Positive Neural Networks in Discrete Time Implement Monotone-Regular Behaviors

Non Peer-reviewed author version

AMELOOT, Tom & VAN DEN BUSSCHE, Jan (2015) Positive Neural Networks in Discrete Time Implement Monotone-Regular Behaviors. In: NEURAL COMPUTATION, 27 (12), p. 2623-2660.

DOI: 10.1162/NECO_a_00789
Handle: http://hdl.handle.net/1942/20662

Positive Neural Networks in Discrete Time Implement Monotone-Regular Behaviors

Tom J. Ameloot* and Jan Van den Bussche
Hasselt University & transnational University of Limburg

Abstract. We study the expressive power of positive neural networks. The model uses positive connection weights and multiple input neurons. Different behaviors can be expressed by varying the connection weights. We show that in discrete time, and in absence of noise, the class of positive neural networks captures the so-called monotone-regular behaviors, that are based on regular languages. A finer picture emerges if one takes into account the delay by which a monotone-regular behavior is implemented. Each monotone-regular behavior can be implemented by a positive neural network with a delay of one time unit. Some monotone-regular behaviors can be implemented with zero delay. And, interestingly, some simple monotone-regular behaviors can not be implemented with zero delay.

1 Introduction

Positive neural networks Based on experimental observations, Douglas and Martin (2004) have proposed an abstract model of the neocortex consisting of interconnected winner-take-all circuits. Each winner-take-all circuit consists of excitatory neurons that, besides exciting each other, indirectly inhibit each other through some inhibition layer. This causes only a few neurons in the circuit to be active at any time. Kappel et al. (2014) further demonstrate through theoretical analysis and simulations that the model of interconnected winner-take-all circuits might indeed provide a deeper understanding of some experimental observations. In this article, we take two inspirational points from this model, that we discuss below.

First, although a biological neural network in general contains both excitatory and inhibitory connections between neurons (Gerstner et al., 2014), excitation and inhibition are not combined in an arbitrary fashion in the above model of interconnected winner-take-all circuits. In that model, the meaning seems to be mostly contained in the excitatory connections whereas inhibitory connections play a more regulatory role such as controlling how many neurons can become simultaneously active. Based on this apparent value of excitatory connections, in this article we are inspired to study neural networks that are simplified to contain

*T.J. Ameloot is a Postdoctoral Fellow of the Research Foundation – Flanders (FWO).

only excitatory connections between neurons. Technically, we consider so-called positive neural networks, where all connections are given a weight that is either strictly positive or zero.

Second, it appears useful to study neural network models with multiple input neurons. In the model of interconnected winner-take-all circuits, each circuit has multiple input neurons that can be concurrently active. This allows each circuit to receive rich input symbols. The input neurons of a circuit could receive stimuli directly from sensory organs, or from other circuits. It would be fascinating to understand how neurons build concepts or recognize patterns over such rich inputs.

Based on the above inspiration, in this article we study a simple positive neural network model with multiple input neurons, operating in discrete time. As mentioned above, the use of nonnegative weights allows only excitation between neurons and no inhibition. In our model, each positive neural network has distinguished sets of input neurons, output neurons, and auxiliary neurons. The network may be recurrent, i.e., the activation of a neuron may indirectly influence its own future activation. As in some previous models (Šíma and Wiedermann, 1998; Šíma and Orponen, 2003), we omit noise and learning. We believe that the omission of inhibition (i.e., negative connection weights) might allow for a better understanding of the foundations of computation in neural networks, where different features gradually increase the expressive power (see also Section 5). Excitation between neurons seems to be a basic feature that we can not omit. The omission of inhibition leads to a notion of monotonicity that we will discuss later in the Introduction.

As a final point for the motivation of the model, we mention that biological neurons seem to mostly encode information in the timing of their activations and not in the magnitude of the activation signals (Gerstner et al., 2014). In this perspective, one may view discrete time models like ours as highlighting the causal steps of the neuronal computation. The discrete time step could in principle be chosen very small.

Expressivity study Our aim in this article is to better understand what computations can or cannot be performed by positive neural networks. We show that positive neural networks represent the class of so-called *monotone-regular behaviors*. The relevance of this result is discussed later in the Introduction. We first provide the necessary context.

Many previous works have investigated the expressive power of various kinds of neural network models (Šíma and Orponen, 2003). A common idea is to relate neural networks to theoretical computation devices like automata (Hopcroft and Ullman, 1979; Sipser, 2006). A lower bound on the expressiveness of a neural network model can be established by simulating automata with neural networks in that model. Conversely, an upper bound on the expressiveness can be established by simulating neural networks with automata. In previous works, simulations with neural networks of both deterministic finite automata (Alon et al., 1991; Indyk, 1995; Omlin and Giles, 1996; Horne and Hush, 1996) and nondeterministic finite automata (Carrasco et al., 1999) have been studied. Some models of neural networks even allow the simulation of arbitrary Turing machines, that are

much more powerful than finite automata, see e.g. (Siegelmann and Sontag, 1995; Maass, 1996). However, the technical constructions used for the simulation of such powerful machines are not necessarily biologically plausible.

In this article, our approach in the expressivity study is to describe the behaviors exhibited by positive neural networks, as follows. An input symbol in our model is a subset of input neurons that are concurrently active. For example, if the symbol $\{a, b, c\}$ is presented as input to the network at time t then this means that a, b, and c are the only input neurons that are (concurrently) active at time t. The empty symbol would mean that no input neurons are active. Output symbols are defined similarly, but over output neurons instead. Now, we define behaviors as functions that transform each sequence of input symbols to a sequence of output symbols. Different behaviors can be expressed by varying the connection weights of a positive neural network. By describing such behaviors, we can derive theoretical upper and lower bounds on the expressivity of positive neural networks. We emphasize that we feed sequences of input symbols to the neural networks, not single symbols.

Our assumption of multiple input neurons that may become concurrently active is in contrast to models of past expressivity studies, where either input encodings were used that *(i)* made only one input neuron active at any given time (Šíma and Wiedermann, 1998; Carrasco et al., 1999, 2000) or *(ii)* presented a single bit string just once over multiple input neurons after which they remained silent (Šíma and Orponen, 2003). Essentially, multiple parallel inputs versus a single sequential input is a matter of input alphabet. One might propose that an external process could transform multiple parallel inputs to a single sequential one (say, a stream of bits), after which previous results might be applied, e.g. (Šíma and Wiedermann, 1998). However, in a biologically plausible setting there is no such external process: in general, it seems that inputs arrive from multiple sensory organs in parallel, and an internal circuit receives inputs from multiple other internal circuits in parallel as remarked at the beginning of the Introduction. Because our aim is to better understand the biologically plausible setting, we therefore have to work with an input alphabet where multiple (parallel) input neurons may become concurrently active.

Monotone-regular behaviors To describe the behaviors exhibited by positive neural networks, we use the class of regular languages, which are those languages recognized by finite automata (Hopcroft and Ullman, 1979; Sipser, 2006). Previously, Šíma and Wiedermann (1998) have shown that neural networks in discrete time that read bit strings over a single input neuron recognize whether prefixes of the input string belong to a regular language or not. In their technical construction, Šíma and Wiedermann essentially simulate nondeterministic finite automata.

In this article, we simulate nondeterministic finite automata in the setting of positive neural networks.[1] Using the simulation, we show that the class of positive neural networks captures the so-called monotone-regular behaviors. A

[1]For example, the finite automaton in Figure 1a is simulated by the positive neural network in Figure 1b.

monotone-regular behavior describes the activations of each output neuron with a regular language over input symbols, where each symbol may contain multiple input neurons as described above. Monotonicity means that each output neuron is activated whenever strings of the regular language are embedded in the input, regardless of any other activations of input neurons. Phrased differently, enriching an input with more activations of input neurons will never lead to fewer activations of output neurons. Monotonicity arises because neurons only excite each other and do not inhibit each other. This notion did not appear explicitly in the work by Šíma and Wiedermann (1998) because their neural networks exactly recognize regular languages over the single input neuron by using inhibition (i.e., negative connection weights): inhibition allows to explicitly test for the absence of input activations at certain times.

Delay is a standard notion in the study of neural networks (Šíma and Orponen, 2003). Intuitively, delay is the number of extra time steps needed by the neural network before it can produce the output symbols prescribed by the behavior. We show that each monotone-regular behavior can be implemented by a positive neural network with a delay of one time unit. This result is in line with the result by Šíma and Wiedermann (1998), but it is based on a new technical construction to deal with the more complex input symbols generated by concurrently active input neurons. We simulate automaton states by neurons as expected, but we design the weights of the incoming connections to a neuron to express simultaneously *(i)* an "or" over context neurons that provide working memory and *(ii)* an "and" over all input neurons mentioned in an input symbol. As in the work by Šíma and Wiedermann (1998), the constructed neural network may activate auxiliary neurons in parallel. Accordingly, our simulation preserves the nondeterminism, or parallelism, of the simulated automaton. As an additional result, we show that a large class of monotone-regular behaviors can be implemented with zero delay. And, interestingly, some simple monotone-regular behaviors can provably not be implemented with zero delay.

To the best of our knowledge, the notion of monotone-regular behaviors is introduced in this article for the first time. But this notion is a natural combination of some previously existing concepts, results, and general intuition, as follows. First, it is likely that both the temporal structure and spatial structure of sensory inputs are important for biological organisms (Buonomano and Maass, 2009). The temporal structure describes the timing of sensory events, and the spatial structure describes which and how many neurons are used to represent each sensory event. Second, the well-known class of regular languages from formal language theory describes symbol sequences that exhibit certain patterns or regularities (Hopcroft and Ullman, 1979); temporal structure is represented by the ordering of symbols, and spatial structure is given by the individual symbols. The relationship between regular languages and neural network models has also been investigated before (Šíma and Wiedermann, 1998). Third, without inhibition, neurons only excite each other and therefore an increased activity of input neurons will not lead to a decreased activity of output neurons. Without inhibition, neurons will respond to patterns embedded in the input stream regardless of any other simultaneous patterns, giving rise to a form of monotonicity on the

resulting behavior.

Relevance We conclude the Introduction by placing our result in a larger picture. The intuition explored in this article, is that neural networks in some sense represent grammars. A grammar is any set of rules describing how to form sequences of symbols over a given alphabet; such sequences may be called sentences.

In an experiment by Reber (1967), subjects were shown sentences generated by an artificial grammar, but the rules of the grammar were not shown. Subjects were better at memorizing and reproducing sentences generated by the grammar when compared to sentences that are just randomly generated. Moreover, subjects were generally able to classify sentences as being grammatical or not. Interestingly, however, subjects could not verbalize the underlying rules of the grammar. This experiment suggests that organisms learn patterns from the environment when the patterns are sufficiently repeated. Those patterns get embedded into the neural network. The resulting grammar can not necessarily be described or explicitly accessed by the organism.

The grammar hypothesis is to some extent confirmed by neuronal recordings of brain areas involved with movement planning in monkeys (Shima and Tanji, 2000; Isoda and Tanji, 2003). These experimental findings suggest that movement sequences are represented by two groups of neurons: the first group represents the temporal structure, and the second group represents individual actions. Neurons in the first group might be viewed as stringing together the output symbols represented by the second group. Hence, the first group might represent the structure of a grammar, indicating the allowed sentences of output symbols.

The above experiments are complemented by Kappel et al. (2014), who have theoretically shown and demonstrated with computer simulations that neural winner-take-all circuits can (learn to) express hidden Markov models. Hidden Markov models are finite state machines with transition probabilities between states, and each state has a certain probability to emit symbols. Such models describe grammars, because each visited state can contribute symbols to an increasing sentence. One of the insights by Kappel et al. is that by repeatedly showing sentences generated by a hidden Markov model to a learning winner-take-all circuit, the states of the Markov model are eventually encoded by global network states, i.e., groups of activated neurons. This way, the neural network builds an internal model of how sentences are formed by the hidden grammar. Interestingly, the computer simulations by Kappel et al. (2014) clearly demonstrate (and visualize) that neurons learn to cooperate in a chain-like fashion, expressing the symbol chains in the hidden grammar. This corresponds well to the earlier predictions (Reber, 1967; Shima and Tanji, 2000; Isoda and Tanji, 2003). We might speculate that, if one assumes a real-world environment to be a (complex) hidden Markov model, organisms with a neural network can learn to understand the patterns, or sentences, generated by that environment.

In this article, we have made the above grammar intuition formal for positive neural networks. By characterizing the expressive power of positive neural networks with monotone-regular behaviors, the activation of an output neuron may be viewed as the recognition of a pattern in the input. This way, each output neu-

ron represents a grammar: the output neuron recognizes which input sentences satisfy the grammar. Moreover, our finding that nondeterministic finite automata can be simulated by positive neural networks is in line with the expressivity result of Kappel et al. (2014) because hidden Markov models generalize nondeterministic automata (Dupont et al., 2005): in a standard nondeterministic automaton, all successor states of a given state are equally likely, whereas a hidden Markov model could assign different transition probabilities to each successor state. The simulation of automata by previous works (Šíma and Orponen, 2003) and the current article, combined with the result by Kappel et al. (2014), might provide a useful intuition: individual neurons or groups of neurons could represent automaton states of a grammar.

Outline This article is organized as follows. We discuss related work in Section 2. We provide in Section 3 the necessary preliminaries, including the formalization of positive neural networks and monotone-regular behaviors. Next, we provide in Section 4 our results regarding the expressivity of positive neural networks. We conclude in Section 5 with topics for future work.

2 Related Work

We now discuss several theoretical works that are related to this article.

The relationship between the semantic notion of monotonicity and the syntactic notion of positive weights is natural, and has been explored in other settings than the current article, see e.g. (Beimel and Weinreb, 2006; Legenstein and Maass, 2008; Daniels and Velikova, 2010). In particular, the paper by Legenstein and Maass (2008) studies more generally the classification ability of sign-constrained neurons. In their setting, fixing some natural number n, there is one output neuron that is given points from $\mathbb{R}^n$ as presynaptic input. Each choice of weights from $\mathbb{R}^n$ allows the output neuron to express a binary (true-false) classification of input points, where "true" is represented by the activation of the output neuron. By imposing sign-constraints on the weights, different families of output neurons are created. For example, one could demand that only positive weights are used. It turns out that the VC-dimension of sign-constrained neurons with n presynaptic inputs is n, which is only one less than unconstrained neurons.[2] Moreover, Legenstein and Maass (2008) characterize the input sets (containing n points from $\mathbb{R}^n$) for which sign-constrained neurons can express all binary classification functions.

Like in the Introduction, we define an input symbol as a set of concurrently active input neurons. The results by Legenstein and Maass (2008) can be used to better understand the nature of input symbols that are distinguishable from each other by a single output neuron having nonnegative presynaptic weights, also referred to as a positive neuron. Indeed, if we would receive a stream of input

[2]For example, for the case of positive weights, the VC-dimension n tells us that there is an input set $S \subseteq \mathbb{R}^n$ with $|S| = n$ that can be *shattered* by the family of positive presynaptic weights, in the following sense: for each classification $h : S \to \{1, 0\}$, there exists a positive presynaptic weight vector in $\mathbb{R}^n$ allowing the resulting single output neuron to express h on S.

symbols and if we would like to individually classify each input symbol by the activation behavior of a positive output neuron (where activation means "true"), the results by Legenstein and Maass (2008) provide sufficient and necessary conditions on the presented input symbols to allow the output neuron to implement the classification. It is also possible, however, to consider a temporal context for each input symbol: then, the decision to activate an output neuron for a certain input symbol depends on which input symbols were shown previously. For example, considering an alphabet of input symbols A, B, and C, we might want to activate the output neuron on symbol B while witnessing the string (A, A, B) but not while only witnessing the string (C, C, B). Hence, the output activation for symbol B depends on the temporal context in which B appears. For a maximum string length k, and assuming that input symbols have maximum size n, one could in principle present a string of k symbols to the output neuron in a single glance, using n times k (new) input neurons. In that approach, the output neuron could even recognize strings of a length $l \leq k$ that are embedded into the presented string, giving rise to the notion of monotonicity discussed in this article. For such cases, the results by Legenstein and Maass (2008) could still be applied to better understand the nature of strings that can be recognized by a positive output neuron. As mentioned in the Introduction, in this article we use regular languages to describe the strings of input symbols upon which an output neuron should become activated. In contrast to fixing a maximum length k on strings, regular languages can describe arbitrarily long strings, by allowing arbitrary repetitions of substrings. By applying our expressivity lower bound (Theorem 4.5), we can for example construct a neural network that activates an output neuron whenever a string of the form A^*B is embedded at the end of the so-far witnessed stream of input symbols, where A^* denotes that symbol A may be repeated an arbitrary number of times. Moreover, the neural network can be constructed in such a way that the output neuron responds with a delay of at most one time unit compared to the pattern's appearance. Results regarding regular languages, in combination with delay, can be analyzed in the framework of the current article. Instead of single output neurons, we consider larger networks where auxiliary neurons can assist the output neurons by reasoning over the temporal context of the input symbols.

Positive neural networks are also related to the monotone acyclic AND-OR boolean circuits, studied e.g. by Alon and Boppana (1987). Concretely, an AND-OR circuit is a directed acyclic graph whose vertices are gates that compute either an OR or an AND of the boolean signals generated by the predecessor gates. The input to the circuit consists of a fixed number of boolean variables. Each AND-OR circuit is a special case of a positive neural network: each AND and OR gate can be translated to a neuron performing the same computation, by applying positive edge weights to presynaptic neurons.

The neurons studied in the current article compute a Boolean linear threshold function of their presynaptic inputs: each neuron computes a weighted sum of the (Boolean) activations of its presynaptic neurons and becomes activated when that sum passes a threshold. Now, the acyclic AND-OR-NOT circuits discussed

by Parberry (1994) are related to the AND-OR circuits mentioned above.[3] It turns out that every Boolean linear threshold function over n input variables can be computed by an acyclic AND-OR-NOT circuit with a number of gates that is polynomial in n and with a depth that is logarithmic in n.[4] One may call such circuits "small", although not of constant size in n. The essential idea in this transformation, is that AND-OR-NOT circuits can compute the sum of the weights for which the corresponding presynaptic input is true, and subsequently compare that sum to a threshold; a binary encoding of the weights and threshold can be embedded into the circuit, but care is taken to ensure that this encoding is of polynomial size in n. It appears, however, that NOT gates play a crucial role in the construction, for handling the carry bit in the summation. The resulting circuit is therefore not positive (or monotone) in the sense of Alon and Boppana (1987). For completeness, we remark that delay is increased if one would replace each Boolean linear threshold neuron with a corresponding AND-OR-NOT sub-circuit, at least if one time unit is consumed for calculating each gate of each sub-circuit. Given that the transformation produces sub-circuits of non-constant depth, it appears nontrivial to describe the overall delay exhibited by the network.

Horne and Hush (1996) show upper and lower bounds on the number of required neurons for simulating deterministic finite automata that read and write sequences of bits. Their approach is to encode the state transition function of an automaton as a Boolean function, that is subsequently implemented by an acyclic neural network.[5] Each execution of the entire acyclic neural network corresponds to one update step of the simulated automaton. A possible advantage of the method by Horne and Hush (1996), is that the required number of neurons could be smaller than the number of automaton states. But, like in the discussion of AND-OR-NOT circuits above, the construction introduces a nontrivial delay in the simulation of the automaton if each neuron (or each layer of neurons) is viewed as consuming one time unit. In this article we are not necessarily concerned with compacting automaton states in as few neurons as possible, but we are instead interested in recognizing a regular language under a maximum delay constraint (of one time unit) in the setting where multiple input neurons can be concurrently active and produce a stream of complex input symbols. The construction by Horne and Hush (1996) can be modified to multiple input neurons that may become concurrently active.

For completeness, we remark that in this article we do not impose the restriction that the (simulated) automata are deterministic. Moreover, in our simulation of automata, we take care to only introduce a polynomial increase in the number of neurons compared to the original number of automaton states (see Theorem 4.5). In particular, if the original automaton is nondeterministic, the constructed neural network for this automaton will preserve that nondeterminism in the form of

[3]Parberry (1994) actually refers to AND-OR-NOT circuits as AND-OR circuits because NOT gates can be pushed to the first layer, which can be used to establish a normal form where layers of AND gates alternate with layers of OR gates (with negation only at the first level).

[4]In particular, we are referring to Theorem 7.4.7 (and subsequently Corollary 6.1.6) of Parberry (1994).

[5]If there are m automaton states then each state can be represented by $\lceil \log_2 m \rceil$ bits.

concurrently active neurons. This stems from our original motivation to propose a construction that could in principle be biologically plausible, where multiple neurons could be active in parallel to explore the different states of the original automaton. From this perspective, implementing a deterministic solution, where only one neuron is active at any given moment, would be less interesting.

Monotonicity in the context of automata has appeared earlier in the work by Gécseg and Imreh (2001). There, an automaton is called *monotone* if there exists a partial order $\leq$ on the automaton states, such that each transition (a, x, a'), going from state a to state a' through symbol x, satisfies $a \leq a'$. Intuitively, this condition prohibits cycles between two different states while parsing a string. In particular, the same state a may not be reused except when the previous state was already a (i.e., self-looping on a is allowed for a while). A language is called monotone when there is a monotone automaton that recognizes it. This notion of monotonicity is not immediately related to the current article, because our notion of monotonicity is not defined on automata (nor on neural networks) but on behaviors, which formalize semantics separate from the actual computation mechanism. Moreover, the positive neural networks studied in this article may reuse the same global state while processing an input string, where a global state is defined as the set of currently activated neurons. For example, the empty global state could occur multiple times while processing an input string, even when this empty global state has precursor states and successor states that are not empty.

3 Preliminaries

3.1 Finite Automata and Regular Languages

We recall the definitions of finite automata and regular languages (Sipser, 2006). An *alphabet* Σ is a finite set. A *string* α *over* Σ is a finite sequence of elements from Σ. The empty string corresponds to the empty sequence. We also refer to the elements of a string as its *symbols*. A *language* $\mathcal{L}$ *over* Σ is a set of strings over Σ. Languages can be finite or infinite.

The length of a string α is denoted $|\alpha|$. For each $i \in \{1, \ldots, |\alpha|\}$, we write α_i to denote the symbol of α at position i. We use the following string notation: $\alpha = (\alpha_1, \ldots, \alpha_{|\alpha|})$. For each $i \in \{1, \ldots, |\alpha|\}$, let $\alpha_{\to i}$ denote the prefix $(\alpha_1, \ldots, \alpha_i)$.

A *(finite) automaton* is a tuple $M = (Q, \Sigma, \delta, q^{\mathrm{s}}, F)$ where

- Q is a finite set of states;
- Σ is an alphabet;
- δ is the transition function, mapping each pair $(q, S) \in Q \times \Sigma$ to a subset of Q;[6]
- q^{s} is the start state, with $q^{\mathrm{s}} \in Q$; and,
- $F \subseteq Q$ is the set of accepting states.

[6]Importantly, this subset could be empty.

Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ be a string over Σ. We call a sequence of states $q_1, \ldots, q_{n+1}$ of M a *run of M on α* if the following conditions are satisfied:

- $q_1 = q^{\mathrm{s}}$; and,
- $q_i \in \delta(q_{i-1}, \alpha_{i-1})$ for each $i \in \{2, \ldots, n+1\}$.

We say that the run $q_1, \ldots, q_{n+1}$ is *accepting* if $q_{n+1} \in F$. We say that the automaton M *accepts* α if there is an accepting run of M on α.[7] Automaton M could be *nondeterministic*: for the same input string α, there could be multiple accepting runs. See also Remark 3.1 below.

We define the language $\mathcal{L}$ over Σ that is *recognized* by M: language $\mathcal{L}$ is the set of all strings over Σ that are accepted by M. Now, a language is said to be *regular* if it is recognized by an automaton.

Remark 3.1. We call an automaton $M = (Q, \Sigma, \delta, q^{\mathrm{s}}, F)$ *deterministic* if $|\delta(q, S)| = 1$ for each $(q, S) \in Q \times \Sigma$, i.e., the successor state is uniquely defined for each combination of a predecessor state and an input symbol. Nondeterministic automata are typically smaller and easier to understand compared to deterministic automata (Sipser, 2006). Moreover, if M is nondeterministic then it represents *parallel* computation. To see this, we can define an alternative but equivalent semantics for M as follows (Sipser, 2006). The *parallel run* of M on an input string $\alpha = (\alpha_1, \ldots, \alpha_n)$ over Σ is the sequence

$$P_1, \ldots, P_{n+1},$$

where $P_1 = \{q^{\mathrm{s}}\}$ and $P_i = \{q_i \in Q \mid \exists q_{i-1} \in P_{i-1} \text{ with } q_i \in \delta(q_{i-1}, \alpha_{i-1})\}$ for each $i \in \{2, \ldots, n+1\}$. We say that M *accepts α under the parallel semantics* if the last state set of the parallel run contains an accepting state. It can be shown that the parallel semantics is equivalent to the semantics of acceptance given earlier. Because non-deterministic automata explore multiple states simultaneously at runtime, they appear to be a natural model for understanding parallel computation in neural networks (see Section 4.2). □

3.2 Behaviors

We use behaviors to describe computations separate from neural networks. Regarding notation, for a set X, let $\mathcal{P}(X)$ denote the *powerset* of X, i.e., the set of all subsets of X.

Let $\mathcal{I}$ and $\mathcal{O}$ be finite sets, whose elements we may think of as representing neurons. In particular, the elements of $\mathcal{I}$ and $\mathcal{O}$ are called *input* and *output* neurons respectively. Now, a *behavior $\boldsymbol{B}$ over input set $\mathcal{I}$ and output set $\mathcal{O}$* is a function that maps each nonempty string over alphabet $\mathcal{P}(\mathcal{I})$ to a subset of $\mathcal{O}$. Regarding terminology, for a string α over $\mathcal{P}(\mathcal{I})$ and an index $i \in \{1, \ldots, |\alpha|\}$,

[7] Our definition of automata omits the special symbol ϵ, that can be used to visit multiple states in sequence without simultaneously reading symbols from the input string. This feature can indeed always be removed from an automaton, without increasing the number of states (Hopcroft and Ullman, 1979).

the symbol α_i says which input neurons are *active* at (discrete) time i. Note that multiple input neurons can be concurrently active.

For an input string $\alpha = (\alpha_1, \ldots, \alpha_n)$ over $\mathcal{P}(\mathcal{I})$, the behavior $\boldsymbol{B}$ implicitly defines the following output string $\beta = (\beta_1, \ldots, \beta_{n+1})$ over $\mathcal{P}(\mathcal{O})$:

- $\beta_1 = \emptyset$, and
- $\beta_i = \boldsymbol{B}(\alpha_{\to i-1})$ for each $i \in \{2, \ldots, n+1\}$.

So, the behavior has access to the preceding input history when producing each output symbol. But an output symbol is never based on future input symbols.

3.3 Monotone-regular Behaviors

Let $\mathcal{I}$ be a set of input neurons. We call a language $\mathcal{L}$ over alphabet $\mathcal{P}(\mathcal{I})$ *founded* when each string of $\mathcal{L}$ is nonempty and has a nonempty subset of $\mathcal{I}$ for its first symbol. Also, for two strings α and β over $\mathcal{P}(\mathcal{I})$, we say that α *embeds* β if α has a suffix γ with $|\gamma| = |\beta|$ such that $\beta_i \subseteq \gamma_i$ for each $i \in \{1, \ldots, |\beta|\}$. Note that β occurs at the *end* of α. Also note that a string embeds itself according to this definition.

Let $\boldsymbol{B}$ be a behavior over an input set $\mathcal{I}$ and an output set $\mathcal{O}$. We call $\boldsymbol{B}$ *monotone-regular* if for each output neuron $x \in \mathcal{O}$ there is a founded regular language $\mathcal{L}(x)$ such that for each nonempty input string α over $\mathcal{P}(\mathcal{I})$,

$$x \in \boldsymbol{B}(\alpha) \Leftrightarrow \alpha \text{ embeds a string } \beta \in \mathcal{L}(x).$$

Intuitively, the regular language $\mathcal{L}(x)$ describes the patterns that output neuron x reacts to. So, the meaning of neuron x is the recognition of language $\mathcal{L}(x)$. We use the term *monotone* to indicate that $\mathcal{L}(x)$ is recognized within surrounding superfluous activations of input neurons, through the notion of embedding. The restriction to founded regular languages expresses that outputs do not emerge spontaneously, i.e, the activations of output neurons are given the opportunity to witness at least one activation of an input neuron.

Remark 3.2. Let M be an automaton that recognizes a founded regular language over $\mathcal{P}(\mathcal{I})$. When reading the symbol $\emptyset$ from the start state of M, we may only enter states from which it is impossible to reach an accepting state; otherwise the recognized language is not founded. See also Lemma 4.4 in Section 4.2. □

Remark 3.3. The definition of monotone-regular behaviors fuses the separate notions of monotonicity and (founded) regular languages. It also seems possible to define monotone-regular behaviors as those behaviors that are both monotone and regular. However, in the formalization of regular behaviors, the regular language of each output neuron x likely has to describe the entire input strings upon which x is activated (at the end). This is in contrast to the current formalization of monotone-regular behaviors, where the (founded) regular language $\mathcal{L}(x)$ could be very small, describing only the patterns that x is really trying to recognize, even when those patterns are embedded in larger inputs. The current formalization

is therefore more insightful for our construction in the expressivity lower bound (Theorem 4.5), where we convert an automaton for $\mathcal{L}(x)$ to a neural network that serves as a pattern recognizer for output neuron x. The current formalization of monotone-regular behaviors allows the pattern recognizer to be as small as possible. □

3.4 Positive Neural Networks

We define a neural network model that is related to previous discrete time models (Šíma and Wiedermann, 1998; Šíma and Orponen, 2003), but with the following differences: we have no inhibition, and we consider multiple input neurons that are allowed to be concurrently active.

Formally, a *(positive) neural network* $\boldsymbol{\mathcal{N}}$ is a tuple $(\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W})$, where

- $\mathcal{I}$, $\mathcal{O}$, and $\mathcal{A}$ are finite and pairwise disjoint sets, containing respectively the *input* neurons, the *output* neurons, and the *auxiliary* neurons;[8]
- we let

$$\begin{aligned} \mathit{edges}(\boldsymbol{\mathcal{N}}) =& (\mathcal{I} \times \mathcal{O}) \cup (\mathcal{I} \times \mathcal{A}) \cup (\mathcal{A} \times \mathcal{O}) \\ & \cup \{(x, y) \in \mathcal{A} \times \mathcal{A} \mid x \neq y\} \end{aligned}$$

  be the set of possible connections; and,
- the function $\mathcal{W}$ is the *weight function* that maps each $(x, y) \in \mathit{edges}(\boldsymbol{\mathcal{N}})$ to a value in $[0, 1]$.

Note that there are direct connections from the input neurons to the output neurons. The weight 0 is used for representing missing connections. Intuitively, the role of the auxiliary neurons is to provide working memory while processing input strings. For example, the activation of an auxiliary neuron could mean that a certain pattern was detected in the input string. Auxiliary neurons can recognize increasingly longer patterns by activating each other (Elman, 1990; Kappel et al., 2014). We refer to Section 4 for constructions involving auxiliary neurons.

We introduce some notations for convenience. If $\boldsymbol{\mathcal{N}}$ is understood from the context, for each $x \in \mathcal{I} \cup \mathcal{O} \cup \mathcal{A}$, we abbreviate

$$\mathit{pre}(x) = \{y \in \mathcal{I} \cup \mathcal{A} \mid (y, x) \in \mathit{edges}(\boldsymbol{\mathcal{N}}) \text{ and } \mathcal{W}(y, x) > 0\}$$

and

$$\mathit{post}(x) = \{y \in \mathcal{O} \cup \mathcal{A} \mid (x, y) \in \mathit{edges}(\boldsymbol{\mathcal{N}}) \text{ and } \mathcal{W}(x, y) > 0\}.$$

We call $\mathit{pre}(x)$ the set of *presynaptic* neurons of x, and $\mathit{post}(x)$ the set of *postsynaptic* neurons of x.

3.1 Operational Semantics

Let $\boldsymbol{\mathcal{N}} = (\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W})$ be a neural network. We formalize how $\boldsymbol{\mathcal{N}}$ processes an input string α over $\mathcal{P}(\mathcal{I})$. We start with the intuition.

[8] Auxiliary neurons are also sometimes called *hidden* neurons (Šíma and Orponen, 2003).

Intuition We do $|\alpha|$ steps, called *transitions*, to process all symbols of α. At each time $i \in \{1, \ldots, |\alpha|\}$, also referred to as transition i, we show the input symbol α_i to $\boldsymbol{\mathcal{N}}$. Specifically, an input neuron $x \in \mathcal{I}$ is active at time i if $x \in \alpha_i$. Input symbols could activate auxiliary and output neurons. Auxiliary neurons could in turn also activate other auxiliary neurons and output neurons. Each time a neuron of $\mathcal{I} \cup \mathcal{A}$ becomes active, it conceptually emits a signal. The signal emitted by a neuron x at time i travels to all postsynaptic neurons y of x, and such received signals are processed by y at the next time $i+1$. Each signal that is emitted by x and received by a postsynaptic neuron y has an associated weight, namely, the weight on the connection from x to y. Subsequently, a postsynaptic neuron y emits a (next) signal if the sum of all received signal weights is larger than or equal to a firing threshold. The firing threshold in our model is 1 for all neurons. All received signals are immediately discarded when proceeding to the next time. In the formalization below, the conceptual signals are not explicitly represented, and instead the transitions directly update sets of activated neurons.

Transitions A *transition of* $\boldsymbol{\mathcal{N}}$ is a triple (N_i, S, N_j) where $N_i \subseteq \mathcal{O} \cup \mathcal{A}$ and $N_j \subseteq \mathcal{O} \cup \mathcal{A}$ are two sets of *activated* neurons, $S \in \mathcal{P}(\mathcal{I})$ is an input symbol, and where

$$N_j = \{y \in \mathcal{O} \cup \mathcal{A} \mid \sum_{z \in pre(y) \cap (N_i \cup S)} \mathcal{W}(z, y) \geq 1\}.$$

We call N_i the *source set*, N_j the *target set*, and S the symbol that is *read*.[9]

Run The *run* $\mathcal{R}$ *of* $\boldsymbol{\mathcal{N}}$ *on input* α is the unique sequence of $|\alpha|$ transitions for which

- the transition with ordinal $i \in \{1, \ldots, |\alpha|\}$ reads input symbol α_i;
- the source set of the first transition is $\emptyset$;
- the target set of each transition is the source set of the next transition.

Note that $\mathcal{R}$ defines $|\alpha| + 1$ sets of activated neurons, including the first source set. We define the *output of* $\boldsymbol{\mathcal{N}}$ *on* α, denoted $\boldsymbol{\mathcal{N}}(\alpha)$, as the set $N \cap \mathcal{O}$ where N is the target set of the last transition in the run of $\boldsymbol{\mathcal{N}}$ on α.

It is possible to consider the behavior $\boldsymbol{B}$ *defined* by $\boldsymbol{\mathcal{N}}$: for each nonempty input string α, we define $\boldsymbol{B}(\alpha) = \boldsymbol{\mathcal{N}}(\alpha)$. So, like a behavior, a neural network implicitly transforms an input string $\alpha = (\alpha_1, \ldots, \alpha_n)$ over $\mathcal{P}(\mathcal{I})$ to an output string $\beta = (\beta_1, \ldots, \beta_{n+1})$ over $\mathcal{P}(\mathcal{O})$:

- $\beta_1 = \emptyset$, and
- $\beta_i = \boldsymbol{\mathcal{N}}(\alpha_{\to i-1})$ for each $i \in \{2, \ldots, n+1\}$.

[9] We include output neurons in transitions only for technical convenience. It is indeed not essential to include output neurons in the source and target sets, because output neurons have no postsynaptic neurons and their activation can be uniquely deduced from the activations of auxiliary neurons and input neurons.

3.2 Design Choices

We discuss the design choices of the formalization of positive neural networks. Although the model is simple, we have some preferences in how to formalize it.

First, the reason for not having connections from output neurons to auxiliary neurons is for simplicity, and so that proofs can more cleanly separate the roles of neurons. However, connections from output neurons to auxiliary neurons can be simulated in the current model by duplicating each output neuron as an auxiliary neuron, including its presynaptic weights.

We exclude self-connections on neurons, i.e., connections from a neuron to itself, because such connections might be less common in biological neural networks.

The connection weights are restricted to the interval $[0, 1]$ to express that there is a maximal strength by which any two neurons can be connected. In biological neural networks, the weight contributed by a single connection, which abstracts a set of synapses, is usually much smaller than the firing threshold (Gerstner et al., 2014). For technical simplicity (cf. Section 4), however, the weights in our model are relatively large compared to the firing threshold.[10] Intuitively, such larger weights represent a hidden assembly of multiple neurons that become active concurrently, causing the resulting sum of emitted weights to be large (Maass, 1996).

We use a normalized firing threshold of 1 for simplicity. Another choice of positive firing threshold could in principle be compensated for by allowing connection weights larger than 1.

3.5 Implementing Behaviors, with Delay

Let $\boldsymbol{\mathcal{N}} = (\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W})$ be a neural network. We say that a behavior $\boldsymbol{B}$ is *compatible* with $\boldsymbol{\mathcal{N}}$ if $\boldsymbol{B}$ is over input set $\mathcal{I}$ and output set $\mathcal{O}$.

Delay is a standard notion in the expressivity study of neural networks (Šíma and Wiedermann, 1998; Šíma and Orponen, 2003). We say that $\boldsymbol{\mathcal{N}}$ *implements a compatible behavior* $\boldsymbol{B}$ *with delay* $k \in \mathbb{N}$ when for each input string α over $\mathcal{P}(\mathcal{I})$,

- if $|\alpha| \leq k$ then $\boldsymbol{\mathcal{N}}(\alpha) = \emptyset$;[11] and,
- if $|\alpha| > k$ then $\boldsymbol{\mathcal{N}}(\alpha) = \boldsymbol{B}(\alpha_{\to m})$ where $m = |\alpha| - k$.

Intuitively, delay is the amount of additional time steps that $\boldsymbol{\mathcal{N}}$ needs before it can conform to the behavior. This additional time is provided by reading more input symbols.[12] Note that a zero delay implementation corresponds to $\boldsymbol{\mathcal{N}}(\alpha) = \boldsymbol{B}(\alpha)$ for all input strings α.

[10] The largest weight is 1, which is equal to the firing threshold; so, a neuron could in principle become activated when only one of its presynaptic neurons is active.

[11] If $k = 0$ then this condition is immediately true because we consider no input strings with length zero.

[12] Suppose $\boldsymbol{\mathcal{N}}$ implements $\boldsymbol{B}$ with delay k. Let α be an input string with $|\alpha| > k$. If we consider α as the entire input to the network $\boldsymbol{\mathcal{N}}$, then the last k input symbols of α may be arbitrary; those symbols only provide additional time steps for $\boldsymbol{\mathcal{N}}$ to compute $\boldsymbol{B}(\alpha_{\to m})$ where $m = |\alpha| - k$.

Letting $\boldsymbol{B}$ be the behavior defined by $\boldsymbol{\mathcal{N}}$, note that $\boldsymbol{\mathcal{N}}$ implements $\boldsymbol{B}$ with zero delay.

Remark 3.4. Šíma and Wiedermann (1998) show that a neural network recognizing a regular language with delay k over a single input neuron can be transformed into a (larger) neural network that recognizes the same language with delay 1. An assumption in the construction, is that the delay k in the original network is caused by paths of length k from the input neuron to output neurons.

The definition of delay in this article is purely semantical: we only look at the timing of output neurons. There could be delay on output neurons, even though there might be direct connections from input neurons to output neurons, because output neurons might cooperate with auxiliary neurons (which might introduce delays).

For completeness, we note that our construction in the expressivity lower bound (Theorem 4.5) does not create direct connections from input neurons to output neurons, and thereby incurs a delay of a least one time unit; but we show that it is actually a delay of precisely one time unit. This construction therefore resembles the syntactical assumption by Šíma and Wiedermann (1998). □

4 Expressivity Results

Our goal is to better understand what positive neural networks can do. Within the discrete-time framework of monotone-regular behaviors, we propose an upper bound on expressivity in Section 4.1; a lower bound on expressivity in Section 4.2; and, in Section 4.3, examples showing that these bounds do not coincide. This separation arises because our analysis takes into account the delay by which a neural network implements a monotone-regular behavior. It turns out that an implementation of zero delay exists for some monotone-regular behaviors, but not for other monotone-regular behaviors. A delay of one time unit is sufficient for implementing all monotone-regular behaviors. As an additional result, we present in Section 4.4 a large class of monotone-regular behaviors that can be implemented with zero delay. If we would ignore delay, however, our upper and lower bound results (Sections 4.1 and 4.2 respectively) intuitively say that the class of positive neural networks captures the class of monotone-regular behaviors: the behavior defined by a positive neural network is monotone-regular, and each monotone-regular behavior can be implemented by a positive neural network.

4.1 Upper Bound

Our expressivity upper bound says that only monotone-regular behaviors can be expressed by positive neural networks. This result is in line with the result by Šíma and Wiedermann (1998), with the difference that we now work with multiple input neurons and the notion of monotonicity.

Theorem 4.1. The behaviors defined by positive neural networks are monotone-regular.

Proof. Intuitively, because a positive neural network only has a finite number of subsets of auxiliary neurons to form its memory, the network behaves like a finite automaton. Hence, as is well-known, the performed computation can be described by a regular language (Šíma and Wiedermann, 1998). An interesting novel aspect, however, is monotonicity, meaning that output neurons recognize patterns even when those patterns are embedded into larger inputs.

Let $\boldsymbol{\mathcal{N}} = (\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W})$ be a positive neural network. Let $\boldsymbol{B}$ denote the behavior defined by $\boldsymbol{\mathcal{N}}$. We show that $\boldsymbol{B}$ is monotone-regular. Fix some $x \in \mathcal{O}$. We define a founded regular language $\mathcal{L}(x)$ such that for each input string α over $\mathcal{P}(\mathcal{I})$ we have

$$x \in \boldsymbol{B}(\alpha) \Leftrightarrow \alpha \text{ embeds a string } \beta \in \mathcal{L}(x).$$

We first define a deterministic automaton M. Let q^{s} and q^{h} be two state symbols where $q^{\mathrm{s}} \neq q^{\mathrm{h}}$ and $\{q^{\mathrm{s}}, q^{\mathrm{h}}\} \cap \mathcal{P}(\mathcal{O} \cup \mathcal{A}) = \emptyset$. We call q^{h} the *halt state* because no useful processing will be performed anymore when M gets into state q^{h} (see below). We concretely define $M = (Q, \Sigma, \delta, q^{\mathrm{s}}, F)$, where

- $Q = \{q^{\mathrm{s}}, q^{\mathrm{h}}\} \cup \mathcal{P}(\mathcal{O} \cup \mathcal{A})$;
- $\Sigma = \mathcal{P}(\mathcal{I})$;
- regarding δ, for each $(q, S) \in Q \times \Sigma$,
  - if $q = q^{\mathrm{s}}$ and $S = \emptyset$ then $\delta(q, S) = \{q^{\mathrm{h}}\}$;
  - if $q = q^{\mathrm{s}}$ and $S \neq \emptyset$ then $\delta(q, S) = \{q'\}$ where

$$q' = \{y \in \mathcal{O} \cup \mathcal{A} \mid \sum_{z \in pre(y) \cap S} \mathcal{W}(z, y) \geq 1\};$$

  - if $q = q^{\mathrm{h}}$ then $\delta(q, S) = \{q^{\mathrm{h}}\}$;
  - if $q \in \mathcal{P}(\mathcal{O} \cup \mathcal{A})$ then $\delta(q, S) = \{q'\}$ where

$$q' = \{y \in \mathcal{O} \cup \mathcal{A} \mid \sum_{z \in pre(y) \cap (q \cup S)} \mathcal{W}(z, y) \geq 1\};$$

- $F = \{q \in \mathcal{P}(\mathcal{O} \cup \mathcal{A}) \mid x \in q\}$.

The addition of state q^{h} is to obtain a founded regular language: strings accepted by M start with a nonempty input symbol. We define $\mathcal{L}(x)$ as the founded regular language recognized by M.[13]

Next, let $\alpha = (S_1, \ldots, S_n)$ be a string over $\mathcal{P}(\mathcal{I})$. We show that

$$x \in \boldsymbol{B}(\alpha) \Leftrightarrow \alpha \text{ embeds a string } \beta \in \mathcal{L}(x).$$

[13]The construction in this proof does not necessarily result in the smallest founded regular language $\mathcal{L}(x)$. The activation of x is based on seeing patterns embedded in a suffix of the input, but our construction also includes strings in $\mathcal{L}(x)$ that are extensions of such patterns with arbitrary prefixes (starting with a nonempty input symbol).

Direction 1 Suppose $x \in \boldsymbol{B}(\alpha)$. Because no neurons are activated on empty symbols, we can consider the smallest index $k \in \{1, \ldots, n\}$ with $S_k \neq \emptyset$. Let $\beta = (S_k, \ldots, S_n)$. Clearly α embeds β. Note that $\boldsymbol{\mathcal{N}}(\beta) = \boldsymbol{\mathcal{N}}(\alpha)$, implying $x \in \boldsymbol{\mathcal{N}}(\beta)$. When giving β as input to automaton M, we do not enter state q^{h} since β starts with a nonempty input symbol. Subsequently, M faithfully simulates the activated neurons of $\boldsymbol{\mathcal{N}}$. The last state q of M reached in this way, corresponds to the last set of activated neurons of $\boldsymbol{\mathcal{N}}$ on β. Since $x \in \boldsymbol{\mathcal{N}}(\beta)$, we have $q \in F$, causing $\beta \in \mathcal{L}(x)$, as desired.

Direction 2 Suppose α embeds a string $\beta \in \mathcal{L}(x)$. Because $\beta \in \mathcal{L}(x)$, there is an accepting run of M on β, where the last state is an element $q \in \mathcal{P}(\mathcal{O} \cup \mathcal{A})$ with $x \in q$. Since M faithfully simulates $\boldsymbol{\mathcal{N}}$, we have $x \in \boldsymbol{\mathcal{N}}(\beta)$. Because the connection weights of $\boldsymbol{\mathcal{N}}$ are nonnegative, if we would extend β with more activations of input neurons both before and during β, like α does, then at least the neurons would be activated that were activated on just β. Hence, $x \in \boldsymbol{\mathcal{N}}(\alpha)$, as desired.

Remark We did not define $Q = \mathcal{O} \cup \mathcal{A}$ because, when reading an input symbol, the activation of a neuron depends in general on multiple presynaptic auxiliary neurons. That context information might be lost when directly casting neurons as automaton states, because an automaton state is already reached by combining just one predecessor state with a new input symbol. □

The following example demonstrates that an implementation with zero delay is at least achievable for some simple monotone-regular behaviors. In Section 4.4 we will also see more advanced monotone-regular behaviors that can be implemented with zero delay.

Example 4.2. Let $\boldsymbol{B}$ be a monotone-regular behavior over an input set $\mathcal{I}$ and an output set $\mathcal{O}$ with the following assumption: for each $x \in \mathcal{O}$, the founded regular language $\mathcal{L}(x)$ contains just one string. The intuition for $\boldsymbol{B}$, is that a simple chain of auxiliary neurons suffices to recognize increasingly larger prefixes of the single string, and the output neuron listens to the last auxiliary neuron and the last input symbol. There is no delay.

We now define a positive neural network $\boldsymbol{\mathcal{N}} = (\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W})$ to implement $\boldsymbol{B}$ with zero delay. For simplicity we assume $|\mathcal{O}| = 1$, and we denote $\mathcal{O} = \{x\}$; we can repeat the construction below in case of multiple output neurons, and the partial results thus obtained can be placed into one network. Denote $\mathcal{L}(x) = \{(S_1, \ldots, S_n)\}$, where $S_1 \neq \emptyset$. If $n = 1$ then we define $\mathcal{A} = \emptyset$ and, letting $m = |S_1|$, we define $\mathcal{W}(u, x) = 1/m$ for each $u \in S_1$; all other weights are set to zero. We can observe that $\boldsymbol{\mathcal{N}}(\alpha) = \boldsymbol{B}(\alpha)$ for each input string α over $\mathcal{P}(\mathcal{I})$.

Now assume $n \geq 2$. We define $\mathcal{A}$ to consist of the pairwise different neurons $y_1, \ldots, y_{n-1}$, with the assumption $x \notin \mathcal{A}$. Intuitively, neuron y_1 should detect symbol S_1. Next, for each $i \in \{2, \ldots, n-1\}$, neuron y_i is responsible for detecting symbol S_i when the prefix $(S_1, \ldots, S_{i-1})$ is already recognized; this is accomplished by letting y_i also listen to y_{i-1}. We specify weight function $\mathcal{W}$ as follows, where any unspecified weights are assumed to be zero:

- For neuron y_1, letting $m = |S_1|$, we define $\mathcal{W}(u, y_1) = 1/m$ for each $u \in S_1$;
- For neuron y_i with $i \in \{2, \ldots, n-1\}$, letting $m = |S_i| + 1$, we define $\mathcal{W}(u, y_i) = 1/m$ for each $u \in \{y_{i-1}\} \cup S_i$;
- For neuron x, letting $m = |S_n| + 1$, we define $\mathcal{W}(u, x) = 1/m$ for each $u \in \{y_{n-1}\} \cup S_n$.

Also for the case $n \geq 2$, we can observe that $\boldsymbol{\mathcal{N}}(\alpha) = \boldsymbol{B}(\alpha)$ for each input string α over $\mathcal{P}(\mathcal{I})$. □

4.2 Lower Bound

The expressivity lower bound (Theorem 4.5 below) complements the expressivity upper bound (Theorem 4.1). We first introduce some additional terminology and definitions.

4.1 Clean Automata

The construction in the expressivity lower bound is based on translating automata to neural networks. The Lemmas below allow us to make certain technical assumptions on these automata, making the translation to neural networks more natural.

We say that an automaton $M = (Q, \Sigma, \delta, q^s, F)$ contains a *self-loop* if there is a pair $(q, S) \in Q \times \Sigma$ such that $q \in \delta(q, S)$. The following Lemma tells us that self-loops can be removed:

Lemma 4.3. Every regular language recognized by an automaton M_1 is also recognized by an automaton M_2 that *(i)* contains no self-loops, and *(ii)* uses at most double the number of states of M_1.[14]

Proof. Denote $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^s, F_1)$. The idea is to duplicate each state involved in a self-loop, so that looping over the same symbol is still possible but now uses two states. Let V be the set of all states of M_1 involved in a self-loop:

$$V = \{q \in Q_1 \mid \exists S \in \Sigma_1 \text{ with } q \in \delta_1(q, S)\}.$$

Let f be an injective function that maps each $q \in V$ to a new state $f(q)$ outside Q_1. To construct M_2, we use the state set $Q_1 \cup \{f(q) \mid q \in V\}$; the same start state as M_1; and, the accepting state set $F_1 \cup \{f(q) \mid q \in F_1 \cap V\}$. For the new transition function, each pair $(q, S) \in Q_1 \times \Sigma_1$ with $q \in \delta_1(q, S)$ is mapped to $\{f(q)\} \cup (\delta_1(q, S) \setminus \{q\})$, and $(f(q), S')$ is mapped to $\delta_1(q, S')$ for each $S' \in \Sigma_1$.[15] An odd number of repetitions over symbol S is possible because we have copied all outgoing transitions of q to $f(q)$. All other pairs $(q, S) \in Q_1 \times \Sigma_1$ with $q \notin \delta_1(q, S)$ are mapped as before. □

[14] Intuitively, the quantification of the number of states indicates that in general M_2 preserves the nondeterminism of M_1.

[15] If $q \in \delta_1(q, S')$ then we can go back from the new state $f(q)$ to the old state q by reading symbol S'.

For founded regular languages, Lemma 4.4 (below), tells us that the symbol $\emptyset$ does not have to be read from the start state. Intuitively, this last assumption means that activated states of an automaton can be simulated by neurons: the activations of input neurons in the first input symbol can be propagated through the neural network to keep track of any further progress, even if subsequent input symbols are empty.

Lemma 4.4. Letting $\mathcal{I}$ be an input set, every founded regular language over $\mathcal{P}(\mathcal{I})$ recognized by an automaton M_1 is also recognized by an automaton $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^{\mathrm{s}}, F_2)$ where *(i)* $\delta_2(q_2^{\mathrm{s}}, \emptyset) = \emptyset$, and *(ii)* M_2 has the same states as M_1.

Proof. The automaton M_2 is almost exactly the same as M_1, except that the state-symbol combination $(q_2^{\mathrm{s}}, \emptyset)$ is mapped by the transition function to $\emptyset$, i.e., it is impossible to read the empty symbol from the start state. We can immediately see that all accepting runs of M_2 are also accepting runs of M_1 because M_1 includes all transition possibilities of M_2.

For the other direction, towards a contradiction, suppose there is an accepting run $q_1, \ldots, q_{n+1}$ of M_1 on a string $\alpha = (S_1, \ldots, S_n)$ but this run is not an accepting run of M_2. Because in M_2 we have only removed the option to read symbol $\emptyset$ from the start state, there has to be some $i \in \{1, \ldots, n\}$ with $S_i = \emptyset$ and q_i is the start state (of M_1, and M_2). Now, note that the state sequence $q_i, \ldots, q_{n+1}$ is an accepting run of M_1 on the suffix $\beta = (S_i, \ldots, S_n)$. But since $S_i = \emptyset$, automaton M_1 would not recognize a founded regular language, which is a contradiction. $\square$

Let M be as above. A state $q \in Q$ is said to be *reachable* if there is string α over Σ and a run of M on α in which q appears; this run does not have to be accepting. Clearly, every regular language recognized by an automaton M_1 is also recognized by an automaton M_2 that keeps only the reachable states of M_1.

Letting $\mathcal{I}$ be an input set, and letting M be an automaton that recognizes a founded regular language over $\mathcal{P}(\mathcal{I})$, we call M *clean* if

- M contains no self-loops;
- M does not read symbol $\emptyset$ from its start state; and,
- M contains only reachable states.

By applying Lemmas 4.3 and 4.4 in order, any automaton recognizing a founded regular language can be converted to a clean one that recognizes the same language; and, the number of states is at most doubled compared to the original automaton (through Lemma 4.3).

For a clean automaton $M = (Q, \Sigma, \delta, q^{\mathrm{s}}, F)$, we define the *pair set* of M, denoted $p(M)$, as the following set

$$\{(q, S) \in Q \times \Sigma \mid q \neq q^{\mathrm{s}} \text{ and } \exists q' \in Q \text{ with } q \in \delta(q', S)\}.$$

In words: the pair set contains the combinations in M of a non-start state and an incoming symbol to that state.

Now, let $\boldsymbol{B}$ be a monotone-regular behavior over an input set $\mathcal{I}$ and an output set $\mathcal{O}$. An *automaton implementation* for $\boldsymbol{B}$ is a function $\boldsymbol{\mathcal{M}}$ mapping each $x \in \mathcal{O}$ to a clean automaton $\boldsymbol{\mathcal{M}}(x)$ that recognizes a founded regular language $\mathcal{L}(x)$ over $\mathcal{P}(\mathcal{I})$ such that for each input string α over $\mathcal{P}(\mathcal{I})$,

$$x \in \boldsymbol{B}(\alpha) \Leftrightarrow \alpha \text{ embeds a string } \beta \in \mathcal{L}(x).$$

Intuitively, an automaton implementation for $\boldsymbol{B}$ is a prototype implementation that can later be converted to a neural network. The *total pair count* of $\boldsymbol{\mathcal{M}}$, denoted $c(\boldsymbol{\mathcal{M}})$, is defined as

$$c(\boldsymbol{\mathcal{M}}) = \sum_{x \in \mathcal{O}} |p(\boldsymbol{\mathcal{M}}(x))| \,.$$

4.2 Lower Bound Result

Theorem 4.5. Every monotone-regular behavior $\boldsymbol{B}$ can be implemented by a positive neural network with delay 1. In particular, each automaton implementation $\boldsymbol{\mathcal{M}}$ for $\boldsymbol{B}$ can be converted to a positive neural network that implements $\boldsymbol{B}$ with delay 1 and that has $c(\boldsymbol{\mathcal{M}})$ auxiliary neurons.[16]

Proof. Let $\boldsymbol{B}$ be a monotone-regular behavior over an input set $\mathcal{I}$ and an output set $\mathcal{O}$. Let $\boldsymbol{\mathcal{M}}$ be an automaton implementation for $\boldsymbol{B}$. For each output neuron x, we translate automaton $\boldsymbol{\mathcal{M}}(x)$ to a neural network. Roughly speaking, we translate state-symbol pairs of the automaton to neurons. A novel aspect, is that each input symbol in our model consists of multiple input neurons. For this reason, our simulation of an automaton state by a neuron uses a nontrivial definition of presynaptic weights allowing us to simultaneously express *(i)* an "or" over auxiliary neurons that provide working memory, and *(ii)* an "and" over all input neurons mentioned in an input symbol. We use only rational weights. There is a delay of one time unit in the construction because the output neuron x listens to neurons that simulate accept states of $\boldsymbol{\mathcal{M}}(x)$.[17] See also the later Remark 4.8. The construction below is illustrated in Example 4.7.

For simplicity, we assume $|\mathcal{O}| = 1$, and we denote $\mathcal{O} = \{x\}$; for the case of multiple output neurons, the construction given below can be repeated, and the neural networks thus obtained can be united to form the overall desired network. Let $M = \boldsymbol{\mathcal{M}}(x)$ and denote $M = (Q, \Sigma, \delta, q^{\mathrm{s}}, F)$ where $\Sigma = \mathcal{P}(\mathcal{I})$. Recall that M is clean.

Positive neural network We now incrementally define the desired positive neural network $\boldsymbol{\mathcal{N}} = (\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W})$ to implement $\boldsymbol{B}$ with delay 1.

[16] Intuitively, the number of auxiliary neurons indicates that in general the constructed neural network preserves the nondeterminism, and thus the parallelism, of the automata in $\boldsymbol{\mathcal{M}}$.

[17] An automaton itself does not introduce delay on string acceptance. In the construction of a neural network, however, all the different accept states should essentially be tunneled through a single output neuron. This requires in general a delay of one time unit (cf. Section 4.3).

Auxiliary neurons First, we define the set of auxiliary neurons:

$$\mathcal{A} = p(M),$$

where $p(M)$ is the pair set of M as defined above. Intuitively, an auxiliary neuron (q, S), where always $q \neq q^{\mathrm{s}}$, represents the automaton state q reached by reading input symbol S from some previous state. We define the set $\mathcal{T} \subseteq \mathcal{A}$ of *trigger neurons*:

$$\mathcal{T} = \{(q, S) \in \mathcal{A} \mid q \in \delta(q^{\mathrm{s}}, S)\}.$$

Intuitively, the neurons in $\mathcal{T}$ are the first auxiliary neurons that become activated by the input; these neurons simulate the event of reading an input symbol from the start state of automaton M. Note that for each $(q, S) \in \mathcal{T}$ we have $S \neq \emptyset$ because $\delta(q^{\mathrm{s}}, \emptyset) = \emptyset$ by assumption on M.

For each $(q, S) \in \mathcal{A} \setminus \mathcal{T}$, we define the set $con(q, S)$ of *context* neurons of (q, S) as follows:

$$con(q, S) = \{(q', S') \in \mathcal{A} \mid q \in \delta(q', S)\}.$$

Intuitively, $con(q, S)$ is the set of auxiliary neurons that recognize prefixes of the strings that neuron (q, S) should recognize, i.e., $con(q, S)$ is the working memory from the viewpoint of (q, S). In the definition of $con(q, S)$, there is no relationship between the symbols S and S'. Note that for each $(q, S) \in \mathcal{A} \setminus \mathcal{T}$, the set $con(q, S)$ is always nonempty because M contains only reachable states.[18]

Weights The design of the connection weights is an intricate part of the construction. For this reason, we spend sufficient attention to the underlying design process. Suppose we have an auxiliary neuron $y = (q, S)$ that should listen to a context $con(q, S) = \{z_1, \ldots, z_m\}$ of auxiliary neurons and to the input symbol $S = \{u_1, \ldots, u_n\}$. We desire weights w_1 and w_2, where w_1 is assigned to each connection (z_i, y) with $i \in \{1, \ldots, m\}$ and w_2 to each connection (u_j, y) with $j \in \{1, \ldots, n\}$, such that the following three properties are satisfied: *(i)* y is not activated if all of $\{z_1, \ldots, z_m\}$ are activated but not yet all of $\{u_1, \ldots, u_n\}$; *(ii)* y is already activated if at least one $z \in \{z_1, \ldots, z_m\}$ is activated while all of $\{u_1, \ldots, u_n\}$ are activated; and, *(iii)* y is not activated if only all of $\{u_1, \ldots, u_n\}$ are activated. This assignment of weights corresponds to the earlier announced "or" and "and", over $\{z_1, \ldots, z_m\}$ and $\{u_1, \ldots, u_n\}$ respectively.

The above desired properties *(i)*, *(ii)*, and *(iii)* are satisfied by the following weight functions w_1 and w_2 that are parameterized by the set cardinalities m and n, denoting $\mathbb{N}_0 = \mathbb{N} \setminus \{0\}$,

$$\begin{aligned} w_1 : \mathbb{N}_0 \times \mathbb{N}_0 \to [0, 1] : &\quad w_1(m, n) = 1/(n \cdot m + 1), \\ w_2 : \mathbb{N}_0 \times \mathbb{N}_0 \to [0, 1] : &\quad w_2(m, n) = m/(n \cdot m + 1). \end{aligned}$$

The design of these functions is documented in Appendix A. The satisfaction of the desired properties is now formalized by the following observations:

[18] Indeed, since $(q, S) \in \mathcal{A}$, there is a reachable state $q' \in Q$ with $q \in \delta(q', S)$. But $(q, S) \notin \mathcal{T}$ implies $q' \neq q^{\mathrm{s}}$, causing $(q', S') \in \mathcal{A}$ for some $S' \in \mathcal{P}(\mathcal{I})$. Hence, $(q', S') \in con(q, S)$.

Claim 4.6. Letting $m, n \in \mathbb{N}_0$,

- $m \cdot w_1(m,n) + (n-1) \cdot w_2(m,n) < 1$;
- $w_1(m,n) + n \cdot w_2(m,n) \geq 1$;
- $n \cdot w_2(m,n) < 1$.

Next, we can define the weights for all connections. We define the weight function $\mathcal{W}$ from the perspective of the neurons in $\mathcal{A} \cup \{x\}$, where any unmentioned weights are assumed to be zero:

- for the output neuron x, and each $(q,S) \in \mathcal{A}$ where q is an accepting state of M (i.e., $q \in F$), we define

$$\mathcal{W}((q,S),x) = 1;$$

- for each $(q,S) \in \mathcal{T}$ and each $y \in S$, letting $n = |S|$, we define

$$\mathcal{W}(y,(q,S)) = 1/n;$$

- for each $(q,S) \in \mathcal{A} \setminus \mathcal{T}$ with $S = \emptyset$, for each $y \in \mathit{con}(q,S)$, we define

$$\mathcal{W}(y,(q,S)) = 1;$$

- for each $(q,S) \in \mathcal{A} \setminus \mathcal{T}$ with $S \neq \emptyset$, letting $m = |\mathit{con}(q,S)|$ and $n = |S|$, for each $y \in \mathit{con}(q,S)$, we define

$$\mathcal{W}(y,(q,S)) = w_1(m,n),$$

and for each $z \in S$, we define

$$\mathcal{W}(z,(q,S)) = w_2(m,n);$$

note in this case that $m > 0$ and $n > 0$.

Intuitively, the role of neurons $(q,S) \in \mathcal{A} \setminus \mathcal{T}$ with $S = \emptyset$ is to propagate past memories forward in time, without requiring new activations of any input neurons.

Correctness We show that $\boldsymbol{\mathcal{N}}$ implements $\boldsymbol{B}$ with a delay of one time unit. Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ be an input string over $\mathcal{P}(\mathcal{I})$. If $n = 1$ then $\boldsymbol{\mathcal{N}}(\alpha) = \emptyset$, as desired, because the output neuron x only listens to auxiliary neurons (that represent accepting states), which makes it impossible for x to become activated on a string with just one symbol. Henceforth, suppose $n \geq 2$. We show that $\boldsymbol{\mathcal{N}}(\alpha) = \boldsymbol{B}(\alpha_{\to n-1})$.

Direction 1 Suppose $x \in \boldsymbol{\mathcal{N}}(\alpha)$. The activation of x means that there is a maximal chain of auxiliary neurons

$$(q_1, S_1), \ldots, (q_k, S_k),$$

that becomes activated when showing α to $\boldsymbol{\mathcal{N}}$ (with $k \geq 1$), where (q_1, S_1) is a trigger neuron; (q_2, S_2), $\ldots$, (q_k, S_k) are non-trigger auxiliary neurons; (q_{i-1}, S_{i-1}) is a presynaptic neuron of (q_i, S_i) for each $i \in \{2, \ldots, k\}$; and, (q_k, S_k) has activated x while the last input symbol α_n was shown. Let $\beta = (S_1, \ldots, S_k)$. By design of the presynaptic weights of the auxiliary neurons (cf. Claim 4.6), we know that the symbols S_1, $\ldots$, S_k effectively occur in α, and more particularly that $\alpha_{\to n-1}$ embeds β. Next, we show that M accepts β. Then, since $\boldsymbol{B}$ is monotone-regular, the embedding of β into $\alpha_{\to n-1}$ implies $x \in \boldsymbol{B}(\alpha_{\to n-1})$.

Based on the above sequence of auxiliary neurons, the state sequence $q^{\mathrm{s}}, q_1, \ldots, q_k$ forms an accepting run of M on β:

- $q_1 \in \delta(q^{\mathrm{s}}, S_1)$ because (q_1, S_1) is a trigger neuron;
- for each $i \in \{2, \ldots, k\}$, we have $q_i \in \delta(q_{i-1}, S_i)$ because (q_{i-1}, S_{i-1}) is a presynaptic neuron of (q_i, S_i);[19]
- q_k must be an accepting state, because we assumed that neuron (q_k, S_k) has activated x.

Direction 2 Suppose $x \in \boldsymbol{B}(\alpha_{\to n-1})$. Because $\boldsymbol{B}$ is monotone-regular, $\alpha_{\to n-1}$ embeds a string β that is accepted by M. Denote $\beta = (S_1, \ldots, S_k)$. We consider an accepting run $q^{\mathrm{s}}, q_1, \ldots, q_k$ of M on β. The string β can be chosen so that $q^{\mathrm{s}} \notin \{q_1, \ldots, q_k\}$.[20] We now consider the following sequence of auxiliary neurons: $(q_1, S_1), \ldots, (q_k, S_k)$.[21] We show that this sequence of auxiliary neurons becomes active in the last k steps of $\boldsymbol{\mathcal{N}}$ on input $\alpha_{\to n-1}$. Let $N_1, \ldots, N_n$ be the sequence of sets of activated neurons while running $\boldsymbol{\mathcal{N}}$ on input $\alpha_{\to n-1}$, where $N_1 = \emptyset$. We show (by induction) for each $i \in \{1, \ldots, k\}$ that $(q_i, S_i) \in N_{n-k+i}$. This results in $(q_k, S_k) \in N_n$, and because (q_k, S_k) simulates an accepting state, on the full string α we thus obtain $x \in \boldsymbol{\mathcal{N}}(\alpha)$, as desired.

Before we continue, note that the embedding of β into $\alpha_{\to n-1}$ concretely means $S_i \subseteq \alpha_{n-1-k+i}$ for each $i \in \{1, \ldots, k\}$. For the base case, we see that (q_1, S_1) is a trigger neuron because $q_1 \in \delta(q^{\mathrm{s}}, S_1)$. So, $S_1 \subseteq \alpha_{n-k}$ implies $(q_1, S_1) \in N_{n-k+1}$.

For the inductive step, we assume $(q_{i-1}, S_{i-1}) \in N_{n-k+i-1}$ where $i \in \{2, \ldots, k\}$. We show that $(q_i, S_i) \in N_{n-k+i}$. If (q_i, S_i) is a trigger neuron then a similar

[19] From the definition of presynaptic neuron, we know that the connection from (q_{i-1}, S_{i-1}) to (q_i, S_i) has a strictly positive weight. This weight could only have been defined if $(q_{i-1}, S_{i-1}) \in \mathit{con}(q_i, S_i)$.

[20] If $q^{\mathrm{s}} = q_i$ for some $i \in \{1, \ldots, k\}$ then $q_i, q_{i+1}, \ldots, q_k$ is an accepting run on the suffix $\beta' = (S_{i+1}, \ldots, S_k)$, and we could instead focus on the smaller string β' that is also embedded into $\alpha_{\to n-1}$.

[21] These are valid auxiliary neurons because *(i)* $q^{\mathrm{s}} \notin \{q_1, \ldots, q_k\}$ by assumption; and,

(ii) because $q^{\mathrm{s}}, q_1, \ldots, q_k$ is an accepting run, we have $q_1 \in \delta(q^{\mathrm{s}}, S_1)$ and $q_i \in \delta(q_{i-1}, S_i)$ for each $i \in \{2, \ldots, k\}$.

reasoning applies as in the base case, using that $S_i \subseteq \alpha_{n-1-k+i}$. If (q_i, S_i) is not a trigger neuron then $(q_{i-1}, S_{i-1}) \in con(q_i, S_i)$ because $q_i \in \delta(q_{i-1}, S_i)$ and $q_{i-1} \neq q^s$, and we distinguish between the following two cases:

- Suppose $S_i = \emptyset$. Then the connection weight from (q_{i-1}, S_{i-1}) to (q_i, S_i) was set to 1, and the activation $(q_{i-1}, S_{i-1}) \in N_{n-k+i-1}$ implies the activation $(q_i, S_i) \in N_{n-k+i}$.
- Suppose $S_i \neq \emptyset$. In that case, the presynaptic weight design of (q_i, S_i) with functions w_1 and w_2 (cf. Claim 4.6), applied to the presynaptic activations $(q_{i-1}, S_{i-1}) \in N_{n-k+i-1}$ and $S_i \subseteq \alpha_{n-1-k+i} = \alpha_{n-k+i-1}$, gives the activation $(q_i, S_i) \in N_{n-k+i}$.

□

Example 4.7. We illustrate the construction of the proof of Theorem 4.5. Let $\mathcal{I}$ consist of four distinct input neurons a, b, c, and d. Let $\mathcal{O} = \{x\}$. We define the following input symbols: $S_1 = \{a, b, c\}$, $S_2 = \{b, c\}$, and $S_3 = \{a, d\}$.

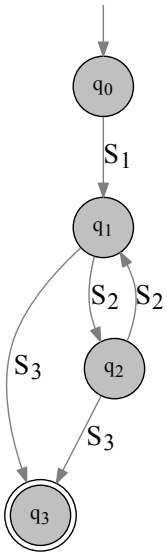
Consider the clean automaton M depicted in Figure 1a, that recognizes a founded regular language over $\mathcal{P}(\mathcal{I})$; we denote this language as $\mathcal{L}(x)$.[22] Language $\mathcal{L}(x)$ is infinite because of the loop between states q_1 and q_2 over symbol S_2. In particular, $\mathcal{L}(x)$ contains all strings of the form (S_1, S_2^*, S_3), where S_2^* denotes an arbitrary number of repetitions of symbol S_2. Let $\boldsymbol{B}$ be the monotone-regular behavior over $\mathcal{I}$ and $\mathcal{O}$ defined by $\mathcal{L}(x)$: for each input string α over $\mathcal{P}(\mathcal{I})$,

$$x \in \boldsymbol{B}(\alpha) \Leftrightarrow \alpha \text{ embeds a string } \beta \in \mathcal{L}(x).$$

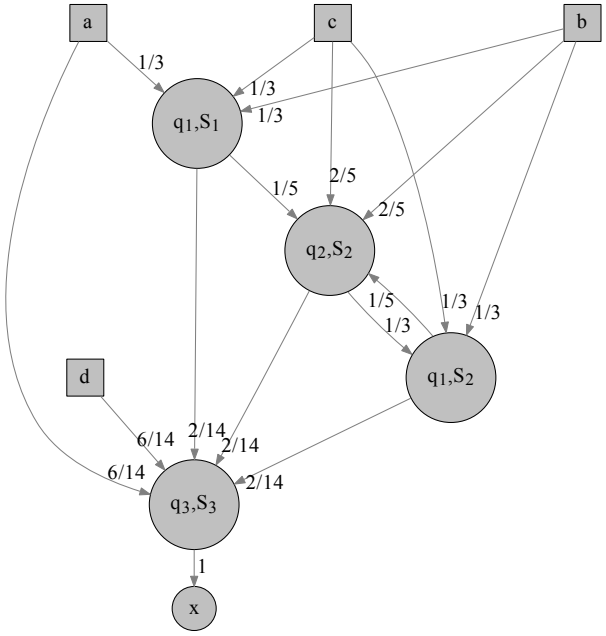
Applying the transformation in the proof of Theorem 4.5 to automaton M results in the positive neural network $\boldsymbol{\mathcal{N}}$ depicted in Figure 1b, where input neurons are indicated by boxes and the nonzero (rational) edge weights are written at the end of a connection. Auxiliary neuron (q_1, S_1) is the only trigger neuron; it listens for symbol S_1. Note that the loop between states q_1 and q_2 of M is preserved as a loop between the auxiliary neurons (q_1, S_2) and (q_2, S_2). We can also see, for example, that the neuron (q_3, S_3) is only activated at time $t \in \mathbb{N}$ when at time $t-1$ both input neurons a and d are active and at least one of the auxiliary neurons (q_1, S_1), (q_2, S_2), and (q_1, S_2); these auxiliary neurons may be viewed as working memory, representing the recognition of prefixes of the desired strings. □

Remark 4.8. In the proof of Theorem 4.5, it is possible to replace the or-and construction of weight functions w_1 and w_2 by a two-stage process, at the cost of an additional delay of one time unit. If we ignore this additional delay, the resulting construction is similar to the one described by Šíma and Wiedermann (1998) in their Theorem 4.1, for the setting with one input neuron, with the difference that we only use positive weights and are thus expressing monotone-regular behaviors.

[22] In Figure 1a, we use the standard notations (Hopcroft and Ullman, 1979; Sipser, 2006): the start state has an entering arrow with no source, and accepting states are indicated with double circles.



(a) A clean automaton recognizing a founded regular language. The input neurons are a, b, c, and d; the considered input symbols are $S_1 = \{a, b, c\}$, $S_2 = \{b, c\}$, and $S_3 = \{a, d\}$.



(b) The positive neural network obtained from the automaton in Figure 1a. The boxes represent the input neurons a, b, c, and d. The output neuron is x. The remaining neurons are auxiliary.

Figure 1: The automaton and positive neural network of Example 4.7.

Concretely, for each symbol $S \in \mathcal{P}(\mathcal{I})$ used by the automaton, with $S \neq \emptyset$, we introduce a *preprocessor* neuron y_S having the following presynaptic weight for each $u \in S$, where $n = |S|$:

$$\mathcal{W}(u, y_S) = 1/n.$$

So, neuron y_S will only be activated when all neurons of S are activated. Next, each auxiliary neuron $(q, S) \in \mathcal{A}$ with $S \neq \emptyset$ is configured to read the preprocessor neuron y_S instead of the input neurons in S directly:

- if $(q, S) \in \mathcal{T}$ then we define $\mathcal{W}(y_S, (q, S)) = 1$;
- if $(q, S) \in \mathcal{A} \setminus \mathcal{T}$ with $S = \emptyset$ then for each $z \in \mathit{con}(q, S)$ we define $\mathcal{W}(z, (q, S)) = 1$ as before;
- if $(q, S) \in \mathcal{A} \setminus \mathcal{T}$ with $S \neq \emptyset$, letting $m = |\mathit{con}(q, S)|$, we define

  $$\mathcal{W}(y_S, (q, S)) = m/(m + 1),$$

  and for each $z \in \mathit{con}(q, S)$,

  $$\mathcal{W}(z, (q, S)) = 1/(m + 1).$$

The total implementation delay now becomes two time units: *(i)* trigger neurons listen to the above preprocessor neurons, and *(ii)* the output neurons listen to auxiliary neurons that simulate accept states as before. We should point out, however, that the construction by Šíma and Wiedermann (1998) only incurs a delay of one time unit because in their setting there is only one input neuron; so, in that setting, all the above preprocessor neurons can be conceptually merged into the single input neuron. □

4.3 Separation

Regarding the expressivity of positive neural networks, the upper bound (Theorem 4.1) and the lower bound (Theorem 4.5) do not coincide. Indeed, as illustrated by the following two examples, there are simple monotone-regular behaviors that can not be implemented with zero delay. The main intuition in these examples, is that the fast reaction speed demanded by zero delay forces too much responsibility on the output neuron, causing this neuron to be erroneously activated. Each example illustrates a different kind of error.

Example 4.9. Let S_1 and S_2 be two disjoint sets of neurons with $|S_1| \geq 2$ and $|S_2| \geq 2$. Let $\mathcal{I} = S_1 \cup S_2$ and $\mathcal{O} = \{x\}$. Let $\mathcal{L}(x)$ be the following founded regular language over $\mathcal{P}(\mathcal{I})$:

$$\mathcal{L}(x) = \{(S_1), (S_2)\}.$$

So, $\mathcal{L}(x)$ is a finite language containing two one-symbol strings.[23] Let $\boldsymbol{B}$ be the following monotone-regular behavior over $\mathcal{I}$ and $\mathcal{O}$ defined by $\mathcal{L}(x)$: for each input

[23] An automaton recognizing $\mathcal{L}(x)$ could have two accepting states q_1 and q_2 besides the start state q^{s}: reading symbol S_i from q^{s} leads to q_i for $i \in \{1, 2\}$.

string α over $\mathcal{P}(\mathcal{I})$, we define

$$\boldsymbol{B}(\alpha) = \begin{cases} \{x\} & \text{if } \alpha \text{ embeds a string } \beta \in \mathcal{L}(x); \\ \emptyset & \text{otherwise.} \end{cases}$$

We show that there is no positive neural network that implements $\boldsymbol{B}$ with zero delay. Towards a contradiction, suppose there is such a neural network $\boldsymbol{\mathcal{N}}$. We show that the connections from S_1 to x and the connections from S_2 to x interfere with each other, causing x to also be triggered on wrong input symbols.

Because $\boldsymbol{\mathcal{N}}$ implements $\boldsymbol{B}$ with zero delay, we have $\boldsymbol{\mathcal{N}}(\alpha) = \boldsymbol{B}(\alpha)$ for all input strings α over $\mathcal{P}(\mathcal{I})$. In particular, $\boldsymbol{\mathcal{N}}((S_1)) = \{x\}$ and $\boldsymbol{\mathcal{N}}((S_2)) = \{x\}$. These fast output reactions imply that neuron x does not rely on auxiliary neurons, and instead reads input neurons directly. So,

$$\sum_{u \in S_1} \mathcal{W}(u, x) \geq 1, \text{ and} \tag{4.1}$$

$$\sum_{u \in S_2} \mathcal{W}(u, x) \geq 1. \tag{4.2}$$

We distinguish between the following cases:

- Suppose there exist some $y \in S_1$ and $z \in S_2$ such that

  $$\mathcal{W}(y, x) + \mathcal{W}(z, x) \geq 1.$$

  Define the symbol $S = \{y, z\}$. Note that $S \in \mathcal{P}(\mathcal{I})$. Because $|S_1| \geq 2$ and $|S_2| \geq 2$, we have $S_1 \not\subseteq S$ and $S_2 \not\subseteq S$. Please note that by choice of y and z,

  $$\sum_{u \in S} \mathcal{W}(u, x) \geq 1.$$

  So, $\boldsymbol{\mathcal{N}}((S)) = \{x\}$. But the string (S) does not embed a string from $\mathcal{L}(x)$, giving $\boldsymbol{B}((S)) = \emptyset$. Hence, $\boldsymbol{\mathcal{N}}((S)) \neq \boldsymbol{B}((S))$, which is a contradiction.

- If the first case does not hold, then we can choose some $y \in S_1$ and $z \in S_2$ for which

  $$\mathcal{W}(y, x) + \mathcal{W}(z, x) < 1.$$

  Define the symbol $S = \mathcal{I} \setminus \{y, z\}$. Note that $S \in \mathcal{P}(\mathcal{I})$. Because $y \in S_1$ and $z \in S_2$, we have $S_1 \not\subseteq S$ and $S_2 \not\subseteq S$. Moreover,

  $$\sum_{u \in S} \mathcal{W}(u, x) = \sum_{u \in S_1} \mathcal{W}(u, x) + \sum_{u \in S_2} \mathcal{W}(u, x) - \mathcal{W}(y, x) - \mathcal{W}(z, x).$$

  By using inequalities (4.1) and (4.2) from above, and $\mathcal{W}(y, x) + \mathcal{W}(z, x) < 1$, we can further obtain:

  $$\begin{aligned} \sum_{u \in S} \mathcal{W}(u, x) &\geq 2 - (\mathcal{W}(y, x) + \mathcal{W}(z, x)) \\ &> 1. \end{aligned}$$

  So, $\boldsymbol{\mathcal{N}}((S)) = \{x\}$. But the string (S) does not embed a string from $\mathcal{L}(x)$, giving $\boldsymbol{B}((S)) = \emptyset$. Again, $\boldsymbol{\mathcal{N}}((S)) \neq \boldsymbol{B}((S))$, which is a contradiction.

□

Example 4.10. Let S_1, S_2, S_3, and S_4 be nonempty sets of neurons that are pairwise disjoint. Let $\mathcal{I} = \bigcup_{i=1}^{4} S_i$ and $\mathcal{O} = \{x\}$. Let $\mathcal{L}(x)$ be the following founded regular language over $\mathcal{P}(\mathcal{I})$:[24]

$$\mathcal{L}(x) = \{(S_1, S_2), (S_3, S_4)\}.$$

Let $\boldsymbol{B}$ be the monotone-regular behavior over $\mathcal{I}$ and $\mathcal{O}$ defined by $\mathcal{L}(x)$: for each input string α over $\mathcal{P}(\mathcal{I})$,

$$\boldsymbol{B}(\alpha) = \begin{cases} \{x\} & \text{if } \alpha \text{ embeds a string } \beta \in \mathcal{L}(x); \\ \emptyset & \text{otherwise.} \end{cases}$$

We show there is no positive neural network that implements $\boldsymbol{B}$ with zero delay. Towards a contradiction, suppose there is such a network $\boldsymbol{\mathcal{N}} = (\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W})$. We show that $\boldsymbol{\mathcal{N}}$ erroneously activates the output neuron on the input string (S_1, S_4) or on the input string (S_3, S_2). Intuitively, the output neuron x confuses the memory contexts emerging from symbols S_1 and S_3.

Because $\boldsymbol{\mathcal{N}}$ implements $\boldsymbol{B}$ with zero delay, we have $\boldsymbol{\mathcal{N}}(\alpha) = \boldsymbol{B}(\alpha)$ for all input strings α over $\mathcal{P}(\mathcal{I})$. In particular, $\boldsymbol{\mathcal{N}}((S_1, S_2)) = \{x\}$ and $\boldsymbol{\mathcal{N}}((S_3, S_4)) = \{x\}$. Let $\mathcal{A}_1 \subseteq \mathcal{A}$ denote the set of auxiliary neurons activated after reading the string (S_1). Similarly, let $\mathcal{A}_3 \subseteq \mathcal{A}$ denote the set of auxiliary neurons activated after reading the string (S_3). Denote, for $i \in \{1, 3\}$,

$$w_i = \sum_{y \in \mathcal{A}_i} \mathcal{W}(y, x).$$

Also denote, for $i \in \{2, 4\}$,

$$w_i = \sum_{y \in S_i} \mathcal{W}(y, x).$$

Now, the output activations $\boldsymbol{\mathcal{N}}((S_1, S_2)) = \{x\}$ and $\boldsymbol{\mathcal{N}}((S_3, S_4)) = \{x\}$ imply

$$w_1 + w_2 \geq 1, \text{ and}$$
$$w_3 + w_4 \geq 1.$$

We distinguish between the following cases:[25]

- Suppose $w_1 + w_4 \geq 1$. This implies $\boldsymbol{\mathcal{N}}((S_1, S_4)) = \{x\}$. But then $\boldsymbol{\mathcal{N}}((S_1, S_4)) \neq \boldsymbol{B}((S_1, S_4))$, which is a contradiction.
- In the other case, we have $w_1 + w_4 < 1$. Together with $w_3 + w_4 \geq 1$ from above, we see that $w_3 > w_1$. Combining $w_3 > w_1$ and $w_1 + w_2 \geq 1$ from above, we obtain $w_3 + w_2 \geq 1$. This implies $\boldsymbol{\mathcal{N}}((S_3, S_2)) = \{x\}$. But then $\boldsymbol{\mathcal{N}}((S_3, S_2)) \neq \boldsymbol{B}((S_3, S_2))$, which is a contradiction.

□

[24] An automaton recognizing this language could splits its computation into two branches from the start state: one branch recognizes the string (S_1, S_2) and the other branch recognizes the string (S_3, S_4).

[25] Although $\boldsymbol{\mathcal{N}}((S_1, S_2)) = \{x\}$ and $\boldsymbol{\mathcal{N}}((S_2)) = \boldsymbol{B}((S_2)) = \emptyset$ imply that $w_1 > 0$, the proof does not really use this fact. Similarly, $w_3 > 0$, but the proof does not use this fact.

4.4 On Zero Delay

The earlier Example 4.2 has provided a zero delay implementation for monotone-regular behaviors whose underlying founded regular language contains only one string. Here, we present a larger class of monotone-regular behaviors that can be implemented with zero delay. First, we call a regular language $\mathcal{L}$ *converging* if all strings in $\mathcal{L}$ end with the same symbol. The following result demonstrates that even monotone-regular behaviors whose underlying founded regular languages are infinite can sometimes be implemented with zero delay:

Theorem 4.11. Every monotone-regular behavior where the founded regular language of each output neuron is also converging, can be implemented by a positive neural network with zero delay.

Proof. Let $\boldsymbol{B}$ be a monotone-regular behavior over an input set $\mathcal{I}$ and an output set $\mathcal{O}$ where the founded regular language of each output neuron is also converging. Let $\boldsymbol{\mathcal{M}}$ be an automaton implementation for $\boldsymbol{B}$. As in the proof of Theorem 4.5, we fix some $x \in \mathcal{O}$. Let $\mathcal{L}(x)$ be the language recognized by $\boldsymbol{\mathcal{M}}(x)$. Denote $\boldsymbol{\mathcal{M}}(x) = (Q, \Sigma, \delta, q^{\mathrm{s}}, F)$, where $\Sigma = \mathcal{P}(\mathcal{I})$. We can modify the construction in the proof of Theorem 4.5 as follows.

First, we define the set V of all state-symbol combinations that lead to an accepting state:

$$V = \{(q, S) \in Q \times \mathcal{P}(\mathcal{I}) \mid \delta(q, S) \cap F \neq \emptyset\}.$$

Because $\mathcal{L}(x)$ is converging, there is one symbol $S \in \mathcal{P}(\mathcal{I})$ such that $S = S_i$ for each $(q_i, S_i) \in V$.[26] We refer to S as the *terminal* symbol. The only difference compared to the proof of Theorem 4.5, is that we now let output neuron x listen to *(i)* the symbol S directly and *(ii)* a different set C of auxiliary neurons. Letting $\mathcal{A}$ be the set of auxiliary neurons as defined in the proof of Theorem 4.5, we define

$$C = \{(q, S') \in \mathcal{A} \mid (q, S) \in V\}.$$

We now specify the presynaptic weights for x, depending on symbol S:

- Suppose $S = \emptyset$. We still have $C \neq \emptyset$: there is always a string $\alpha \in \mathcal{L}(x)$ ending with S, for which there is an accepting run $q_1, \ldots, q_n, q_{n+1}$ where $q_{n+1} \in \delta(q_n, S) \cap F$; and, $q_n \neq q^{\mathrm{s}}$ because $\boldsymbol{\mathcal{M}}(x)$ does not read $S = \emptyset$ from its start state, implying $(q_n, S') \in C$ for some $S' \in \mathcal{P}(\mathcal{I})$. Now, for each $y \in C$, we define
$$\mathcal{W}(y, x) = 1.$$
- Suppose $S \neq \emptyset$. If $C = \emptyset$ then x only has to detect symbol S; accordingly, letting $n = |S|$, for each $z \in S$, we define
$$\mathcal{W}(z, x) = 1/n.$$

[26] For each $(q_i, S_i) \in V$, there is an input string α over $\mathcal{P}(\mathcal{I})$ and a run of $\boldsymbol{\mathcal{M}}(x)$ on α ending with q_i because q_i is a reachable state by assumption on $\boldsymbol{\mathcal{M}}(x)$. Since $(q_i, S_i) \in V$, the extension of α with S_i belongs to $\mathcal{L}(x)$. So, for any $(q_1, S_1) \in V$ and $(q_2, S_2) \in V$, there are strings in $\mathcal{L}(x)$ ending with S_1 and S_2; but convergence of $\mathcal{L}(x)$ implies $S_1 = S_2$.

If $C \neq \emptyset$, then we reuse the or-and construction with weight functions w_1 and w_2; concretely, letting $m = |C|$ and $n = |S|$, for $y \in C$, we define

$$\mathcal{W}(y, x) = w_1(m, n),$$

and for each $z \in S$, we define

$$\mathcal{W}(z, x) = w_2(m, n).$$

All other connections from auxiliary neurons to x are set to zero. So, instead of listening to auxiliary neurons that simulate accept states, the output neuron x *(i)* listens to auxiliary neurons that simulate the states preceding accept states, and *(ii)* also verifies that the terminal symbol S effectively occurs. □

The following example demonstrates that the converse of Theorem 4.11 does not hold, so we do not yet have a precise characterization of the monotone-regular behaviors that can be implemented with zero delay.

Example 4.12. Let $S_1 = \{a, b\}$ and $S_2 = \{b, c\}$ where a, b, and c are pairwise different neurons. Let $\mathcal{I} = S_1 \cup S_2$ and $\mathcal{O} = \{x\}$. Let $\mathcal{L}(x)$ be the following founded regular language over $\mathcal{P}(\mathcal{I})$:

$$\mathcal{L}(x) = \{(S_1), (S_2)\}.$$

Note that $\mathcal{L}(x)$ is not converging. Let $\boldsymbol{B}$ be the monotone-regular behavior over $\mathcal{I}$ and $\mathcal{O}$ defined by $\mathcal{L}(x)$: for each input string α over $\mathcal{P}(\mathcal{I})$,

$$\boldsymbol{B}(\alpha) = \begin{cases} \{x\} & \text{if } \alpha \text{ embeds a string } \beta \in \mathcal{L}(x); \\ \emptyset & \text{otherwise.} \end{cases}$$

The following positive neural network $\boldsymbol{\mathcal{N}} = (\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W})$ implements $\boldsymbol{B}$ with zero delay: $\mathcal{A} = \emptyset$, and

$$\begin{aligned} \mathcal{W}(a, x) &= 1/3, \\ \mathcal{W}(b, x) &= 2/3, \\ \mathcal{W}(c, x) &= 1/3. \end{aligned}$$

In contrast to Example 4.9, we can not fool this network to trigger x on a wrong input symbol like $\{a, c\}$. That is because $\mathcal{W}$ assigns a heavier weight to connection (b, x), which renders the input neuron b crucial for the activation of x. □

5 Conclusion and Future Work

We have studied the expressivity of positive neural networks with multiple input neurons. Within the framework of monotone-regular behaviors, we have suggested both an upper and lower bound on the expressivity. These bounds do not coincide when we take into account the delay by which a behavior is implemented. We now discuss several avenues for further work.

Single input neurons If there is only a single input neuron, Šíma and Wiedermann (1998) show that all regular languages can be recognized by a neural network with a delay of one time unit. Our article has shown a similar result for monotone-regular behaviors, but in the case of multiple input neurons. It might be interesting to better understand the relationship between these results.

Symbols over multiple input neurons could be translated to a single input neuron as follows: supposing there are n ordered input neurons, each subset of input neurons can be represented as a binary code over n bits. This way, each sequence of input symbols can be translated to a sequence of binary codes, and the resulting sequence may be viewed as a single bit string. However, this construction would increase output delay. Moreover, it is not clear if this technical construction can be achieved inside a positive neural network itself, because on every time step an entirely new symbol arrives over the multiple input neurons; the positive neural network might not be able to buffer the new symbols while it is translating the previous symbols.

Characterizing zero delay We have seen that seemingly simple monotone-regular behaviors already require a delay of one time unit (Section 4.3). We have also made some first steps towards identifying the class of monotone-regular behaviors that can be implemented with zero delay (Section 4.4). However, a precise characterization is missing. Example 4.12 suggests that in case of multiple terminal symbols in the underlying regular languages, we could seek for an assignment of nonuniform weights to the input neurons. Perhaps the existence of such nonuniform weights can be related to the syntactical properties of the accompanying automata.

Minimal network size Like previous complexity-theoretic analyses of neural networks (Šíma and Orponen, 2003), one could examine what minimal number of auxiliary neurons is necessary for implementing certain monotone-regular behaviors. Note that a lower bound on the number of states in an automaton implementation of a behavior does not directly provide a lower bound on the number of neurons, because clever design of the weights could perhaps pack more functionality into fewer neurons than the number of automaton states (or symbol-state combinations). Such efficient implementations were previously studied, e.g. by Horne and Hush (1996) for the simulation of deterministic automata by recurrent neural networks. For positive neural networks, it might be possible to explore the relationship with (monotone) AND-OR boolean circuits, where Alon and Boppana (1987) have previously obtained lower bounds on the number of gates (neurons) for implementing certain boolean functions.

We should note, however, that some of the existing constructions, e.g. (Horne and Hush, 1996) introduce delays in which the overall neural network would process incoming input symbols. To compare such constructions with the results regarding delay in this article, perhaps some of the constructed sub-circuits could be viewed as being computed instantaneously, and would thus not contribute to the overall delay.

Inhibition Previous works on the expressive power of neural networks have often assumed negative connection weights between neurons, allowing neurons to inhibit the activation of their postsynaptic neurons (Šíma and Orponen, 2003). It is interesting to extend our work with this feature, but in such a way that it is still biologically plausible. In particular, one should make a distinction between excitatory and inhibitory neurons (Gerstner et al., 2014): the postsynaptic weights of excitatory neurons are always positive and the postsynaptic weights of inhibitory neurons are always negative. Both neuron types are used in winner-take-all circuits (Kappel et al., 2014).

As suggested by the findings of Šíma and Wiedermann (1998), inhibitory neurons could allow the neural network to test for the explicit absence of input activations, lifting the expressive power to "regular" behaviors that, in contrast to monotone-regular behaviors, depend on very precise input symbols that are not embedded in surrounding input noise. For example, a neural network might activate an output neuron whenever the input symbol $\{a, b, c\}$ occurs in its pure form, i.e., no other input neurons are active besides a, b, and c.

Another view, is that inhibitory neurons have a stabilizing effect, at least in a winner-take-all setting (Kappel et al., 2014): inhibitory neurons let the most strongly recognized patterns survive; otherwise perhaps too many insignificant pattern pieces will be floating around in the limited working memory.

Possibly, multiple biologically plausible topologies with inhibition are possible. The expressivity of the resulting neural network models, including any results regarding delays, could strongly depend on the manner by which inhibitory and excitatory neurons are connected.

Noise and continuous time Noise is an important aspect of real biological neurons (Gerstner et al., 2014), and it might be an important resource for expressing nondeterministic computations (Maass, 2014). It would be interesting to see how the results regarding regular languages can be extended to this framework. One possibility is to study the quality by which a noisy positive neural network approximates a true monotone-regular behavior. Here, quality might be formalized as the probability of producing correct output activations given a certain probability distribution on the noise.

Moreover, the model studied in this article is based on discrete time steps. Again, real-world neurons do not obey this restriction, so it appears interesting to investigate if our results can be extended to a setting with continuous time. However, the restriction to discrete time steps may enable an understanding of neurons that operate in continuous time by focusing on the causal relationships between neuron activations. From this viewpoint, regular languages could also provide insights into the workings of neurons operating in continuous time.

Learning An important aspect of biological neurons is that they modify their presynaptic weights over time through a learning mechanism called STDP, that depends on the relative timing of neuron activations (Gerstner et al., 2014).[27] One

[27]The acronym "STDP" stands for spike-timing-dependent plasticity.

could for example consider reward-modulated STDP, where connection weights are updated at some time point when the overall performance of the neural network has recently improved (Gerstner et al., 2014). In a biologically plausible setting, it seems intriguing to understand how overall behavior and consciousness could emerge from dopamine neurons signaling reward to an organism (Schultz, 2013).

Forbidding recurrent connections Weak recurrent connections in biological neural networks might already be sufficient to provide an interaction of working memory with new inputs (Buonomano and Maass, 2009). So, pure looping behavior as needed in the recognition of regular languages might not be really needed by an organism. So, in a further expressivity study, one could simplify positive neural networks by forbidding recurrent connections. This way, only finite regular languages can be recognized. It seems interesting to understand the resulting model from a practical perspective. In particular, one might verify if the resulting networks are still useful for real-world tasks. It seems that memories of larger stimuli require more neurons, and longer activation chains between those neurons.

Sharing auxiliary neurons The construction for the expressivity lower bound (Theorem 4.5) builds a separate network of auxiliary neurons for each output neuron. In biological networks, multiple output neurons share a pool of auxiliary neurons (Buonomano and Maass, 2009). It seems interesting to understand the impact of sharing on the behaviors exhibited by the individual output neurons.

Multiple interconnected networks In this article, we have investigated the expressiveness of single networks where all neurons are directly connected to each other. However, when the number of neurons increases, the number of direct connections increases quadratically. This would become impractical to implement in biological neural networks. Indeed, one hypothesis is that the brain is composed of many small networks that are connected strongly internally, but perhaps only weakly externally (Kappel et al., 2014). It is interesting to understand how such an organization of the connections influences the expressivity.

Acknowledgments

The first author thanks Robert Brijder for suggestions regarding the formalization of finite automata.

References

Alon, N. and Boppana, R. B. (1987). The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1).

Alon, N., Dewdney, A., and Ott, T. (1991). Efficient simulation of finite automata by neural nets. *Journal of the ACM*, 38(2):495–514.

Beimel, A. and Weinreb, E. (2006). Monotone circuits for monotone weighted threshold functions. *Information Processing Letters*, 97(1):12–18.

Buonomano, D. and Maass, W. (2009). State-dependent computations: spatiotemporal processing in cortical networks. *Nature Reviews Neuroscience*, 10(2):113–125.

Carrasco, R., Forcada, M., Ángeles Valdés-Muñoz, M., and Ñeco, R. (2000). Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. *Neural Computation*, 12(9):2129–2174.

Carrasco, R., Oncina, J., and Forcada, M. (1999). Efficient encoding of finite automata in discrete-time recurrent neural networks. In *Ninth International Conference on Artificial Neural Networks*, volume 2, pages 673–677.

Daniels, H. and Velikova, M. (2010). Monotone and partially monotone neural networks. *Neural Networks, IEEE Transactions on*, 21(6):906–917.

Douglas, R. and Martin, K. (2004). Neuronal circuits of the neocortex. *Annual Review of Neuroscience*, 27(1):419–451.

Dupont, P., Denis, F., and Esposito, Y. (2005). Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371.

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

Gécseg, F. and Imreh, B. (2001). On monotone automata and monotone languages. *Journal of Automata, Languages, and Combinatorics*, 7(1):71–82.

Gerstner, W., Kistler, W., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press.

Hopcroft, J. and Ullman, J. (1979). *Introduction to automata theory, languages, and computation*. Addison-wesley.

Horne, B. and Hush, D. (1996). Bounds on the complexity of recurrent neural network implementations of finite state machines. *Neural Networks*, 9(2):243–252.

Indyk, P. (1995). Optimal simulation of automata by neural nets. In Mayr, E. and Puech, C., editors, *STACS 95*, volume 900 of *Lecture Notes in Computer Science*, pages 337–348. Springer Berlin Heidelberg.

Isoda, M. and Tanji, J. (2003). Contrasting neuronal activity in the supplementary and frontal eye fields during temporal organization of multiple saccades. *Journal of Neurophysiology*, 90(5):3054–3065.

Kappel, D., Nessler, B., and Maass, W. (2014). STDP installs in winner-take-all circuits an online approximation to hidden markov model learning. *PLOS Computational Biology*, 10(3).

Legenstein, R. and Maass, W. (2008). On the classification capability of sign-constrained perceptrons. *Neural Computation*, 20(1):288–309.

Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, 8(1).

Maass, W. (2014). Noise as a resource for computation and learning in networks of spiking neurons. *Proceedings of the IEEE*, 102(5):860–880.

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

Parberry, I. (1994). *Circuit Complexity and Neural Networks*. The MIT Press.

Reber, A. (1967). Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior*, 6(6):855–863.

Schultz, W. (2013). Updating dopamine reward signals. *Current Opinion in Neurobiology*, 23(2):229–238.

Shima, K. and Tanji, J. (2000). Neuronal activity in the supplementary and presupplementary motor areas for temporal organization of multiple movements. *Journal of Neurophysiology*, 84(4):2148–2160.

Siegelmann, H. and Sontag, E. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150.

Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology.

Šíma, J. and Orponen, P. (2003). General-purpose computation with neural networks: A survey of complexity theoretic results. *Neural Computation*, 15(12):2727–2778.

Šíma, J. and Wiedermann, J. (1998). Theory of neuromata. *Journal of the ACM*, 45(1):155–178.

A Design of the Weights (Claim 4.6)

Denote $\mathbb{N}_0 = \mathbb{N} \setminus \{0\}$. Let $m \in \mathbb{N}_0$ and $n \in \mathbb{N}_0$. Suppose we have two sets Y and Z with $m = |Y|$ and $n = |Z|$. Both sets should form the presynaptic neurons of a neuron x. We want to find weights w_1 and w_2, to be assigned to the neurons in Y and Z respectively, such that

1) $m \cdot w_1 + (n-1) \cdot w_2 < 1$;

2) $w_1 + n \cdot w_2 \geq 1$;

3) $n \cdot w_2 < 1$.

Condition 1 expresses that all neurons from Z should be activated before x may be activated, regardless of how many neurons in Y are activated. Condition 2 expresses that if all neurons in Z are activated then a single neuron from Y suffices to activate x; but Condition 3 stipulates that at least one neuron of Y should be activated. So, neuron x requires all neurons of Z and just a single neuron from Y. Our design of such weights is based on a denominator $f \in \mathbb{N}_0$:

$$\begin{aligned} w_1 &= 1/f, \\ w_2 &= (1 - 1/f)/n. \end{aligned}$$

We see that Condition 2 is satisfied for any $f \in \mathbb{N}_0$:

$$\begin{aligned} 1/f + n\left((1 - 1/f)/n\right) &= 1/f + (1 - 1/f) \\ &= 1 \geq 1. \end{aligned}$$

Also, Condition 3 is satisfied for any $f \in \mathbb{N}_0$:

$$n\left((1 - 1/f)/n\right) = 1 - 1/f < 1.$$

For Condition 1, we solve for f:

$$\begin{aligned} m \cdot w_1 + (n-1) \cdot w_2 &< 1; \\ m/f + (1 - 1/f)\left(\frac{n-1}{n}\right) &< 1; \\ &\vdots \\ f &> n(m-1) + 1. \end{aligned}$$

So, we can choose $f = n \cdot m + 1$.[28]

[28]Because $n > 0$, we can make the following derivation: $m - 1 < m$; $n(m-1) < n \cdot m$; $n(m-1) + 1 < n \cdot m + 1$.