cooperate to learn the pattern. When a is received, the values in $M0$, $M1$ and $M2$ are increased. The read weights for $M0$, $M1$ and $M2$ are all around 0.3 as shown in Fig.10(b). From this experiment we can see that $M1$ and $M2$ are redundant here.

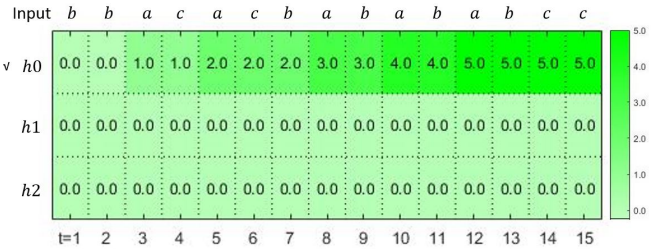


Figure 7: Task1: vRNN: Internal memory content

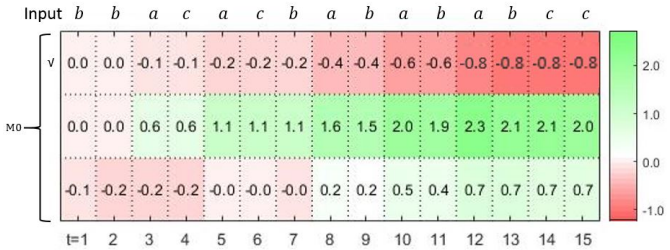


Figure 8: Task1: LSTM: External memory content

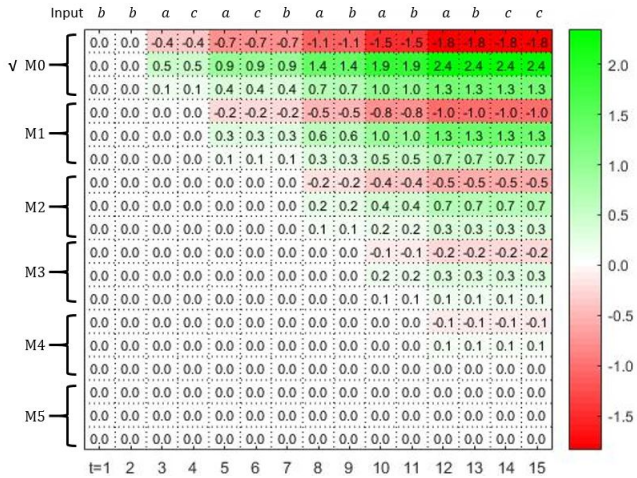


Figure 9: Task1: Neural stack: stack content

D.2 COUNTING WITH INTERFERENCE

The experiment settings are same as "counting" task. Fig.11 to Fig.13 shows the memory usage of LSTM, neural stack and neural RAM. Fig.11 shows that when a is received, the third element of the memory content would increase by around 0.2. Fig.12 and Fig.13 also show the similar incremental patterns of neural stack and neural RAM. An notable difference between Fig.11-Fig.12 and Fig.8-Fig.9 is the usage of the memory. When dealing with counting task, the output gates are always 1, however, when dealing with counting with interference task, the output gates are 0 when inputting b and c, this helps to cut off the interference from the memory. Similarly, the read vector are always around [0.3, 0.3, 0.3] in Fig.10, however, in Fig.13, the read vector's elements are almost zeros when encountering b and c. The read vector here works as the output gate in LSTM and neural stack. It also shows why neural RAM does not need an output gate.

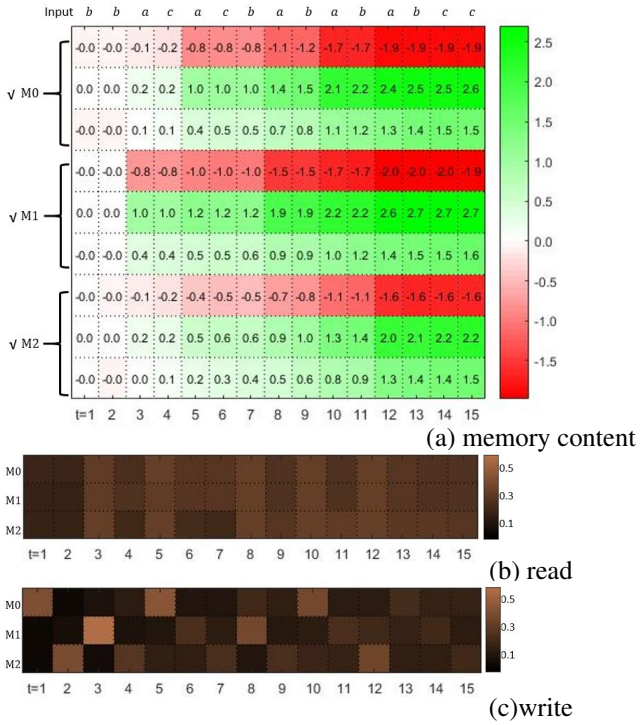


Figure 10: Task1: Neural RAM: Memory bank content and corresponding read and write operation

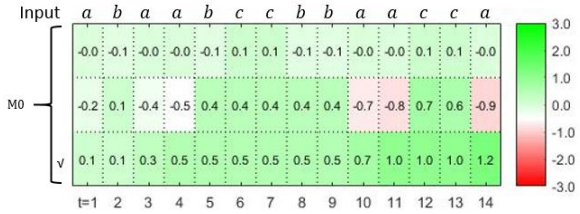


Figure 11: Task2: LSTM: External memory content

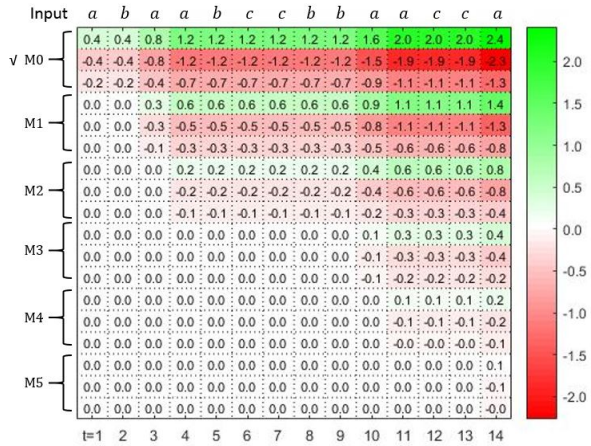


Figure 12: Task2: Neural stack: stack content

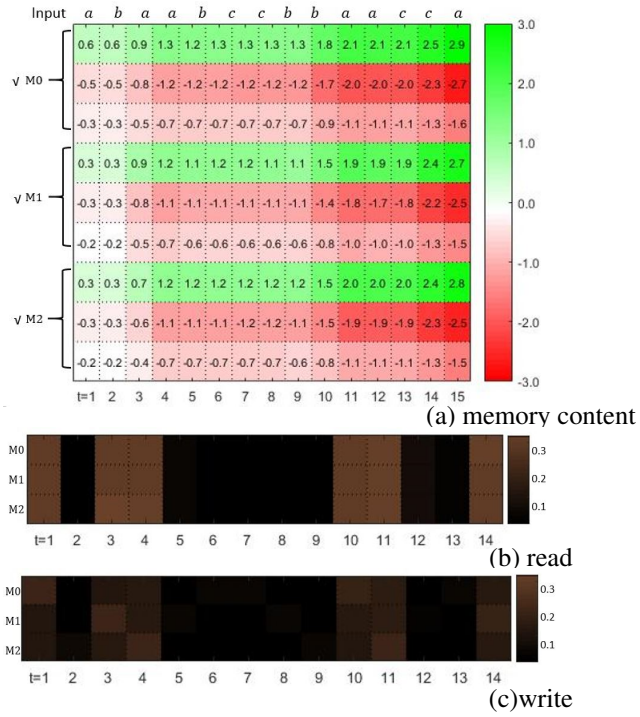


Figure 13: Task2: Neural RAM: Memory bank content and corresponding read and write operation

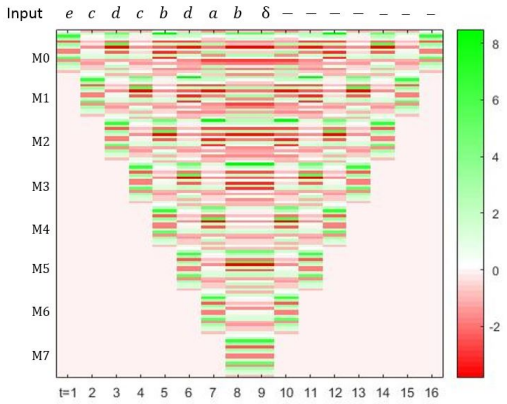
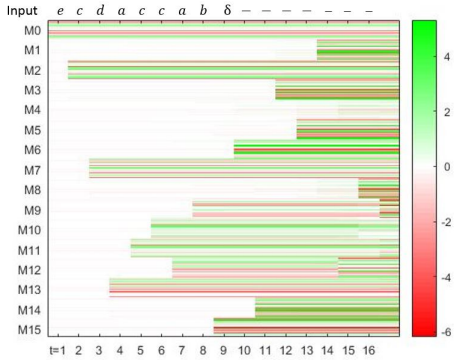


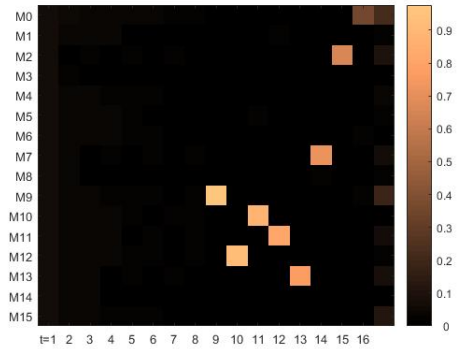
Figure 14: Task3: Neural stack: stack content

D.3 REVERSING

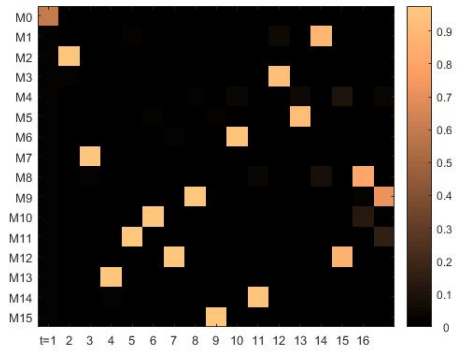
Fig.14 shows how neural stack utilizes its stack memory to solve this problem. Since each memory bank's word size is 16, here we only use colors instead of the specific numbers to show the values of contents in memory. Different from the first two tasks, the function of the stack is finally exploited. In the first half sequence, the input symbols are encoded as 16-elements vectors and pushed into the stack. In the second half of the sequence, the contents in the stack are popped out sequentially. It should be noticed that as long as the the contents are popped out, they can not be revisited anymore. Different from neural stack, the contents in neural stack are never wiped as shown in Fig.15. The contents in the memory bank are only wiped if they are useless in the future or they have to be wiped to make space for new stuffs. Another feature of the memory bank for neural RAM is the memory banks are not used in order such as M0, M1, M2...In this example, the memory banks are used in the order M0, M2, M7, M13.... But as long as the network knows the writing order, the task can be accomplished. Fig15(b)(c) shows the reading and writing weights, we can see that the second half of



(a) memory content



(b) read



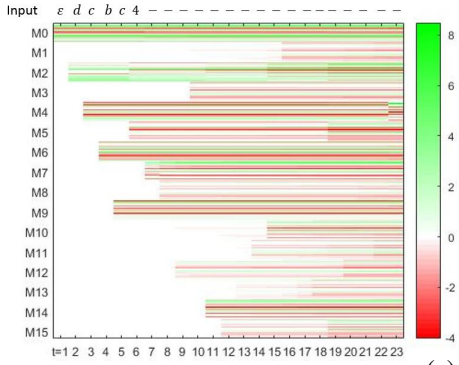
(c) write

Figure 15: Task3: Neural RAM: Memory bank content and corresponding read and write operation

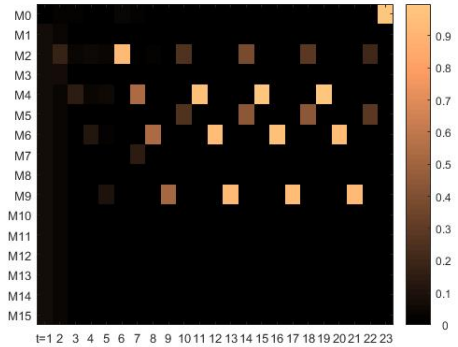
the reading weights is the mirror of the first half of the sequence of the writing weights, which means the network learns to reverse.

D.4 REPEAT COPYING

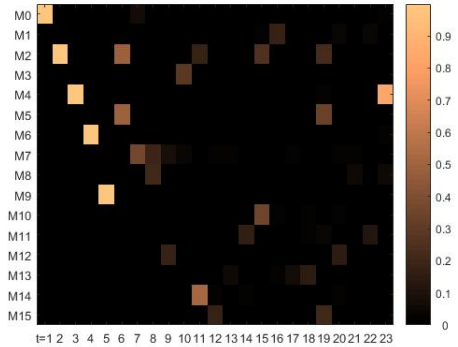
Since neural RAM is the only network that can handle this problem, Fig.16 shows how the neural RAM solves this problem. From the writing weights we can see that, the starting symbol is saved in M0, and symbols needed to be repeated are save in M2, M4, M6, M9. After t=4, the network would read from M2/M5, M4,M6,M9. At the beginning of every loop, the network reads from both M2 and M5 probably because the repeating time symbol δ is saved in M5. The value in M5 can tell the network whether to continue repeating or not. We can see from Fig.16(b), at time t=22, after reading from M2 and M5, the network stops reading from M4 to M9 and turns to M0.



(a) memory content



(b) read



(c) write

Figure 16: Task4: Neural RAM: Memory bank content and corresponding read and write operation