A Taxonomy for Neural Memory Networks

Ying Ma and Jose C. Principe, *Life Fellow, IEEE*

***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 selected. Moreover, we use some signal processing applications and two natural language processing applications to evaluate the methodology in a realistic setting.**

***Index Terms*— Long short-term memory (LSTM), neural RAM, neural stack, recurrent neural network (RNN).**

I. Introduction

RECURRENT neural networks (RNNs) have been extensively studied and enjoy success in a lot of sequence learning problems. Elman [1] and Jordan [2] proposed the first classic version of recurrent network (RNN) that introduces memory by adding a feedback from the hidden layer to itself for sequence recognition. They are often referred to as vanilla RNN (vRNN) nowadays. Although vRNN is theoretically Turing complete if well-trained [3], it is usually ineffective when the sequence is long.

Many dynamic neural networks (recurrent, dynamic, and memory neural network are used interchangeably in this paper) have emerged recently to improve the vRNN architecture. Some of them adopt internal memory, some adopt external memory, and some adopt logic gates, while others adopt an attention mechanism. As expected, all of them have advantages for some specific tasks, but it is 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 still remain not fully understood in the current works. Thus, the major goal of this paper is to open the RNNs' 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 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.

Manuscript received June 30, 2018; revised March 19, 2019; accepted June 2, 2019. Date of publication August 20, 2019; date of current version June 2, 2020. This work was supported in part by the Lifelong Learning Machines Program of the Defense Advanced Research Projects Agency, Microsystems Technology Office, under Grant FA9453-18-1-0039. *(Corresponding author: Ying Ma.)*

The authors are with the Computational NeuroEngineering Laboratory (CNEL), Department of Electronic and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: mayingbit2011@gmail.com; principe@cnel.ufl.edu).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2926466

II. Related Work

Among the abundance of recurrent network papers, a very few papers focus on understanding and analysis. Omlin and Giles [4] discussed how vRNN behaves like deterministic finite-state automata, while [5]–[7] compared long short-term memory (LSTM) [8] and vRNN's performance on some context-free/context-sensitive language. In [9], the capacity of recurrent nets and how difficult they are to train is studied, while [10] visualized long-term interactions and representations learned by recurrent networks. Greff *et al.* [11] empirically studied the importance of various computational components of LSTM, and Jozefowicz *et al.* [12] evaluated a variety of RNN architectures and tried to find the best one. Finally, [13] evaluated GRU [14] compared to LSTMs and [15] tested and compared The performances of sequential, random access, and stack memory architectures on the language modeling data set. These works usually study the performance of networks based on the output error, and this paper focuses more on how these networks encoded information in order to solve a problem.

III. Memory Structure Analysis

Memory analysis is not an easy issue because "memory" is a very abstract concept and the specific memory requirements for a specific task are implicit, which means that quantitatively conceptualizing and analyzing memory is a hard problem. Memory capacity is used to quantify how much information can be stored at a specific time, but it fails to include the time information; in other words, it cannot be used to measure how many time steps of information can be stored. However, memory exists in the space of events that build the collected signal and in time; therefore, it is a spatiotemporal concept.

In order to incorporate time information in the measure of memory, de Vries and Principe [16] proposed a methodology to quantify memory with an analytic expression for the compromise between memory depth and memory resolution.

However, there are a lot of complex dynamic systems that not only evolve in time but are also affected by useful past events. For example, in natural language processing, if a paragraph is given:

"John was born and raised in China. He went to the UK to study after 18 years old and now lives in the United States. John's mother language is –".

The word in – should be Chinese because of the fact that John was raised in China. All other information is irrelevant. Hence, instead of formulating the data generating system as

$$h_n = f(h_{n-1}, x_n) + v_n \tag{1}$$

$$y_n = g(h_n) + n_n \tag{2}$$

where h_n is the state at time n, and a more efficient formula should be

$$h_n = f(e_{m_1}, e_{m_2}, e_{m_3}, \ldots, x_n) + v_n \tag{3}$$

where e_{m_i} is a relevant event happening at time m_i and $m_i < n$, where n is the current time. The relevant events at each time step could be different.

Equation (1) is one popular representation of state-space model, which describes a system with input x_n and output y_n (x_n and y_n are also called cause and observation) in terms of a latent Markovian state h_n. $f(\bullet)$ is the transition model and $g(\bullet)$ is the observation model.

Equation (1) can be used to describe many classic control problems, where memory of past is encoded in the state variable h_n. However, for some real sequence learning problem (e.g., video or language), although we can still formulate them as a complex dynamic system, building such a state transition model is almost impossible, and the state space would be very large.

Thus, we decide to describe the dynamic of some complex dynamic system with (3), which decouples state samples from memory events. Instead of encoding the memory with a state variable, the memory is encoded in multiple past events e_{m_i}. Hence, e_{m_i} is a function of the state and input variables and some previous extracted events E

$$e_{m_i} = t(h_{m_i-1}, x_{m_i}, E) + n'_n \tag{4}$$

where n'_n is a noise term. Since useful events do not happen all the time, we use m_i instead of i here. In the above-mentioned example, e_{m_1} is "China." $m_1 = 7$ (1: Jone, 2: was, 3: born, 4: and, 5: raised, 6: in, and 7: China). Equation (1) can be seen as a special case of (3) when there is only one useful event at every time step

$$h_n = f(e_{n-1}, x_n) + v_n \tag{5}$$

$$e_{n-1} = t(h_{n-1}, x_n) + n'_n. \tag{6}$$

Comparing (1) and (3), (3) brings us several advantages as follows.

1) Avoiding the difficulty of building a state variable x_t for complex dynamic systems.
2) The working mechanism of the underlying dynamic system is more like the way human solving with a sequence learning problem (when humans make decisions or predictions, they usually ignore useless intermediate samples and recall relevant past events).
3) It provides a new framework to analyze memory structure for the memory system. When the problem is posed in this way, we immediately can see three fundamental steps in any memory system as follows:
   a) how to select the relevant events from the flow of time samples, i.e., extracting useful events e_{m_i} from the time series and storing them for future use;
   b) how to select the time interval where relevant events affect the current processing, i.e., at each time step, the relevant events need to be selected from the past stored events' set;
   c) how to effectively use this information for the current task, i.e., how to store the least number of events to solve a specific task.

Actually, unlike vRNN [the underlying dynamic model of vRNN is (1)], a lot of recent memory architectures have the capability of extracting events from time flow. "How many events can be stored and accessed" is an important property of various memory network architectures and can help to distinguish these different network architectures. The capacity of an architecture can be enhanced a lot by hyperparameter selection (for example, the number of neurons in vRNN), but the "number of distinct events that can be stored" depends upon the network architecture. Thus, in this section, we will investigate the characteristics of memory implemented by four popular RNNs: vRNN, LSTM, neural stack, and neural RAM from "how many events can be stored" perspective. Attention is paid to how their underlying memory organizations lead to different features and expressive power.

A. vRNN

The vRNN network [2] is composed of three layers: input, hidden recurrent, and output layers. Besides all the feedforward 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

$$\mathbf{h}_t = f\left(\mathbf{w}_{xh}^T \mathbf{x}_t + \mathbf{w}_{hh}^T \mathbf{h}_{t-1} + \mathbf{b}_h\right) \tag{7}$$

$$\mathbf{o}_t = f\left(\mathbf{w}_{ho}^T \mathbf{h}_t + \mathbf{b}_o\right) \tag{8}$$

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

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.

This memory mixed all the past events in its hidden state. It cannot recover these distinct events from the hidden state. Hence, the state can also be seen as one single compound event, which is updated at each time step

$$h_n = f(e_{n-1}). \tag{9}$$

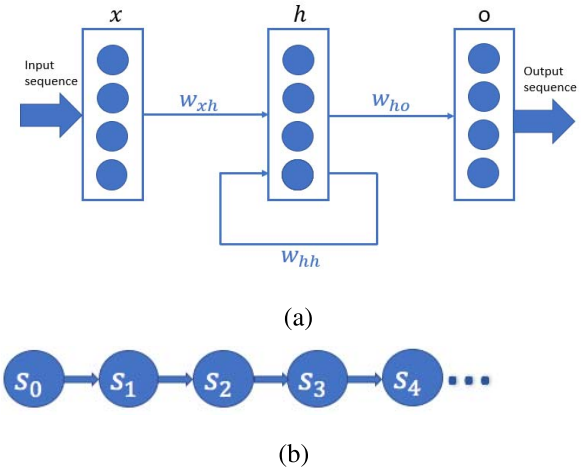


Fig. 1. vRNN.

Compared to (3), current state h_n only depends on one compound event, which happens at time $n-1$.

Fig. 1(b) shows the state transition diagram of vRNN, where $s_0, s_1, \ldots, s_4$ represent the state at time $t_0, t_1, \ldots, t_4$, respectively. The arrows show the variables' dependence relationship. Here, state s_1 is decided only by s_0, s_2 is decided only by s_1, 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 [16]. 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 of 1) where the current state is decided only by the 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 do not 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.

B. 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 $\mathbf{m}_t$

$$\mathbf{c}_t = f\left(\mathbf{w}_{hc}^T \mathbf{h}_{t-1} + \mathbf{b}_c\right) \tag{10}$$

$$\mathbf{m}_t = g_{i,t}\mathbf{c}_t + g_{f,t}\mathbf{m}_{t-1} \tag{11}$$

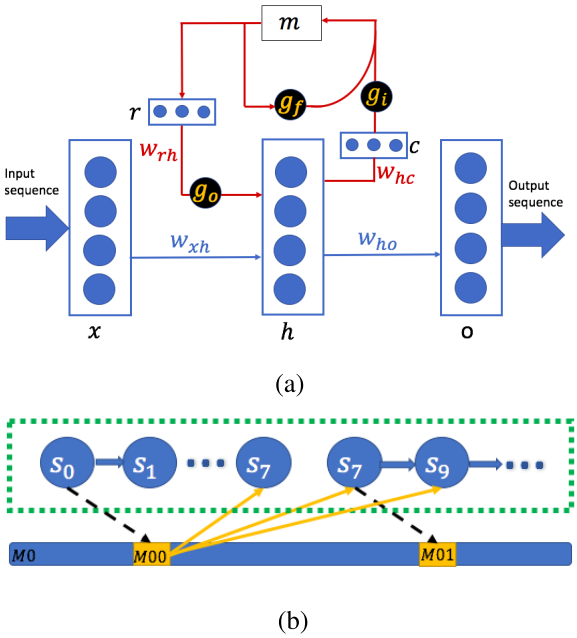


Fig. 2. LSTM. Blue line (called here a belt) named $M0$ represents the external memory over time. At t_1, memory $M00$ is generated and stored, and at time t_9, $M00$ is updated to $M01$. Black dashed arrows represent the effect of the current state on the external memory. The state index is also the time index. (a) Network architecture. (b) Memory visualization.

$$\mathbf{r}_t = \mathbf{m}_t \tag{12}$$

$$\mathbf{h}_t = f\left(\mathbf{w}_{xh}^T \mathbf{x}_t + \mathbf{w}_{rh}^T g_{o,t}\mathbf{r}_t + \mathbf{b}_h\right) \tag{13}$$

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.

This external memory $\mathbf{m}_t$ adds more flexibility to the state transition diagram, and in fact, the LSM is the first neural memory system that was working with memory elements of (3), although this was never mentioned. 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 $g_{f,t} = 0$ and input gate $g_{i,t} = 1$, s_t depends on s_{t-1}, if $g_{f,t} = 1$ and $g_{i,t} = 0$, s_t depends on $\mathbf{m}_{t-1}$, and if $0 < g_{f,t} < 1$ and $0 < g_{i,t} < 1$, s_t depends on both $\mathbf{m}_{t-1}$ and s_{t-1}; The calculation details of these gates are in Appendix A-A). For example, $s1$ depends on one previous state s_0 illustrated by the blue arrows, s_7 and s_8 depend on the long-term memory $M00$ illustrated by the yellow arrows, and s_9 depends on both the previous state s_8 and the long-term memory $M00$. The introduced external memory circumvents the compromise between the memory depth and memory resolution that is always present in the state memory in vRNN. For instance, for a tenth-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 ten states into a new state); however, LSTM only needs to learn a state model with two states and an external memory storing the state information ten steps before. By constructing

this short path between the long-term memory and the current state, LSTM works much better than vRNN for sequences that skip intermediate values of time dependences.

In other words, LSTM is capable of extracting a useful event at a specific time and storing it in an external memory which will not be affected by the intermediate irrelevant information. This long-term memory can be seen as the event extracted from the input time series

$$h_n = f(e_{m_1}). \tag{14}$$

Note that here, the useful event happened at time m_1 other than $n-1$ as in vRNN (9).

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., $s_t = f(s_{t-1}, s_{t-2}, \ldots, s_{t-10})$, LSTM and vRNN have the same expressive power. This also tells 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 do not have access to memory $M00$ any more. According to this property, this architecture is extremely useful when the previous states do not need to be addressed again after they are updated.

C. 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 stack property 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; and no-operation, keeping the stack unchanged.

The diagram for the neural stack network is shown in Fig. 3(a), where we use the architecture in [17]. Elements in the stack would be updated as follows:

$$\mathbf{m}_t(i) = \begin{cases} d_t^{\text{push}}\mathbf{c} + d_t^{\text{pop}}\mathbf{m}_{t-1}(1) + d_t^{\text{no-op}}\mathbf{m}_{t-1}(0) \\ \qquad\qquad \text{if } i = 0 \\ d_t^{\text{push}}\mathbf{m}_{t-1}(i-1) + d_t^{\text{pop}}\mathbf{m}_{t-1}(i+1) \\ + d_t^{\text{no-op}}\mathbf{m}_{t-1}(i), \qquad \text{otherwise} \end{cases} \tag{15}$$

where $\mathbf{m}_t(i)$ is the content of the stack at time t in position i, $m_t(0)$ is the topmost content at time t, $\mathbf{c}$ is the candidate content to be pushed onto the stack, and d_t^{push}, d_t^{pop}, and $d_t^{\text{no-op}}$ are push, pop, and no-operation signals, respectively. In order to train the network with back propagation through time (BPTT), all operations have to be implemented by continuous functions over a continuous domain. The calculation details of the stack contents and the corresponding operators are in Appendix A-B. Since the recurrence is introduced by the stack memory, the dynamics of the model are

$$\mathbf{h}_t = g\left(\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T\mathbf{r}_t + \mathbf{b}_h\right) \tag{16}$$

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

$$\mathbf{r}_t = g_o\mathbf{m}_t(0). \tag{17}$$

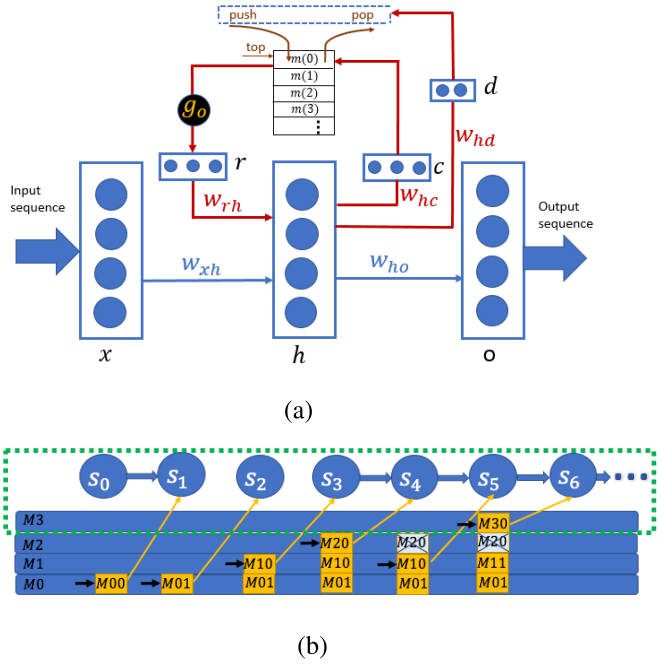


Fig. 3. Neural stack: the network first saves state $M00$ in belt $M0$ and updates it to $M01$. At time t_2, instead of replacing $M01$ with a new state $M10$, a new belt $M1$ is created to save $M10$. In this way, both $M01$ and $M10$ are kept. Similarly, at time t_5, $M30$ is saved in another belt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

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 $M0$, and 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, and 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. Hence, the realization of state update of neural stack is

$$h_n = f(\underbrace{e_{m_1}, e_{m_2}, \ldots, e_{m_{i-1}}}_{\text{not accessible at step n}}, e_{m_i}) \tag{18}$$

where e_{m_i} is the event stored on the top of stack. Comparing (14) and (18), LSTM can be seen as a special case of the neural stack.

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

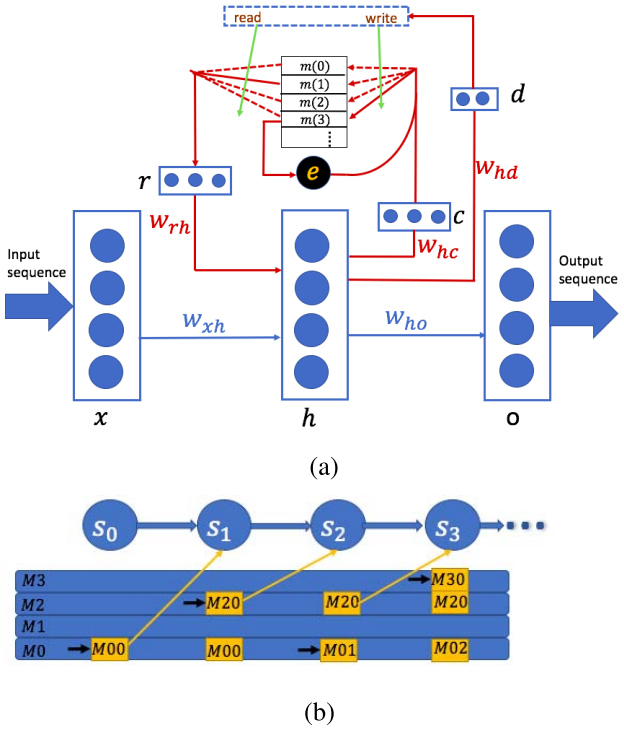


Fig. 4. Neural RAM. (a) Network architecture. (b) Memory visualization.

LSTM as shown in Fig. 3(b) with a green dashed box. The stack operators (push, pop, and no-op) in the 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 the 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. First, it cannot 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. Second, 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, and therefore, in the future time steps, content in belt $M2$ cannot be accessed and updated anymore.

From the state transition analysis mentioned earlier, 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.

D. 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 (NTM) [18] 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

$$\mathbf{r}_t = \sum_{i=0}^{M-1} w_t^r(i)\mathbf{m}_t(i) \tag{19}$$

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

$$\mathbf{c}_t = f\left(\mathbf{w}_{hc}^T\mathbf{h}_t + \mathbf{b}_c\right) \tag{20}$$

$$\mathbf{m}_t(i) = w_t^w(i)\mathbf{c}_t(i) + e_t(i)\mathbf{m}_{t-1}(i). \quad \forall i \tag{21}$$

where $w_t^r(i)$ 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-C. The dynamics of the hidden layer are

$$\mathbf{h}_t = f\left(\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T\mathbf{r}_{t-1} + \mathbf{b}_h\right). \tag{22}$$

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. Realization of the state update formula (3) of neural RAM is

$$h_n = f(e_{m_1}, e_{m_2}, e_{m_3} \ldots). \tag{23}$$

It is not hard to see that neural RAM is more powerful compared to all other architectures.

For example, in Fig. 4(b), at time t_0, memory $M00$ is stored in belt $M0$, and at time t_1, system control can directly jump to belt $M2$ to store $M20$. What is 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 above-mentioned analysis, it is not hard to see that neural RAM is the most powerful network among all the models discussed in this paper.

IV. Memory Network Taxonomy

From the analysis in Section III, 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, and the innovation of neural RAM is to remove

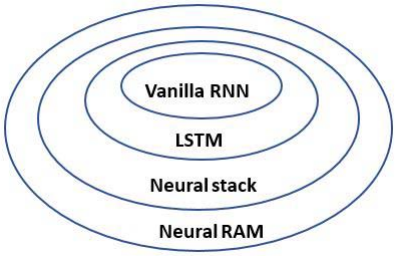


Fig. 5. Memory network taxonomy.

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 to have a different expressive power. In this section, a taxonomy of RNN 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$\subseteq$ LSTM$\subseteq$neural stack$\subseteq$neural RAM. Some classes are named after a typical model. For example, vRNN class also includes an RNN that is composed of ReLUs and initialized with the identity matrix (IRNN) [19] and highway network [20], LSTM class also includes GRU [14] and peephole network [21]. Neural stack class includes the architecture in [17] and [22]–[24]; neural RAM class includes NTM [18], differentiable neural computers (DNC) [25], enhanced LSTM [26], [27], and attention model [28]. The classification of these four types of networks is based on the their memory characteristics, i.e., internal memory, one external memory slot, external memory slots with a restricted visiting order, and 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. NTM [18] and attention model [28] belong to the same class since both of them have multiple external memory slots without restricted visiting order, although NTM [18] uses the continuous read and write head to access the memory and attention model [28] stores all past contents and train a weight to pay attention to the useful past content.

In Sections IV-A and IV-B, 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.

A. Inclusion Relationship Derivations

Theorem 1: theorem]lkofbknbg Neural RAM can be degraded to a neural stack if the following holds.

1) All the reading weights except that for the topmost memory slot are set to zeros, $w_t^r(i) = 0, if\ i \neq 0$.
2) Only the writing weight for the topmost memory slot is learned, and all others are copied from it, $w_t^r(i) = w_t^r(0)$, if $i \neq 0$.
3) In the writing process, instead of learning all the contents to be written to the stack as in (20), only the content of M0 is learned as $\mathbf{c}_t(0) = t(\mathbf{w}_{hc}^{'\mathrm{T}}\mathbf{h}_{t-1} + b_c) + \gamma\,\mathbf{m}_{t-1}(1)$ and all others are calculated as $\mathbf{c}_t(i) = \mathbf{m}_{t-1}(i-1) + \gamma\,\mathbf{m}_{t-1}(i+1)$, if $i \neq 0$.
4) Only the writing and erasing weights for the topmost element are learned, and 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, $d_t^{\text{pop}} = 0$. d_t^{push} in neural stack works as the input gate in LSTM, and $d_t^{\text{no-op}}$ in neural stack works as the forget gate in LSTM.

Theorem 3: The LSTM is degraded to the vRNN if the following holds.

1) All three gates are set as constants, $g_o = 0$, $g_i = 1$, and $g_f = 0$.
2) Weight $\mathbf{w}_{hc}$ and bias $\mathbf{b}_c$ are set as constants $\mathbf{w}_{hc} = \mathbf{I}$ and $\mathbf{b}_c = \mathbf{0}$.
3) The activation function $t(x)$ is set as linear activation function $t_1(x) = x$.

Proof: All the proofs are in Appendix B. ■

B. Mapping From Network Types to Task Types

It is not hard to see that the proposed taxonomy resembles, but it is different from, the hierarchical organization of automata: vRNN$\Leftrightarrow$Finite state machine, neural stack$\Leftrightarrow$Deterministic pushdown automaton, and neural RAM$\Leftrightarrow$Turing machine. In fact, notice that all these neural models are all universal machines, unlike the automata. The fundamental issue is that we have to find the appropriate architecture to simplify the enormous problem of learning from data efficiently. For instance, 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 also do some more sequence learning tasks, such as prediction. In this case, sequences do not need to satisfy the restrictive grammars, which will depend upon the time structure of the signal and it is unknown *a priori*. Hence, in order to make our taxonomy fit into these more general applications, we divide all the sequence learning tasks into four classes according to their memory requirements, as summarized in Table I. 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 selected: counting, counting with interference, reversing, and repeat copying. We will analyze the memory requirements of them one by one.

1) Counting: For the counting task, the input sequences are composed of a', b', and c'. The output sequence is trying to count the number of a'. 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'. 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 that the output error is almost zero.

TABLE I
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

2) Counting With Interference: For the counting with the 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', 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 inputs b and 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.

3) Reversing: The third task reverses the order of the symbols in time. 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, and thus, they can solve this task.

4) Repeat Copying: The hardest task is repeat copying, by which we mean that 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 to be stored, but also they should be retrieved more than one time, and here, the number is 3. Since all the saved information in the 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 hyperparameters. As shown in Section V, 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.

V. 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 Section IV. We also use them to visualize how each network encodes information in order to solve a problem in the Supplementary Material. Then, we analyze how to implement some basic signal processing operations with neural memory networks, and the details are in Section V-B. Finally, 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.

A. Synthetic Tasks

1) Experiment Parameters' Setting:

a) 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 is 3. All the weights are initialized as random numbers with mean 0 and variance 1, and 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 as follows.

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]

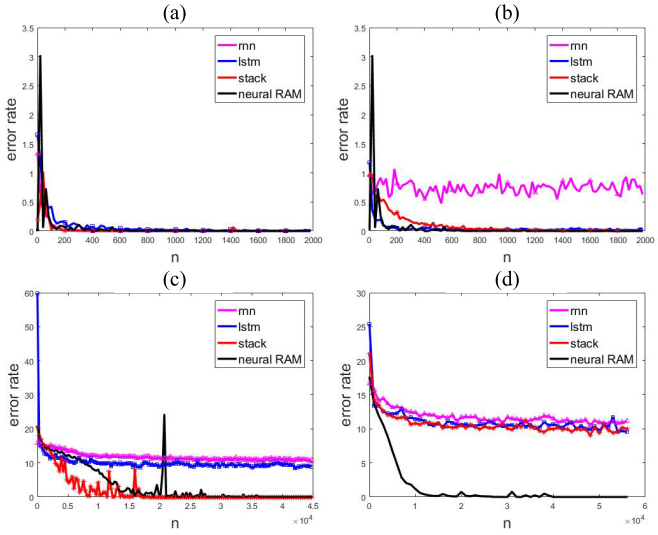


Fig. 6. Learning curve for four synthetic tasks. (a) Counting. (b) Counting with interference. (c) Reversing. (d) Repeating.

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

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]

b) 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 (mse) as the cost function. In neural RAM, the word size and memory depth are set as 16. The length of read and write vectors is also set as 16. The number of input neurons, hidden neurons, and output neurons is 6, 64, and 6, respectively. The model is trained with sequences up to the length 20. In the 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, and e are one-hot encoded with on-value 1and off-value 0; δ is encoded with on-value n and off-value 0, and n is the repeating number.

2) Experiment Result: 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 the same as in Fig. 6 except for a different non-zero error value when the network is not capable to accomplish the task.

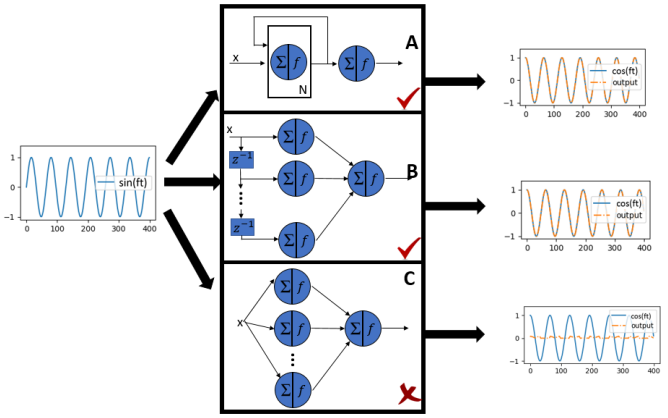


Fig. 7. Time delay network.

B. Signal Processing

In this section, we will test how different memory networks realize basic signal processing operations: addition, multiplication, time shifting, time scaling, time reversing, and signal generating. The purpose is to show the capabilities of different memory networks which are consistent with the taxonomy proposed in this paper. We use the low-dimensional signals: sine waves and cosine waves to conduct all the experiments in this section. Since realization of addition and multiplication does not need memories, we leave the analysis and experiments for them in Appendix C and save the main body of this paper for memory networks.

1) Time Shifting: In order to realize time shifting, the system should have access to The previous samples. Hence, either a TDNN or all the memory networks introduced in this paper can implement it, i.e., the mse is less than a specified threshold, here 0.01. In experiments, when we input sin(ft), the network is expected to output cos(ft). The frequencies of the input sine waves are in the range $[\pi, 4\pi]$. The sampling rate is 32 Hz. The mse of different networks is shown in Table II. The result is the average of 20 runs. Fig. 7 shows a realization for vRNN (A), TDNN (B), and feedforward network (C). In Figs. 7, 8, and 10–12, the black arrow denotes weighted connection and the bias is omitted in figures. From the results, we can see that when the network has memory (vRNN and feedforward network), time shifting can be implemented.

2) Time Scaling: For time scaling, we will talk about time contraction and expansion.

For a periodic signal, contraction is realized by the combination of time-shifting network and multiplication network. For example, in order to generate sin2t given sint, a signal containing component cost should be generated by time-shifting network, and then, multiplication, sin2t $=$ 2sintcost, should be implemented with a multiplication network (explained in Appendix C). Hence, a network without the capability of time shifting, such as feedforward network or signal multiplication,

TABLE II
MSE FOR SIGNAL PROCESSING OPERATORS

operator		one layer FF	two layer FF	TDNN	vanilla RNN	LSTM	neural stack	neural RAM
addition		**2e-6**	**1e-5**	**1e-5**	**3e-5**	**1e-5**	**2e-5**	**3e-5**
multiplication		0.35	**2e-3**	**3e-4**	**1e-4**	**2e-3**	**4e-4**	**6e-3**
time shifting		0.48	0.52	**3e-3**	**3e-5**	**4e-4**	**5e-3**	**4e-3**
signal generating		0.61	0.44	0.48	**8e-3**	**5e-4**	**3e-3**	**4e-3**
time scaling	periodic signal contraction	0.45	0.38	**3e-3**	**5e-3**	**1e-3**	**7e-3**	**2e-3**
	periodic signal expansion	0.48	0.52	0.54	0.49	**2e-3**	**3e-3**	**3e-3**
	uncorrelated signal	0.33	0.41	0.38	0.32	0.41	0.44	**4e-3**
time reversing		0.44	0.49	0.52	0.58	0.44	**1e-3**	**8e-4**

FF means feedforward network. For all the networks in this table, the number of hidden neurons are 10, output neuron is 1. For LSTM, neural stack and neural RAM, the external memories width is 10. For neural RAM, the external memory memory length is 64. In Fig.7 to Fig.12, the sizes of the networks are same as this table unless otherwise specified. From this table, we get see that the power of these networks for signal processing operators is, one layer FF⊆two layer FF⊆TDNN⊆vanilla RNN⊆LSTM⊆neural stack⊆neural RAM.

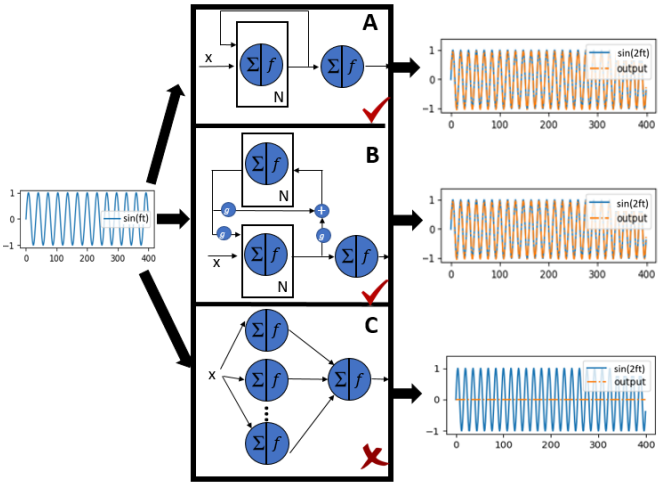


Fig. 8. Time scaling network: double frequency.

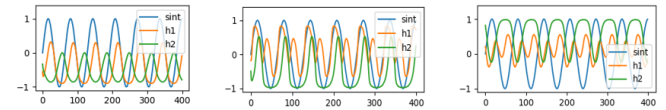


Fig. 9. Hidden units output for RNN.

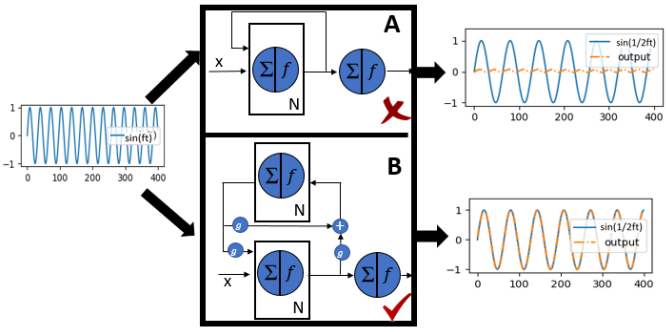


Fig. 10. Time scaling network: half frequency.

can never finish this job no matter how many hidden neurons and layers are included. In Fig. 8, we show an example for frequency doubling of a sine wave with RNN (A), LSTM (B), and feedforward network (C). In RNN and LSTM, the number of hidden units is set to 2, and the results are pretty good. The output of the network and the desired signal sin2ft are almost overlapped. However, in the feedforward network, even if we increase the number of hidden neurons to 200, it still cannot do frequency doubling.

In order to see the working details of RNN, we draw the outputs of the two hidden units in Fig. 9. The three subfigures are with different initialization for all weights and bias. From these figures, we can see that in all of them, at least one of the hidden units has a delayed version of the input signal: the orange line in the first figure and the green line in the second and third figures. Although the delayed signal does not have the exact same shape as the input signal, as long as it has cost, the filter in the multiplication network (explained in the appendix) can filter the useless components out. Hence, we can draw the conclusion that memory networks implement signal contraction for periodic signal by adopting a time-shifting network and multiplication network. It means that all the memory network with at least two hidden layers can do periodic signal contraction, as shown in Table II.

Frequency division is a much harder problem since the combination of time-shifting and multiplication networks cannot generate terms with lower frequency. Thus, signal expansion should be realized by function approximation. Since the input (sinft) and output [sin(ft/2)] are one-to-many mapping, correct prediction can only be made if the network is considering all the input points from the very first one if we use vRNN. Due to the compromise of memory depth and memory resolution, RNN with a limited number of hidden neurons cannot accomplish this task. However, LSTM can easily solve this problem since it has an external memory to help it differentiate the odd and the even input period, which makes the input/output mapping a one-to-one mapping. Fig. 10 shows an example for vRNN (A) and LSTM (B). In LSTM, the number of hidden neurons is 10, and in vRNN, the number of hidden neurons is set to 500. From Fig. 10, we can see that the output of LSTM (orange line) is overlapped with the desired signal sin(ft/2), but the vRNN cannot learn the pattern. The results of all other networks are in Table II. From the results, we can see that the mse for all the networks with external memories can finish this task with small errors.

For signal with low correlation in their time structure (the lower limit is white noise, i.e., no correlation in time) within one period, the architectures introduced before do not apply. The only way to perform time scaling for white noise is

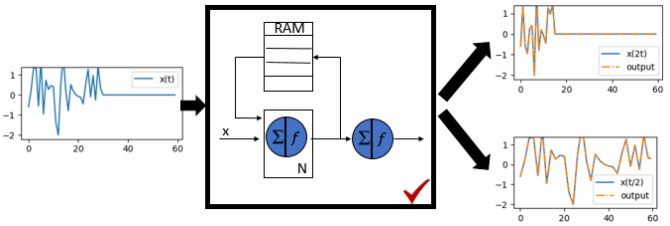


Fig. 11. Time scaling with neural RAM.

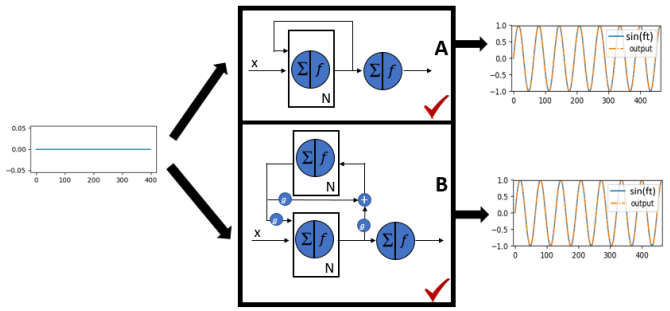


Fig. 12. Frequency generator.

to use multiple memory slots to save the waveform in a period and then output it with another frequency based on the decimation and interpolation. According to our analysis of different memory networks, neural RAM is the only network that can solve this problem. Fig. 11 shows an example of frequency doubling and dividing of the white Gaussian noise. The left figure is a randomly generated Gaussian noise $x(t)$ with length 30. The right two figures show the output of the neural RAM for desire $x(2t)$ and $x(t/2)$, respectively. From the result, we can see that neural RAM learned how to output signal with either double or half input signal frequency. Although neural RAM does not have high accuracy if the frequency change is a rational number, it is the only network that does time scaling for uncorrelated signal according to our knowledge.

3) Signal Generating: The idea to generate periodic signals with neural networks is to make the network oscillate at some resonant frequency. As long as there are at least two neurons connected with each other, the energy can flow back and forth between the neurons with a certain frequency. It can be seen as an analog of LC circuit to gain physical intuition. The difference in neural networks is that there is no resistance, so there is no dissipation of energy. Hence, the network can oscillate forever without damping if one uses infinite precision. Practically, for finite precision arithmetic, there will be a very slow degradation of the waveform characteristics. Fig. 12 shows an example of generating sine wave at a certain frequency with zero input using vRNN (A) and LSTM (B). In this example, we only use two neurons in the hidden layer to make the network oscillate. From the results, we can see that the output of both of these two networks can give good results. We also test the performance after 40 000, and the phase shift is still less than one sample. Since neural RAM and neural stack also have these interconnected neurons, they can also do signal generating with this mechanism.

4) Time Reversing: Neural stack and neural RAM are the networks suitable for time reversing since they have the external memory bank to save the input samples and output them in the reverse order. The experiment is similar to the one in Section IV-B3, and hence, we leave out the details and only present the results in Table II.

From all the experiments for signal processing operators in this section, we can verify our taxonomy.

C. 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, which can be expected because all the networks are universal machines, with different constraints created by the architectures. Hence, we will illustrate how to select the minimum network resources to accomplish the task with a relatively better performance in this section.

1) Sentiment Analysis: The first experiment is a sentiment analysis problem that will infer emotional tone of the text as negative or positive.

An example from the lmdb movie review data set [29] with negative emotion is:

"Outlandish premise that rates low on plausibility and unfortunately also struggles feebly to raise laughs or interest. Only Hawn's well-known charm allows it to skate by on very thin ice. Goldie's gotta be a contender for an actress who's done so much in her career with very little quality material at her disposal."

And a positive text is:

"I absolutely loved this movie. I bought it as soon as I could find a copy of it. This movie had so much emotion, and felt so real, I could really sympathize with the characters. Every time I watch it, the ending makes me cry. I can really identify with Busy Phillip's character, and how I would feel if the same thing had happened to me. I think that all high schools should show this movie, maybe it will keep people from wanting to do the same thing. I recommend this movie to everybody and anybody. Especially those who have been affected by any school shooting. It truly is one of the greatest movies of all time."

In our experiments, the number of input neurons, hidden neurons, and output neurons is 50, 64, and 2, respectively, 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 [29] 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 about the paragraph's tone will be made at the end. The output is [1, 0] for the positive text and [0, 1] for the negative text. The data set adopted here is the lmdb movie review data [30], which has 12 500 positive reviews and 12 500 negative reviews. Here, we use 11 500 reviews for training and 1000 data for testing. In our experiments, the nonlinear activation functions

TABLE III
ERROR RATE FOR MOVIE REVIEW

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

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 is 4 and 1, respectively. The results are the average of 20 runs with random initializations.

In order to judge the emotional tone of the text at its end, an external memory whose value would be affected by some key words is useful. 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. However, the memory bank that can store multiple contents does not show more advantages here. We test these networks' performance on the lmdb movie review data set [29]. The results are in Table III, which shows that vRNN performs worst. LSTM, neural stack, and neural RAM have similar performances. Thus, our analysis is verified.

2) Question Answering: Next, we investigate the performance of these four networks on three question answering tasks from the bAbI data set [31]. 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 traveled 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 traveled to the hallway. What is more, since the machine does not 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.

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 in [25], and the results for these two networks are from [25]. 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 is 150, 64, and 150, respectively. The memory width for the neural stack is 64.

From the results in Table IV, we can see that neural RAM achieves the best performance. One thing to be mentioned here is that the variance is larger than all others, although the mean error rate of the neural RAM is the lowest. We believe that the reason for this is the complexity of the NTM, which leads to too many local minima. Since the point here is to check the capabilities of different neural memory networks, we should understand that architectures suitable for this problem should always belong to neural RAM class. References [32]–[34] show the results for some other architectures from this class.

TABLE IV
ERROR RATE FOR THREE TASKS FROM BABI TASKS

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

VI. 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 the 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.

APPENDIX A
NETWORK ARCHITECTURE COMPONENTS

A. LSTM

The three gates are in LSTM are calculated as follows:

$$g_{i,t} = s\left(\mathbf{w}_{hg_i}^T \mathbf{h}_{t-1} + \mathbf{w}_{xg_i}^T \mathbf{x}_t + \mathbf{b}_{g_i}\right) \tag{24}$$

$$g_{f,t} = s\left(\mathbf{w}_{hg_f}^T \mathbf{h}_{t-1} + \mathbf{w}_{xg_f}^T \mathbf{x}_t + \mathbf{b}_{g_f}\right) \tag{25}$$

$$g_{o,t} = s\left(\mathbf{w}_{hg_o}^T \mathbf{h}_{t-1} + \mathbf{w}_{xg_o}^T \mathbf{x}_t + \mathbf{b}_{g_o}\right) \tag{26}$$

where $\mathbf{w}_{hg_i}$, $\mathbf{w}_{hg_f}$, and $\mathbf{w}_{hg_o}$ are $K_h \times 1$ weights, $\mathbf{w}_{hg_i}$, $\mathbf{w}_{hg_f}$, and $\mathbf{w}_{hg_o}$ 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.

B. Neural Stack

Neural network interacts with the stack memory by d_t^{push}, d_t^{pop}, $d_t^{\text{no-op}}$, $\mathbf{c}_t$, and $\mathbf{r}_t$. According to [22], 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 $d_{\text{push}} = 1$, the current vector will be pushed into the stack as it is, if $d_{\text{push}} = 0.8$, the current vector is first multiplied by 0.8 and then pushed into the stack.

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

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

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

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

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

C. Neural RAM

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

$$\boldsymbol{w}_t^r = f\left(\mathbf{w}_{ha}^T \mathbf{h}_{t-1} + \mathbf{b}_r\right) \tag{27}$$

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

$$\boldsymbol{w}_t^w = s\left(\mathbf{w}_{hb}^T \mathbf{h}_{t-1} + \mathbf{w}_{xb}^T \mathbf{x}_t + \mathbf{b}_w\right) \tag{28}$$

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

The erase weighting is learned as

$$\mathbf{e}_t = s\left(\mathbf{w}_{he}^T \mathbf{h}_{t-1} + \mathbf{w}_{xe}^T \mathbf{x}_t + \mathbf{b}_e\right) \tag{29}$$

where $\mathbf{e}_t = [e_t(1), e_t(2), \ldots, e_t(M)]^T$, $\mathbf{w}_{\text{he}}$ 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 NTM [18], $e_t(i)$ 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, and 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 and 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 first blend with the previous weight and then shifted for several steps

$$w_t^r(i) = g_t w_{t-1}^r(i) + (1 - g_t) w_t^r(i) \tag{30}$$

$$w_t^r(i) = w_t^r([i-n]_M) \tag{31}$$

where g_t is the gate to balance the previous weight and current weight, n is the shifting steps, and $[i-n]_M$ means the circular shift for M entities. Since the shifting operation is not differentiable, the method in [18] should be utilized as an approximation.

Another example is [25], which improves the performance even more. To be specific, for reading, a matrix to remember the order of memory locations that 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 that they are written to. 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 the 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.

APPENDIX B
THEOREM PROOFS

A. 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

$$w_t^r(i) = \begin{cases} 0, & \text{if } i \neq 0 \\ t\left(\mathbf{w}_{ha}^{'T} \mathbf{h}_{t-1} + b_a\right), & \text{if } i = 0 \end{cases} \tag{32}$$

where $\mathbf{w}'_{\text{ha}}$ is $K_h \times 1$ vector and b_a is the scalar. Equation (32) is a special case of (27)

$$\mathbf{c}_t(0) = t\left(\mathbf{w}_{hc}^{'\text{T}} \mathbf{h}_{t-1} + b_c\right) + \gamma\, \mathbf{m}_{t-1}(1) \tag{33}$$

and all others are calculated as

$$\mathbf{c}_t(i) = \mathbf{m}_{t-1}(i-1) + \gamma\, \mathbf{m}_{t-1}(i+1), \quad \text{if } i \neq 0. \tag{34}$$

In the writing process, (33) and (34) can be seen as a special case of (20) because $\mathbf{h}_{t-1}$ in (20) is a function of $\mathbf{m}_{t-1}$. Substituting (33) and (34) into the memory update equation of neural RAM (21), we get

$$\begin{aligned} &\mathbf{m}_t(i) \\ &= w_t^w(i)\mathbf{c}_t(i) + e_t(i)\mathbf{m}_{t-1}(i) \\ &= \begin{cases} w_t^w(i) t(\mathbf{w}_{hc}^{'\text{T}} \mathbf{h}_{t-1} + b_c) + \\ \gamma\, w_t^w(i)\mathbf{m}_{t-1}(1) + e_t(i)\mathbf{m}_{t-1}(i), & \text{if } i = 0 \\ w_t^w(i)\mathbf{m}_{t-1}(i-1) + \\ \gamma\, w_t^w(i)\mathbf{m}_{t-1}(i+1)\mathbf{c}_t(i) + e_t(i)\mathbf{m}_{t-1}(i), & \text{otherwise} \end{cases} \\ &= \begin{cases} w_t^w(i) t(\mathbf{w}_{hc}^{'\text{T}} \mathbf{h}_{t-1} + b_c) + \\ \gamma\, w_t^w(i)\mathbf{m}_{t-1}(1) + e_t(i)\mathbf{m}_{t-1}(i), & \text{if } i = 0 \\ w_t^w(i)\mathbf{m}_{t-1}(i-1) + \\ \gamma\, w_t^w(i)\mathbf{m}_{t-1}(i+1)\mathbf{c}_t(i) + e_t(i)\mathbf{m}_{t-1}(i), & \text{otherwise.} \end{cases} \end{aligned} \tag{35}$$

Finally since (4) and (4) can be seen as a special case of (28) and (29), the neural stack can be treated as a special case of neural RAM. Comparing the memory writing operation of neural stack (15) and neural RAM (35), we can see that $w_t^w(0)$, $\gamma\, w_t^w(0)$, and $e_t(0)$ work as the push, pop, and no-operate operations, respectively. Comparing the memory reading operation of neural stack and neural RAM, we can see that $w_t^r(0)$ in neural RAM (32) works as the output gate in neural stack (17).

B. Proof of Theorem 2

The dynamic of LSTM is

$$\mathbf{h}_t = f\left(\mathbf{w}_{xh}^T \mathbf{x}_t + \mathbf{w}_{rh}^T g_{o,t} \mathbf{r}_t + \mathbf{b}_h\right) \tag{36}$$

and the dynamic of neural stack is

$$\mathbf{h}_t = g\left(\mathbf{w}_{xh}^T \mathbf{x}_t + \mathbf{w}_{rh}^T \mathbf{r}_t + \mathbf{b}_h\right). \tag{37}$$

According to (36) and (37), the dynamics of the neural stack have a similar form as LSTM except for the reading vector, i.e., the reading vector is $\mathbf{r}_t = g_o \mathbf{m}_t(0)$. If we set the pop signal as zero, $d_t^{\text{pop}} = 0$, and no operation on the stack contents except for the topmost elements is available, then

$$\begin{aligned} \mathbf{m}_t(0) &= d_t^{\text{push}} \mathbf{c} + d_t^{\text{no-op}} \mathbf{m}_{t-1}(0) \\ \mathbf{m}_t(i) &= 0, \quad \text{if } i \neq 0. \end{aligned}$$

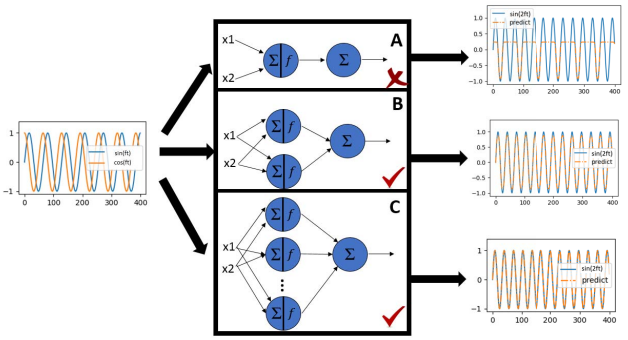


Fig. 13. Multiplication neural network.

Since d_t^{push} and $d_t^{\text{no-op}}$ are calculated in the same way that the input gate $g_{i,t}$ and forget gate $g_{f,t}$ are calculated in LSTM as shown in (24) and (25), the read vector would be

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

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

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

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

C. Proof of Theorem 3

Compared to vRNN, LSTM introduces an external memory and the gate operation mechanism. Thus, 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:

$$\begin{aligned}\mathbf{h}_t &= t\big(\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T g_o\mathbf{r}_t + \mathbf{b}_h\big) && (38)\\ &= t\big[\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T\mathbf{r}_r + \mathbf{b}_h\big] && (39)\\ &= t\big[\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T\mathbf{m}_t + \mathbf{b}_h\big] \\ &= t\big[\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T(g_i\mathbf{c}_t + g_f\mathbf{m}_{t-1}) + \mathbf{b}_h\big] \\ &= t\big(\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T\mathbf{c}_t + \mathbf{b}_h\big) && (40)\\ &= t\big[\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T t_1\big(\mathbf{w}_{hc}^T\mathbf{h}_{t-1} + \mathbf{b}_c\big) + \mathbf{b}_h\big] && (41)\\ &= t\big(\mathbf{w}_{xh}^T\mathbf{x}_t + \mathbf{w}_{rh}^T\mathbf{h}_{t-1} + \mathbf{b}_h\big). && (42)\end{aligned}$$

Here, (39) is due to $g_o = 0$, (40) is due to $g_i = 1$ and $g_f = 0$, and (42) 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

$$\begin{aligned}\mathbf{w}_{\text{hc}} &= \mathbf{I} \\ \mathbf{b}_c &= \mathbf{0} \\ t_1(x) &= x.\end{aligned}$$

Since (38) is the dynamic of LSTM and (42) is the dynamic of vRNN, the argument that vRNN is a special case of LSTM is proven.

APPENDIX C
ADDITION AND MULTIPLICATION

By addition of signals, we mean if the two sources of the input of the network are x_1 and x_2, the expected output is $y = c_1x_1 + c_2x_2$, where c_1 and c_2 are coefficients and n is the time index. Since there is an addition operator in each neuron in all the memory networks, the addition of signals can be easily realized. However, there is no explicit multiplication operator in neural networks, the multiplication ($y = x_1x_2$) should be implemented through the nonlinear activation function and a weighted summation in the following layer. To be specific, in the first step, the multiplication term is generated by the nonlinear activation function. For example, the output of neuron 1 in the first hidden layer is

$$\begin{aligned}y_1 &= f(c_1x_1 + c_2x_2 + b) && (43)\\ &= f(x_0) + f'(x_0)(c_1x_1 + c_2x_2 - x_0) \\ &\quad + \underbrace{f''(x_0)(c_1x_1 + c_2x_2 - x_0)^2}_{x_1x_2} \\ &\quad + \underbrace{f'''(x_0)(c_1x_1 + c_2x_2 - x_0)^3}_{x_1x_2} + \cdots && (44)\\ &= c_1^{0'} + c_1^{1'}x_1 + c_1^{2'}x_2 + c_1^{3'}x_1x_2 + c_1^{4'}x_1^2x_2 + \cdots && (45)\end{aligned}$$

$c_1^{0'}, c_1^{1'}, \ldots,$ are the coefficient after combining of like terms. The product appears in all the higher order (more than first) derivative terms. Next, in order to filter out all other terms except for $c_1^{3'}x_1x_2$, a weighted combination of the outputs of different neurons is implemented

$$\begin{aligned} w_1c_1^{0'} + w_2c_2^{0'} + \cdots + w_nc_n^{0'} &= 0 \\ w_1c_1^{1'} + w_2c_2^{1'} + \cdots + w_nc_n^{1'} &= 0 \\ &\vdots \\ w_1c_1^{3'} + w_2c_2^{3'} + \cdots + w_nc_n^{3'} &= 1 \\ w_1c_1^{4'} + w_2c_2^{4'} + \cdots + w_nc_n^{4'} &= 0 \\ &\vdots \end{aligned} \quad (46)$$

where n is the number of hidden neurons in the first hidden layer. Equation (46) is underdetermined (it includes infinite number of equations but n unknowns). Since we cannot reduce the number of equations, we have to increase the number of unknowns which means increasing the number of neurons, which leads to better solutions. Fig. 13 shows a multiplication example. The input to the network is $x_1 = \text{sint}$ and $x_2 = \text{cost}$. The desired network output is $y = \text{sin2t}$, which is the product of the two input signals. We are trying to test the capabilities of the feedforward network only with one nonlinear active function in the hidden layer, with two nonlinear activation functions and a weighted sum of them, and with 20 nonlinear activation functions and a weighted sum of them. Fig. 13 shows the results.

Network A has the nonlinear activate function, and hence, it can generate the higher order derivative terms. Since it only has one unit in the next layer, the product term x_1x_2 cannot be filtered through it. Hence, it cannot successfully generate sin2t. Compared with A, network B has two hidden neurons, which

means that there are two unknowns in (46). Since the coordinates of the higher order derivative terms are relatively small, the output of network B can follow the trend of sin2t, even though outputs around peaks are not well learned. Network C tries to improve the filter performance by increasing the number of hidden neurons to 20. Since there are more unknowns in (46), the solution can be approximated better as shown in the output. From the earlier analysis along with the results in Fig. 13, we can draw the conclusion that multiplication can be realized by the nonlinear activation function and the weighted combination of the hidden layer neurons. The more hidden neurons involved, the better the performance we get.

REFERENCES

[1] J. L. Elman, "Finding structure in time," *Cognit. Sci.*, vol. 14, no. 2, pp. 179–211, Apr./Jun. 1990.

[2] M. I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Proc. 8th Annu. Conf. Cogn. Sci. Soc.* Hillsdale, NJ, USA: Erlbaum, 1986, pp. 531–546.

[3] K. Doya, "Universality of fully connected recurrent neural networks," Dept. Biol., Univ. California San Diego, San Diego, CA, USA, Tech. Rep., 1993.

[4] C. W. Omlin and C. L. Giles, "Constructing deterministic finite-state automata in recurrent neural networks," *J. ACM*, vol. 43, no. 6, pp. 937–972, Nov. 1996.

[5] F. A. Gers and E. Schmidhuber, " LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1333–1340, Nov. 2001.

[6] P. Rodriguez, "Simple recurrent networks learn context-free and context-sensitive languages by counting," *Neural Comput.*, vol. 13, no. 9, pp. 2093–2118, Sep. 2001.

[7] J. Schmidhuber, F. Gers, and D. Eck, "Learning nonregular languages: A comparison of simple recurrent networks and LSTM," *Neural Comput.*, vol. 14, no. 9, pp. 2039–2041, 2002.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[9] J. Collins, J. Sohl-Dickstein, and D. Sussillo, "Capacity and trainability in recurrent neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[10] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[11] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[12] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, Jun. 2015, pp. 2342–2350.

[13] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," Dec. 2014, *arXiv:1412.3555*. [Online]. Available: https://arxiv.org/abs/1412.3555

[14] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," Jun. 2014, *arXiv:1406.1078*. [Online]. Available: https://arxiv.org/abs/1406.1078

[15] D. Yogatama *et al.*, "Memory architectures in recurrent neural network language models," in *Proc. Int. Conf. Learn. Representations, (ICLR)*, 2018, pp. 1–10. [Online]. Available: https://openreview.net/forum?id=SkFqf0lAZ

[16] B. de Vries and J. C. Principe, "The gamma model—A new neural model for temporal processing," *Neural Netw.*, vol. 5, no. 4, pp. 565–576, Jul./Aug. 1992.

[17] A. Joulin and T. Mikolov, "Inferring algorithmic patterns with stack-augmented recurrent nets," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 190–198.

[18] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," Oct. 2014, *arXiv:1410.5401*. [Online]. Available: https://arxiv.org/abs/1410.5401

[19] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," Apr. 2015, *arXiv:1504.00941*. [Online]. Available: https://arxiv.org/abs/1504.00941

[20] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," May 2015, *arXiv:1505.00387*. [Online]. Available: https://arxiv.org/abs/1505.00387

[21] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proc. IEEE-INNS-ENNS Int. Joint Conf. Neural Netw.*, Jul. 2000, vol. 3, pp. 189–194.

[22] G.-Z. Sun, "Learning context-free grammar with enhanced neural network pushdown automaton," in *Proc. IEE Colloq. Grammatical Inference, Theory, Appl. Alternatives*, Apr. 1993, pp. P6-1–P6-13.

[23] G. Z. Sun, C. L. Giles, H. H. Chen, and Y. C. Lee, "The neural network pushdown automaton: Model, stack and learning simulations," Nov. 2017, *arXiv:1711.05738*. [Online]. Available: https://arxiv.org/abs/1711.05738

[24] E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom, "Learning to transduce with unbounded memory," in *Proc. Adv. Neural Inf. Processing Syst. (NIPS)*, 2015, pp. 1828–1836.

[25] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, Oct. 2016.

[26] A. Graves, "Generating sequences with recurrent neural networks," Aug. 2013, *arXiv:1308.0850*. [Online]. Available: https://arxiv.org/abs/1308.0850

[27] J. Weston, S. Chopra, and A. Bordes, "Memory networks," Oct. 2014, *arXiv:1410.3916*. [Online]. Available: https://arxiv.org/abs/1410.3916

[28] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 2440–2448.

[29] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. Empirical Methods Natural Lang. Process. (EMNLP)*, Oct. 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

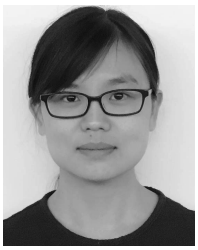
[30] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics, Hum. Lang. Technol.*, Portland, Oregon, USA, Jun. 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[31] J. Weston *et al.*, "Towards ai-complete question answering: A set of prerequisite toy tasks," Feb. 2015, *arXiv:1502.05698*. [Online]. Available: https://arxiv.org/abs/1502.05698

[32] A. Kumar *et al.*, "Ask me anything: Dynamic memory networks for natural language processing," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jun. 2016, pp. 1378–1387.

[33] M. Hu, Y. Peng, Z. Huang, X. Qiu, F. Wei, and M. Zhou, "Reinforced mnemonic reader for machine reading comprehension," May 2017, *arXiv:1705.02798*. [Online]. Available: https://arxiv.org/abs/1705.02798

[34] B. Pan, H. Li, Z. Zhao, B. Cao, D. Cai, and X. He, "Memen: Multi-layer embedding with memory networks for machine comprehension," Jul. 2017, *arXiv:1707.09098*. [Online]. Available: https://arxiv.org/abs/1707.09098



Ying Ma received the B.E. degree in electronic engineering from the Changchun Institute of Technology, Changchun, China, in 2011. She is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, University of Florida, Gainesville, FL, USA, under the supervision of Prof. J. Principe.

From July 2015 to July 2016, she visited the Department of Electrical and Computer Engineering, University of Southampton, Southampton, U.K. as a Visiting Student under the supervision of Prof. L. Hanzo and Prof. S. Chen. She was a Summer Intern at Apple Siri understanding in 2019. Her current research interests include machine learning and deep learning, especially sequence learning with application in NLP and video games.

Jose C. Principe (M'83–SM'90–LF'00) received the Ph.D. degree in electrical and computer engineering from the University of Florida, Gainesville, FL, USA, in 1979.

He is currently a Distinguished Professor of electrical and computer engineering with the University of Florida, where he is also the Don D. and Ruth S. Eckis Chair with the Department of Electrical and Computer Engineering and the Department of Biomedical Engineering. He is also the Founding Director of the Computational NeuroEngineering Laboratory, University of Florida. His current research interests include advanced signal processing and machine learning, brain machine interfaces, and the modeling and applications of cognitive systems.