On the Expressivity of Recurrent Neural Cascades

Nadezda Alexandrovna Knorozova[1], **Alessandro Ronca**[2]

[1]RelationalAI
[2]University of Oxford
nadezda.knorozova@relational.ai, alessandro.ronca@cs.ox.ac.uk

Abstract

Recurrent Neural Cascades (RNCs) are the recurrent neural networks with no cyclic dependencies among recurrent neurons. This class of recurrent networks has received a lot of attention in practice. Besides training methods for a fixed architecture such as backpropagation, the cascade architecture naturally allows for constructive learning methods, where recurrent nodes are added incrementally one at a time, often yielding smaller networks. Furthermore, acyclicity amounts to a structural prior that even for the same number of neurons yields a more favourable sample complexity compared to a fully-connected architecture.

A central question is whether the advantages of the cascade architecture come at the cost of a reduced expressivity. We provide new insights into this question. We show that the regular languages captured by RNCs with sign and tanh activation with positive recurrent weights are the *star-free* regular languages. In order to establish our results we develop a novel framework where capabilities of RNCs are assessed by analysing which semigroups and groups a single neuron is able to implement. A notable implication of our framework is that RNCs can achieve the expressivity of all regular languages by introducing neurons that can implement groups.

Introduction

Recurrent Neural Cascades (RNCs) are a class of recurrent networks that has been successfully applied in many different areas, including information diffusion in social networks (Wang et al. 2017), geological hazard predictions (Zhu et al. 2020), automated image annotation (Shin et al. 2016), brain-computer inference (Zhang et al. 2018), and optics (Xu et al. 2020). In the cascade architecture neurons can be layed out into a sequence so that every neuron has access to the output of all preceding neurons as well as to the external input; and at the same time, it has no dependency on the subsequent neurons. Compared to fully-connected networks, the cascade architecture has half of the connections. It immediately implies that RNCs have a more favourable *sample complexity*, or dually better generalisation capabilities. This is evident from the fact that the VC dimension of recurrent networks depends directly on the number of connections (Koiran and Sontag 1998).

The acyclic structure of the cascade architecture naturally allows for so-called *constructive learning* methods, cf. (Fahlman 1990; Reed and Marks II 1999). These methods construct the network architecture dynamically during the training, often yielding smaller networks, faster training and improved generalisation. One such method is *recurrent cascade correlation*, which builds the architecture incrementally adding one recurrent neuron at a time (Fahlman 1990). RNCs emerge naturally here from the fact that existing nodes will not depend on nodes added later. RNCs also admit learning methods for fixed architectures, such as *backpropagation through time*, cf. (Werbos 1990), where only the weights are learned. For these methods the advantage of the cascade architecture comes from the reduced number of weights.

A central question is whether the advantages of the cascade architecture come at the cost of a reduced *expressivity* compared to the fully-connected architecture. The studies so far have shown that there exist regular languages that are not captured by RNCs with monotone activation such as tanh (Giles et al. 1995). However, an exact characterisation of their expressitivity is still missing. Furthermore, it is unclear whether the inability to capture all regular languages is a limitation of the cascade architecture, or rather of the considered activation functions. We continue this investigation and provide new insights in the capabilities of RNCs to capture regular languages.

Our contribution. We develop an analysis of the capabilities of RNCs establishing the following expressivity results.

- RNCs with sign or tanh activations capture the star-free regular languages. The expressivity result already holds when recurrent weights are restricted to be positive.
- RNCs with sign or tanh activations and positive recurrent weights do not capture any regular language that is not star-free.
- Allowing for negative recurrent weights properly extends the expressivity of RNCs with sign and tanh activations beyond the star-free regular languages.
- We show that in principle the expressivity of RNCs can be extended to all regular languages. It suffices to identify appropriate recurrent neurons. In particular, neurons that can implement finite simple groups. As a first step,

we show that second-order sign and tanh neurons can implement the cyclic group of order two.

Our expressivity results establish an important connection between recurrent neural networks and the wide range of formalisms whose expressivity is the star-free regular languages. Such formalisms include *star-free regular expressions* from where they take their name, cf. (Ginzburg 1968), *Monadic First-order Logic* on finite linearly-ordered domains, cf. (McNaughton and Papert 1971), *Past Temporal Logic*, cf. (Manna and Pnueli 1991), and *Linear Temporal Logic* on finite traces (De Giacomo and Vardi 2013). They are also the languages recognised by *counter-free automata* as well as *group-free automata*, cf. (Schützenberger 1965; Ginzburg 1968; McNaughton and Papert 1971). On one hand, our results introduce an opportunity of employing RNCs for learning targets that one would describe in any of the above formalisms. For such targets, RNCs are sufficiently expressive and, compared to fully-connected recurrent neural networks, offer a more favorable sample complexity along with a wider range of learning algorithms. On the other hand, it places RNCs alongside well-understood formalisms with the possibility of establishing further connections and leveraging many existing fundamental results.

As a result of our investigation we develop a novel framework where recurrent neural networks are analysed through the lens of Semigroup and Group Theory. We establish a formal correspondence between continuous systems such as recurrent neural networks and discrete abstract objects such as semigroups and groups. Effectively we bridge RNCs with Algebraic Automata Theory, two fields that developed independently, and so far have not been considered to have any interaction. Specifically, our framework allows for establishing the expressivity of RNCs by analysing the capabilities of a single neuron from the point of view of which semigroups and groups it can implement. If a neuron can implement the so-called *flip-flop monoid*, then cascades of such neurons capture the star-free regular languages. To go beyond that, it is sufficient to introduce neurons that implement *groups*. Our framework can be readily used to analyse the expressivity of RNCs with neurons that have not been considered in this work. In particular, we introduce abstract flip-flop and group neurons, which are the neural counterpart of the flip-flop monoid and of any given group. To show expressivity results, it is sufficient to instantiate our abstract neurons. Specifically in this work we show how to instantiate flip-flop neurons with (first-order) sign and tanh, as well as a family of grouplike neurons with second-order sign and tanh. In a similar way, other results can be obtained by instantiating the abstract neurons with different activation functions.

The extended version of this paper provides proofs of all our results, a more extensive background on the required notions from semigroup and group theory, and examples of star-free regular languages as found in two different applications (Knorozova and Ronca 2023).

Part I: Background

We introduce the necessary background.

Dynamical Systems

Dynamical systems provide us with a formalism where to cast both neural networks and automata. The kind of dynamical systems relevant to us are described next. They are discrete-time, and they have some continuity properties. Specifically, a *dynamical system* S is a tuple

$$S = \langle U, X, f, x^{\text{init}}, Y, h \rangle,$$

where U is a set of elements called *inputs*, X is a set of elements called *states*, $f : X \times U \to X$ is called *dynamics function*, $x^{\text{init}} \in X$ is called *initial state*, Y is a set of elements called *outputs*, and $h : X \times U \to Y$ is called *output function*; furthermore, sets U, X, Y are metric spaces, and functions f, h are continuous. At every time point $t = 1, 2, \ldots$, the system receives an input $u_t \in U$. The state x_t of the system at time t is defined as follows. At time $t = 0$, before receiving any input, the system is in state $x_0 = x^{\text{init}}$. Then, the state x_t and output y_t are determined by the previous state x_{t-1} and the current input u_t as

$$x_t = f(x_{t-1}, u_t), \qquad y_t = h(x_{t-1}, u_t).$$

The *dynamics* of S are the tuple $D = \langle U, X, f \rangle$. *Subdynamics* of D are any tuple $\langle U', X', f \rangle$ such that $U' \subseteq U$, $X' \subseteq X$, and $f(X', U') \subseteq X'$. Note that $f(X', U') = \{f(x, u) \mid x \in X', u \in U'\}$. The function *implemented* by system S is the function that maps every input sequence $u_1, \ldots, u_\ell$ to the output sequence $y_1, \ldots, y_\ell$. We write $S(u_1, \ldots, u_\ell) = y_1, \ldots, y_\ell$. Two systems are *equivalent* if they implement the same function.

Architectures. A *network* is a dynamical system N with a factored state space $X = X_1 \times \cdots \times X_d$ and dynamics function of the form

$$f(\mathbf{x}, u) = \langle f_1(\mathbf{x}, u), \ldots, f_d(\mathbf{x}, u) \rangle,$$
$$\text{where } \mathbf{x} = \langle x_1, \ldots, x_d \rangle.$$

Note that f_i determines the i-th component of the state by reading the entire state vector and the input. A network can be expressed in a modular way by expressing the dynamics function as

$$f(\mathbf{x}, u) = \langle f_1(x_1, u_1), \ldots, f_d(x_d, u_d) \rangle,$$
$$\text{where } u_i = \langle u, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d \rangle.$$

It is a *modular view* because now the dynamics of N can be seen as made of the d dynamics of the form

$$D_i = \big\langle (U \times X_1 \times \cdots \times X_{i-1} \times X_{i+1} \times \cdots \times X_d), X_i, f_i \big\rangle.$$

We call every D_i a *component* of the dynamics of N. When there are no cyclic dependencies among the components of N, the network can be expressed as a *cascade*, where the dynamics function is of the form

$$f(\mathbf{x}, u) = \langle f_1(x_1, u_1), \ldots, f_d(x_d, u_d) \rangle,$$
$$\text{where } u_i = \langle u, x_1, \ldots, x_{i-1} \rangle.$$

In a cascade, every component has access to the state of the preceding components, in addition to the external input. Differently, in a network, every component has access to the state of all components, in addition to the external input.

Recurrent Neural Cascades and Networks

A *core recurrent neuron* is a triple $N = \langle V, X, f \rangle$ where $V \subseteq \mathbb{R}$ is the input domain, $X \subseteq \mathbb{R}$ are the states, and function f is of the form

$$f(x, u) = \alpha((w \cdot x) \oplus v),$$

with $w \in \mathbb{R}$ called *weight*, $\oplus$ a binary operator over $\mathbb{R}$, and $\alpha : \mathbb{R} \to \mathbb{R}$ called *activation function*. A *recurrent neuron* is the composition of a core recurrent neuron N with an *input function* $\beta : U \subseteq \mathbb{R}^a \to V$ that can be implemented by a feedforward neural network. Namely, it is a triple $\langle U, X, f_\beta \rangle$ where $f_\beta(x, u) = f(x, \beta(u))$.

We often omit the term 'recurrent' as it is the only kind of neuron we consider explicitly. By default we will assume that the operator $\oplus$ is addition. We will also consider the case where $\oplus$ is product; in this case we refer to the neuron as a *second-order neuron*.

A neuron is a form of dynamics, so the notions introduced for dynamical systems apply. A *Recurrent Neural Cascade (RNC)* is a cascade whose components are recurrent neurons and whose output function is a feedforward neural network. A *Recurrent Neural Network (RNN)* is a network whose components are recurrent neurons and whose output function is a feedforward neural network.

Automata

Automata are dynamical systems with a finite input domain, a finite set of states, and a finite output domain. The terminology used for automata is different from the one used for dynamical systems.

The input and output domains are called *alphabets*, and their elements are called *letters*. Input and output sequences are seen as *strings*, where a string $\sigma_1 \dots \sigma_\ell$ is simply a concatenation of letters. The set of all strings over an alphabet Σ is written as Σ^*.

An automaton is a tuple $A = \langle \Sigma, Q, \delta, q^{\text{init}}, \Gamma, \theta \rangle$ where Σ is called *input alphabet* (rather than input domain), Q is the set of states, $\delta : Q \times \Sigma \to Q$ is called *transition function* (rather than dynamics function), $q^{\text{init}} \in Q$ is the initial state, Γ is called *output alphabet* (rather than output domain), and $\theta : Q \times \Sigma \to \Gamma$ is the output function. Again, the requirement is that Σ, Q, Γ are finite. The tuple $D = \langle \Sigma, Q, \delta \rangle$ is called a *semiautomaton*, rather than dynamics.

In order to analyse automata, it is convenient to introduce the notion of state transformation. A *state transformation* is a function $\tau : Q \to Q$ from states to states, and it is called: (i) a *permutation* if $\tau(Q) = Q$, (ii) a *reset* if $\tau(Q) = \{q\}$ for some $q \in Q$, (iii) an *identity* if $\tau(q) = q$ for every $q \in Q$. Note that an identity transformation is, in particular, a permutation transformation. Every input letter $\sigma \in \Sigma$ induces the state transformation $\delta_\sigma(q) = \delta(q, \sigma)$. Such state transformation δ_σ describes all state updates triggered by the input letter σ. The *set of state transformations* of semiautomaton D is $\{\delta_\sigma \mid \sigma \in \Sigma\}$. Any two letters that induce the same state transformations are equivalent for the semiautomaton, in the sense that they trigger the same state updates. Such equivalence can be made explicit by writing a semiautomaton as consisting of two components. The first component is an *input function* that translates input letters into letters of an *internal alphabet* Π, where each letter represents an equivalence class of inputs. The second component is a semiautomaton operating on the internal alphabet Π. This way, the internal letters induce distinct state transformations.

Definition 1. *Given a function $\phi : \Sigma \to \Pi$, and a semiautomaton $\langle \Pi, Q, \delta \rangle$ their* composition *is the semiautomaton $\langle \Sigma, Q, \delta_\phi \rangle$ where δ_ϕ is defined as $\delta_\phi(q, \sigma) = \delta(q, \phi(\sigma))$. We call ϕ the* input function *of the resulting semiautomaton, and we call Π its* internal alphabet.

We will often write semiautomata with an explicit input function—w.l.o.g. since we can always choose identity.

Fundamentals of Algebraic Automata Theory

Semiautomata can be represented as networks or cascades. This is a structured alternative to state diagrams—the unstructured representation of transitions as a labelled graph. For the cascade architecture, the fundamental theorem by Krohn and Rhodes shows that every semiautomaton can be expressed as a cascade of so-called *prime semiautomata* (Krohn and Rhodes 1965). Moreover, using only *some* prime semiautomata, one obtains specialised expressivity results. Prime semiautomata can be partitioned into two classes. The first class of prime semiautomata are *flip-flops*. At their core, they have a semiautomaton that corresponds to the standard notion of flip-flop from digital circuits, and hence they provide the fundamental functionality of storing one bit of information, with the possibility of setting and resetting.

Definition 2. *Let set, $reset$, $read$, and $high$, low be distinguished symbols. The core flip-flop semiautomaton is the semiautomaton $\langle \Pi, Q, \delta \rangle$ where the input alphabet is $\Pi = \{set, reset, read\}$, the states are $Q = \{high, low\}$, and the identities below hold:*

$$\delta(read, q) = q, \quad \delta(set, q) = high, \quad \delta(reset, q) = low.$$

A flip-flop semiautomaton is the composition of an input function with the core flip-flop semiautomaton.

Note that the state transformations of a flip-flop characterise its intuitive functionalities. In particular, $read$ induces an identity transformation, set induces a reset to $high$, and $reset$ induces a reset to low.

The second class of prime semiautomata are the *simple grouplike semiautomata*.

Definition 3. *Let $G = (D, \circ)$ be a finite group. The core G semiautomaton is the semiautomaton $\langle D, D, \delta \rangle$ where $\delta(g, h) = g \circ h$. A G semiautomaton is the composition of an input function with the core G semiautomaton. A semiautomaton is* (simple) grouplike *if it is a G semiautomaton for some (simple) group G.*

Of particular interest to us is the class of *group-free* semiautomata. Intuitively, they are the semiautomata that do not involve groups, and do not show any periodic behaviour. The formal definition requires additional notions from semigroup theory, and hence we defer it to the appendix.

Next we state a direct implication of the *Krohn-Rhodes decomposition theorem*—see Theorem 3.1 of (Dömösi and

Nehaniv 2005). The statement requires the notion of *homomorphic representation* which is given later in Definition 4, in a more general form that applies to arbitrary dynamical systems. Intuitively, if a semiautomaton A is homomorphically represented by a semiautomaton B, it means that the capabilities of A are captured by the capabilities of B.

Theorem 1 (Krohn-Rhodes)**.** *Every semiautomaton is homomorphically represented by a cascade of prime semiautomata. Every group-free semiautomaton is homomorphically represented by a cascade of flip-flop semiautomata.*

The converse of both statements in Theorem 1 holds as well. Thus, group-free semiautomata can be characterised as the semiautomata that are homomorphically represented by a cascade of flip-flop semiautomata. If one allows for cyclic dependencies, then flip-flop semiautomata suffice to capture all semiautomata. This a direct implication of the *Letichevsky decomposition theorem*—see Theorem 2.69 of (Dömösi and Nehaniv 2005).

Theorem 2 (Letichevsky)**.** *Every semiautomaton is homomorphically represented by a network of flip-flop semiautomata.*

Classes of Languages and Functions

A *language* L over Σ is a subset of Σ^*. It can also be seen the indicator function $f_L : \Sigma^* \to \{0,1\}$ where $f_L(x) = 1$ iff $x \in L$. In general we will be interested in functions $f : \Sigma^* \to \Gamma$ for Γ an arbitrary output alphabet. An *acceptor* is a dynamical system whose output domain is $\{0,1\}$. The language *recognised* by an acceptor is the set of strings on which the acceptor returns 1 as its last output.

The *regular languages* are the ones recognised by automaton acceptors (Kleene 1956). The *group-free regular languages* are the ones recognised by automaton acceptors with a group-free semiautomaton, and they coincide with the *star-free regular languages*, cf. (Ginzburg 1968). These notions can be naturally generalised to functions. The *regular functions* are the ones implemented by automata. The *group-free regular functions* are the ones implemented by automata with a group-free semiautomaton.

Part II: Our Framework

In this section we present our framework for analysing RNCs. First, we introduce a notion of homomorphism for dynamical systems. Then, we formalise the notion of symbol grounding. Finally, we introduce abstract neurons which are the neural counterpart of prime semiautomata.

Homomorphisms for Dynamical Systems

We introduce a new notion of homomorphism which allows us to compare systems by comparing their dynamics. Homomorphisms are a standard notion in automata theory, cf. (Arbib 1969). However, there, they do not deal with the notion of *continuity*, which holds trivially for all functions involved in automata, since they are over finite domains. Here we introduce homomorphisms for dynamical systems, with the requirement that they must be continuous functions. This allows one to infer results for continuous dynamical systems such as recurrent neural networks, as stated by our Propositions 1 and 2, which are instrumental to our results.

Definition 4. *Consider two system dynamics* $D_1 = \langle U, X_1, f_1\rangle$ *and* $D_2 = \langle U, X_2, f_2\rangle$*. A* homomorphism *from* D_1 *to* D_2 *is a continuous surjective function* $\psi : X_1 \to X_2$ *satisfying the equality*

$$\psi\big(f_1(x,u)\big) = f_2\big(\psi(x),u\big)$$

for every state $x \in X_1$ *and every input* $u \in U$*. We say that* D_1 homomorphically represents D_2 *if* D_1 *has subdynamics* D_1' *such that there is a homomorphism from* D_1' *to* D_2*.*

The relevance of the notion of homomorphic representation is made clear by the two following propositions.

Proposition 1. *If dynamics* D_1 *homomorphically represent dynamics* D_2*, then every system with dynamics* D_2 *admits an equivalent system with dynamics* D_1*.*

For the second proposition, the following notions are needed, which are borrowed from automata theory, cf. (Arbib 1969), but apply to dynamical systems as well.

Definition 5. *A state* x *of a system* S *is* reachable *if there is an input sequence* $u_1, \ldots, u_\ell$ *such that the system is in state* x *at time* ℓ*. A system is* connected *if every state is reachable. Given a system* S *and one of its states* x*, the system* S^x *is the system obtained by setting* x *to be the initial state. Two states* x *and* x' *of* S *are equivalent if the systems* S^x *and* $S^{x'}$ *are equivalent. A system is in* reduced form *if it has no distinct states which are equivalent. A system is* canonical *if it is connected and in reduced form.*

Proposition 2. *If a system* S_1 *is equivalent to a canonical system* S_2 *with a discrete output domain, then the dynamics of* S_1 *homomorphically represent the dynamics of* S_2*.*

Symbol Grounding

Our goal is to establish expressivity results for recurrent networks. Given an input alphabet Σ and an output alphabet Γ, we want to establish which functions from Σ^* to Γ^* can be implemented by a recurrent network. However, recurrent networks operate on a real-valued input domain $U \subseteq \mathbb{R}^n$ and a real-valued output domain $Y \subseteq \mathbb{R}^m$. In order to close the gap, we introduce the notion of symbol grounding.

Definition 6. *Given a domain* $Z \subseteq \mathbb{R}^n$ *and an alphabet* Λ*, a* symbol grounding *from* Z *to* Λ *is a continuous surjective function* $\lambda : Z \to \Lambda$*.*

Symbol groundings can be seen as connecting the subymbolic level $Z \subseteq \mathbb{R}^n$ to the symbolic level Λ. For an element z at the subsymbolic level, the letter $\lambda(z)$ is its meaning at the symbolic level. Assuming that a symbol grounding λ is surjective means that every letter corresponds to at least one element $z \in Z$. The assumption is w.l.o.g. because we can remove the letters that do not represent any element of the subsymbolic level.

Symbol groundings can be robust to noise, when every letter corresponds to a ball in $\mathbb{R}^n$ rather than to a single point, so as to allow some tolerance—any noise that stays within the ball does not affect the symbolic level.

Definition 7. *A symbol grounding $\lambda : Z \to \Lambda$ is* robust *if, for every $a \in \Lambda$, there exists a ball $B \subseteq Z$ of non-zero radius such that $\lambda(B) = \{a\}$.*

In the following sections we establish expressivity results considering fixed, but arbitrary, input alphabet Σ, input domain $U \subseteq \mathbb{R}^n$, input symbol grounding $\lambda_\Sigma : U \to \Sigma$, output alphabet Γ, output domain $Y \subseteq \mathbb{R}^m$, and output symbol grounding $\lambda_\Gamma : Y \to \Gamma$. Whenever we relate an RNC (or RNN) to an automaton, we mean that the RNC (or RNN) operates at the subsymbolic level and then its output is mapped to the symbolic level, while the automaton operates entirely at the symbolic level. Formally, given an RNC or RNN $\langle U, X, f, x^{\text{init}}, Y, g\rangle$ and an automaton $\langle \Sigma, Q, \delta, q^{\text{init}}, \Gamma, \theta\rangle$, we relate the corresponding dynamical sytems $\langle U, X, f, x^{\text{init}}, \Gamma, g \circ \lambda_\Gamma\rangle$ and $\langle U, Q, \delta_{\lambda_\Sigma}, q^{\text{init}}, \Gamma, \theta\rangle$ where $\delta_{\lambda_\Sigma}(q, u) = \delta(q, \lambda_\Sigma(u))$. Note that both systems take inputs in U and return letters in Γ.

Assumptions. We make the mild technical assumptions that U is a *compact*, and that the output symbol grounding λ_Γ is *robust*. These assumptions, together with continuity, allow us to make use of the Universal Approximation Theorem for feedforward neural networks, cf. (Hornik 1991).

Abstract Neurons

We first introduce an abstract class of neurons that model the behaviour of a flip-flop or grouplike semiautomaton. This allows us to state general results about cascades and networks of such abstract neurons. Then, we show that these results will transfer to cascades and networks of any concrete instantiation of such neurons.

Definition 8. *A* core flip-flop neuron *is a core neuron $\langle V, X, f\rangle$ where the set V of inputs is expressed as the union of three disjoint closed intervals $V_{\text{set}}, V_{\text{reset}}, V_{\text{read}}$ of non-zero length, the set X of states is expressed as the union of two disjoint closed intervals $X_{\text{low}}, X_{\text{high}}$, and the following conditions hold:*

$$f(X, V_{\text{set}}) \subseteq X_{\text{high}},$$
$$f(X, V_{\text{reset}}) \subseteq X_{\text{low}},$$
$$f(X_{\text{high}}, V_{\text{read}}) \subseteq X_{\text{high}},$$
$$f(X_{\text{low}}, V_{\text{read}}) \subseteq X_{\text{low}}.$$

A flip-flop neuron *is the composition of a core flip-flop neuron with an input function. The* state interpretation *of a flip-flop neuron is the function ψ defined as $\psi(x) = high$ for $x \in X_{\text{high}}$ and $\psi(x) = low$ for $x \in X_{\text{low}}$.*

Definition 9. *Let $G = (D, \circ)$ be a group with $D = \{1, \dots, n\}$. A* core G neuron *is a core neuron $\langle V, X, f\rangle$ where the set V of inputs is expressed as the union of n disjoint closed intervals $V_1, \dots, V_n$ of non-zero length, the set X of states is expressed as the union of n disjoint closed intervals $X_1, \dots, X_n$, and the following condition holds for every $i, j \in D$:*

$$f(X_i, V_j) \subseteq X_{i \circ j}.$$

A G neuron *is the composition of a core G neuron with an input function. The* state interpretation *of a G neuron is the function ψ defined as $\psi(x) = i$ for $x \in X_i$. A neuron is* (simple) grouplike *if it is a G neuron for some (simple) group G.*

Abstract neurons are designed to be the neural counterpart of flip-flop and grouplike semiautomata. Specifically, they are designed to guarantee the existence of a homomorphism as stated in Lemma 1 below. The lemma and all the following results involving abstract neurons hold regardless of the specific way the core of an abstract neuron is instantiated. We highlight this aspect in the claims by referring to an abstract neuron *with arbitrary core*.

Lemma 1. *Every flip-flop semiautomaton is homomorphically represented by a flip-flop neuron with arbitrary core. Similarly, every G semiautomaton is homomorphically represented by a G neuron with arbitrary core. In either case, the homomorphism is given by the state interpretation of the neuron.*

The lemma is based on three key observations. First, the inclusion requirements in the definition of a flip-flop neuron determine a correspondence with transitions of a flip-flop semiautomaton; the same holds for grouplike neurons. Second, the fact that input intervals have non-zero length introduces sufficient tolerance to approximate the input function of a semiautomaton by a feedforward neural network making use of Universal Approximation Theorems, cf. (Hornik 1991). Third, the fact that state partitions are closed intervals ensures continuity of a homomorphism.

The previous lemma extends to cascades and networks.

Lemma 2. *Every cascade (or network) of flip-flop or grouplike semiautomata $A_1, \dots, A_d$ is homomorphically represented by a cascade (network, resp.) of d neurons $N_1, \dots, N_d$ where N_i is a flip-flop neuron if A_i is a flip-flop semiautomaton and N_i is a G neuron if A_i is a G semiautomaton.*

Part III: Expressivity Results

We present our results for RNCs of sign and tanh activation.

Implementation of Flip-Flop Neurons

We give precise conditions under which neurons with sign or tanh activation are flip-flop neurons. Following the definition of the abstract flip-flop neuron the goal is to partition the state space of both sign and tanh into low and high states and then find inputs inducing read, set and reset transitions. For sign activation the choice is simple, we interpret -1 as the low state and $+1$ as the high state. Since states are bounded, we know the maximum and minimum value that can be achieved by $w \cdot x$ for any possible state x. Therefore we can find inputs that will either maintain the sign or make it the desired one.

Proposition 3. *Let $w > 0$. A core neuron with sign activation and weight w is a core flip-flop neuron if its state partition is*

$$X_{\text{low}} = \{-1\},\ X_{\text{high}} = \{+1\},$$

for some real number $a \in (0, 1)$ and its inputs partition satisfies

$$V_{\text{reset}} \in \big(-\infty,\ w \cdot (-a - 1)\big],$$
$$V_{\text{read}} \in \big[w \cdot (a - 1),\ w \cdot (1 - a)\big],$$
$$V_{\text{set}} \in \big[w \cdot (a + 1),\ +\infty\big).$$

Tanh activation requires a more careful treatment. We represent the low and the high states as closed disjoint intervals including -1 and $+1$ respectively. Then using the values of state boundaries and the monotonicity property of tanh we can find inputs allowing for read, set and reset transitions without violating the state boundaries.

Proposition 4. *Let $w > 1$, and let $f(x) = \tanh(w \cdot x)$. A core neuron with tanh activation and weight w is a core flip-flop neuron if its state partition is*

$$X_{\text{low}} = [-1, f(a)], \qquad X_{\text{high}} = [f(b), +1],$$

for some real numbers $a < b$ satisfying $a - f(a) > b - f(b)$, and its input partition satisfies

$$\begin{aligned} V_{\text{reset}} &\in \big(-\infty,\, w \cdot (a-1)\big], \\ V_{\text{read}} &\in \big[w \cdot (b - f(b)),\, w \cdot (a - f(a))\big], \\ V_{\text{set}} &\in \big[w \cdot (b+1),\, +\infty\big). \end{aligned}$$

Differently from sign activation, the low and high states of tanh are not partitioned based on their sign. In fact, the low states can include positive values and high states can include negative values. This is determined entirely by the values of a and b defining the state boundaries. We remark that the range of valid a, b values increases with the increasing value of w. The quantities a, b also determine the length of the V_{read} interval, that impacts the robustness or the noise tolerance of the neuron. It is possible to choose the values of a and b that maximise the length of the V_{read} interval. In particular, these are the points where the derivative of $f(x)$ is equal to one.

Expressivity of RNCs

We are now ready to present our expressivity results. We state them in terms of the functions that can be implemented by an RNC. The results apply to languages as well, since they correpond to indicator functions as discussed in the background section.

As a positive expressivity result, we show that RNCs of flip-flop neurons can implement all group-free regular functions.

Theorem 3. *Every group-free regular function can be implemented by an RNC of flip-flop neurons with arbitrary core. In particular, it can be implemented by an RNC of neurons with sign or tanh activation, where it is sufficient to consider positive weights.*

The result is obtained by applying results from the previous sections. We have that every group-free regular function F is implemented by a group-free automaton, whose semiautomaton is homomorphically represented by a cascade of flip-flop semiautomata (Theorem 1), which is in turn homomorphically represented by a cascade of flip-flop neurons (Lemma 2); therefore, F is implemented by a system whose dynamics are a cascade of flip-flop neurons (Proposition 1) and whose output function is some continuous output function; we replace the output function with a feedforward neural network making use of the Universal Approximation Theorem, relying on the fact that approximation will not affect the result because the output symbol grounding is assumed to be robust.

In the rest of this section we show that RNCs of sign or tanh neurons with positive weight do not implement regular functions that are not group-free. We start by establishing a necessary condition. In order to go beyond group-free regular functions, it is necessary for the dynamics to show a periodic, alternating behaviour.

Lemma 3. *If a semiautomaton that is not group-free is homomorphically represented by dynamics $\langle U, X, f\rangle$, with homomorphism ψ, then there exist $u \in U$ and $x_0 \in X$ such that, for $x_i = f(x_{i-1}, u)$, the disequality $\psi(x_i) \neq \psi(x_{i+1})$ holds for every $i \geq 0$.*

Then we show that, for sign or tanh neurons with positive weight, a constant input yields a convergent sequence of states—in fact, more generally, such a sequence is convergent even when the input is not constant but itself convergent, and this stronger property is required in the proof of the lemma below. Together with the lemma above, it implies that a cascade of sign or tanh neurons with positive weight can capture a group-free semiautomaton only at the cost of generating a sequence of *converging alternating* states. This would amount to an essential discontinuity for any candidate homomorphism.

Lemma 4. *Every semiautomaton that is not group-free is not homomorphically represented by a cascade where each component is a neuron with sign or tanh activation and positive weight.*

Then the expressivity result follows from the lemma by Proposition 2.

Theorem 4. *For any regular function F that is not group-free, there is no RNC implementing F whose components are neurons with sign or tanh activation and positive weight.*

In light of Theorem 3 and Theorem 4, we identify a class of RNCs that can implement all group-free regular functions and no other regular function.

Theorem 5. *The class of regular functions that can be implemented by RNCs of sign or tanh neurons with positive weight is the group-free regular functions.*

Necessary Conditions for Group-freeness

We show that both acyclicity and positive weights of sign and tanh are necessary to stay within the group-free functions. First, recurrent neural networks, with arbitrary dependencies among their neurons, implement all regular functions, including the ones that are not group-free.

Theorem 6. *Every regular function can be implemented by an RNN of flip-flop neurons with arbitrary core. In particular, it can be implemented by an RNN of neurons with sign or tanh activation.*

The theorem is proved similarly to Theorem 3, using Theorem 2 in place of Theorem 1. The above Theorem 6 seems to be folklore. However we are not aware of an existing formal proof for the case of a differentiable activation function such as tanh. We discuss it further in the related work section.

Next we show that the restriction to positive weights is necessary to stay within the expressivity of group-free regular functions.

Theorem 7. *There is an RNC consisting of a single tanh (or sign) neuron with negative weight that implements a regular function that is not group-free regular.*

The proof amounts to showing that a tanh (or sign) neuron with negative weight captures a semiautomaton that is not group-free. It is a two-state semiautomaton with one non-identity permutation transformation. We conjecture that single sign or tanh neurons are not able to capture an actual grouplike semiautomaton.

Implementation of Group Neurons

We give an instantiation of a group neuron as per Definition 9. In particular, we show when second-order neurons with sign or tanh activation are instances of the C_2 neuron, the neuron implementing the cyclic group of order two.

Proposition 5. *Let w, a be real numbers either satisfying $a, w > 0$ or $a, w < 0$. A core second-order neuron with* sign *activation and weight w is a core C_2 neuron, if its states partition is*

$$X_0 = \{-1\},\ X_1 = \{+1\},$$

and its input partition satisfies

$$\begin{aligned} V_1 \in (-\infty, -a], V_0 \in [a, +\infty), &\qquad \text{if } a, w > 0,\\ V_0 \in (-\infty, a], V_1 \in [-a, +\infty), &\qquad \text{if } a, w < 0. \end{aligned}$$

Proposition 6. *Let w, a be real numbers either satisfying $a, w > 0$ or $a, w < 0$. Let $f(x) = \tanh(w \cdot x)$. A core second-order neuron with $\tanh$ activation and weight w is a core C_2 neuron, if its state partition is*

$$X_0 = [-1, -f(a)],\ \ X_1 = [f(a), +1]$$

and its input partition satisfies

$$\begin{aligned} V_1 \in (-\infty, -a/f(a)],\ V_0 \in [a/f(a), +\infty) &\ \ \text{if } a, w > 0,\\ V_0 \in (-\infty, a/f(a)],\ V_1 \in [-a/f(a), +\infty) &\ \ \text{if } a, w < 0. \end{aligned}$$

Then by Lemma 1 the above neurons homomorphically represent C_2 semiautomata. By Lemma 2 an RNC containing these neurons can homomorphically represent a cascade of C_2 semiautomata. In particular, such RNCs can recognise languages that are not star-free, cf. (Ginzburg 1968).

Related Work

In our work, the connection between RNNs and automata plays an important role. Interestingly, the connection appears to exist from the beginning of automata theory (Arbib 1969): "*In 1956 the series Automata Studies (Shannonon and McCarthy [1956]) was published, and automata theory emerged as a relatively autononmous discipline. [...] much interest centered on finite-state sequential machines, which first arose not in the abstract form [...], but in connection with the input-output behaviour of a McCulloch-Pitts net [...]*". The relationship between automata and the networks by (McCulloch and Pitts 1943) is discussed both in (Kleene 1956) and (Minsky 1967). Specifically, an arbitrary automaton can be captured by a McCulloch-Pitts network. Our Theorem 6 reinforces this result, extending it to sign and tanh activation. The extension to tanh is important because of its differentiability, and it requires a different set of techniques since it is not binary, but rather real-valued. Furthermore, our results extend theirs by showing a correspondence between RNCs and group-free automata.

The Turing-completeness of RNNs as an *offline model* of computation are studied in (Siegelmann and Sontag 1995; Kilian and Siegelmann 1996; Hobbs and Siegelmann 2015; Chung and Siegelmann 2021). In this setting, an RNN is allowed to first read the entire input sequence, and then return the output after an arbitrary number of iterations, triggered by blank inputs. This differs from our study, which focuses on the capabilities of RNNs as *online machines*, which process the input sequence one element at a time, outputting a value at every step. This is the way they are used in many practical applications such as Reinforcement Learning, cf. (Bakker 2001; Ha and Schmidhuber 2018; Hausknecht and Stone 2015; Kapturowski et al. 2019).

The expressivity of RNNs in terms of whether they capture all *rational series* or not has been analysed in (Merrill et al. 2020). This is a class of functions that includes all regular functions. Thus, it is a coarse-grained analysis compared to ours, which focuses on subclasses of the regular languages.

The problem of *latching* one bit of information has been studied in (Bengio, Simard, and Frasconi 1994) and (Frasconi et al. 1995). This problem is related to star-free regular languages, as it amounts to asking whether there is an automaton recognising a language of the form sr^* where s is a set command and r is a read command. This is a subset of the functionalities implemented by a flip-flop semiautomaton. Their work established conditions under which a tanh neuron can latch a bit. Here we establish conditions guaranteeing that a tanh neuron homomorphically represents a flip-flop semiautomaton, implying that it can latch a bit. An architecture that amounts to a restricted class of RNCs has been considered in (Frasconi, Gori, and Soda 1992).

Automata cascades are considered in (Ronca, Knorozova, and De Giacomo 2023), where they are shown to yield favourable sample complexity results for automata learning.

Conclusions and Future Work

We developed a new methodology that provides a fresh perspective on RNCs as systems implementing semigroups and groups. This enabled us to establish new expressivity results for RNCs with sign and tanh activations. We believe our methodology has a potential that extends beyond our current results. In particular, we believe it provides a principled way to identify new classes of recurrent networks that incorporate different priors based on groups.

We have covered sign and tanh activation, postponing the study of other activation functions such as logistic curve, ReLU, GeLU. Beyond that, one could identify neurons that can homomorphically represent grouplike semiautomata. This will allow to capture specific subclasses of regular functions that are beyond group-free. To showcase this direction we have presented second-order sign and tanh neurons, as instances of neurons homomorphically representing the cyclic group of order two.

Acknowledgments

Alessandro Ronca is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT).

References

Arbib, M. 1969. *Theories of Abstract Automata*. Automatic Computation. Prentice-Hall.

Bakker, B. 2001. Reinforcement Learning with Long Short-Term Memory. In *NeurIPS*.

Bengio, Y.; Simard, P. Y.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2).

Chung, S.; and Siegelmann, H. T. 2021. Turing Completeness of Bounded-Precision Recurrent Neural Networks. In *NeurIPS*.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*.

Dömösi, P.; and Nehaniv, C. L. 2005. *Algebraic theory of automata networks: An introduction*. SIAM.

Fahlman, S. E. 1990. The Recurrent Cascade-Correlation Architecture. In *NIPS*.

Frasconi, P.; Gori, M.; Maggini, M.; and Soda, G. 1995. Unified Integration of Explicit Knowledge and Learning by Example in Recurrent Networks. *IEEE Trans. Knowl. Data Eng.*, 7(2).

Frasconi, P.; Gori, M.; and Soda, G. 1992. Local Feedback Multilayered Networks. *Neural Comput.*, 4(1).

Giles, C.; Chen, D.; Sun, G.-Z.; Chen, H.-H.; Lee, Y.-C.; and Goudreau, M. 1995. Constructive learning of recurrent neural networks: Limitations of recurrent cascade correlation and a simple solution. *IEEE Transactions on Neural Networks*, 6(4).

Ginzburg, A. 1968. *Algebraic Theory of Automata*. Academic Press.

Ha, D.; and Schmidhuber, J. 2018. Recurrent World Models Facilitate Policy Evolution. In *NeurIPS*.

Hausknecht, M. J.; and Stone, P. 2015. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposia*.

Hobbs, J. N.; and Siegelmann, H. T. 2015. Implementation of universal computation via small recurrent finite precision neural networks. In *IJCNN*.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2).

Kapturowski, S.; Ostrovski, G.; Quan, J.; Munos, R.; and Dabney, W. 2019. Recurrent Experience Replay in Distributed Reinforcement Learning. In *ICLR*.

Kilian, J.; and Siegelmann, H. T. 1996. The Dynamic Universality of Sigmoidal Neural Networks. *Inf. Comput.*, 128(1).

Kleene, S. C. 1956. Representation of events in nerve nets and finite automata. *Automata studies*, 34.

Knorozova, N. A.; and Ronca, A. 2023. On The Expressivity of Recurrent Neural Cascades. *CoRR*, abs/2312.09048.

Koiran, P.; and Sontag, E. D. 1998. Vapnik-Chervonenkis Dimension of Recurrent Neural Networks. *Discret. Appl. Math.*, 86(1).

Krohn, K.; and Rhodes, J. 1965. Algebraic Theory of Machines. I. Prime Decomposition Theorem for Finite Semigroups and Machines. *Trans. Am. Math. Soc.*, 116.

Manna, Z.; and Pnueli, A. 1991. Completing the Temporal Picture. *Theor. Comput. Sci.*, 83(1).

McCulloch, W. S.; and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4).

McNaughton, R.; and Papert, S. A. 1971. *Counter-Free Automata*. The MIT Press.

Merrill, W.; Weiss, G.; Goldberg, Y.; Schwartz, R.; Smith, N. A.; and Yahav, E. 2020. A Formal Hierarchy of RNN Architectures. In *ACL*.

Minsky, M. L. 1967. *Computation: Finite and Infinite Machines*. Prentice-Hall.

Reed, R.; and Marks II, R. J. 1999. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press.

Ronca, A.; Knorozova, N. A.; and De Giacomo, G. 2023. Sample Complexity of Automata Cascades. In *AAAI*.

Schützenberger, M. P. 1965. On Finite Monoids Having Only Trivial Subgroups. *Inf. Control.*, 8(2).

Shin, H.-C.; Roberts, K.; Lu, L.; Demner-Fushman, D.; Yao, J.; and Summers, R. M. 2016. Learning to read chest x-rays: Recurrent neural cascade model for automated image annotation. In *IEEE/CVF CVPR*.

Siegelmann, H. T.; and Sontag, E. D. 1995. On the Computational Power of Neural Nets. *J. Comput. Syst. Sci.*, 50(1).

Wang, J.; Zheng, V. W.; Liu, Z.; and Chang, K. C.-C. 2017. Topological recurrent neural network for diffusion prediction. In *IEEE ICDM*.

Werbos, P. 1990. Backpropagation through time: What it does and how to do it. *Proc. of the IEEE*, 78(10).

Xu, Z.; Sun, C.; Ji, T.; Manton, J. H.; and Shieh, W. 2020. Cascade recurrent neural network-assisted nonlinear equalization for a 100 Gb/s PAM4 short-reach direct detection system. *Optics Letters*, 45(15).

Zhang, D.; Yao, L.; Zhang, X.; Wang, S.; Chen, W.; Boots, R.; and Benatallah, B. 2018. Cascade and Parallel Convolutional Recurrent Neural Networks on EEG-based Intention Recognition for Brain Computer Interface. In *AAAI*.

Zhu, L.; Huang, L.; Fan, L.; Huang, J.; Huang, F.; Chen, J.; Zhang, Z.; and Wang, Y. 2020. Landslide susceptibility prediction modeling based on remote sensing and a novel deep learning algorithm of a cascade-parallel recurrent neural network. *Sensors*, 20(6).