Turing Completeness of Bounded-Precision Recurrent Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Previous works have proved that recurrent neural networks (RNNs) are Turing-complete. However, in the proofs, it is assumed that the RNNs can have neurons with unbounded precision, which is neither practical in implementation nor biologically plausible. To remove this assumption, we propose a dynamically growing memory module made of neurons of fixed precision. The memory module dynamically recruits new neurons when more memories are needed, and releases them when memories become irrelevant. To illustrate the memory module's capacity, we prove that a 54-neuron bounded-precision RNN with growing memory modules can simulate any Turing Machines. Furthermore, we analyze the Turing completeness of both unbounded-precision RNNs and bounded-precision RNNs, revisiting and extending the theoretical foundation of RNNs.

1 Introduction

Symbolic (such as Turing machines) and sub-symbolic processing (such as adaptive neural networks) are two competing methods for representing and processing information, each with its pros and cons. An ultimate way to combine symbolic and sub-symbolic capabilities is by enabling the running of algorithms on a neural substrate, which means a neural network that can simulate a Universal Turing Machine. Previous works [1, 2, 3] have shown that this is possible – there exists a recurrent neural network (RNN) that can simulate a Universal Turing Machine. However, these proofs assumed a couple of neurons with unbounded precision that hold the same number of symbols used when processing the Turing machine's tape. Here we provide another simulation of Turing machines by an RNN with bounded-precision neurons only.

The general idea works as follows: the Turing machine's tape is stored in a growing memory module - a stack of neurons with pushing and popping operations controlled by neurons in the RNN. The size of the growing memory module is determined by the usage of the tape – the RNN dynamically recruits new neurons when more memory is needed (i.e. for a task that is more memory-intensive) and releases them when memories become irrelevant. The neurons in the stack (except the top neuron) are not updated in the steps of RNN, saving computational cost for memories that are not in the focus of computing and thus do not require changing. The capability of the growing memory module is illustrated by the existence of a 54-neuron bounded-precision RNN with growing memory modules that can simulate any Turing machines.

[4, 5] proposed an RNN with a memory bank, called Neural Turing Machine, that is fully differentiable and thus can be trained by gradient descent, sharing similarities with previous work [6]. Though inspired by Turing Machines, bounded-precision Neural Turing Machines are not Turing-complete since it has a fixed-size memory, but can likely be used for training the one we propose here.

Our proposed growing memory module is inspired by biological memory systems. The process of dynamically recruiting new neurons when more memory is necessary is also observed in biological

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

memory systems. Neurogenesis is the process by which new neurons are produced in the central nervous system. It is most active during early development, but continuous throughout life. In adult vertebrates, neurogenesis is known to occur in the dentate gyrus (DG) of the hippocampal formation [7] and the subventricular zone (SVZ) of the lateral ventricles [8]. Since DG is well-known in neuroscience for its role in pattern separation for memory encoding [9, 10], this suggests that biological memory systems also dynamically recruit new neurons. The rate of neurogenesis in adult mice has been shown to be higher if they are exposed to a wider variety of experiences [11]. This further suggests a role for self-regulated neurogenesis in scaling up the number of new memories that can be encoded and stored during one's lifetime without catastrophic forgetting of previously consolidated memories. Besides the mechanism of recruiting new neurons, the process of storing neurons in the growing memory module also shares some similarities with biological memory consolidation, a process by which short-term memory is transformed into long-term memory [12, 13]. Compared to short-term memory, long-term memory is more long-lasting and robust to interference. This is similar to the neurons stored in the growing memory module – the values of these neurons (except the top neuron in the stack) remain unchanged and cannot be interfered by the RNN, providing a mechanism to store information stably.

Simulation of a Turing machine by an RNN with growing memory modules poses a practical and biologically inspired way to combine symbolic and sub-symbolic capabilities. All neurons in the RNN and growing memory modules have fixed precision, and the enhanced RNN can thus be implemented easily. The number of neurons required in the RNN is constant in the tape's length, and the time complexity of the simulation is linear in the number of steps required for the Turing machine to compute the output. And most interestingly, the number of computations required for the simulation does not grow with the memory size or the tape's length since neurons in the growing memory module (except the top neuron in the stack) are not updated.

Besides the design of the growing memory module, this paper also analyzes the Turing completeness of both unbounded-precision RNN and bounded-precision RNN, revisiting and extending the theoretical foundation of RNNs. For unbounded-precision RNNs, we prove that there exists a 40-neuron unbounded-precision RNN that is Turing-complete, which is the smallest Turing-complete RNN to date. For bounded-precision RNNs, we analyze the relationship between the number of neurons and the precision of an RNN when simulating a Turing machine, showing that the number of bounded-precision neurons required to simulate a Turing machine is linear in the tape's length.

The remainder of the paper is structured as follows. Section 2 describes the preliminary of the paper, including the definition of Turing machines and RNNs. Section 3 revisits and extends theories relating to simulating a Turing machine with unbounded-precision RNNs, proving the existence of a 40-neuron unbounded-precision RNN that is Turing-complete. Section 4 presents the growing memory module and proves the existence of a 54-neuron bounded-precision RNN with two growing memory modules that is Turing-complete. Section 5 establishes new theories relating to the relationship between the number of neurons and the precision of an RNN when simulating a Turing machine. Section 6 concludes the paper.

2 Background and Notation

A Turing machine is a 7-tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where Q is a finite set of states, Σ is a finite set of input symbols, Γ is a finite set of tape symbols (note that $\Sigma \subset \Gamma$), $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition rule, $q_0 \in Q$ is the initial starting state, B is the blank symbol (note that $B \in \Gamma$ but $B \notin \Sigma$), and $F \subset Q$ is the set of final state. We only consider deterministic Turing machines in this paper. The *instantaneous description* of a Turing machine can be represented by a 3-tuple $(q, t_l, t_r) \in (Q, \Gamma^*, \Gamma^*)$ where q denotes the state, t_l and t_r denotes the string of symbols in the left and right tape respectively. We assume the leftmost symbol is the closest symbol to the read/write head for both t_l and t_r (that is, the left tape is reversed in the representation) and the infinite blank symbols are omitted. The set of all possible instantaneous description is denoted as $\mathcal{X} := (Q, \Gamma^*, \Gamma^*)$. The *complete dynamic map* of $\mathcal{M}$, denoted as $\mathcal{P}_{\mathcal{M}} : \mathcal{X} \to \mathcal{X}$, is defined by: 1. determinate the next transition by $q', r', d = \delta(q, r)$ where r is the leftmost symbol in t_l, denoted by $t_{l,1}$, and q' is the next state; 2. replace r by r'; 3. move $t_{l,1}$ to the leftmost of t_r if $d = L$ and $t_{r,1}$ to the leftmost of t_l if $d = R$ (If there are no symbols left in t_l or t_r, append a blank symbol B to it). The *partial input-output function* of $\mathcal{M}$, denoted as $\mathcal{P}^*_{\mathcal{M}} : \mathcal{X} \to \mathcal{X}$, is defined by applying $\mathcal{P}_{\mathcal{M}}$ repeatedly until $q \in F$, and is undefined if it is not possible to have $q \in F$ by applying $\mathcal{P}_{\mathcal{M}}$ repeatedly.

A recurrent neural network (RNN) is a neural network consisting of n neurons. The value of neuron i at time $t \in \{1, 2, ...\}$, denoted as $x_i(t) \in \mathbb{Q}$ ($\mathbb{Q}$ is the set of rational numbers), is computed by an affine transformation of the values of all neurons in the previous state followed by an activation function σ, i.e. $x_i(t) = \sigma(\sum_{j=1}^{n} w_{ij} x_j(t) + b_i)$, where w_{ij} is the weight and b_i is the bias; or in vector form:

$$\mathbf{x}(t) = \sigma(W\mathbf{x}(t-1) + \mathbf{b}), \tag{1}$$

where $\mathbf{x}(t) \in \mathbb{Q}^n$, $W \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^n$. This defines a mapping $\mathcal{T}_{W,\mathbf{b}} : \mathbb{Q}^n \to \mathbb{Q}^n$ which characterizes an RNN. Also, we only consider the saturated-linear function in this paper; that is:

$$\sigma(x) := \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } 0 \le x \le 1 \\ 1, & \text{if } x > 1. \end{cases} \tag{2}$$

We call a neuron $x_i(t)$ to have precision p in base b if for all $t > 0$, $x_i(t)$ can be expressed as $\sum_{k=1}^{p} \frac{a_k(t)}{\prod_{j=1}^{k} b_j(t)}$ for some strings $a(t) \in \{0, 1, ..., b\}^p$ and $b(t) \in \{1, ..., b\}^p$.

For a string a, we use a_k to denote the k^{th} symbol in a and $a_{j:k}$ to denote the string $a_j a_{j+1} ... a_k$. For a function f that maps from a set $\mathbb{Y}$ to a subset of $\mathbb{Y}$, we denote f^n as the n^{th} iterate of f where $1 \le n < \infty$. For any two vectors $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$, we denote $\mathbf{x} \oplus \mathbf{y} \in \mathbb{R}^{m+n}$ as the concatenation of the two vectors.

3 Turing Completeness of Unbounded-Precision RNNs

To simulate a Turing machine $\mathcal{M}$ by an RNN, we first consider how to encode the instantaneous description $(q, t_l, t_r) \in \mathcal{X}$ by a vector of rational numbers, with which an RNN can be initialized. For the state $q \in Q$, we encode it with $\lceil \log_2 |Q| \rceil$ binary values, denoted as $\rho^q : Q \to \{0, 1\}^{\lceil \log_2 |Q| \rceil}$, with each possible combination of binary values representing a specific state.

For the left tape $t_l \in \Gamma^*$ and the right tape $t_r \in \Gamma^*$, we use fractal encoding to encode them into two rational numbers. The fractal encoding we use is similar to [1, 2], but we generalize here to an arbitrary number of symbols. Without loss of generosity, assume that the tape symbols are $\Gamma = \{1, 2, ..., |\Gamma|\}$. Then, define the fractal encoding $\rho^t : \Gamma^* \to \mathbb{Q}$ by:

$$\rho^t(t_m) := \left(\sum_{i=1}^{|t_m|} \frac{2t_{m,i} - 1}{(2|\Gamma|)^i} \right) + \frac{B}{(2|\Gamma|)^{|t_m|} \cdot (2|\Gamma| - 1)}. \tag{3}$$

Example. Let $|\Gamma| = 4$, $t_l = (2, 3, 4, 2, 3)$, and $B = 1$. Then $\rho^t(t_l) = \frac{3}{8} + \frac{5}{8^2} + \frac{7}{8^3} + \frac{3}{8^4} + \frac{5}{8^5} + \frac{1}{8^6} + \frac{1}{8^7} + \frac{1}{8^8} + ... = \frac{3}{8} + \frac{5}{8^2} + \frac{7}{8^3} + \frac{3}{8^4} + \frac{5}{8^5} + \frac{1}{8^5 \cdot 7}$. Note that the last term in (3) represents the infinite blank symbols of a tape.

The particular choice of this fractal encoding is due to the easy manipulation of the top symbols, and the details can be found in Appendix A. The possible values of fractal encoding also have a close relationship with the Cantor set [1, 2].

However, this fractal encoding assumes that neurons can have the same precision as $|t_m|$, the tape's size, in base $2|\Gamma|$. As the tape in a Turing machine has an unbounded length, it means that we require neurons with unbounded precision. Note that this is different from infinite precision - at any steps of the Turing machine, the precision of $\rho^t(t_m)$ is finite. Still, the tape's size may be large, and it is difficult to compute with the high-precision encoding. We will discuss how to remove this unbounded-precision assumption in Section 4.

Finally, we need to have a specific encoding for the top symbol in each tape: $\rho^r : \Gamma \to \{0, 1\}^{|\Gamma|-1}$, defined by:

$$\rho^r_i(s) := 1\{s > i\}, \tag{4}$$

where $1 \le i \le |\Gamma| - 1$. That is, the i coordinate of $\rho^r(s)$ is one if and only if the symbol s has a value larger than i. This fully encodes the symbol s and is used to encode both $t_{l,1}$ and $t_{r,1}$.

Together, define the encoding function $\rho : \mathcal{X} \to \mathbb{Q}^{2|\Gamma| + \lceil \log_2 |Q| \rceil + |Q||\Gamma| + 5}$ by:

$$\rho(q, t_l, t_r) = \rho^q(q) \oplus \rho^t(t_l) \oplus \rho^t(t_r) \oplus \rho^r(t_{l,1}) \oplus \rho^r(t_{r,1}) \oplus \mathbf{0}, \tag{5}$$

where $\mathbf{0}$ is a zero vector of size $|Q||\Gamma| + 5$. Also, let $\rho^{-1} : \rho(\mathcal{X}) \to \mathcal{X}$ be the function such that $\rho^{-1}(\rho(x)) = x$ for all $x \in \mathcal{X}$ (note that ρ is injective), and we call ρ^{-1} the decoder function.

It should be noted that $\rho^q(q) \oplus \rho^t(t_l) \oplus \rho^t(t_r)$ is sufficient to decode the instantaneous description. We include $\rho^r(t_{l,1}) \oplus \rho^r(t_{r,1}) \oplus \mathbf{0}$ only because it facilitates the operation of the RNN. For example, some neurons that are initialized with the zero values will compute the values required in the intermediate steps of updates.

Given an instantaneous description $x \in \mathcal{X}$, we can then initialize neurons in an RNN with values $\rho(x)$. Then, it is possible to construct the parameters of RNN such that the update given by the RNN on these neurons is the same as the update given by the Turing machine:

Theorem 1. *Given a Turing machine $\mathcal{M}$, there exists an injective function $\rho : \mathcal{X} \to \mathbb{Q}^N$ and an n-neuron unbounded-precision RNN $\mathcal{T}_{W,\mathbf{b}} : \mathbb{Q}^n \to \mathbb{Q}^n$, where $n = 2|\Gamma| + \lceil \log_2 |Q| \rceil + |Q||\Gamma| + 5$, such that for all instantaneous descriptions $x \in \mathcal{X}$,*

$$\rho^{-1}(\mathcal{T}^3_{W,\mathbf{b}}(\rho(x))) = \mathcal{P}_{\mathcal{M}}(x). \tag{6}$$

The proof can be found in Appendix A. In other words, for any Turing machine $\mathcal{M}$, there exists an RNN such that every three steps of the RNN yield the same result as one step of the Turing Machine. This means that we can simulate any Turing machines using an RNN with a linear time. To be specific, the *partial input-output function* of an RNN, denoted as $\mathcal{T}^*_{W,\mathbf{b}} : \mathbb{Q}^N \to \mathbb{Q}^N$, is defined by applying $\mathcal{T}^3_{W,\mathbf{b}}$ repeatedly until $q \in F$ (where q is the state that the RNN simulates), and is undefined if it is not possible to have $q \in F$ by applying $\mathcal{T}^3_{W,\mathbf{b}}$ repeatedly. This is similar to the definition of $\mathcal{P}^*_{\mathcal{M}}$. Based on this definition and Theorem 1, it follows that:

Corollary 1.1. *Given a Turing machine $\mathcal{M}$, there exists an injective function $\rho : \mathcal{X} \to \mathbb{Q}^n$ and an n-neuron unbounded-precision RNN $\mathcal{T}_{W,\mathbf{b}} : \mathbb{Q}^n \to \mathbb{Q}^n$, where $n = 2|\Gamma| + \lceil \log_2 |Q| \rceil + |Q||\Gamma| + 5$, such that for all instantaneous descriptions $x \in \mathcal{X}$, the following holds: If $\mathcal{P}^*_{\mathcal{M}}(x)$ is defined, then*

$$\rho^{-1}(\mathcal{T}^*_{W,\mathbf{b}}(\rho(x))) = \mathcal{P}^*_{\mathcal{M}}(x), \tag{7}$$

and if $\mathcal{P}^*_{\mathcal{M}}(x)$ is not defined, then $\mathcal{T}^*_{W,\mathbf{b}}(\rho(x))$ is also not defined.

Moreover, if $\mathcal{P}^*_{\mathcal{M}}$ is defined and computed in T steps by $\mathcal{M}$, then $\mathcal{T}^*_{W,\mathbf{b}}(\rho(x))$ is computed in $3T$ steps by the RNN.

Corollary 1.1 shares some similarities with the Theorem 1 proposed in [1]. However, our theorem states that $3T$, instead of $4T$ plus a linear order of length of the tape, is sufficient to simulate a Turing machine. Our theorem also gives a relationship between the number of neurons required for the RNN and the size of Q and Γ in the Turing machine. The RNN constructed is also a direct simulation of a Turing Machine instead of a two-stack machine with a binary tape.

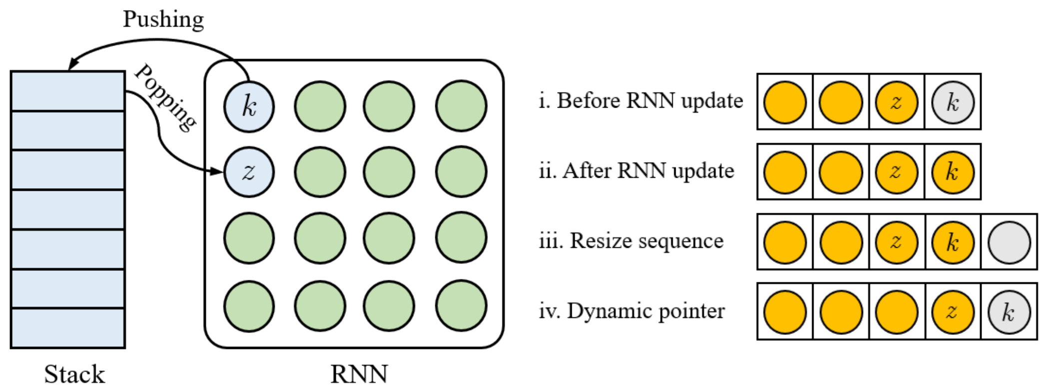
[14] proposed a Universal Turing Machine (UTM), a Turing machine with 6 states and 4 symbols denoted by $U_{6,4}$, that can simulate any Turing machines in time $\mathcal{O}(T^6)$, where T is the number of steps required for the Turing machine (the one to be simulated) to compute the result. As $U_{6,4}$ is also a Turing machine, we can thus apply Corollary 1.1 to simulate $U_{6,4}$, leading to a Turing-complete RNN. Plugging in $|Q| = 6$ and $|\Gamma| = 4$, we obtain the following result:

Theorem 2. *There exists a 40-neuron unbounded-precision RNN that can simulate any Turing machines in $\mathcal{O}(T^6)$, where T is the number of steps required for the Turing machine to compute the result.*

Note that [1] proved that there exists a Turing-complete RNN with 1058 neurons, while [15] proposed a Turing-complete RNN with 52 neurons. However, [15] does not have a rigorous proof, and some key formulas seem to be missing. As far as we are aware, our proposed 40-neuron RNN is the smallest Turing-complete RNN to date.

4 Turing Completeness of Bounded-Precision RNNs with Growing Memories

In the following, we consider how to remove the assumption of unbounded precision that is required in all the results in Section 3. If we assume all neurons to have precision bounded by p in base $2|\Gamma|$,



(a) An RNN with a growing memory module

(b) Pushing with dynamic pointer

Figure 1: (a) An RNN with a growing memory module. The RNN controls the pushing and popping of the stack by two neurons $k(t)$ and $z(t)$. (b) Illustration of pushing in growing memory module as a dynamic pointer. Orange (dark) and gray (light) circles represent non-zero neurons and zero neurons respectively. i. Before RNN update, $k(t-1) = 0$. ii. RNN set $k(t) = c$ where $c > 0$ is the value to be pushed. iii. One zero neuron is appended to the sequence since there are no zero neurons left. iv. Both z and k point to different neurons in the sequence, ready to be read by other neurons in the next update of RNN.

then the tape can be encoded by $\lceil |t_m|/p \rceil$ neurons using fractal encoding, where $m \in \{l, r\}$. We do this by encoding every p symbols in the tape with a single neuron. However, notice that most of these neurons do not require updating when simulating a single step of the Turing machine. This is similar to how most parts of the tape are not updated during one step of a Turing machine. It is thus not efficient to include all these neurons (which may be large in number) in the RNN. Instead, we propose a *growing memory module* that can be equipped to an RNN to manipulate neurons in a stack-like mechanism:

Definition 3. *A growing memory module is a stack of non-zero neurons with push and pop operations controlled by two neurons in an RNN, denoted as push neuron $k(t)$ and pop neuron $z(t)$, in the following way: for every step t after the RNN finished updating the values of neurons, (i) if $k(t) > 0$, then a new neuron of value $k(t)$ is pushed to the stack and $k(t)$ is set to 0; (ii) if $z(t) = 0$ and the stack is not empty, then the top neuron is popped from the stack and $z(t)$ is set to the value of the top neuron in the updated stack; (iii) if $z(t) = 0$ and the stack is empty, then $z(t)$ is set to a default value c.*

With this definition, an RNN with a growing memory module is a mapping $\mathcal{T}_{W,\mathbf{b}} : (\mathbb{Q}^N, \mathbb{Q}^*) \rightarrow (\mathbb{Q}^N, \mathbb{Q}^*)$ where the first element of the tuple corresponds to values of neurons in the RNN and the second element of the tuple corresponds to the stack of the growing memory module. We can also equip an RNN with multiple growing memory modules, with each module having its own push and pop neurons controlled by the RNN. For example, an RNN with two growing memory modules defines a mapping $\mathcal{T}_{W,\mathbf{b}} : (\mathbb{Q}^N, \mathbb{Q}^*, \mathbb{Q}^*) \rightarrow (\mathbb{Q}^N, \mathbb{Q}^*, \mathbb{Q}^*)$.

We can also view the growing memory module as a way of dynamically pointing to a sequence of neurons - consider a sequence of non-zero neurons appended by a zero neuron. $z(t)$ can be viewed as the pointer for the last non-zero neuron in the sequence, and $k(t)$ can be viewed as the pointer for the zero neuron in the sequence. Also, let m be the number of zero neurons in the sequence. If $m = 0$, then we add one zero neuron at the end of the sequence. If $m > 1$, then we remove $m - 1$ zero neurons from the end of sequence. In this way, $k(t)$ is always defined. An illustration of this is shown in Figure 1.

The advantage of the proposed growing memory module is that it can dynamically recruit new neurons when more memories are needed, and releases them when memories become irrelevant. We therefore do not need to estimate the number of neurons required before simulating a Turing machine. It also saves computational cost significantly since neurons in the stack (except the top neuron) are not updated.

Based on the growing memory module, we can construct an RNN with bounded-precision neurons that can simulate any Turing Machines. We require two growing memory modules in the RNN: one for the left tape and one for the right tape. Similar to the previous section, we first describe how we can encode the instantaneous description $(q, t_l, t_r) \in \mathcal{X}$ by a vector of rational numbers and two stacks, with which an RNN and its growing memory modules can be initialized. In the following discussion, we assume that all neurons have precision bounded by $p \geq 2$ in base $2|\Gamma|$; that is, each neuron can encode p symbols at most.

Both the state q and the top symbols $t_{l,1}, t_{r,1}$ are encoded with binary values using the same method as in Section 3. For the tape t_m ($m \in \{l, r\}$), we define the fractal encoding $\rho^t : \Gamma^* \to \mathbb{Q}$ by:

$$\rho^t(t) := \sum_{i=1}^{|t|} \frac{2t_i}{(2|\Gamma|)^i}, \tag{8}$$

which is the same as (3) but with the encoding for infinite blank symbols removed. Then, we encode the tape t_m into a stack of neurons as follows: First, encode the rightmost p symbols of it with ρ^t and push it into an empty stack, denoted as M_m. Then, encode the next rightmost p symbols and push it to M_m again, and repeat until there are no more than p symbols left in the tape. Denote this encoding function for the tape as $\rho^M : \Gamma^* \to \mathbb{Q}^*$. The remaining symbols in the tape, denoted as $t_{m,1:h(|t_m|)}$, where $h(y) := ((y-1) \bmod p) + 1$, will be encoded with fractal encoding ρ^t as well but appears in the initialization values of the RNN's neurons instead of the stacks.

The general idea is that the symbols closest to the read/write head ($t_{m,1:h(|t_m|)}$) resides in the RNN since they are the symbols that require updating, and the remaining symbols will be stored in a stack. If the number of symbols residing in the RNN reaches 0 or p, then we pop or push from the stack respectively to ensure that at least 1 and at most p symbols reside in the RNN. It is also interesting to note that the k^{th} neuron in the stack (from the top) will require at least kp steps of the Turing machine before it is changed, so the values of neurons near the bottom of the stack will not be changed for many steps.

Finally, we need to encode $h(|t_m|)$, which keeps track of the number of symbols residing in the RNN, such that the RNN can know when to push or pop from the stack. We use the following function: $\rho^h : \{1, 2, ..., p\} \to \mathbb{Q}$ by:

$$\rho^h(y) = \frac{y}{p+1}. \tag{9}$$

Together, define the encoding function $\rho : \mathcal{X} \to (\mathbb{Q}^{2|\Gamma|+\lceil \log_2 |Q| \rceil + |Q||\Gamma| + 19}, \mathbb{Q}^*, \mathbb{Q}^*)$ by:

$$\begin{aligned}\rho(q, t_l, t_r) = (&\rho^q(q) \oplus \rho^t(t_{l,1:h(|t_l|)}) \oplus \rho^t(t_{r,1:h(|t_r|)}) \oplus \rho^t(t_{l,h(|t_l|)+1:h(|t_l|)+p}) \oplus \\ &\rho^t(t_{r,h(|t_r|)+1:h(|t_r|)+p}) \oplus \rho^r(t_{l,1}) \oplus \rho^r(t_{r,1}) \oplus \rho^h(h(|t_l|)) \oplus \rho^h(h(|t_r|)) \oplus \\ &\mathbf{0}, \rho^M(t_l), \rho^M(t_r)),\end{aligned} \tag{10}$$

where $\mathbf{0}$ is a zero vector of size $|Q||\Gamma| + 15$. The first element of the tuple is for initializing the neurons in the RNN, while the second and third element of the tuple is for initializing stacks of the two growing memory modules. Note that all encoded values have precision of p in the above definition. Similar to the previous section, ρ is injective and so we can define the decoder function $\rho^{-1} : \rho(\mathcal{X}) \to \mathcal{X}$.

With the new encoder function ρ and the growing memory module, we can prove an alternative version of Theorem 1 that only requires an RNN with bounded precision:

Theorem 4. *Given a Turing machine $\mathcal{M}$, there exists an injective function $\rho : \mathcal{X} \to (\mathbb{Q}^n, \mathbb{Q}^*, \mathbb{Q}^*)$ and an n-neuron p-precision (in base $2|\Gamma|$) RNN with two growing memory modules $\mathcal{T}_{W,\mathbf{b}} : (\mathbb{Q}^n, \mathbb{Q}^*, \mathbb{Q}^*) \to (\mathbb{Q}^n, \mathbb{Q}^*, \mathbb{Q}^*)$, where $n = 2|\Gamma| + \lceil \log_2 |Q| \rceil + |Q||\Gamma| + 19$ and $p \geq 2$, such that for all instantaneous descriptions $x \in \mathcal{X}$,*

$$\rho^{-1}(\mathcal{T}^3_{W,\mathbf{b}}(\rho(x))) = \mathcal{P}_{\mathcal{M}}(x). \tag{11}$$

Proof. The detailed proof is in Appendix B. To illustrate the construction of the RNN, the parameters for the neuron initialized with $\rho^h(h(|t_l|))$, called the left-selector neuron $s_l(t)$, will be described here.

We assume all neurons in the RNN are initialized with $\rho(x)$ at time $t = 1$. The selector neuron $s_l(t)$ encodes the number of left-tape symbols residing in the RNN. In three steps of an RNN, we need to update its value from $s_l(1) = h(|t_l|)/(p+1)$ to $s_l(4) = h(|t_l'|)/(p+1)$, where t_l' is the left tape after one step of the Turing machine. First, notice that $h(|t_l'|)$ can be expressed as:

$$h(|t_l'|) = \begin{cases} h(|t_l|) - 1, & \text{if } d = L \text{ and } h(|t_l|) \geq 2 \\ p, & \text{if } d = L \text{ and } h(|t_l|) = 1 \\ h(|t_l|) + 1, & \text{if } d = R \text{ and } h(|t_l|) \leq p - 1 \\ 1, & \text{if } d = R \text{ and } h(|t_l|) = p, \end{cases} \tag{12}$$

where d is the direction that the Turing machine is moving. For example, if the Turing machine is moving left, then there will be one less left-tape symbol residing in the RNN after one step of the Turing machine. However, if there are no left-tape symbols left in the RNN, the top neuron will be popped from the left-tape stack to the RNN, so there will be p left-tape symbols residing in the RNN instead. A similar process holds when the Turing machine is moving right.

Then we consider how to implement (12) with an RNN: Define stage neurons as:

$$c_1(t+1) = \sigma(1 - c_1(t) - c_2(t)), \tag{13}$$
$$c_2(t+1) = \sigma(c_1(t)), \tag{14}$$

with both neurons initialized to be zero. Define $\mathbf{c}(t) := [c_1(t), c_2(t), c_3(t)]$ where $c_3(t) := 1 - c_1(t) - c_2(t)$, then $\mathbf{c}(1) = [0, 0, 1]$; $\mathbf{c}(2) = [1, 0, 0]$; $\mathbf{c}(3) = [0, 1, 0]$. These stage neurons signal which one of the three steps that the RNN is in.

Also, in the construction of our RNN, there exists a linear sum of neurons, denoted as, $\mathbf{d}_m(t) = [d_l(t), d_r(t)]$, such that if the Turing machine is moving left, $\mathbf{d}(1) = [0, 0]$; $\mathbf{d}(2) = [1, 0]$; $\mathbf{d}(3) = [0, 0]$; and if the Turing machine is moving right, $\mathbf{d}(1) = [0, 0]$; $\mathbf{d}(2) = [0, 1]$; $\mathbf{d}(3) = [0, 0]$; this signals which direction the Turing machine is moving to (formulas of $\mathbf{d}_m(t)$ appears in Appendix B).

Then consider the following update rule for the left-selector neurons:

$$s_l(t+1) = \sigma(s_l(t) + d_r(t) - d_l(t) - ps_l'(t) + ps_l''(t)), \tag{15}$$
$$s_l'(t+1) = \sigma((p+1)s_l(t) + d_r(t) - p - c_2(t) - c_3(t)), \tag{16}$$
$$s_l''(t+1) = \sigma(2 - (p+1)s_l(t) - d_r(t) - c_2(t) - c_3(t)), \tag{17}$$

where $s_l(1) = h(|t_l|)/(p+1)$, $s_l'(1) = s_l''(1) = 0$. It can be verified that $s_l(4) = h(|t_l'|)/(p+1)$ as defined by (12), completing the proof for $s_l(t)$. The proof for the remaining neurons can be found in Appendix B. $\square$

Similar to Corollary 1.1, it follows that:

Corollary 4.1. *Given a Turing machine $\mathcal{M}$, there exists an injective function $\rho : \mathcal{X} \to (\mathbb{Q}^n, \mathbb{Q}^*, \mathbb{Q}^*)$ and an n-neuron p-precision (in base $|2\Gamma|$) RNN with two growing memory modules $\mathcal{T}_{W,\mathbf{b}} : (\mathbb{Q}^n, \mathbb{Q}^*, \mathbb{Q}^*) \to (\mathbb{Q}^n, \mathbb{Q}^*, \mathbb{Q}^*)$, where $n = 2|\Gamma| + \lceil \log_2 |Q| \rceil + |Q||\Gamma| + 19$ and $p \geq 2$, such that for all instantaneous descriptions $x \in \mathcal{X}$, the following holds: If $\mathcal{P}_{\mathcal{M}}^*(x)$ is defined, then*

$$\rho^{-1}(\mathcal{T}_{W,\mathbf{b}}^*(\rho(x))) = \mathcal{P}_{\mathcal{M}}^*(x), \tag{18}$$

and if $\mathcal{P}_{\mathcal{M}}^*(x)$ is not defined, then $\mathcal{T}_{W,\mathbf{b}}^*(\rho(x))$ is also not defined.
Moreover, if $\mathcal{P}_{\mathcal{M}}^*(x)$ is defined and computed in T steps by $\mathcal{M}$, then $\mathcal{T}_{W,\mathbf{b}}^*(\rho(x))$ is computed in $3T$ steps by the RNN.

Finally, applying Corollary 4.1 to $U_{6,4}$, we obtain:

Theorem 5. *There exists a 54-neuron p-precision (in base 8) RNN with two growing memory modules that can simulate any Turing machines in $\mathcal{O}(T^6)$, where T is the number of steps required for the Turing machine to compute the result and $p \geq 2$.*

The architecture of the Turing-complete 54-neuron RNN follows the one used in the proof of Theorem 4, which can be illustrated as a 4-layer model as shown in Figure 2.

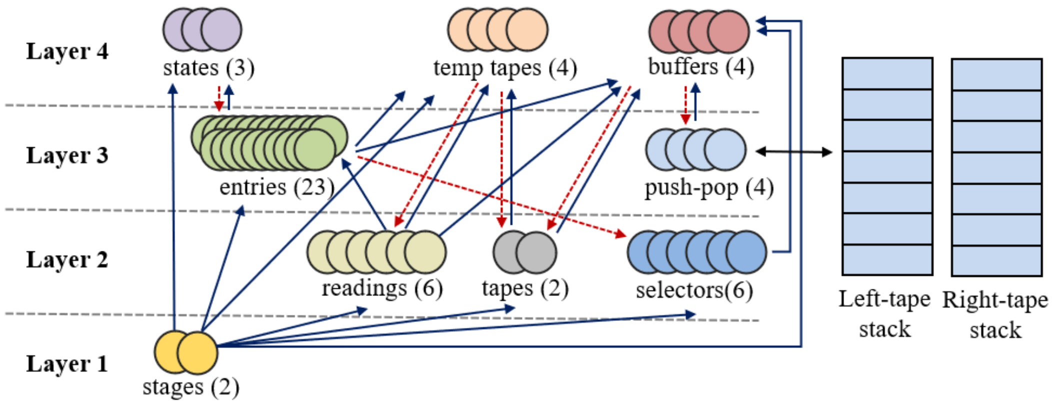


Figure 2: The architecture of the Turing-complete 54-neuron RNN with two growing memory modules. Blue (solid) lines denote bottom-up connections, and red (dotted) lines denote top-down connections. Notable neurons include: state neurons - represent the current state; tape neurons - represent the tape symbols near the read/write head; selector neurons - represent the number of symbols residing in the RNN; push and pop neurons - control the pushing and popping of left-tape and right-tape stacks; stage neurons - represent the current stage and inhibits computation of other neurons if the neurons do not require updating in a stage. A detailed description of these neurons can be found in Appendix B.

5 Turing Completeness of Bounded-Precision RNNs

As discussed previously, it is inefficient to update all neurons that encode the tape in an RNN, since most of the neurons do not require updating. However, we do not need the growing memory modules if we update all neurons at the same time. To extend the theoretical foundation of RNNs, we also construct a bounded-precision RNN to simulate a Turing machine without any growing memory modules.

We define a Turing machine with a bounded tape as a Turing machine that halts if the read/write head reaches the end of the tape, which is assumed to have a finite size. Then it is possible to construct an RNN with $\mathcal{O}(\lceil F/p \rceil)$ p-precision neurons to simulate a Turing machine with a bounded tape of size F:

Theorem 6. *Given a Turing machine $\mathcal{M}$ with a bounded tape of size F, there exists an injective function $\rho : \mathcal{X} \to \mathbb{Q}^N$ and an n-neuron p-precision (in base $|2\Gamma|$) RNN $\mathcal{T}_{W,\mathbf{b}} : \mathbb{Q}^n \to \mathbb{Q}^n$, where $n = \mathcal{O}(\lceil F/p \rceil)$ and $p \geq 2$, such that for all instantaneous descriptions $x \in \mathcal{X}$,*

$$\rho^{-1}(\mathcal{T}^3_{W,\mathbf{b}}(\rho(x))) = \mathcal{P}_{\mathcal{M}}(x). \tag{19}$$

The proof can be found in Appendix C. The general idea of the proof is to implement the growing memory module in Section 4 by an RNN as well and place all neurons inside the RNN. A corollary similar to Corollary 1.1 and 4.1 follows and is omitted here.

This theorem shows that there is a trade-off between the number of neurons and precision when trying to simulate a Truing machine with an RNN. It also establishes that with the same precision p, an RNN with more neurons has a larger capability since it can simulate a Turing machine with a larger tape. Taking it to the extreme, we can use an infinite-neuron bounded-precision RNN to simulate $U_{6,4}$, which has an unbounded tape. This leads to the following theorem:

Theorem 7. *There exists an infinite-neuron bounded-precision RNN that can simulate any Turing machines in $\mathcal{O}(T^6)$, where T is the number of steps required for the Turing machine to compute the result.*

6 Discussion and Conclusion

It is inevitable that either unbounded precision (Theorem 2) or an infinite number of neurons (Theorem 7) is required to construct a Turing-complete RNN, since the tape in a Turing machine has an unbounded length, and an RNN can only encode information by neurons. However, both unbounded precision or infinite neurons are not practical in implementation. This highlights a major limitation of RNNs. To provide a practical solution, we propose a growing memory module that allows an RNN to control a stack with two neurons and prove the Turing completeness of the enhanced RNN (Theorem 5).

All the RNNs consider in this paper have a fixed parameter. Given the input and target output, we may also consider learning the parameters of RNNs with error backpropagation. However, the operation of the growing memory module is not fully differentiable. Nonetheless, it may be possible to construct a differentiable version of the growing memory module and learn the parameters of RNNs directly, which is left for future work. The growing memory module may facilitate RNNs to learn tasks that are more memory-intensive.

In conclusion, we prove the Turing completeness of a 40-neuron unbounded-precision RNN, which is the smallest Turing-complete RNN to date. We propose a novel growing memory module that can be equipped to an RNN to remove the unbounded-precision requirement. With the growing memory module, a 54-neuron bounded-precision RNN can be Turing-complete. We also analyze the relationship between the number of neurons and the precision of an RNN when simulating a Turing machine. Finally, we prove the Turing completeness of an infinite-neuron bounded-precision RNN.

References

[1] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

[2] Hava T Siegelmann. Computation beyond the turing limit. *Science*, 268(5210):545–548, 1995.

[3] Hava T Siegelmann. *Neural networks and analog computation: beyond the Turing limit*. Springer Science & Business Media, 2012.

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

[5] 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.

[6] Joe Kilian and Hava T Siegelmann. The dynamic universality of sigmoidal neural networks. *Information and computation*, 128(1):48–56, 1996.

[7] H Georg Kuhn, Heather Dickinson-Anson, and Fred H Gage. Neurogenesis in the dentate gyrus of the adult rat: age-related decrease of neuronal progenitor proliferation. *Journal of Neuroscience*, 16(6):2027–2033, 1996.

[8] Arturo Alvarez-Buylla and Jose Manuel Garcıa-Verdugo. Neurogenesis in adult subventricular zone. *Journal of Neuroscience*, 22(3):629–634, 2002.

[9] David Marr, David Willshaw, and Bruce McNaughton. Simple memory: a theory for archicortex. In *From the Retina to the Neocortex*, pages 59–128. Springer, 1991.

[10] MW Jung and BL McNaughton. Spatial selectivity of unit activity in the hippocampal granular layer. *Hippocampus*, 3(2):165–182, 1993.

[11] Gerd Kempermann, H Georg Kuhn, and Fred H Gage. Experience-induced neurogenesis in the senescent dentate gyrus. *Journal of Neuroscience*, 18(9):3206–3212, 1998.

[12] Georg Elias Müller and Alfons Pilzecker. *Experimentelle beiträge zur lehre vom gedächtniss*, volume 1. JA Barth, 1900.

[13] James L McGaugh. Memory–a century of consolidation. *Science*, 287(5451):248–251, 2000.

[14] Turlough Neary and Damien Woods. Four small universal turing machines. *Fundamenta Informaticae*, 91(1):123–144, 2009.

[15] J Nicholas Hobbs and Hava Siegelmann. Implementation of universal computation via small recurrent finite precision neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–5. IEEE, 2015.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] This is a theoretical work and does not present any foreseeable societal consequences.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]