A Formal Hierarchy of RNN Architectures

William Merrill[*] **Gail Weiss**[†] **Yoav Goldberg**[*‡]
Roy Schwartz[*§] **Noah A. Smith**[*§] **Eran Yahav**[†]
[*]Allen Institute for AI [†]Technion [‡]Bar Ilan University [§]University of Washington
{willm,yoavg,roys,noah}@allenai.org
{sgailw,yahave}@cs.technion.ac.il

Abstract

We develop a formal hierarchy of the expressive capacity of RNN architectures. The hierarchy is based on two formal properties: space complexity, which measures the RNN's memory, and rational recurrence, defined as whether the recurrent update can be described by a weighted finite-state machine. We place several RNN variants within this hierarchy. For example, we prove the LSTM is not rational, which formally separates it from the related QRNN (Bradbury et al., 2016). We also show how these models' expressive capacity is expanded by stacking multiple layers or composing them with different pooling functions. Our results build on the theory of "saturated" RNNs (Merrill, 2019). While formally extending these findings to unsaturated RNNs is left to future work, we hypothesize that the practical learnable capacity of unsaturated RNNs obeys a similar hierarchy. Experimental findings from training unsaturated networks on formal languages support this conjecture.

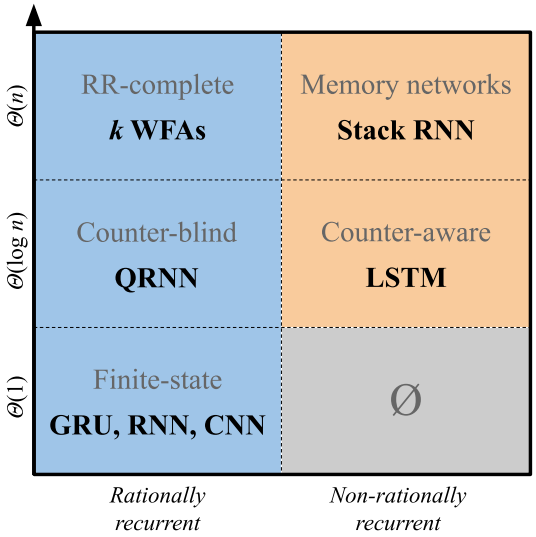

Figure 1: Hierarchy of state expressiveness for saturated RNNs and related models. The y axis represents increasing space complexity. $\emptyset$ means provably empty. Models are in bold with qualitative descriptions in gray.

1 Introduction

While neural networks are central to the performance of today's strongest NLP systems, theoretical understanding of the formal properties of different kinds of networks is still limited. It is established, for example, that the Elman (1990) RNN is Turing-complete, given infinite precision and computation time (Siegelmann and Sontag, 1992, 1994; Chen et al., 2018). But tightening these unrealistic assumptions has serious implications for expressive power (Weiss et al., 2018), leaving a significant gap between classical theory and practice, which theorems in this paper attempt to address.

Recently, Peng et al. (2018) introduced **rational RNNs**, a subclass of RNNs whose internal state can be computed by independent weighted finite automata (WFAs). Intuitively, such models have a computationally simpler recurrent update than conventional models like long short-term memory networks (LSTMs; Hochreiter and Schmidhuber, 1997). Empirically, rational RNNs like the quasi-recurrent neural network (QRNN; Bradbury et al., 2016) and unigram rational RNN (Dodge et al., 2019) perform comparably to the LSTM, with a smaller computational budget. Still, the underlying simplicity of rational models raises the question of whether their expressive power is fundamentally limited compared to other RNNs.

In a separate line of work, Merrill (2019) introduced the **saturated RNN**[1] as a formal model for analyzing the capacity of RNNs. A saturated RNN is a simplified network where all activation functions have been replaced by step functions. The saturated network may be seen intuitively as a "stable" version of its original RNN, in which the in-

[1]Originally referred to as the *asymptotic RNN*.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 443–459

ternal activations act discretely. A growing body of work—including this paper—finds that the saturated theory predicts differences in practical learnable capacity for various RNN architectures (Weiss et al., 2018; Merrill, 2019; Suzgun et al., 2019a).

We compare the expressive power of rational and non-rational RNNs, distinguishing between *state expressiveness* (what kind and amount of information the RNN states can capture) and *language expressiveness* (what languages can be recognized when the state is passed to a classifier). To do this, we build on the theory of saturated RNNs.

State expressiveness We introduce a unified hierarchy (Figure 1) of the functions expressible by the states of rational and non-rational RNN encoders. The hierarchy is defined by two formal properties: space complexity, which is a measure of network memory,[2] and rational recurrence, whether the internal structure of the RNN can be described by WFAs. The hierarchy reveals concrete differences between LSTMs and QRNNs, and further separates both from a class containing convolutional neural networks (CNNs, Lecun and Bengio, 1995; Kim, 2014), Elman RNNs, and gated recurrent units (GRU; Cho et al., 2014).

We provide the first formal proof that LSTMs can encode functions that rational recurrences cannot. On the other hand, we show that the saturated Elman RNN and GRU are rational recurrences with constant space complexity, whereas the QRNN has unbounded space complexity. We also show that an unrestricted WFA has rich expressive power beyond any saturated RNN we consider—including the LSTM. This difference potentially opens the door to more expressive RNNs incorporating the computational efficiency of rational recurrences.

Language expressiveness When applied to classification tasks like language recognition, RNNs are typically combined with a "decoder": additional layer(s) that map their hidden states to a prediction. Thus, despite differences in state expressiveness, rational RNNs might be able to achieve comparable empirical performance to non-rational RNNs on NLP tasks. In this work, we consider the setup in which the decoders only view the final hidden state of the RNN.[3] We demonstrate that a sufficiently strong decoder can overcome some of the differences in state expressiveness between different models. For example, an LSTM can recognize $a^n b^n$ with a single decoding layer, whereas a QRNN provably cannot until the decoder has two layers. However, we also construct a language that an LSTM can recognize without a decoder, but a QRNN cannot recognize with any decoder. Thus, no decoder can fully compensate for the weakness of the QRNN compared to the LSTM.

Experiments Finally, we conduct experiments on formal languages, justifying that our theorems correctly predict which languages unsaturated recognizers trained by gradient descent can learn. Thus, we view our hierarchy as a useful formal tool for understanding the relative capabilities of different RNN architectures.

Roadmap We present the formal devices for our analysis of RNNs in Section 2. In Section 3 we develop our hierarchy of state expressiveness for single-layer RNNs. In Section 4, we shift to study RNNs as language recognizers. Finally, in Section 5, we provide empirical results evaluating the relevance of our predictions for unsaturated RNNs.

2 Building Blocks

In this work, we analyze RNNs using formal models from automata theory—in particular, WFAs and counter automata. In this section, we first define the basic notion of an encoder studied in this paper, and then introduce more specialized formal concepts: WFAs, counter machines (CMs), space complexity, and, finally, various RNN architectures.

2.1 Encoders

We view both RNNs and automata as *encoders*: machines that can be parameterized to compute a set of functions $f : \Sigma^* \to \mathbb{Q}^k$, where Σ is an input alphabet and $\mathbb{Q}$ is the set of rational reals. Given an encoder M and parameters θ, we use M_θ to represent the specific function that the parameterized encoder computes. For each encoder, we refer to the set of functions that it can compute as its *state expressiveness*. For example, a deterministic finite state acceptor (DFA) is an encoder whose parameters are its transition graph. Its state expressiveness is the indicator functions for the regular languages.

2.2 WFAs

Formally, a WFA is a non-deterministic finite automaton where each starting state, transition, and

[2] Space complexity measures the number of different configurations an RNN can reach as a function of input length. Formal definition deferred until Section 2.

[3] This is common, but not the only possibility. For example, an attention decoder observes the full sequence of states.

final state is weighted. Let Q denote the set of states, Σ the alphabet, and $\mathbb{Q}$ the rational reals.[4] This weighting is specified by three functions:

1. Initial state weights $\lambda : Q \to \mathbb{Q}$
2. Transition weights $\tau : Q \times \Sigma \times Q \to \mathbb{Q}$
3. Final state weights $\rho : Q \to \mathbb{Q}$

The weights are used to encode any string $x \in \Sigma^*$:

Definition 1 (Path score)**.** Let π be a path of the form $q_0 \to_{x_1} q_1 \to_{x_2} \cdots \to_{x_t} q_t$ through WFA A. The score of π is given by

$$A[\pi] = \lambda(q_0) \left(\prod_{i=1}^{t} \tau(q_{i-1}, x_i, q_i)\right) \rho(q_t).$$

By $\Pi(x)$, denote the set of paths producing x.

Definition 2 (String encoding)**.** The encoding computed by a WFA A on string x is

$$A[x] = \sum_{\pi \in \Pi(x)} A[\pi].$$

Hankel matrix Given a function $f : \Sigma^* \to \mathbb{Q}$ and two enumerations α, ω of the strings in Σ^*, we define the Hankel matrix of f as the infinite matrix

$$[H_f]_{ij} = f(\alpha_i \cdot \omega_j). \tag{1}$$

where $\cdot$ denotes concatenation. It is sometimes convenient to treat H_f as though it is directly indexed by Σ^*, e.g. $[H_f]_{\alpha_i, \omega_j} = f(\alpha_i \cdot \omega_j)$, or refer to a sub-block of a Hankel matrix, row- and column-indexed by prefixes and suffixes $P, S \subseteq \Sigma^*$. The following result relates the Hankel matrix to WFAs:

Theorem 1 (Carlyle and Paz, 1971; Fliess, 1974)**.** *For any $f : \Sigma^* \to \mathbb{Q}$, there exists a WFA that computes f if and only if H_f has finite rank.*

Rational series (Sakarovitch, 2009) For all $k \in \mathbb{N}$, $\mathbf{f} : \Sigma^* \to \mathbb{Q}^k$ is a *rational series* if there exist WFAs $A_1, \cdots, A_k$ such that, for all $x \in \Sigma^*$ and $1 \leq i \leq k$, $A_i[x] = f_i(x)$.

2.3 Counter Machines

We now turn to introducing a different type of encoder: the real-time counter machine (CM; Merrill, 2020; Fischer, 1966; Fischer et al., 1968). CMs are deterministic finite-state machines augmented with finitely many integer counters. While processing a string, the machine updates these counters, and may use them to inform its behavior.

We view counter machines as encoders mapping $\Sigma^* \to \mathbb{Z}^k$. For $m \in \mathbb{N}, \circ \in \{+, -, \times\}$, let $\circ m$ denote the function $f(n) = n \circ m$.

Definition 3 (General CM; Merrill, 2020)**.** A k-counter CM is a tuple $\langle \Sigma, Q, q_0, u, \delta \rangle$ with

1. A finite alphabet Σ
2. A finite set of states Q, with initial state q_0
3. A counter update function

$$u : \Sigma \times Q \times \{0, 1\}^k \to \{\times 0, -1, +0, +1\}^k$$

4. A state transition function

$$\delta : \Sigma \times Q \times \{0, 1\}^k \to Q$$

A CM processes input tokens $\{x_t\}_{t=1}^n$ sequentially. Denoting $\langle q_t, \mathbf{c}_t \rangle \in Q \times \mathbb{Z}^k$ a CM's configuration at time t, define its next configuration:

$$q_{t+1} = \delta\left(x_t, q_t, \vec{\mathbb{1}}_{=0}(\mathbf{c}_t)\right) \tag{2}$$

$$\mathbf{c}_{t+1} = u\left(x_t, q_t, \vec{\mathbb{1}}_{=0}(\mathbf{c}_t)\right)(\mathbf{c}_t), \tag{3}$$

where $\vec{\mathbb{1}}_{=0}$ is a broadcasted "zero-check" operation, i.e., $\vec{\mathbb{1}}_{=0}(\mathbf{v})_i \triangleq \mathbb{1}_{=0}(v_i)$. In (2) and (3), note that the machine only views the zeroness of each counter, and not its actual value. A general CM's encoding of a string x is the value of its counter vector $\mathbf{c}_t$ after processing all of x.

Restricted CMs

1. A CM is Σ-*restricted* iff u and δ depend only on the current input $\sigma \in \Sigma$.
2. A CM is $(\Sigma \times Q)$-*restricted* iff u and δ depend only on the current input $\sigma \in \Sigma$ and the current state $q \in Q$.
3. A CM is Σ^w-*restricted* iff it is $(\Sigma \times Q)$-restricted, and the states Q are windows over the last w input tokens, e.g., $Q = \Sigma^{\leq w}$.[5]

These restrictions prevent the machine from being "counter-aware": u and δ cannot condition on the counters' values. As we will see, restricted CMs have natural parallels in the realm of rational RNNs. In Subsection 3.2, we consider the relationship between counter awareness and rational recurrence.

2.4 Space Complexity

As in Merrill (2019), we also analyze encoders in terms of state space complexity, measured in bits.

Definition 4 (Bit complexity)**.** An encoder $M : \Sigma^* \to \mathbb{Q}^k$ has $T(n)$ space iff

$$\max_{\theta} \left|\{s_{M_\theta}(x) \mid x \in \Sigma^{\leq n}\}\right| = 2^{T(n)},$$

[4] WFAs are often defined over a generic semiring; we consider only the special case when it is the field of rational reals.

[5] The states $q \in \Sigma^{<w}$ represent the beginning of the sequence, before w input tokens have been seen.

where $s_{M_\theta}(x)$ is a minimal representation[6] of M's internal configuration immediately after x.

We consider three asymptotic space complexity classes: $\Theta(1)$, $\Theta(\log n)$, and $\Theta(n)$, corresponding to encoders that can reach a constant, polynomial, and exponential (in sequence length) number of configurations respectively. Intuitively, encoders that can dynamically count but cannot use more complex memory like stacks–such as all CMs–are in $\Theta(\log n)$ space. Encoders that can uniquely encode every input sequence are in $\Theta(n)$ space.

2.5 Saturated Networks

A saturated neural network is a discrete approximation of neural network considered by Merrill (2019), who calls it an "asymptotic network." Given a parameterized neural encoder $M_\theta(x)$, we construct the saturated network s-$M_\theta(x)$ by taking

$$\text{s-}M_\theta(x) = \lim_{N\to\infty} M_{N\theta}(x) \tag{4}$$

where $N\theta$ denotes the parameters θ multiplied by a scalar N. This transforms each "squashing" function (sigmoid, tanh, etc.) to its extreme values (0, ± 1). In line with prior work (Weiss et al., 2018; Merrill, 2019; Suzgun et al., 2019b), we consider saturated networks a reasonable approximation for analyzing practical expressive power. For clarity, we denote the saturated approximation of an architecture by prepending it with s, e.g., s-LSTM.

2.6 RNNs

A recurrent neural network (RNN) is a parameterized update function $g_\theta : \mathbb{Q}^k \times \mathbb{Q}^{d_x} \to \mathbb{Q}^k$, where θ are the rational-valued parameters of the RNN and d_x is the dimension of the input vector. g_θ takes as input a current state $\mathbf{h} \in \mathbb{Q}^k$ and input vector $\mathbf{x} \in \mathbb{Q}^{d_x}$, and produces the next state. Defining the initial state as $\mathbf{h}_0 = \mathbf{0}$, an RNN can be applied to an input sequence $x \in (\mathbb{Q}^{d_x})^*$ one vector at a time to create a sequence of states $\{\mathbf{h}_t\}_{t\leq|x|}$, each representing an encoding of the prefix of x up to that time step. RNNs can be used to encode sequences over a finite alphabet $x \in \Sigma^*$ by first applying a mapping (embedding) $e : \Sigma \to \mathbb{Q}^{d_x}$.

Multi-layer RNNs "Deep" RNNs are RNNs that have been arranged in L stacked layers $R_1, ..., R_L$. In this setting, the series of output states $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_{|x|}$ generated by each RNN on its input is fed as input to the layer above it, and only the first layer receives the original input sequence $x \in \Sigma^*$ as input.

The recurrent update function g can take several forms. The original and most simple form is that of the *Elman RNN*. Since then, more elaborate forms using gating mechanisms have become popular, among them the LSTM, GRU, and QRNN.

Elman RNNs (Elman, 1990) Let $\mathbf{x}_t$ be a vector embedding of x_t. For brevity, we suppress the bias terms in this (and the following) affine operations.

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}). \tag{5}$$

We refer to the saturated Elman RNN as the *s-RNN*. The s-RNN has $\Theta(1)$ space (Merrill, 2019).

LSTMs (Hochreiter and Schmidhuber, 1997) An LSTM is a gated RNN with a state vector $\mathbf{h}_t \in \mathbb{Q}^k$ and memory vector $\mathbf{c}_t \in \mathbb{Q}^k$. [7]

$$\mathbf{f}_t = \sigma(\mathbf{W}^f\mathbf{x}_t + \mathbf{U}^f\mathbf{h}_{t-1}) \tag{6}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}^i\mathbf{x}_t + \mathbf{U}^i\mathbf{h}_{t-1}) \tag{7}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}^o\mathbf{x}_t + \mathbf{U}^o\mathbf{h}_{t-1}) \tag{8}$$
$$\tilde{\mathbf{c}}_\mathbf{t} = \tanh(\mathbf{W}^c\mathbf{x}_t + \mathbf{U}^c\mathbf{h}_{t-1}) \tag{9}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_\mathbf{t} \tag{10}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \tag{11}$$

The LSTM can use its memory vector $\mathbf{c}_t$ as a register of counters (Weiss et al., 2018). Merrill (2019) showed that the s-LSTM has $\Theta(\log n)$ space.

GRUs (Cho et al., 2014) Another kind of gated RNN is the GRU.

$$\mathbf{z}_t = \sigma(\mathbf{W}^z\mathbf{x}_t + \mathbf{U}^z\mathbf{h}_{t-1}) \tag{12}$$
$$\mathbf{r}_t = \sigma(\mathbf{W}^r\mathbf{x}_t + \mathbf{U}^r\mathbf{h}_{t-1}) \tag{13}$$
$$\mathbf{u}_t = \tanh\left(\mathbf{W}^u\mathbf{x}_t + \mathbf{U}^u(\mathbf{r}_t \odot \mathbf{h}_{t-1})\right) \tag{14}$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \mathbf{u}_t. \tag{15}$$

Weiss et al. (2018) found that, unlike the LSTM, the GRU cannot use its memory to count dynamically. Merrill (2019) showed the s-GRU has $\Theta(1)$ space.

[6] I.e., the minimal state representation needed to compute M_θ correctly. This distinction is important for architectures like attention, for which some implementations may retain unusable information such as input embedding order.

[7] With respect to our presented definition of RNNs, the concatenation of $\mathbf{h}_t$ and $\mathbf{c}_t$ can be seen as the recurrently updated state. However in all discussions of LSTMs we treat only $\mathbf{h}_t$ as the LSTM's 'state', in line with common practice.

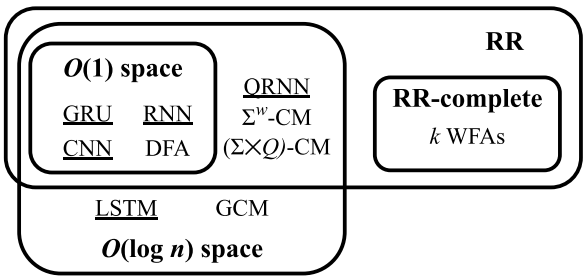

Figure 2: Diagram of the relations between encoders. Neural networks are underlined. We group by asymptotic upper bound (O), as opposed to tight (Θ).

QRNNs Bradbury et al. (2016) propose QRNNs as a computationally efficient hybrid of LSTMs and CNNs. Let $*$ denote convolution over time, let $\mathbf{W}^z, \mathbf{W}^f, \mathbf{W}^o \in \mathbb{Q}^{d_x \times w \times k}$ be convolutions with window length w, and let $\mathbf{X} \in \mathbb{Q}^{n \times d_x}$ denote the matrix of n input vectors. An *ifo*-QRNN (henceforth referred to as a QRNN) with *window length* w is defined by $\mathbf{W}^z$, $\mathbf{W}^f$, and $\mathbf{W}^o$ as follows:

$$\mathbf{Z} = \tanh(\mathbf{W}^z * \mathbf{X}) \tag{16}$$

$$\mathbf{F} = \sigma(\mathbf{W}^f * \mathbf{X}) \tag{17}$$

$$\mathbf{O} = \sigma(\mathbf{W}^o * \mathbf{X}) \tag{18}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{z}_t \tag{19}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t \tag{20}$$

where $\mathbf{z}_t, \mathbf{f}_t, \mathbf{o}_t$ are respectively rows of $\mathbf{Z}, \mathbf{F}, \mathbf{O}$. A QRNN Q can be seen as an LSTM in which all uses of the state vector $\mathbf{h}_t$ have been replaced with a computation over the last w input tokens–in this way it is similar to a CNN.

The s-QRNN has $\Theta(\log n)$ space, as the analysis of Merrill (2019) for the s-LSTM directly applies. Indeed, any s-QRNN is also a (Σ^w)-restricted CM extended with $=\pm 1$ ("set to ± 1") operations.

3 State Expressiveness

We now turn to presenting our results. In this section, we develop a hierarchy of single-layer RNNs based on their state expressiveness. A set-theoretic view of the hierarchy is shown in Figure 2.

Let $\mathcal{R}$ be the set of rational series. The hierarchy relates $\Theta(\log n)$ space to the following sets:

- **RR** As in Peng et al. (2018), we say that An encoder is *rationally recurrent (RR)* iff its state expressiveness is a subset of $\mathcal{R}$.
- **RR-hard** An encoder is *RR-hard* iff its state expressiveness contains $\mathcal{R}$. A Turing machine is RR-hard, as it can simulate any WFA.
- **RR-complete** Finally, an encoder is *RR-complete* iff its state expressiveness is equivalent to $\mathcal{R}$. A trivial example of an RR-complete encoder is a vector of k WFAs.

The different RNNs are divided between the intersections of these classes. In Subsection 3.1, we prove that the s-LSTM, already established to have $\Theta(\log n)$ space, is not RR. In Subsection 3.2, we demonstrate that encoders with restricted counting ability (e.g., QRNNs) are RR, and in Subsection 3.3, we show the same for all encoders with finite state (CNNs, s-RNNs, and s-GRUs). In Subsection 3.4, we demonstrate that none of these RNNs are RR-hard. In Appendix F, we extend this analysis from RNNs to self attention.

3.1 Counting Beyond RR

We find that encoders like the s-LSTM—which, as discussed in Subsection 2.3, is "aware" of its current counter values—are not RR. To do this, we construct $f_0 : \{a, b\}^* \to \mathbb{N}$ that requires counter awareness to compute on strings of the form a^*b^*, making it not rational. We then construct an s-LSTM computing f_0 over a^*b^*.

Let $\#_{a-b}(x)$ denote the number of as in string x minus the number of bs.

Definition 5 (Rectified counting)**.**

$$f_0 : x \mapsto \begin{cases} \#_{a-b}(x) & \text{if } \#_{a-b}(x) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 1. *For all $f : \{a, b\}^* \to \mathbb{N}$, if $f(a^i b^j) = f_0(a^i b^j)$ for all $i, j \in \mathbb{N}$, then $f \notin \mathcal{R}$.*

Proof. Consider the Hankel sub-block $\mathbf{A}_n$ of H_f with prefixes $P_n = \{a^i\}_{i \leq n}$ and suffixes $S_n = \{b^j\}_{j \leq n}$. $\mathbf{A}_n$ is lower-triangular:

$$\begin{pmatrix} 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & \cdots \\ 2 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{21}$$

Therefore $\operatorname{rank}(\mathbf{A}_n) = n-1$. Thus, for all n, there is a sub-block of H_f with rank $n-1$, and so $\operatorname{rank}(H_f)$ is unbounded. It follows from Theorem 1 that there is no WFA computing f. □

Theorem 2. *The s-LSTM is not RR.*

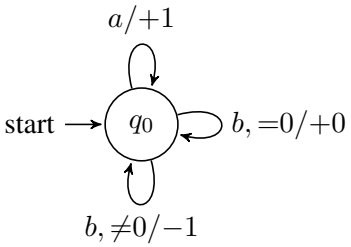

Figure 3: A 1-CM computing f_0 for $x \in \{a^i b^j \mid i, j \in \mathbb{N}\}$. Let $\sigma/\pm m$ denote a transition that consumes σ and updates the counter by $\pm m$. We write $\sigma, =0/\pm m$ (or $\neq$) for a transition that requires the counter is 0.

Proof. Assume the input has the form $a^i b^j$ for some i, j. Consider the following LSTM [8]:

$$i_t = \sigma\big(10N h_{t-1} - 2N\mathbb{1}_{=b}(x_t) + N\big) \quad (22)$$

$$\tilde{c}_t = \tanh\big(N\mathbb{1}_{=a}(x_t) - N\mathbb{1}_{=b}(x_t)\big) \quad (23)$$

$$c_t = c_{t-1} + i_t \tilde{c}_t \quad (24)$$

$$h_t = \tanh(c_t). \quad (25)$$

Let $N \to \infty$. Then $i_t = 0$ iff $x_t = b$ and $h_{t-1} = 0$ (i.e. $c_{t-1} = 0$). Meanwhile, $\tilde{c}_t = 1$ iff $x_t = a$. The update term becomes

$$i_t \tilde{c}_t = \begin{cases} 1 & \text{if } x_t = a \\ -1 & \text{if } x_t = b \text{ and } c_{t-1} > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (26)$$

For a string $a^i b^j$, the update in (26) is equivalent to the CM in Figure 3. Thus, by Lemma 1, the s-LSTM (and the general CM) is not RR. □

3.2 Rational Counting

While the counter awareness of a general CM enables it to compute non-rational functions, CMs that cannot view their counters are RR.

Theorem 3. *Any Σ-restricted CM is RR.*

Proof. We show that any function that a Σ-restricted CM can compute can also be computed by a collection of WFAs. The CM update operations ($-1, +0, +1$, or $\times 0$) can all be reexpressed in terms of functions $\mathbf{r}(x), \mathbf{u}(x) : \Sigma^* \to \mathbb{Z}^k$ to get:

$$\mathbf{c}_t = \mathbf{r}(x_t)\mathbf{c}_{t-1} + \mathbf{u}(x_t) \quad (27)$$

$$\mathbf{c}_t = \textstyle\sum_{i=1}^{t} \left(\prod_{j=i+1}^{t} \mathbf{r}(x_j)\right) \mathbf{u}(x_i). \quad (28)$$

A WFA computing $[\mathbf{c}_t]_i$ is shown in Figure 4. □

[8] In which f_t and o_t are set to 1, such that $c_t = c_{t-1} + i_t \tilde{c}_t$.

The WFA in Figure 4 also underlies unigram rational RNNs (Peng et al., 2018). Thus, Σ-restricted CMs are actually a special case of unigram WFAs. In Appendix A, we show the more general result:

Theorem 4. *Any $(\Sigma \times Q)$-restricted CM is RR.*

In many rational RNNs, the updates at different time steps are independent of each other outside of a window of w tokens. Theorem 4 tells us this independence is not an essential property of rational encoders. Rather, any CM where the update is conditioned by finite state (as opposed to being conditioned by a local window) is in fact RR.

Furthermore, since (Σ^w)-restricted CMs are a special case of $(\Sigma \times Q)$-restricted CMs, Theorem 4 can be directly applied to show that the s-QRNN is RR. See Appendix A for further discussion of this.

3.3 Finite-Space RR

Theorem 4 motivates us to also think about finite-space encoders: i.e., encoders with no counters" where the output at each prefix is fully determined by a finite amount of memory. The following lemma implies that any finite-space encoder is RR:

Lemma 2. *Any function $f : \Sigma^* \to \mathbb{Q}$ computable by a $\Theta(1)$-space encoder is a rational series.*

Proof. Since f is computable in $\Theta(1)$ space, there exists a DFA A_f whose accepting states are isomorphic to the range of f. We convert A_f to a WFA by labelling each accepting state by the value of f that it corresponds to. We set the starting weight of the initial state to 1, and 0 for every other state. We assign each transition weight 1. □

Since the CNN, s-RNN, and s-GRU have finite state, we obtain the following result:

Theorem 5. *The CNN, s-RNN, and s-GRU are RR.*

While Schwartz et al. (2018) and Peng et al. (2018) showed the CNN to be RR over the max-plus semiring, Theorem 5 shows the same holds for $\langle \mathbb{Q}, \cdot, + \rangle$.

3.4 RR Completeness

While "rational recurrence" is often used to indicate the simplicity of an RNN architecture, we find in this section that WFAs are surprisingly computationally powerful. Figure 5 shows a WFA mapping binary string to their numeric value, proving WFAs have $\Theta(n)$ space. We now show that none of our RNNs are able to simulate an arbitrary WFA, even in the unsaturated form.

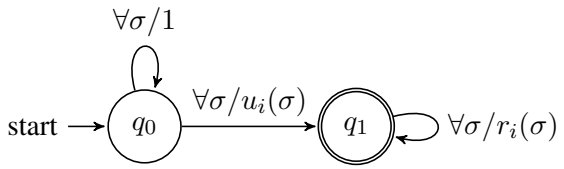

Figure 4: WFA simulating unit i of a Σ-restricted CM. Let $\forall\sigma/w(\sigma)$ denote a set of transitions consuming each token σ with weight $w(\sigma)$. We use standard DFA notation to show initial weights $\lambda(q_0) = 1, \lambda(q_1) = 0$ and accepting weights $\rho(q_0) = 0, \rho(q_1) = 1$.

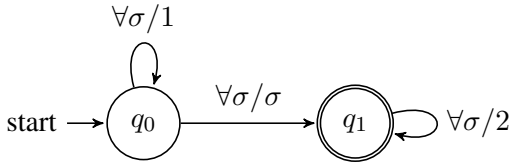

Figure 5: A WFA mapping binary strings to their numeric value. This can be extended for any base > 2. Cortes and Mohri (2000) present a similar construction. Notation is the same as Figure 4.

Theorem 6. *Both the saturated and unsaturated RNN, GRU, QRNN, and LSTM*[9] *are not RR-hard.*

Proof. Consider the function f_b mapping binary strings to their value, e.g. $101 \mapsto 5$. The WFA in Figure 5 shows that this function is rational.

The value of f_b grows exponentially with the sequence length. On the other hand, the value of the RNN and GRU cell is bounded by 1, and QRNN and LSTM cells can only grow linearly in time. Therefore, these encoders cannot compute f_b. □

In contrast, memory networks can have $\Theta(n)$ space. Appendix G explores this for stack RNNs.

3.5 Towards Transformers

Appendix F presents preliminary results extending saturation analysis to self attention. We show saturated self attention is not RR and consider its space complexity. We hope further work will more completely characterize saturated self attention.

4 Language Expressiveness

Having explored the set of functions expressible internally by different saturated RNN encoders, we turn to the languages recognizable when using them with a decoder. We consider the following setup:

1. An s-RNN encodes x to a vector $\mathbf{h}_t \in \mathbb{Q}^k$.
2. A *decoder* function maps the last state $\mathbf{h}_t$ to an accept/reject decision, respectively: $\{1, 0\}$.

[9] As well as CMs.

We say that a language L is decided by an encoder-decoder pair $\mathbf{e}, \mathbf{d}$ if $\mathbf{d}(\mathbf{e}(x)) = 1$ for every sequence $x \in L$ and otherwise $\mathbf{d}(\mathbf{e}(x)) = 0$. We explore which languages can be decided by different encoder-decoder pairings.

Some related results can be found in Cortes and Mohri (2000), who study the expressive power of WFAs in relation to CFGs under a slightly different definition of language recognition.

4.1 Linear Decoders

Let $\mathbf{d}_1$ be the single-layer linear decoder

$$\mathbf{d}_1(\mathbf{h}_t) \triangleq \mathbb{1}_{>0}(\mathbf{w} \cdot \mathbf{h}_t + b) \in \{0, 1\} \tag{29}$$

parameterized by $\mathbf{w}$ and b. For an encoder architecture E, we denote by $D_1(E)$ the set of languages decidable by E with $\mathbf{d}_1$. We use $D_2(E)$ analogously for a 2-layer decoder with $\mathbb{1}_{>0}$ activations, where the first layer has arbitrary width.

4.2 A Decoder Adds Power

We refer to sets of strings using regular expressions, e.g. $a^* = \{a^i \mid i \in \mathbb{N}\}$. To illustrate the purpose of the decoder, consider the following language:

$$L_{\leq} = \{x \in \{a, b\}^* \mid \#_{a-b}(x) \leq 0\}. \tag{30}$$

The Hankel sub-block of the indicator function for $L_{\leq}$ over $P = a^*$, $S = b^*$ is lower triangular. Therefore, no RR encoder can compute it.

However, adding the D_1 decoder allows us to compute this indicator function with an s-QRNN, which is RR. We set the s-QRNN layer to compute the simple series $c_t = \#_{a-b}(x)$ (by increasing on a and decreasing on b). The D_1 layer then checks $c_t \leq 0$. So, while the indicator function for $L_{\leq}$ is not itself rational, it can be easily recovered from a rational representation. Thus, $L_{\leq} \in D_1$(s-QRNN).

4.3 Case Study: $a^n b^n$

We compare the language expressiveness of several rational and non-rational RNNs on the following:

$$a^n b^n \triangleq \{a^n b^n \mid n \in \mathbb{N}\} \tag{31}$$

$$a^n b^n \Sigma^* \triangleq \{a^n b^n (a|b)^* \mid 0 < n\}. \tag{32}$$

$a^n b^n$ is more interesting than $L_{\leq}$ because the D_1 decoder cannot decide it simply by asking the encoder to track $\#_{a-b}(x)$, as that would require it to compute the non-linearly separable $=0$ function. Thus, it appears at first that deciding $a^n b^n$ with D_1

might require a non-rational RNN encoder. However, we show below that this is not the case.

Let $\circ$ denote stacking two layers. We will go on to discuss the following results:

$$a^n b^n \in D_1(\text{WFA}) \quad (33)$$
$$a^n b^n \in D_1(\text{s-LSTM}) \quad (34)$$
$$a^n b^n \notin D_1(\text{s-QRNN}) \quad (35)$$
$$a^n b^n \in D_1(\text{s-QRNN} \circ \text{s-QRNN}) \quad (36)$$
$$a^n b^n \in D_2(\text{s-QRNN}) \quad (37)$$
$$a^n b^n \Sigma^* \in D_1(\text{s-LSTM}) \quad (38)$$
$$a^n b^n \Sigma^* \notin D\,(\text{s-QRNN}) \text{ for } \textit{any } D \quad (39)$$
$$a^n b^n \Sigma^* \cup \{\epsilon\} \in D_1(\text{s-QRNN} \circ \text{s-QRNN}) \quad (40)$$

WFAs (Appendix B) In Theorem 8 we present a function $f : \Sigma^* \to \mathbb{Q}$ satisfying $f(x) > 0$ iff $x \in a^n b^n$, and show that H_f has finite rank. It follows that there exists a WFA that can decide $a^n b^n$ with the D_1 decoder. Counterintuitively, $a^n b^n$ can be recognized using rational encoders.

QRNNs (Appendix C) Although $a^n b^n \in D_1(\text{WFA})$, it does not follow that every rationally recurrent model can also decide $a^n b^n$ with the help of D_1. Indeed, in Theorem 9, we prove that $a^n b^n \notin D_1(\text{s-QRNN})$, whereas $a^n b^n \in D_1(\text{s-LSTM})$ (Theorem 13).

It is important to note that, with a more complex decoder, the QRNN *could* recognize $a^n b^n$. For example, the s-QRNN can encode $c_1 = \#_{a-b}(x)$ and set c_2 to check whether x contains ba, from which a D_2 decoder can recognize $a^n b^n$ (Theorem 10).

This does not mean the hierarchy dissolves as the decoder is strengthened. We show that $a^n b^n \Sigma^*$—which seems like a trivial extension of $a^n b^n$—is not recognizable by the s-QRNN with *any* decoder.

This result may appear counterintuitive, but in fact highlights the s-QRNN's lack of counter awareness: it can only passively encode the information needed by the decoder to recognize $a^n b^n$. Failing to recognize that a valid prefix has been matched, it cannot act to preserve that information after additional input tokens are seen. We present a proof in Theorem 11. In contrast, in Theorem 14 we show that the s-LSTM can directly encode an indicator for $a^n b^n \Sigma^*$ in its internal state.

Proof sketch: $a^n b^n \Sigma^* \notin D(\text{s-QRNN})$. A sequence $s_1 \in a^n b^n \Sigma^*$ is shuffled to create $s_2 \notin a^n b^n \Sigma^*$ with an identical multi-set of counter updates.[10] Counter updates would be order agnostic if not for reset operations, and resets mask all history, so extending s_1 and s_2 with a single suffix s containing all of their w-grams reaches the same final state. Then for any D, $D(\text{s-QRNN})$ cannot separate them. We formalize this in Theorem 11.

We refer to this technique as the *suffix attack*, and note that it can be used to prove for multiple other languages $L \in D_2(\text{s-QRNN})$ that $L \cdot \Sigma^*$ is not in $D(\text{s-QRNN})$ for any decoder D.

2-layer QRNNs Adding another layer overcomes the weakness of the 1-layer s-QRNN, at least for deciding $a^n b^n$. This follows from the fact that $a^n b^n \in D_2(\text{s-QRNN})$: the second QRNN layer can be used as a linear layer.

Similarly, we show in Theorem 10 that a 2-layer s-QRNN can recognize $a^n b^n \Sigma^* \cup \{\epsilon\}$. This suggests that adding a second s-QRNN layer compensates for some of the weakness of the 1-layer s-QRNN, which, by the same argument for $a^n b^n \Sigma^*$ cannot recognize $a^n b^n \Sigma^* \cup \{\epsilon\}$ with any decoder.

4.4 Arbitrary Decoder

Finally, we study the theoretical case where the decoder is an arbitrary recursively enumerable (RE) function. We view this as a loose upper bound of stacking many layers after a rational encoder. What information is inherently lost by using a rational encoder? WFAs can uniquely encode each input, making them Turing-complete under this setup; however, this does not hold for rational s-RNNs.

RR-complete Assuming an RR-complete encoder, a WFA like Figure 5 can be used to encode each possible input sequence over Σ to a unique number. We then use the decoder as an oracle to decide any RE language. Thus, an RR-complete encoder with an RE decoder is Turing-complete.

Bounded space However, the $\Theta(\log n)$ space bound of saturated rational RNNs like the s-QRNN means these models cannot fully encode the input. In other words, some information about the prefix $x_{:t}$ must be lost in $\mathbf{c}_t$. Thus, rational s-RNNs are not Turing-complete with an RE decoder.

5 Experiments

In Subsection 4.3, we showed that different saturated RNNs vary in their ability to recognize $a^n b^n$ and $a^n b^n \Sigma^*$. We now test empirically whether

[10] Since QRNN counter updates depend only on the w-grams present in the sequence.

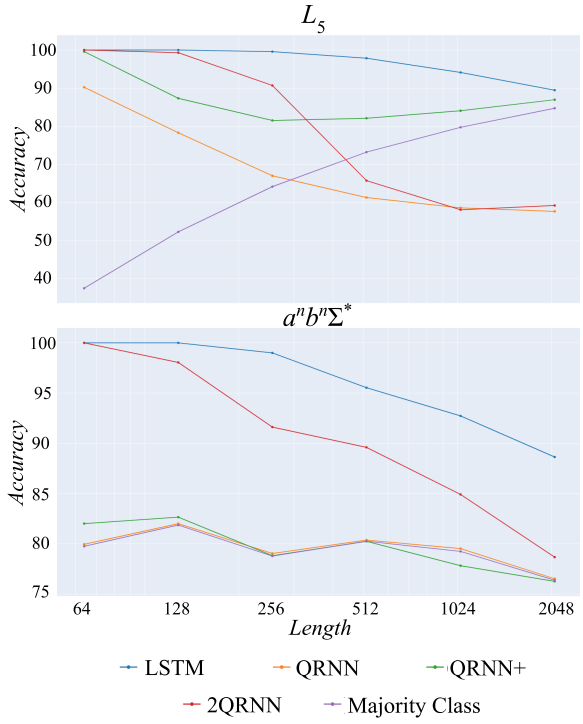

Figure 6: Accuracy recognizing L_5 and $a^n b^n \Sigma^*$. "QRNN+" is a QRNN with a 2-layer decoder, and "2QRNN" is a 2-layer QRNN with a 1-layer decoder.

these predictions carry over to the learnable capacity of unsaturated RNNs.[11] We compare the QRNN and LSTM when coupled with a linear decoder D_1. We also train a 2-layer QRNN ("QRNN2") and a 1-layer QRNN with a D_2 decoder ("QRNN+").

We train on strings of length 64, and evaluate generalization on longer strings. We also compare to a baseline that always predicts the majority class. The results are shown in Figure 6. We provide further experimental details in Appendix E.

Experiment 1 We use the following language, which has similar formal properties to $a^n b^n$, but with a more balanced label distribution:

$$L_5 = \left\{ x \in (a|b)^* \mid |\#_{a-b}(x)| < 5 \right\}. \quad (41)$$

In line with (34), the LSTM decides L_5 perfectly for $n \leq 64$, and generalizes fairly well to longer strings. As predicted in (35), the QRNN cannot fully learn L_5 even for $n = 64$. Finally, as predicted in (36) and (37), the 2-layer QRNN and the QRNN with D_2 do learn L_5. However, we see that they do not generalize as well as the LSTM for longer strings. We hypothesize that these multi-layer models require more epochs to reach the same generalization performance as the LSTM.[12]

Experiment 2 We also consider $a^n b^n \Sigma^*$. As predicted in (38) and (40), the LSTM and 2-layer QRNN decide $a^n b^n \Sigma^*$ flawlessly for $n = 64$. A 1-layer QRNN performs at the majority baseline for all n with both a 1 and 2-layer decoder. Both of these failures were predicted in (39). Thus, the only models that learned $a^n b^n \Sigma^*$ were exactly those predicted by the saturated theory.

6 Conclusion

We develop a hierarchy of saturated RNN encoders, considering two angles: space complexity and rational recurrence. Based on the hierarchy, we formally distinguish the state expressiveness of the non-rational s-LSTM and its rational counterpart, the s-QRNN. We show further distinctions in state expressiveness based on encoder space complexity.

Moreover, the hierarchy translates to differences in language recognition capabilities. Strengthening the decoder alleviates some, but not all, of these differences. We present two languages, both recognizable by an LSTM. We show that one can be recognized by an s-QRNN only with the help of a decoder, and that the other cannot be recognized by an s-QRNN with the help of any decoder.

While this means existing rational RNNs are fundamentally limited compared to LSTMs, we find that it is not necessarily being rationally recurrent that limits them: in fact, we prove that a WFA can perfectly encode its input—something no saturated RNN can do. We conclude with an analysis that shows that an RNN architecture's strength must also take into account its space complexity. These results further our understanding of the inner working of NLP systems. We hope they will guide the development of more expressive rational RNNs.

Acknowledgments

We appreciate Amir Yehudayoff's help in finding the WFA used in Theorem 8, and the feedback of researchers at the Allen Institute for AI, our anonymous reviewers, and Tobias Jaroslaw. The project was supported in part by NSF grant IIS-1562364, Israel Science Foundation grant no.1319/16, and the European Research Council under the EU's Horizon 2020 research and innovation program, grant agreement No. 802774 (iEXTRACT).

[11] https://github.com/viking-sudo-rm/rr-experiments

[12] As shown by the baseline, generalization is challenging because positive labels become less likely as strings get longer.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Borja Balle, Xavier Carreras, Franco M. Luque, and Ariadna Quattoni. 2014. Spectral learning of weighted automata. *Machine Learning*, 96(1):33–63.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2016. Quasi-recurrent neural networks.

J. W. Carlyle and A. Paz. 1971. Realizations by stochastic finite automata. *J. Comput. Syst. Sci.*, 5(1):26–40.

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. 2018. Recurrent neural networks as weighted language recognizers. In *Proc. of NAACL*, pages 2261–2271.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proc. of EMNLP*, pages 1724–1734.

Corinna Cortes and Mehryar Mohri. 2000. Context-free recognition with weighted automata. *Grammars*, 3(2/3):133–150.

Jesse Dodge, Roy Schwartz, Hao Peng, and Noah A. Smith. 2019. RNN architecture learning with sparse regularization. In *Proc. of EMNLP*, pages 1179–1184.

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

Patrick C Fischer. 1966. Turing machines with restricted memory access. *Information and Control*, 9(4):364–379.

Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. 1968. Counter machines and counter languages. *Mathematical Systems Theory*, 2(3):265–283.

Michel Fliess. 1974. Matrices de Hankel. *J. Math. Pures Appl*, 53(9):197–222.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

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

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proc. of EMNLP*, pages 1746–1751.

Yann Lecun and Yoshua Bengio. 1995. *The Handbook of Brain Theory and Neural Networks*, chapter "Convolutional Networks for Images, Speech, and Time Series". MIT Press.

William Merrill. 2019. Sequential neural networks as automata. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 1–13.

William Merrill. 2020. On the linguistic capacity of real-time counter automata.

Hao Peng, Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. Rational recurrences. In *Proc. of EMNLP*, pages 1203–1214.

Jacques Sakarovitch. 2009. Rational and recognisable power series. In *Handbook of Weighted Automata*, pages 105–174. Springer.

Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. Bridging CNNs, RNNs, and weighted finite-state machines. In *Proc. of ACL*, pages 295–305.

Hava T. Siegelmann and Eduardo D. Sontag. 1992. On the computational power of neural nets. In *Proc. of COLT*, pages 440–449.

Hava T. Siegelmann and Eduardo D. Sontag. 1994. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360.

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019a. LSTM networks can perform dynamic counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M. Shieber. 2019b. Memory-augmented recurrent neural networks can learn generalized Dyck languages.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition.

A Rational Counting

We extend the result in Theorem 3 as follows.

Theorem 7. *Any $(\Sigma \times Q)$-restricted CM is rationally recurrent.*

Proof. We present an algorithm to construct a WFA computing an arbitrary counter in a $(\Sigma \times Q)$-restricted CM. First, we create two independent copies of the transition graph for the restricted CM. We refer to one copy of the CM graph as the *add graph*, and the other as the *multiply graph*.

The initial state in the add graph receives a starting weight of 1, and every other state receives a starting weight of 0. Each state in the add graph receives an accepting weight of 0, and each state in the multiply graph receives an accepting weight of 1. In the add graph, each transition receives a weight of 1. In the multiply graph, each transition receives a weight of 0 if it represents $\times 0$, and 1 otherwise. Finally, for each non-multiplicative update $\sigma/{+m}$[13] from q_i to q_j in the original CM, we add a WFA transition σ/m from q_i in the add graph to q_j in the multiply graph.

Each counter update creates one path ending in the multiply graph. The path score is set to 0 if that counter update is "erased" by a $\times 0$ operation. Thus, the sum of all the path scores in the WFA equals the value of the counter. □

This construction can be extended to accommodate $=m$ counter updates from q_i to q_j by adding an additional transition from the initial state to q_j in the multiplication graph with weight m. This allows us to apply it directly to s-QRNNs, whose update operations include $=1$ and $=-1$.

B WFAs

We show that while WFAs cannot directly encode an indicator for the language $a^n b^n = \{a^n b^n | \mid n \in \mathbb{N}\}$, they can encode a function that can be thresholded to recognize $a^n b^n$, i.e.:

Theorem 8. *The language $a^n b^n = \{a^n b^n \mid n \in \mathbb{N}\}$ over $\Sigma = \{a, b\}$ is in D_1(WFA).*

We prove this by showing a function whose Hankel matrix has finite rank that, when combined with the identity transformation (i.e., $w = 1, b = 0$) followed by thresholding, is an indicator for $a^n b^n$. Using the shorthand $\sigma(x) = \#_\sigma(x)$, the function is:

$$f(w) = \begin{cases} 0.5 - 2(a(x) - b(x))^2 & \text{if } x \in a^* b^* \\ -0.5 & \text{otherwise.} \end{cases} \tag{42}$$

Immediately f satisfies $\mathbb{1}_{>0}(f(x)) \iff x \in a^n b^n$. To prove that its Hankel matrix, H_f, has finite rank, we will create 3 infinite matrices of ranks 3, 3 and 1, which sum to H_f. The majority of the proof will focus on the rank of the rank 3 matrices, which have similar compositions.

We now show 3 series r, s, t and a set of series they can be combined to create. These series will be used to create the base vectors for the rank 3 matrices.

$$a_i = \frac{i(i+1)}{2} \tag{43}$$
$$b_i = i^2 - 1 \tag{44}$$
$$r_i = \text{fix}_0(i, a_{i-2}) \tag{45}$$
$$s_i = \text{fix}_1(i, -b_{i-1}) \tag{46}$$
$$t_i = \text{fix}_2(i, a_{i-1}) \tag{47}$$

where for every $j \leq 2$,

$$\text{fix}_j(i, x) = \begin{cases} x & \text{if } i > 2 \\ 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{48}$$

Lemma 3. *Let $c_i = 1 - 2i^2$ and $\{c^{(k)}\}_{k \in \mathbb{N}}$ be the set of series defined $c_i^{(k)} = c_{|i-k|}$. Then for every $i, k \in \mathbb{N}$,*

$$c_i^{(k)} = c_0^{(k)} r_i + c_1^{(k)} s_i + c_2^{(k)} t_i.$$

Proof. For $i \in \{0, 1, 2\}$, r_i, s_i and t_i collapse to a 'select' operation, giving the true statement $c_i^{(k)} = c_i^{(k)} \cdot 1$. We now consider the case $i > 2$. Substituting the series definitions in the right side of the equation gives

$$c_k a_{i-2} + c_{|k-1|}(-b_{i-1}) + c_{k-2} a_{i-1} \tag{49}$$

which can be expanded to

$$\begin{aligned} &(1 - 2k^2) && \cdot \frac{i^2 - 3i + 2}{2} && + \\ &(1 - 2(k-1)^2) && \cdot (1 - (i-1)^2) && + \\ &(1 - 2(k-2)^2) && \cdot \frac{(i-1)i}{2}. \end{aligned}$$

[13] Note that $m = -1$ for the -1 counter update.

Reordering the first component and partially opening the other two gives

$$(-2k^2+1)\frac{i^2-3i+2}{2}+$$
$$(-2k^2+4k-1)(2i-i^2)+$$
$$(-k^2+4k-3.5)(i^2-i)$$

and a further expansion gives

$$\begin{aligned}
-k^2i^2+ \quad & 0.5i^2+3k^2i-1.5i-2k^2+1+\\
2k^2i^2-4ki^2+ \quad & i^2-4k^2i+8ki-2i+\\
-k^2i^2+4ki^2- \quad & 3.5i^2+k^2i-4ki+3.5i
\end{aligned}$$

which reduces to

$$-2i^2+4ki-2k^2+1=1-2(k-i)^2=c_i^{(k)}.$$

□

We restate this as:

Corollary 1. *For every $k\in\mathbb{N}$, the series $c^{(k)}$ is a linear combination of the series r, s and t.*

We can now show that f is computable by a WFA, proving Theorem 8. By Theorem 1, it is sufficient to show that H_f has finite rank.

Lemma 4. *H_f has finite rank.*

Proof. For every $P,S\subseteq\{a,b\}^*$, denote

$$[H_f|_{P,S}]_{u,v}=\begin{cases}[H_f]_{u,v} & \text{if } u\in P \text{ and } v\in S\\ 0 & \text{otherwise}\end{cases}$$

Using regular expressions to describe P,S, we create the 3 finite rank matrices which sum to H_f:

$$A=(H_f+0.5)|_{a^*,a^*b^*} \tag{50}$$
$$B=(H_f+0.5)|_{a^*b^+,b^*} \tag{51}$$
$$C=(-0.5)|_{u,v}. \tag{52}$$

Intuitively, these may be seen as a "split" of H_f into sections as in Figure 7, such that A and B together cover the sections of H_f on which $u{\cdot}v$ does not contain the substring ba (and are equal on them to $H_f+0.5$), and C is simply the constant matrix -0.5. Immediately, $H_f=A+B+C$, and $\mathrm{rank}(C)=1$.

We now consider A. Denote $P_A=a^*, S_A=a^*b^*$. A is non-zero only on indices $u\in P_A, v\in S_A$, and for these, $u{\cdot}v\in a^*b^*$ and $A_{u,v}=0.5+f(u{\cdot}v)=1-2(a(u)+a(v)-b(v))^2$. This gives that for every $u\in P_A, v\in S_A$,

$$A_{u,v}=c_{|a(u)-(b(v)-a(v))|}=c^{(a(u))}_{b(v)-a(v)}. \tag{53}$$

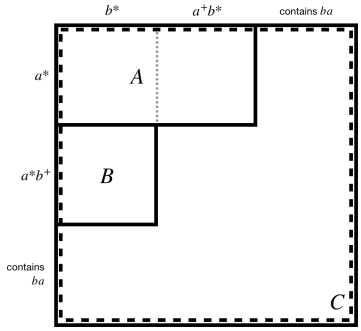

Figure 7: Intuition of the supports of A, B and C.

For each $\tau\in\{r,s,t\}$, define $\tilde{\tau}\in\mathbb{Q}^{\{a,b\}^*}$ as

$$\tilde{\tau}_v=\mathbb{1}_{v\in a^*b^*}\cdot\tau_{b(v)-a(v)}. \tag{54}$$

We get from Corollary 1 that for every $u\in a^*$, the uth row of A is a linear combination of $\tilde{r}$, $\tilde{s}$, and $\tilde{t}$. The remaining rows of A are all $\mathbf{0}$ and so also a linear combination of these, and so $\mathrm{rank}(A)\leq 3$.

Similarly, we find that the nonzero entries of B satisfy

$$B_{u,v}=c_{|b(v)-(a(u)-b(u))|}=c^{(b(v))}_{a(u)-b(u)} \tag{55}$$

and so, for $\tau\in\{r,s,t\}$, the columns of B are linear combinations of the columns $\tau'\in\mathbb{Q}^{\{a,b\}^*}$ defined

$$\tau'_u=\mathbb{1}_{u\in a^*b^+}\cdot\tau_{a(u)-b(u)}. \tag{56}$$

Thus we conclude $\mathrm{rank}(B)\leq 3$.

Finally, $H_f=A+B+C$, and so by the subadditivity of rank in matrices,

$$\mathrm{rank}(H_f)\leq\sum_{M=A,B,C}\mathrm{rank}(M)=7. \tag{57}$$

□

In addition, the rank of $\tilde{H}_f\in\mathbb{Q}^{\{a,b\}^{\leq 2},\{a,b\}^{\leq 2}}$ defined $[\tilde{H}_f]_{u,v}=[H_f]_{u,v}$ is 7, and so we can conclude that the bound in the proof is tight, i.e., $\mathrm{rank}(H_f)=7$. From here $\tilde{H}_f$ is a complete subblock of H_f and can be used to explicitly construct a WFA for f, using the spectral method described by Balle et al. (2014).

C s-QRNNs

Theorem 9. *No s-QRNN with a linear threshold decoder can recognize $a^nb^n=\{a^nb^n\mid n\in\mathbb{N}\}$, i.e., $a^nb^n\notin D_1$(s-QRNN).*

Proof. An *ifo* s-QRNN can be expressed as a Σ^k-restricted CM with the additional update operations $\{:=-1, :=1\}$, where k is the window size of the QRNN. So it is sufficient to show that such a machine, when coupled with the decoder D_1 (linear translation followed by thresholding), cannot recognize $a^n b^n$.

Let $\mathcal{A}$ be some such CM, with window size k and h counters. Take $n = k + 10$ and for every $m \in \mathbb{N}$ denote $w_m = a^n b^m$ and the counter values of $\mathcal{A}$ after w_m as $c^m \in \mathbb{Q}^h$. Denote by u_t the vector of counter update operations made by this machine on input sequence w_m at time $t \leq n + m$. As $\mathcal{A}$ is dependent only on the last k counters, necessarily all u_{k+i} are identical for every $i \geq 1$.

It follows that for all counters in the machine that go through an assignment (i.e., $:=$) operation in u_{k+1}, their values in c^{k+i} are identical for every $i \geq 1$, and for every other counter j, $c_j^{k+i} - c_j^k = i \cdot \delta$ for some $\delta \in \mathbb{Z}$. Formally: for every $i \geq 1$ there are two sets I, $J = [h] \setminus I$ and constant vectors $\mathbf{u} \in \mathbb{N}^I, \mathbf{v} \in \mathbb{N}^J$ s.t. $c^{k+i}|_I = \mathbf{u}$ and $[c^{k+i} - c^k]|_J = \mathbf{i} \cdot v$.

We now consider the linear thresholder, defined by weights and bias $\mathbf{w}, b$. In order to recognise $a^n b^n$, the thresholder must satisfy:

$$\mathbf{w} \cdot c^{k+9} + b < 0 \quad (58)$$
$$\mathbf{w} \cdot c^{k+10} + b > 0 \quad (59)$$
$$\mathbf{w} \cdot c^{k+11} + b < 0 \quad (60)$$

Opening these equations gives:

$$\mathbf{w}|_J(\cdot c^k|_J + 9\mathbf{v}|_J) + \mathbf{w}|_I \cdot \mathbf{u} < 0 \quad (61)$$
$$\mathbf{w}|_J(\cdot c^k|_J + 10\mathbf{v}|_J) + \mathbf{w}|_I \cdot \mathbf{u} > 0 \quad (62)$$
$$\mathbf{w}|_J(\cdot c^k|_J + 11\mathbf{v}|_J) + \mathbf{w}|_I \cdot \mathbf{u} < 0 \quad (63)$$

but this gives $9w|_J \cdot \mathbf{v}|_J < 10w|_J \cdot \mathbf{v}|_J > 11w|_J \cdot \mathbf{v}|_J$, which is impossible.

□

However, this does not mean that the s-QRNN is entirely incapable of recognising $a^n b^n$. Increasing the decoder power allows it to recognise $a^n b^n$ quite simply:

Theorem 10. *For the two-layer decoder D_2, $a^n b^n \in D_2$(*s-QRNN*).*

Proof. Let $\#_{ba}(x)$ denote the number of ba 2-grams in x. We use s-QRNN with window size 2 to maintain two counters:

$$[\mathbf{c}_t]_1 = \#_{a-b}(x) \quad (64)$$
$$[\mathbf{c}_t]_2 = \#_{ba}(x). \quad (65)$$

$[\mathbf{c}_t]_2$ can be computed provided the QRNN window size is ≥ 2. A two-layer decoder can then check

$$0 \leq [\mathbf{c}_t]_1 \leq 0 \wedge [\mathbf{c}_t]_2 \leq 0. \quad (66)$$

□

Theorem 11 (Suffix attack)**.** *No s-QRNN and decoder can recognize the language $a^n b^n \Sigma^* = a^n b^n (a|b)^*$, $n > 0$, i.e., $a^n b^n \Sigma^* \notin L$(s-QRNN) for any decoder L.*

The proof will rely on the s-QRNN's inability to "freeze" a computed value, protecting it from manipulation by future input.

Proof. As in the proof for Theorem 9, it is sufficient to show that no Σ^k-restricted CM with the additional operations $\{:=-1, :=1\}$ can recognize $a^n b^n \Sigma^*$ for any decoder L.

Let $\mathcal{A}$ be some such CM, with window size k and h counters. For every $w \in \Sigma^n$ denote by $c(w) \in \mathbb{Q}^h$ the counter values of $\mathcal{A}$ after processing w. Denote by u_t the vector of counter update operations made by this machine on an input sequence w at time $t \leq |w|$. Recall that $\mathcal{A}$ is Σ^k restricted, meaning that u_i depends exactly on the window of the last k tokens for every i.

We now denote $j = k + 10$ and consider the sequences $w_1 = a^j b^j a^j b^j a^j b^j$, $w_2 = a^j b^{j-1} a^j b^{j+1} a^j b^j$. w_2 is obtained from w_1 by removing the $2j$-th token of w_1 and reinserting it at position $4j$.

As all of w_1 is composed of blocks of $\geq k$ identical tokens, the windows preceding all of the other tokens in w_1 are unaffected by the removal of the $2j$-th token. Similarly, being added onto the end of a substring b^k, its insertion does not affect the windows of the tokens after it, nor is its own window different from before. This means that overall, the set of all operations u_i performed on the counters is identical in w_1 and in w_2. The only difference is in their ordering.

w_1 and w_2 begin with a shared prefix a^k, and so necessarily the counters are identical after processing it. We now consider the updates to the counters after these first k tokens, these are determined by the windows of k tokens preceding each update.

First, consider all the counters that undergo some assignment ($:=$) operation during these sequences, and denote by $\{w\}$ the multiset of windows in $w \in \Sigma^k$ for which they are reset. w_1 and w_2 only contain k-windows of types $a^x b^{k-x}$ or $b^x a^{k-x}$, and so these must all re-appear in the shared suffix $b^j a^j b^j$ of w_1 and w_2, at which point they will be synchronised. It follows that these counters all finish with identical value in $c(w_1)$ and $c(w_2)$.

All the other counters are only updated using addition of $-1, 1$ and 0, and so the order of the updates is inconsequential. It follows that they too are identical in $c(w_1)$ and $c(w_2)$, and therefore necessarily that $c(w_1) = c(w_2)$.

From this we have w_1, w_2 satisfying $w_1 \in a^n b^n \Sigma^*, w_2 \notin a^n b^n \Sigma^*$ but also $c(w_1) = c(w_2)$. Therefore, it is not possible to distinguish between w_1 and w_2 with the help of any decoder, despite the fact that $w_1 \in a^n b^n \Sigma^*$ and $w_2 \notin a^n b^n \Sigma^*$. It follows that the CM and s-QRNN cannot recognize $a^n b^n \Sigma^*$ with any decoder. □

For the opposite extension $\Sigma^* a^n b^n$, in which the language is augmented by a *prefix*, we cannot use such a "suffix attack". In fact, $\Sigma^* a^n b^n$ can be recognized by an s-QRNN with window length $w \geq 2$ and a linear threshold decoder as follows: a counter counts $\#_{a-b}(x)$ and is reset to 1 on appearances of ba, and the decoder compares it to 0.

Note that we define decoders as functions from the final state to the output. Thus, adding an additional QRNN layer does not count as a "decoder" (as it reads multiple states). In fact, we show that having two QRNN layers allows recognizing $a^n b^n \Sigma^*$.

Theorem 12. *Let ϵ be the empty string. Then,*

$$a^n b^n \Sigma^* \cup \{\epsilon\} \in D_1(\text{s-QRNN} \circ \text{s-QRNN}).$$

Proof. We construct a two-layer s-QRNN from which $a^n b^n \Sigma^*$ can be recognized. Let \$ denote the left edge of the string. The first layer computes two quantities d_t and e_t as follows:

$$d_t = \#_{ba}(x) \tag{67}$$
$$e_t = \#_{\$b}(x). \tag{68}$$

Note that e_t can be interpreted as a binary value checking whether the first token was b. The second layer computes c_t as a function of d_t, e_t, and x_t (which can be passed through the first layer). We will demonstrate a construction for c_t by creating linearly separable functions for the gate terms f_t and z_t that update c_t.

$$f_t = \begin{cases} 1 & \text{if } d_t \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{69}$$

$$z_t = \begin{cases} 1 & \text{if } x_t = a \vee e_t \\ -1 & \text{otherwise.} \end{cases} \tag{70}$$

Now, the update function u_t to c_t can be expressed

$$u_t = f_t z_t = \begin{cases} +0 & \text{if } 0 < d_t \\ +1 & \text{if } d_t \leq 0 \wedge (x_t = a \vee e_t) \\ -1 & \text{otherwise.} \end{cases} \tag{71}$$

Finally, the decoder accepts iff $c_t \leq 0$. To justify this, we consider two cases: either x starts with b or a. If x starts with b, then $e_t = 0$, so we increment c_t by 1 and never decrement it. Since $0 < c_t$ for any t, we will reject x. If x starts with a, then we accept iff there exists a sequence of bs following the prefix of as such that both sequences have the same length. □

D s-LSTMs

In contrast to the s-QRNN, we show that the s-LSTM paired with a simple linear and thresholding decoder can recognize both $a^n b^n$ and $a^n b^n \Sigma^*$.

Theorem 13.

$$a^n b^n \in D_1(\text{s-LSTM}).$$

Proof. Assuming a string $a^i b^i$, we set two units of the LSTM state to compute the following functions using the CM in Figure 3:

$$[\mathbf{c}_t]_1 = \text{ReLU}(i - j) \tag{72}$$
$$[\mathbf{c}_t]_2 = \text{ReLU}(j - i). \tag{73}$$

We also add a third unit $[\mathbf{c}_t]_3$ that tracks whether the 2-gram ba has been encountered, which is equivalent to verifying that the string has the form $a^i b^i$. Allowing $\mathbf{h}_t = \tanh(\mathbf{c}_t)$, we set the linear threshold layer to check

$$[\mathbf{h}_t]_1 + [\mathbf{h}_t]_2 + [\mathbf{h}_t]_3 \leq 0. \tag{74}$$

□

Theorem 14.

$$a^n b^n \Sigma^* \in D_1(\text{s-LSTM}).$$

Proof. We use the same construction as Theorem 13, augmenting it with

$$[\mathbf{c}_t]_4 \triangleq [\mathbf{h}_{t-1}]_1 + [\mathbf{h}_{t-1}]_2 + [\mathbf{h}_{t-1}]_3 \leq 0. \quad (75)$$

We decide x according to the (still linearly separable) equation

$$\big(0 < [\mathbf{h}_t]_4\big) \vee \big([\mathbf{h}_t]_1 + [\mathbf{h}_t]_2 + [\mathbf{h}_t]_3 \leq 0\big). \quad (76)$$

□

E Experimental Details

Models were trained on strings up to length 64, and, at each index t, were asked to classify whether or not the prefix up to t was a valid string in the language. Models were then tested on independent datasets of lengths $64, 128, 256, 512, 1024$, and 2048. The training dataset contained 100000 strings, and the validation and test datasets contained 10000. We discuss task-specific schemes for sampling strings in the next paragraph. All models were trained for a maximum of 100 epochs, with early stopping after 10 epochs based on the validation cross entropy loss. We used default hyperparameters provided by the open-source AllenNLP framework (Gardner et al., 2018). The code is available at `https://github.com/viking-sudo-rm/rr-experiments`.

Sampling strings For the language L_5, each token was sampled uniformly at random from $\Sigma = \{a, b\}$. For $a^n b^n \Sigma^*$, half the strings were sampled in this way, and for the other half, we sampled n uniformly between 0 and 32, fixing the first $2n$ characters of the string to $a^n b^n$ and sampling the suffix uniformly at random.

Experimental cost Experiments were run for 20 GPU hours on Quadro RTX 8000.

F Self Attention

Architecture We place saturated self attention (Vaswani et al., 2017) into the state expressiveness hierarchy. We consider a single-head self attention encoder that is computed as follows:

1. At time t, compute queries $\mathbf{q}_t$, keys $\mathbf{k}_t$, and values $\mathbf{v}_t$ from the input embedding $\mathbf{x}_t$ using a linear transformation.
2. Compute attention head $\mathbf{h}_t$ by attending over the keys and values up to time t ($\mathbf{K}_{:t}$ and $\mathbf{V}_{:t}$) with query $\mathbf{q}_t$.
3. Let $\|\cdot\|_L$ denote a layer normalization operation (Ba et al., 2016).

$$\mathbf{h}'_t = \mathrm{ReLU}\left(\mathbf{W}^h \cdot \|\mathbf{h}_t\|_L\right) \quad (77)$$

$$\mathbf{c}_t = \left\|\mathbf{W}^c \mathbf{h}'_t\right\|_L. \quad (78)$$

This simplified architecture has only one attention head, and does not incorporate residual connections. It is also masked (i.e., at time t, can only see the prefix $\mathbf{X}_{:t}$), which enables direct comparison with unidirectional RNNs. For simplicity, we do not add positional information to the input embeddings.

Theorem 15. *Saturated masked self attention is not RR.*

Proof. Let $\#_\sigma(x)$ denote the number of occurences of $\sigma \in \Sigma$ in string x. We construct a self attention layer to compute the following function over $\{a, b\}^*$:

$$f(x) = \begin{cases} 0 & \text{if } \#_a(x) = \#_b(x) \\ 1 & \text{otherwise.} \end{cases} \quad (79)$$

Since the Hankel sub-block over $P = a^*, S = b^*$ has infinite rank, $f \notin \mathcal{R}$.

Fix $\mathbf{v}_t = \mathbf{x}_t$. As shown by Merrill (2019), saturated attention over a prefix of input vectors $\mathbf{X}_{:t}$ reduces to sum of the subsequence for which key-query similarity is maximized, i.e., denoting $I = \{i \in [t] \mid \mathbf{k}_i \cdot \mathbf{q}_t = m\}$ where $m = \max\{\mathbf{k}_i \cdot \mathbf{q}_t | i \in [t]\}$:

$$\mathbf{h}_t = \frac{1}{|I|} \sum_{i \in I} \mathbf{x}_{t_i}. \quad (80)$$

For all t, set the key and query $k_t, q_t = 1$. Thus, all the key-query similarities are 1, and we obtain:

$$\mathbf{h}_t = \frac{1}{t} \sum_{t'=1}^{t} \mathbf{x}_{t'} \quad (81)$$

$$= \frac{1}{t} \left(\#_a(x),\ \#_b(x)\right)^\top. \quad (82)$$

Applying layer norm to this quantity preserves equality of the first and second elements. Thus, we set the layer in (77) to independently check $0 < [\mathbf{h}^0_t]_1 - [\mathbf{h}^0_t]_2$ and $[\mathbf{h}^0_t]_1 - [\mathbf{h}^0_t]_2 < 0$ using ReLU. The final layer c_t sums these two quantities, returning 0 if neither condition is met, and 1 otherwise.

Since saturated self attention can represent $f \notin \mathcal{R}$, it is not RR. □

Space Complexity We show that self attention falls into the same space complexity class as the LSTM and QRNN. Our method here extends Merrill (2019)'s analysis of attention.

Theorem 16. *Saturated single-layer self attention has* $\Theta(\log n)$ *space.*

Proof. The construction from Theorem 15 can reach a linear (in sequence length) number of different outputs, implying a linear number of different configurations, and so that the space complexity of saturated self attention is $\Omega(\log n)$. We now show the upper bound $O(\log n)$.

A sufficient representation for the internal state (configuration) of a self-attention layer is the unordered group of key-value pairs over the prefixes of the input sequence.

Since $f_k : x_t \mapsto \mathbf{k}_t$ and $f_v : x_t \mapsto \mathbf{v}_t$ have finite domain (Σ), their images $K = \text{image}(f_k), V = \text{image}(f_v)$ are finite.[14] Thus, there is also a finite number of possible key-value pairs $\langle \mathbf{k}_t, \mathbf{v}_t \rangle \in K \times V$. Recall that the internal configuration can be specified by the number of occurrences of each possible key-value pair. Taking n as an upper bound for each of these counts, we bound the number of configurations of the layer as $n^{|K \times V|}$. Therefore the bit complexity is

$$\log_2 \left(n^{|K \times V|}\right) = O(\log n). \tag{83}$$

□

Note that this construction does not apply if the "vocabulary" we are attending over is not finite. Thus, using unbounded positional embeddings, stacking multiple self attention layers, or applying attention over other encodings with unbounded state might reach $\Theta(n)$.

While it eludes our current focus, we hope future work will extend the saturated analysis to self attention more completely. We direct the reader to Hahn (2020) for some additional related work.

G Memory Networks

All of the standard RNN architectures considered in Section 3 have $O(\log n)$ space in their saturated form. In this section, we consider a stack RNN encoder similar to the one proposed by Suzgun et al. (2019b) and show how it, like a WFA, can encode binary representations from strings. Thus, the stack RNN has $\Theta(n)$ space. Additionally, we find that it is not RR. This places it in the upper-right box of Figure 1.

Classically, a stack is a dynamic list of objects to which elements $v \in V$ can be added and removed in a LIFO manner (using *push* and *pop* operations). The stack RNN proposed in Suzgun et al. (2019b) maintains a differentiable variant of such a stack, as follows:

Differentiable Stack In a differentiable stack, the update operation takes an element s_t to push and a distribution π_t over the update operations push, pop, and no-op, and returns the weighted average of the result of applying each to the current stack. The averaging is done elementwise along the stacks, beginning from the top entry. To facilitate this, differentiable stacks are padded with infinite 'null entries'. Their elements must also have a weighted average operation defined.

Definition 6 (Geometric k-stack RNN encoder)**.** Initialize the stack $\mathbf{S}$ to an infinite list of null entries, and denote by S_t the stack value at time t. Using 1-indexing for the stack and denoting $[S_{t-1}]_0 \triangleq \mathbf{s}_t$, the geometric k-stack RNN recurrent update is:[15]

$$\begin{aligned} \mathbf{s}_t &= \mathbf{f}_s(x_t, \mathbf{c}_{t-1}) \\ \pi_t &= \mathbf{f}_\pi(x_t, \mathbf{c}_{t-1}) \\ \forall i \geq 1 \quad [\mathbf{S}_t]_i &= \sum_{a=1}^{3} [\pi_t]_a [\mathbf{S}_{t-1}]_{i+a-2}. \end{aligned}$$

In this work we will consider the case where the null entries are $\mathbf{0}$ and the encoding $\mathbf{c}_t$ is produced as a geometric-weighted sum of the stack contents,

$$\mathbf{c}_t = \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^{i-1} [\mathbf{S}_t]_i.$$

This encoding gives preference to the latest values in the stack, giving initial stack encoding $\mathbf{c}_0 = \mathbf{0}$.

Space Complexity The memory introduced by the stack data structure pushes the encoder into $\Theta(n)$ space. We formalize this by showing that, like a WFA, the stack RNN can encode binary strings to their value.

Lemma 5. *The saturated stack RNN can compute the converging binary encoding function, i.e.,* $101 \mapsto 1 \cdot 1 + 0.5 \cdot 0 + 0.25 \cdot 1 = 1.25$.

[14] Note that any periodic positional encoding will also have finite image.

[15] Intuitively, $[\pi_t]_a$ corresponds to the operations push, no-op, and pop, for the values $a = 1, 2, 3$ respectively.

Proof. Choose $k = 1$. Fix the controller to always push x_t. Then, the encoding at time t will be

$$\mathbf{c}_t = \sum_{i=1}^{t} \left(\frac{1}{2}\right)^{i-1} x_i. \tag{84}$$

This is the value of the prefix $x_{:t}$ in binary. □

Rational Recurrence We provide another construction to show that the stack RNN can compute non-rational series. Thus, it is not RR.

Definition 7 (Geometric counting)**.** Define $f_2 : \{a, b\}^* \to \mathbb{N}$ such that

$$f_2(x) = \exp_{\frac{1}{2}} \left(\#_{a-b}(x)\right) - 1.$$

Like similar functions we analyzed in Section 3, the Hankel matrix H_{f_2} has infinite rank over the sub-block $a^i b^j$.

Lemma 6. *The saturated stack RNN can compute* f_2.

Proof. Choose $k = 1$. Fix the controller to push 1 for $x_t = a$, and pop otherwise. □