Positive Neural Networks in Discrete Time
Implement Monotone-Regular Behaviors

Tom J. Ameloot
Hasselt University

Abstract

Many works have investigated the expressive power of various kinds of neural networks. We continue this study with inspiration from biologically plausible models. In particular, we study positive neural networks with multiple input neurons, and where neurons only excite each other and do not inhibit each other. Different behaviors can be expressed by varying the connection strengths between the neurons. 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.

Contents
1 Introduction 2
2 Preliminaries 5
   2.1 Finite Automata and Regular Languages 5
   2.2 Behaviors 6
   2.3 Monotone-regular Behaviors 7
   2.4 Positive Neural Networks 7
   2.5 Implementing Behaviors, with Delay 9
3 Expressivity Results 9
   3.1 Upper Bound 10
   3.2 Lower Bound 12
   3.3 Separation 18
   3.4 On Zero Delay 20
4 Conclusion and Future Work 22
A Proof of Theorem 3.1 26
B Proof of Lemma 3.3 28
1 Introduction

Many works have investigated the expressive power of various kinds of neural networks \[18\]. A frequently recurring idea for understanding neural networks is to relate them to finite automata. In particular, simulations of finite automata by neural networks have been explored, along with complexity issues, e.g., regarding the minimal number of neurons that are needed for a certain task \[18\]. Simulations of deterministic automata \[1, 10, 15, 9\] and nondeterministic automata \[4\] have been studied.

It was suggested by Šíma and Orponen that the study of expressive power could gradually consider increasingly more biologically realistic models \[18\]. In this paper, we take inspiration from the work by Kappel et al. \[12\], who study the expressive power of winner-take-all (WTA) networks. Such WTA networks are a biologically plausible model for small modules of functionality in the brain \[5\]. Kappel et al. theoretically prove and experimentally demonstrate that WTA networks can learn to express a hidden Markov model. One hypothesis, is that the brain is composed of many interconnected WTA networks \[12\]. It therefore seems natural to use this model as inspiration for the further study of the expressive power of neural networks.

Multiple input neurons One important aspect of the WTA model considered by Kappel et al. \[12\], is that a neural network has multiple input neurons that may become concurrently active, thus feeding more complex input symbols to the network. This is in contrast to models in the past expressivity study, where either input encodings were used that (i) made only one input neuron active at any given time \[19, 4, 5\] or (ii) presented a single bit string just once over multiple input neurons after which they remained silent \[18\]. We therefore believe that more attention should be given to the expressive power of neural networks for the case of multiple input neurons that may be concurrently active. This provides a biologically realistic setting, where stimuli from multiple sensory organs, and from multiple cells within each organ, arrive concurrently. It is fascinating to understand how neurons could form concepts or recognize patterns over such rich inputs.

Positive neural networks In this paper, we study positive neural networks, where all connections between neurons use a positive weight. So, neurons only excite each other and they do not inhibit each other. Each network has distinguished sets of input neurons, output neurons, and auxiliary neurons. Importantly, taking inspiration from the model by Kappel et al. \[12\], we consider multiple input neurons that are allowed to be concurrently active. The network may also be recurrent, i.e., the activation of a neuron may indirectly influence its own future activation.
A symbol in our model is a set of 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, a behavior transforms each sequence of input symbols to a sequence of output symbols. By varying the connection weights between neurons of a network, different behaviors can be expressed. By describing such behaviors, we can derive theoretical upper and lower bounds on the expressivity of positive neural networks.

Our model is still strongly related to some previous models \[19, 18\]: our model is based on discrete time, is noise-free, and omits learning. We have additionally omitted inhibition to better understand the foundations of computation in neural networks, where different layers of features build upon each other to gradually increase the expressive power (see also Section 4). In contrast to inhibition, excitation between neurons seems to be a basic feature that we can not omit.

Regular languages  
Previously, Šíma and Wiedermann \[19\] have investigated the expressive power of neuromata, which are neural networks in discrete time that read bit strings over a single input neuron and that can recognize whether prefixes of the input string belong to a regular language or not. The expressive power of neuromata was investigated by relating them to finite automata. In particular, Šíma and Wiedermann convert regular expressions to neuromata, essentially simulating nondeterministic automata.

In this paper, we elaborate the relationship between neural networks and nondeterministic finite automata by considering the novel setting with multiple input neurons that are allowed to be concurrently active. Concretely, we show that the class of positive neural networks captures the so-called monotone-regular behaviors. The monotone-regular behaviors describe the intended output of a network with regular languages over the input neurons. The term monotonicity here means that each output neuron is activated whenever certain patterns 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 from the fact that neurons only excite each other and do not inhibit each other. This notion did not appear explicitly in the work by Šíma and Wiedermann \[19\] because their neural networks exactly recognize regular languages over the single input neuron by using inhibition (in the form of negative connection weights): regular languages can explicitly test for the absence of input activations at certain times.

A finer picture emerges if one takes into account the delay by which a positive neural network implements a monotone-regular behavior. Delay is a standard notion in the expressivity study of neural networks \[19, 18\], and is defined as the number of extra time steps needed by the neural network before it can officially produce the output symbols prescribed by the behavior. Now, it turns out 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 \[19\], but it is based on a new technical construction.
to deal with multiple input neurons (see below). Moreover, 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.

**Simulation** Our expressivity lower bound, saying that all monotone-regular behaviors can be implemented by a positive neural network with a delay of one time unit, is based on a new simulation of nondeterministic finite automata by positive neural networks. We simulate automaton states by neurons as expected, but the technical challenge is to deal with the more complex input symbols generated by concurrently active input neurons. 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 Šima and Wiedermann [19], the constructed neural network may activate auxiliary neurons in parallel. Accordingly, our simulation preserves the nondeterminism, or parallelism, of the simulated automaton.

**Relevance** For the WTA model, Kappel et al. [12] have provided the following lower bound on expressivity: WTA networks can (learn to) express hidden Markov models. Our finding that nondeterministic finite automata can be simulated with positive neural networks is in line with the expressivity result of Kappel et al. because hidden Markov models are a generalization of nondeterministic automata: 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. One of the insights by Kappel et al. is that the states of a hidden Markov model are encoded by the global state of a neural network, i.e., the collective whole of all neurons being activated at any time. Interestingly, however, the simulation of automata as provided by previous works [18] and the current paper, might provide intuition about the role played by individual neurons. Indeed, it is typical to simulate automaton states by individual neurons and not by global network states. We believe that such a fine-grained viewpoint is informative to understanding computation in biological neural networks.

As a final point, we mention that biological neurons mostly encode information in the timing of their activations and not in the magnitude of the activation signals [7]. Hence, a study of expressivity of neural networks might initially focus on only the timing of neuron activations. In this regard, one may view discrete time models like ours as highlighting the causal steps of the neuronal computation, to draw conclusions about how neurons cooperate and how the activations of neurons evolve over a network topology. The discrete time step could in principle be chosen very small, so any results obtained in discrete time could still inform or inspire results in continuous time, with or without noise.

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

2.1 Finite Automata and Regular Languages

We recall some basic definitions from finite automata and regular languages \cite{HopcroftUllman:2003}.

A string \( \alpha \) over a finite set \( X \) is a finite sequence of elements from \( X \). The empty string corresponds to the empty sequence. We also refer to the elements of a string as its symbols. The length of \( \alpha \) is denoted \( |\alpha| \). For each \( i \in \{1, \ldots, |\alpha|\} \), we write \( \alpha_i \) to denote the symbol of \( \alpha \) at position \( i \). We use the following string notation:

\[
\alpha = (\alpha_1, \ldots, \alpha_{|\alpha|}).
\]

For each \( i \in \{1, \ldots, |\alpha|\} \), let \( \alpha_{\rightarrow i} \) denote the prefix \((\alpha_1, \ldots, \alpha_i)\).

Let \( \epsilon \) be a special symbol, called jump. An alphabet \( \Sigma \) is a finite set not containing \( \epsilon \). A language \( L \) over \( \Sigma \) is a set of strings over \( \Sigma \). Languages can be finite or infinite.

A (finite) automaton is a tuple \( M = (Q, \Sigma, \delta, q_s, F) \) where

- \( Q \) is a set of states,
- \( \Sigma \) is an alphabet,
- \( \delta \) is the transition function, mapping each pair \((q, a) \in Q \times (\Sigma \cup \{\epsilon\})\) to a subset of \( Q \)
- \( q_s \in Q \) is the start state, and
- \( F \subseteq Q \) is the set of accepting states.

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

- \( q_1 = q_s \); 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 call a string \( \beta \) over \( \Sigma \cup \{\epsilon\} \) an \( \epsilon \)-extension of a string \( \alpha \) over \( \Sigma \) if the removal of all \( \epsilon \)-symbols from \( \beta \) results in \( \alpha \). We say that the automaton \( M \) accepts \( \alpha \) if there is an accepting run of \( M \) on an \( \epsilon \)-extension of \( \alpha \). Automaton \( M \) could be nondeterministic: for the same input string \( \alpha \), there could be multiple \( \epsilon \)-extensions of \( \alpha \) for each of which there could again be multiple accepting runs.

We define the language \( L \) over \( \Sigma \) that is recognized by \( M \): language \( L \) is the set of all strings over \( \Sigma \) that are accepted by \( M \). Now, a language is said to be regular if it is recognized by an automaton.

We say that automaton \( M \) is jump-free if \( \delta(q, \epsilon) = \emptyset \) for each \( q \in Q \). In this context, we recall the following Lemma:

**Lemma 2.1** (Hopcroft and Ullman \cite{HopcroftUllman:2003}, pages 26–27). Every regular language recognized by an automaton \( M_1 \) is also recognized by an automaton \( M_2 \) that is jump-free and that has the same states as \( M_1 \).

\footnote{Importantly, this subset could be empty.}
Remark 2.2. We call an automaton $M = (Q, \Sigma, \delta, q_0, F)$ deterministic if $|\delta(q, a)| = 1$ for each $(q, a) \in Q \times \Sigma$ and $M$ is jump-free. Nondeterministic automata are typically smaller and easier to understand compared to deterministic automata \cite{17}. 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. For simplicity of discussion, we assume $M$ is jump-free (otherwise we first apply Lemma 2.1). The parallel run of $M$ on an input string $\alpha = (\alpha_1, \ldots, \alpha_n)$ over $\Sigma$ is the sequence

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

where $P_1 = \{q_0\}$ 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 $\alpha$ under the parallel semantics if the last state set of the parallel run contains an accepting state.

We now argue the equivalence of the semantics. First, we observe that any accepting run $q_1, \ldots, q_{n+1}$ of $M$ on $\alpha$ is embedded into $P_1, \ldots, P_{n+1}$, i.e., $q_i \in P_i$ for each $i \in \{1, \ldots, n+1\}$; this causes $M$ to accept $\alpha$ under the parallel semantics too.

For the converse direction, suppose $M$ accepts $\alpha$ under the parallel semantics, i.e., for the parallel run $P_1, \ldots, P_{n+1}$ of $M$ on $\alpha$ there is a state $q_{n+1} \in P_{n+1} \cap F$. Now, using the definition of $P_{n+1}$, there is a state $q_n \in P_n$ such that $q_{n+1} \in \delta(q_n, \alpha_n)$. Using a similar reasoning, it is possible to select a complete sequence of states $q_1, \ldots, q_{n+1}$ where $q_i \in P_i$ for each $i \in \{1, \ldots, n+1\}$, implying $q_1 = q_0^\alpha$, and such that $q_i \in \delta(q_{i-1}, \alpha_{i-1})$ for each $i \in \{2, \ldots, n+1\}$. We see that $q_1, \ldots, q_{n+1}$ is an accepting run of $M$ on $\alpha$.

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 3.2). \hfill $\Box$

2.2 Behaviors

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

Let $I$ and $O$ be finite sets, whose elements we may think of as representing neurons. In particular, the elements of $I$ and $O$ are called input and output neurons respectively. Now, a behavior $B$ over input set $I$ and output set $O$ is a function that maps each nonempty string over alphabet $\mathcal{P}(I)$ to a subset of $O$.\footnote{If $M$ is not jump-free then each $P_i$ would be extended with all states $q_i' \in Q$ that are reachable from a state $q_i \in P_i$ by a path of $\epsilon$-transitions.} Regarding terminology, for a string $\alpha$ over $\mathcal{P}(I)$ and an index $i \in \{1, \ldots, |\alpha|\}$, the symbol $\alpha_i$ says which input neurons are active at (discrete) time $i$; the neurons in $\alpha_i$ are also said to spike, or emit a spike, at time $i$. Note that multiple input neurons can be concurrently active.

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

- $\beta_1 = \emptyset$, and
- $\beta_i = B(\alpha_{i-1})$ 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.\footnote{We will always tacitly assume that $\epsilon \notin \mathcal{P}(I)$.}
2.3 Monotone-regular Behaviors

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

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

\[
x \in B(\alpha) \iff \alpha \text{ embeds a string } \beta \in L(x).
\]

Intuitively, the regular language \( L(x) \) describes the patterns that output neuron \( x \) reacts to. So, the meaning of neuron \( x \) is the recognition of language \( L(x) \). We use the term monotone to indicate that \( L(x) \) is recognized within surrounding superfluous input spikes, through the notion of embedding. The restriction to founded regular languages expresses that outputs do not emerge spontaneously, i.e., each output spike is given the opportunity to witness at least one preceding input spike.

**Remark 2.3.** Let \( M \) be an automaton that recognizes a founded regular language over \( P(I) \). If one reads the symbol \( \emptyset \) from the start state of \( M \), then one may only enter states \( q \) from which it is impossible to reach an accepting state; otherwise the recognized language would not be founded (cf. Lemma 3.4). □

2.4 Positive Neural Networks

We define a neural network model that is related to previous discrete time models [19, 18], 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 \( N \) is a tuple \((I, O, A, W)\), where

- \( I, O, \) and \( A \) are pairwise disjoint sets, containing respectively the input neurons, the output neurons, and the auxiliary neurons;\(^4\)
- we let

\[
\text{edges}(N) = \{(I \times O) \cup (I \times A) \cup (A \times O) \\
\cup \{(x, y) \in A \times A \mid x \neq y\}
\]

be the set of possible connections; and,

- the function \( W \) is the weight function that maps each \((x, y) \in \text{edges}(N)\) to a value in \([0, 1]\).

Note that there are direct connections from the input neurons to the output neurons. 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.

\(^4\)Auxiliary neurons are also sometimes called hidden neurons [18].
Auxiliary neurons can recognize increasingly longer patterns by activating each other \[6, 12\]. We refer to Section 3 for constructions involving auxiliary neurons.

We introduce some notations for convenience. If \(N\) is understood from the context, for each \(x \in I \cup O \cup A\), we abbreviate
\[
\text{pre}(x) = \{y \in I \cup A \mid (y, x) \in \text{edges}(N) \text{ and } W(y, x) > 0\}
\]
and
\[
\text{post}(x) = \{y \in O \cup A \mid (x, y) \in \text{edges}(N) \text{ and } W(x, y) > 0\}.
\]
We call \(\text{pre}(x)\) the set of presynaptic neurons of \(x\), and \(\text{post}(x)\) the set of postsynaptic neurons of \(x\).

### 2.4.1 Operational Semantics

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

**Intuition** We do \(|\alpha|\) steps, called transitions, to process all symbols of \(\alpha\). At each time \(i \in \{1, \ldots, |\alpha|\}\), an input neuron \(x\) spikes if \(x \in \alpha_i\). A spike emitted by any neuron at time \(i\) travels to all its postsynaptic neurons, and such received spikes will be processed at the next time \(i + 1\). The strength by which the spike is received, depends on the weight of the edge towards a postsynaptic neuron. A postsynaptic neuron emits a spike of its own if the sum of all received strengths is larger than or equal to a firing threshold. The firing threshold in our model is 1. All received spikes are immediately discarded when proceeding to the next time.

**Transitions** A transition of \(\mathcal{N}\) is a triple \((N_i, S, N_j)\) where \(N_i \subseteq O \cup A\) and \(N_j \subseteq O \cup A\) are two sets of activated neurons, \(S \in P(I)\) is an input symbol, and where
\[
N_j = \{y \in O \cup A \mid \sum_{z \in \text{pre}(y) \cap (N_i \cup S)} W(z, y) \geq 1\}.
\]
We call \(N_i\) the source set and \(N_j\) the target set of the transition\(^5\).

**Remark 2.4.** In biological neurons, the weight contributed by a single edge, which abstracts a set of synapses, is usually much smaller than the firing threshold \[7\]. For technical simplicity (cf. Section 3), however, the weights in our model can be relatively large compared to the firing threshold \[6\]. In biologically more realistic networks, large weights could be simulated by arranging for multiple neurons to spike concurrently, causing the resulting sum of emitted weights to be large \[13\]. \(\Box\)

\(^5\)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.

\(^6\)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.
Run  The run $R$ of $N$ on input $\alpha$ is the unique sequence of $|\alpha|$ transitions for which

- the transition with ordinal $i \in \{1, \ldots, |\alpha|\}$ reads input symbol $\alpha_i$;
- 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 $R$ defines $|\alpha| + 1$ sets of activated neurons, including the first source set. We define the output of $N$ on $\alpha$, denoted $N(\alpha)$, as the set $N \cap O$ where $N$ is the target set of the last transition in the run of $N$ on $\alpha$.

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

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

### 2.5 Implementing Behaviors, with Delay

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

Delay is a standard notion in the expressivity study of neural networks [19, 18]. We say that $N$ implements a compatible behavior $B$ with delay $k \in \mathbb{N}$ when for each input string $\alpha$ over $P(I)$,

- if $|\alpha| \leq k$ then $N(\alpha) = \emptyset$ and,
- if $|\alpha| > k$ then $N(\alpha) = B(\alpha \rightarrow m)$ where $m = |\alpha| - k$.

Intuitively, delay is the amount of additional time steps that $N$ needs before it can officially conform to the behavior. This additional time is provided by reading more input symbols.\footnote{If $k = 0$ then this condition is immediately true because we consider no input strings with length zero.} Note that a zero delay implementation corresponds to $N(\alpha) = B(\alpha)$ for all input strings $\alpha$.

Letting $B$ be the behavior defined by $N$, note that $N$ implements $B$ with zero delay.

### 3 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 3.1; a lower bound on expressivity in Section 3.2; and, in Section 3.3, 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 3.4 a large class of monotone-regular behaviors that can be implemented with zero delay. If delay is ignored, we could roughly say that positive neural networks capture the monotone-regular behaviors.

3.1 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 Šima and Wiedermann [19], with the difference that we now work with multiple input neurons and the notion of monotonicity.

Theorem 3.1. 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 [19]. An interesting novel aspect, however, is monotonicity, meaning that output neurons recognize patterns even when those patterns are embedded into larger inputs.

Let \( N = (I, O, A, W) \) be a positive neural network. Let \( B \) denote the behavior defined by \( N \). We show that \( B \) is monotone-regular. Fix some \( x \in O \).

We define a founded regular language \( L(x) \) such that for each input string \( \alpha \) over \( P(I) \) we have

\[
x \in B(\alpha) \iff \alpha \text{ embeds a string } \beta \in L(x).
\]

We first define a deterministic automaton \( M \). Let \( q^s \) and \( q^h \) be two state symbols where \( q^s \neq q^h \) and \( \{q^s, q^h\} \cap P(O \cup A) = \emptyset \). 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 \( M = (Q, \Sigma, \delta, q^s, F) \), where

- \( Q = \{q^s, q^h\} \cup P(O \cup A) \);
- \( \Sigma = P(I) \);
- regarding \( \delta \), for each \( (q, S) \in Q \times \Sigma \),
  - if \( q = q^s \) and \( S = \emptyset \) then \( \delta(q, S) = \{q^h\} \);
  - if \( q = q^s \) and \( S \neq \emptyset \) then \( \delta(q, S) = \{q'\} \) where
    \[
    q' = \{y \in O \cup A \mid \sum_{z \in \text{pre}(y) \cap S} W(z, y) \geq 1\};
    \]
  - if \( q = q^h \) then \( \delta(q, S) = \{q^h\} \);
  - if \( q \in P(O \cup A) \) then \( \delta(q, S) = \{q'\} \) where
    \[
    q' = \{y \in O \cup A \mid \sum_{z \in \text{pre}(y) \cap (q \cup S)} W(z, y) \geq 1\};
    \]
The addition of state \( q^\alpha \) 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 \( M \). We show in Appendix A that for all input strings \( \alpha \) over \( \mathcal{P}(I) \),

\[ x \in B(\alpha) \iff \alpha \text{ embeds a string } \beta \in \mathcal{L}(x). \]

The notion of monotonicity becomes apparent in the direction “\( \Rightarrow \)”.

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 neurons. That information would be lost when casting neurons as states.

The following example demonstrates that an implementation with zero delay is at least achievable for some simple monotone-regular behaviors. In Section 3.4 we will also see more advanced monotone-regular behaviors that can be implemented with zero delay.

Example 3.2. Let \( B \) be a monotone-regular behavior over an input set \( I \) 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 \( B \), 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 \( \mathcal{N} = (I, \mathcal{O}, \mathcal{A}, W) \) to implement \( B \) 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 \( W(u, x) = 1/m \) for each \( u \in S_1 \); all other weights are set to zero. We can observe that \( \mathcal{N}(\alpha) = B(\alpha) \) for each input string \( \alpha \) over \( \mathcal{P}(I) \).

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 \( y_i \) should detect symbol \( S_i \). 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 \( W \) as follows, where any unspecified weights are assumed to be zero:

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

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

The technical construction in the expressivity lower bound (Theorem 3.6 below) may provide intuition about the roles played by individual neurons. We first introduce some additional terminology and definitions.

3.2.1 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 \((q, S) \in Q \times (\Sigma \cup \{\epsilon\})\) such that \( q \in \delta(q, S) \). The following Lemma tells us that self-loops can be removed without introducing symbol \( \epsilon \):

**Lemma 3.3.** Every regular language recognized by an automaton \( M_1 \) is also recognized by an automaton \( M_2 \) that (i) contains no self-loops, (ii) uses at most double the number of states of \( M_1 \), and (iii) is jump-free if \( M_1 \) is jump-free.\(^{10}\) (The proof is given in Appendix B.)

For founded regular languages, Lemma 3.4 (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 first input symbol that activates the automaton provides initial spikes that can be propagated through the neural network to keep track of any further progress, even if subsequent input symbols are empty.

**Lemma 3.4.** Letting \( I \) be an input set, every founded regular language over \( \mathcal{P}(I) \) recognized by a jump-free automaton \( M_1 \) is also recognized by a jump-free automaton \( M_2 = (Q_2, \Sigma_2, \delta_2, q_{s2}, F_2) \) where (i) \( \delta_2(q_{s2}, \emptyset) = \emptyset \), and (ii) \( M_2 \) has the same states as \( M_1 \).\(^{11}\) (The proof is given in Appendix C.)

Let \( M \) be as above. A state \( q \in Q \) is said to be reachable if there is string \( \alpha \) over \( \Sigma \cup \{\epsilon\} \) and a run of \( M \) on \( \alpha \) in which \( q \) appears; this run does not have to be accepting. Clearly,

**Lemma 3.5.** 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 \( I \) be an input set, and letting \( M \) be an automaton that recognizes a founded regular language over \( \mathcal{P}(I) \), we call \( M \) clean if

- \( M \) is jump-free;
- \( M \) contains no self-loops;
- \( M \) does not read symbol \( \emptyset \) from its start state; and,
- \( M \) contains only reachable states.

\(^{10}\)Intuitively, the quantification of the number of states indicates that in general \( M_2 \) preserves the nondeterminism of \( M_1 \).

\(^{11}\)The restriction to jump-free automata is only for technical convenience, and is not crucial.
By applying Lemmas 2.1, 3.3, 3.4, and 3.5 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 3.3).

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

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

In words: the pair count is the number of combinations in $M$ of a non-start state and an incoming symbol to that state.

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

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

Intuitively, an automaton implementation for $B$ is a prototype implementation that can later be converted to a neural network. The total pair count of $M$, denoted $p(M)$, is defined as

$$p(M) = \sum_{x \in O} p(M(x)).$$

### 3.2.2 Lower Bound Result

**Theorem 3.6.** Every monotone-regular behavior $B$ can be implemented by a positive neural network with delay 1. In particular, each automaton implementation $M$ for $B$ can be converted to a positive neural network that implements $B$ with delay 1 and that has $p(M)$ auxiliary neurons.\(^{\text{12}}\)

**Proof.** (The construction below is illustrated in Example 3.8.) Let $B$ be a monotone-regular behavior over an input set $I$ and an output set $O$. Let $M(x)$ be an automaton implementation for $B$. Intuitively, for each output neuron $x$, we translate automaton $M(x)$ to a neural network. 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 only use 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 $M(x)$.\(^{\text{13}}\) See also the later Remark 3.9.

For simplicity, we assume $|O| = 1$, and we denote $O = \{x\}$; for the case of multiple output neurons, the construction given below can be repeated, and the

\(^{\text{12}}\)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 $M$.

\(^{\text{13}}\)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 3.3).
networks thus obtained can be united to form the overall desired network. Let $M = \mathcal{M}(x)$ and denote $M = (Q, \Sigma, \delta, q^0, F)$ where $\Sigma = \mathcal{P}(I)$. Recall that $M$ is clean.

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

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

$$\mathcal{A} = \{(q, S) \in Q \times \mathcal{P}(\mathcal{I}) \mid q \neq q^0 \text{ and } \exists q' \in Q \text{ with } q \in \delta(q', S)\}.$$ 

Note that $|\mathcal{A}| = p(M)$. Intuitively, an auxiliary neuron $(q, S)$ represents the automaton state $q$ reached by reading input symbol $S$. We define the set $T \subseteq \mathcal{A}$ of trigger neurons:

$$T = \{(q, S) \in \mathcal{A} \mid q \in \delta(q^0, S)\}.$$ 

Intuitively, the neurons in $T$ are the first auxiliary neurons to 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 $(q, S) \in T$ we have $S \neq \emptyset$ because $\delta(q^0, \emptyset) = \emptyset$ by assumption on $M$.

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

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

Intuitively, $\mathcal{C}(q, S)$ is the set of auxiliary neurons that recognize prefixes of the strings that $(q, S)$ should recognize, i.e., $\mathcal{C}(q, S)$ is the working memory from the viewpoint of $(q, S)$. In the definition of $\mathcal{C}(q, S)$, there is no relationship between the symbols $S$ and $S'$. Note that for each $(q, S) \in \mathcal{A} \setminus T$, the set $\mathcal{C}(q, S)$ is always nonempty because $M$ contains only reachable states.\(^{14}\)

Weights Denote $\mathbb{N}_0 = \mathbb{N} \setminus \{0\}$. To specify the weights, we consider the following three functions:

$$f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0 : \quad f(m, n) = n \cdot m + 1;$$  
$$w_1 : \mathbb{N}_0 \times \mathbb{N}_0 \to [0, 1] : \quad w_1(m, n) = \frac{1}{f(m, n);}$$  
$$w_2 : \mathbb{N}_0 \times \mathbb{N}_0 \to [0, 1] : \quad w_2(m, n) = \left(1 - \frac{1}{f(m, n)}\right)/n.$$ 

Intuitively, if we have an auxiliary neuron $y$ where we assign $w_1(m, n)$ as presynaptic weight to each of $m$ other auxiliary neurons $\{z_1, \ldots, z_m\}$ and $w_2(m, n)$ as presynaptic weight to each of $n$ input neurons $\{u_1, \ldots, u_n\}$, then Claim 3.7 (below) tells us the following: (i) $y$ is not activated if not 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.

\(^{14}\)Indeed, since $(q, S) \in \mathcal{A}$, there is a reachable state $q' \in Q$ with $q \in \delta(q', S)$. But $(q, S) \notin T$ implies $q' \neq q^0$, causing $(q', S') \in \mathcal{A}$ for some $S' \in \mathcal{P}(\mathcal{I})$. Hence, $(q', S') \in \mathcal{C}(q, S)$.
Claim 3.7. Let $m,n \in \mathbb{N}_0$. We have

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

(The proof is given in Appendix D.)

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

- for the output neuron $x$, and each $(q,S) \in A$ with $q \in F$, we define $W((q,S),x) = 1$;
- for each $(q,S) \in T$ and each $y \in S$, letting $n = |S|$, we define $W(y,(q,S)) = 1/n$;
- for each $(q,S) \in A \setminus T$ with $S = \emptyset$, for each $y \in \text{con}(q,S)$, we define $W(y,(q,S)) = 1$;
- for each $(q,S) \in A \setminus T$ with $S \neq \emptyset$, letting $m = |\text{con}(q,S)|$ and $n = |S|$, for each $y \in \text{con}(q,S)$, we define $W(y,(q,S)) = w_1(m,n)$, and for each $z \in S$, we define $W(z,(q,S)) = w_2(m,n)$;

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

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

Correctness Intuitively, $\mathcal{N}$ simulates the automaton $M$. And because $M$ recognizes the language $L(x)$ that describes the role of $x$ in $B$, the network $\mathcal{N}$ conforms to $B$. There is a delay of one time unit because $x$ computes an “or” over auxiliary neurons that simulate accept states of $M$. The formal proof that $\mathcal{N}$ implements $B$ with delay 1 is given in Appendix E. ⊓⊔

Example 3.8. We illustrate the construction of the proof of Theorem 3.6. Let $I$ consist of four distinct input neurons $a$, $b$, $c$, and $d$. Let $O = \{x\}$. 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 1 that recognizes a founded regular language over $\mathcal{P}(I)$: we denote this language as $L(x)$. Language $L(x)$ is infinite because of the loop between states $q_1$ and $q_2$ over symbol $S_2$. In particular, $L(x)$ contains all strings of the form $(S_1,S_2^*,S_3)$, where

---

15In Figure 1 we use the standard notations [8, 17]: the start state has an entering arrow with no source, and accepting states are indicated with double circles.
$S^2_n$ denotes an arbitrary number of repetitions of symbol $S_2$. Let $B$ be the monotone-regular behavior over $I$ and $O$ defined by $\mathcal{L}(x)$: for each input string $\alpha$ over $\mathcal{P}(I)$,

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

Applying the transformation in the proof of Theorem 3.6 to automaton $M$ results in the positive neural network $N$ depicted in Figure 2, 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 $t \in N$ 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 3.9. In the proof of Theorem 3.6, 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 [19] 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. Concretely, for each symbol $S \in \mathcal{P}(I)$ 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 $n = |S|$:

$$W(u, y_S) = 1/n.$$ 

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

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

$$W(y_S, (q, S)) = m/(m + 1),$$

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

$$W(z, (q, S)) = 1/(m + 1).$$

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 [19] 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. □
Figure 1: A clean automaton recognizing a founded regular language (see Example 3.8).

Figure 2: The positive neural network obtained from the automaton in Figure 1 (see Example 3.8).
3.3 Separation

Regarding the expressivity of positive neural networks, the upper bound (Theorem 3.1) and the lower bound (Theorem 3.6) do not coincide. Indeed, as illustrated by the following examples, there are simple monotone-regular behaviors that cannot 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 triggered.

Example 3.10. Let $S_1$ and $S_2$ be two disjoint sets of neurons with $|S_1| \geq 2$ and $|S_2| \geq 2$. Let $I = S_1 \cup S_2$ and $O = \{x\}$. Let $L(x)$ be the following founded regular language over $P(I)$:

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

So, $L(x)$ is a finite language containing two one-symbol strings. Let $B$ be the following monotone-regular behavior over $I$ and $O$ defined by $L(x)$: for each input string $\alpha$ over $P(I)$, we define

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

We show that there is no positive neural network that implements $B$ with zero delay. Towards a contradiction, suppose there is such a neural network $\mathcal{N}$. 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 $\mathcal{N}$ implements $B$ with zero delay, we have $\mathcal{N}(\alpha) = B(\alpha)$ for all input strings $\alpha$ over $P(I)$. In particular, $\mathcal{N}((S_1)) = \{x\}$ and $\mathcal{N}((S_2)) = \{x\}$. These fast output reactions imply that neuron $x$ does not rely on auxiliary neurons, and instead reads input neurons directly. So,

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

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

We distinguish between the following cases:

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

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

Define the symbol $S = \{y, z\}$. Note that $S \in P(I)$. 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 $z$,

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

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

\[\text{An automaton recognizing } L(x) \text{ could have two accepting states } q_1 \text{ and } q_2 \text{ besides the start state } q^0; \text{ reading symbol } S_i \text{ from } q^0 \text{ leads to } q_i \text{ for } i \in \{1, 2\}.\]

18
• If the first case does not hold, then we can choose some \( y \in S_1 \) and \( z \in S_2 \) for which

\[
W(y, x) + W(z, x) < 1.
\]

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,

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

By using \( \sum_{u \in S_1} W(u, x) \geq 1 \) and \( \sum_{u \in S_2} W(u, x) \geq 1 \) from above, and \( W(y, x) + W(z, x) < 1 \), we can further obtain:

\[
\sum_{u \in S} W(u, x) \geq 2 - (W(y, x) + W(z, x)) > 1.
\]

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

\[\Box\]

Example 3.11. Let \( S_1, S_2, S_3, \) and \( S_4 \) be nonempty sets of neurons that are pair-wise 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 \( \mathcal{P}(\mathcal{I}) \):

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

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

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

We show there is no positive neural network that implements \( B \) with zero delay. Towards a contradiction, suppose there is such a network \( \mathcal{N} = (\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{W}) \). We show that \( \mathcal{N} \) erroneously produces an output spike 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 \( \mathcal{N} \) implements \( B \) with zero delay, we have \( \mathcal{N}(\alpha) = B(\alpha) \) for all input strings \( \alpha \) over \( \mathcal{P}(\mathcal{I}) \). In particular, \( \mathcal{N}((S_1, S_2)) = \{x\} \) and \( \mathcal{N}((S_3, S_4)) = \{x\} \). 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 \( i \in \{1, 3\} \),

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

Also denote, for \( i \in \{2, 4\} \),

\[
w_i = \sum_{y \in \mathcal{S}_i} W(y, x).
\]

\[\text{An automaton recognizing this language could splits its computation into two branches from the start state: one branch recognizes the string (}\mathcal{S}_1, \mathcal{S}_2\) and the other branch recognizes the string (}\mathcal{S}_3, \mathcal{S}_4\).\]
Now, the output activations $N((S_1, S_2)) = \{x\}$ and $N((S_3, S_4)) = \{x\}$ imply
\[ w_1 + w_2 \geq 1, \text{ and} \]
\[ w_3 + w_4 \geq 1. \]

We distinguish between the following cases:\(^{18}\)

- Suppose $w_1 + w_4 \geq 1$. This implies $N((S_1, S_4)) = \{x\}$. But then $N((S_1, S_4)) \neq B((S_1, S_4))$, 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 $N((S_3, S_2)) = \{x\}$. But then $N((S_3, S_2)) \neq B((S_4, S_2))$, which is a contradiction.

\[ □ \]

### 3.4 On Zero Delay

The earlier Example 3.2 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 3.12.** 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 $B$ 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 $M$ be an automaton implementation for $B$. As in the proof of Theorem 3.6, we fix some $x \in \mathcal{O}$. Let $\mathcal{L}(x)$ be the language recognized by $M(x)$. Denote $M(x) = (Q, \Sigma, \delta, q^0, F)$, where $\Sigma = \mathcal{P}(\mathcal{I})$. We can modify the construction in the proof of Theorem 3.6 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 $S = S_1$ for each $(q_i, S_i) \in V$.\(^ {19}\) We refer to $S$ as the terminal symbol. The only difference compared to the proof of Theorem 3.6 is that we now let output neuron $x$ listen to (i) the symbol $S$ directly and (ii) a different set $C$ of auxiliary neurons.

\(^{18}\)Although $N((S_1, S_2)) = \{x\}$ and $N((S_3, S_4)) = B((S_4)) = \emptyset$ 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.

\(^{19}\)For each $(q_i, S_i) \in V$, there is an input string $\epsilon$ over $\mathcal{P}(\mathcal{I})$ and a run of $M(x)$ on $\alpha$ ending with $q_i$ because $q_i$ is a reachable state by assumption on $M(x)$. Since $(q_i, S_i) \in V$, the extension of $\alpha$ 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 $S_1 = S_2$. 
Letting $A$ be the set of auxiliary neurons as defined in the proof of Theorem 3.6, we define
\[ C = \{(q, S') \in A \mid (q, S) \in V\}. \]

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 $\alpha \in L(x)$ 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, $q_n \neq q^*$ because $A(x)$ does not read $S = \emptyset$ from its start state, implying $(q_n, S') \in C$ for some $S' \in \mathcal{P}(I)$. Now, for each $y \in C$, we define $W(y, x) = 1$.

- 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 $W(z, x) = 1/n$.

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 $W(y, x) = w_1(m, n)$, and for each $z \in S$, we define $W(z, x) = w_2(m, n)$.

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. $\square$

The following example demonstrates that the converse of Theorem 3.12 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 3.13.** Let $S_1 = \{a, b\}$ and $S_2 = \{b, c\}$ where $a$, $b$, and $c$ are pairwise different neurons. Let $I = S_1 \cup S_2$ and $O = \{x\}$. Let $L(x)$ be the following founded regular language over $\mathcal{P}(I)$:
\[ L(x) = \{(S_1), (S_2)\}. \]

Note that $L(x)$ is not converging. Let $B$ be the monotone-regular behavior over $I$ and $O$ defined by $L(x)$: for each input string $\alpha$ over $\mathcal{P}(I)$,
\[ B(\alpha) = \begin{cases} \{x\} & \text{if } \alpha \text{ embeds a string } \beta \in L(x); \\ \emptyset & \text{otherwise.} \end{cases} \]

The following positive neural network $N = (I, O, A, W)$ implements $B$ with zero delay: $A = \emptyset$, and
\[
W(a, x) = 1/3, \\
W(b, x) = 2/3, \\
W(c, x) = 1/3.
\]
In contrast to Example 3.10, we can not fool this network to trigger \( x \) on a wrong input symbol like \( \{a,c\} \). That is because \( W \) assigns a heavier weight to the connection \((b,x)\), which renders the input neuron \( b \) crucial for the activation of \( x \).

\[\square\]

4 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.

Simulation optimality  As in previous studies of neural networks [18], one could study whether the number of auxiliary neurons in the construction of our lower bound (Theorem 3.6) is optimal or not. Currently, we quantify the number of auxiliary neurons based on the number of symbol-state combinations occurring in the original automata. But perhaps more intricate weight designs could pack more functionality into fewer neurons.

Single input neurons  If there is only a single input neuron, the work by Šíma and Wiedermann [19] shows that all regular languages can be recognized by a neural network with a delay of one time unit. Our paper 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. For example, it is not entirely clear how symbols over multiple input neurons could be translated to a single input neuron while not increasing the output delay.

Characterizing zero delay  We have seen that seemingly simple monotone-regular behaviors already require a delay of one time unit (Section 3.3). We have also made some first steps towards identifying the class of monotone-regular behaviors that can be implemented with zero delay (Section 3.4). However, a precise characterization is missing. Example 3.13 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.

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 [18]. 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 [7]: 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 (WTA) networks [12].

As suggested by the findings of Šíma and Wiedermann [19], 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 WTA setting [12]: 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 networks, 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 [7], and it might be an important resource for expressing nondeterministic computations [14, 11]. 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 paper 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 [7]. One could for example consider reward-modulated STDP, where connection weights are updated at at time point when the overall performance of the neural network has recently improved [7]. 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 [16].

Forbidding recurrent connections Weak recurrent connections in biological neural networks might already be sufficient to provide an interaction of working memory with new inputs [2]. 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, on could simplify positive 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

---

[20]The acronym “STDP” stands for spike-timing-dependent plasticity.
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 3.6) builds a separate network of auxiliary neurons for each output neuron. In biological networks, multiple output neurons share a pool of auxiliary neurons [2]. It seems interesting to understand the impact of sharing on the behaviors exhibited by the individual output neurons.

**Multiple interconnected networks** In this paper, we have investigated the expressiveness of single networks in which the 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 [12]. It is interesting to understand how such an organization of the connections influences the expressivity.

**Acknowledgments**

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

**References**

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

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

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

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

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

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

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

[8] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-wesley, 1979.
[9] B.G. Horne and D.R. Hush. Bounds on the complexity of recurrent neural network implementations of finite state machines. *Neural Networks*, 9(2):243–252, 1996.

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

[11] Z. Jonke, S. Habenschuss, and W. Maass. A theoretical basis for efficient computations with noisy spiking neurons. *CoRR*, abs/1412.5862, 2014.

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

[13] W. Maass. Lower bounds for the computational power of networks of spiking neurons. *Electronic Colloquium on Computational Complexity*, 1(19), 1994.

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

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

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

[17] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2006.

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

[19] J. Šíma and J. Wiedermann. Theory of neuromata. *Journal of the ACM*, 45(1):155–178, 1998.
A  Proof of Theorem 3.1

Let $x \in \mathcal{O}$ be the output neuron for which the automaton $M = (Q, \Sigma, \delta, q^s, F)$ and language $\mathcal{L}(x)$ are defined.

A.1  Direction 1

Let $\alpha$ be an input string over $\mathcal{P}(I)$. Suppose $x \in \mathcal{B}(\alpha)$. We show that $\alpha$ embeds a string $\beta \in \mathcal{L}(x)$.

By definition of $\mathcal{B}$, we have $x \in \mathcal{N}(\alpha)$. Let $k \in \{1, \ldots, |\alpha|\}$ be the smallest index for which $\alpha_k \neq \emptyset$. Note that $k$ exists: in the operational semantics, the output neuron $x$ can only produce a spike if it receives spikes from its presynaptic neurons, which in turn means that originally there must have been input spikes.

Consider the suffix $\beta = (\alpha_k, \alpha_{k+1}, \ldots, \alpha_{|\alpha|})$. Surely $\alpha$ embeds $\beta$. We now show that $\beta \in \mathcal{L}(x)$, i.e., $\beta$ is accepted by the automaton $M$. Denote $m = |\beta|$.

First, note that the prefix $\alpha \rightarrow k - 1$ does not activate neurons because it contains only empty input symbols; so, the network is still in the start state after reading $\alpha \rightarrow k - 1$. Hence, $\mathcal{N}(\beta) = \mathcal{N}(\alpha)$. We consider the following sequence of sets of activated neurons during the run of $\mathcal{N}$ on $\beta$:

$$N_1, N_2, \ldots, N_{m+1},$$

where $N_1 = \emptyset$. Next, because $M$ is deterministic, for each previous state and incoming symbol there is one successor state. So, we can consider the unique sequence of states that automaton $M$ visits while reading $\beta$:

$$q_1, q_2, \ldots, q_{m+1},$$

where $q_1 = q^s$. We show by induction on $i = 2, \ldots, m + 1$ that

$$q_i = N_i.$$

Since $x \in \mathcal{N}(\beta)$, we would obtain that $x \in N_{m+1} = q_{m+1}$, which implies $q_{m+1} \in F$. Then we may conclude that $q_1, \ldots, q_{m+1}$ is an accepting run of $M$ on $\beta$, as desired.

**Base case**  Let $i = 2$. Since $q_1 = q^s$ and $\beta_1 = \alpha_k \neq \emptyset$, we have defined

$$q_2 = \{ y \in \mathcal{O} \cup \mathcal{A} | \sum_{z \in \text{pre}(y) \cap \beta_1} W(z, y) \geq 1 \}.$$

Since $N_1 = \emptyset$, we may write:

$$q_2 = \{ y \in \mathcal{O} \cup \mathcal{A} | \sum_{z \in \text{pre}(y) \cap (N_1 \cup \beta_1)} W(z, y) \geq 1 \}.$$

We see that the right hand side matches the definition of $N_2$ as given by the operational semantics. So, $q_2 = N_2$. 

26
Inductive step  Let \( i \in \{2, \ldots, m\} \), and assume that \( q_i = N_i \). We show that \( q_{i+1} = N_{i+1} \). In \( M \), it is not possible to go back to the start state. So, \( q_i \neq q^s \), and we have defined
\[
q_{i+1} = \{ y \in \mathcal{O} \cup A \mid \sum_{z \in \text{pre}(y) \cap (q_i \cup \beta_i)} W(z, y) \geq 1 \}.
\]
Since \( q_i = N_i \), the right hand side matches the definition of \( N_{i+1} \) as given by the operational semantics. So, \( q_{i+1} = N_{i+1} \).

A.2 Direction 2

Let \( \alpha \) be an input string over \( \mathcal{P}(\mathcal{I}) \). Suppose \( \alpha \) embeds a string \( \beta \in \mathcal{L}(x) \). We show that \( x \in B(\alpha) \). We concretely show \( x \in N(\alpha) \), which, by definition of \( B \), implies \( x \in B(\alpha) \). Denote \( n = |\alpha| \) and \( m = |\beta| \).

Because \( \beta \in \mathcal{L}(x) \), we can consider the sequence of states that automaton \( M \) visits while reading \( \alpha \):
\[
q_1, q_2, \ldots, q_{m+1},
\]
where \( q_1 = q^s \) and \( q_{m+1} \in F \). By definition of \( F \), we have \( x \in q_{m+1} \). Note that \( q_i \in \mathcal{P}(\mathcal{O} \cup \mathcal{A}) \) for each \( i \in \{2, \ldots, m+1\} \) because \( (i) \) after leaving \( q^s \), it is not possible to go back to \( q^s \); and, \( (ii) \) if \( q_j = q^h \) for any \( j \in \{2, \ldots, m+1\} \) then \( q_{m+1} = q^h \notin F \). We can also consider the sequence of sets of activated neurons when running \( \mathcal{N} \) on input \( \alpha \):
\[
N_1, N_2, \ldots, N_{n+1}.
\]
We show by induction on \( i = 2, \ldots, m+1 \) that
\[
q_i \subseteq N_{n-m+i}.
\]
Here, \( n-m+i \) just abbreviates the expression \( n+1-(m+1)+i \). For \( i = m+1 \) this gives \( q_{m+1} \subseteq N_{n+1} \), which implies \( x \in N_{n+1} \cap \mathcal{O} = N(\alpha) \), as desired.

Base case  Let \( i = 2 \). Note that \( 1 \leq m \leq n \) implies \( 1 \leq m-n+2 \leq n+1 \). So, \( N_{n-m+2} \) is defined. Now, since \( q_1 = q^s \) and \( \beta_1 \neq \emptyset \) by foundedness of \( \mathcal{L}(x) \), we have defined
\[
q_2 = \{ y \in \mathcal{O} \cup A \mid \sum_{z \in \text{pre}(y) \cap \beta_1} W(z, y) \geq 1 \}.
\]
Since \( \beta \) is embedded in \( \alpha \), we have \( \beta_1 \subseteq \alpha_{n-m+1} \), which implies \( \beta_1 \subseteq N_{n-m+1} \cup \alpha_{n-m+1} \). So, because there is no inhibition,
\[
q_2 \subseteq \{ y \in \mathcal{O} \cup A \mid \sum_{z \in \text{pre}(y) \cap (N_{n-m+1} \cup \alpha_{n-m+1})} W(z, y) \geq 1 \}.
\]
We see that the right hand side matches the definition of \( N_{n-m+2} \) as given by the operational semantics. So, \( q_2 \subseteq N_{n-m+2} \).

21First, from \( m \geq 1 \) we can deduce: \(-m \leq -1; n-m \leq n-1; n-m+2 \leq n+1\). Second, from \( n \geq m \) we can deduce: \( n-m \geq 0; n-m+2 \geq n-m+1 \geq 1 \).
**Inductive step** Let \( i \in \{2, \ldots, m\} \). For the induction hypothesis, assume that \( q_i \subseteq N_{n-m+i} \). Since \( q_i \in \mathcal{P}(O \cup A) \) (see above), we have defined
\[
q_{i+1} = \{ y \in O \cup A \mid \sum_{z \in \text{pre}(y) \cap (q_i \cup \beta_i)} W(z, y) \geq 1 \}.
\]
Again, because \( \beta \) is embedded in \( \alpha \), we have \( \beta_i \subseteq \alpha_{n-m+i} \). Combined with the induction hypothesis, we obtain \( q_i \cup \beta_i \subseteq N_{n-m+i} \cup \alpha_{n-m+i} \). So, because there is no inhibition,
\[
q_{i+1} = \{ y \in O \cup A \mid \sum_{z \in \text{pre}(y) \cap (N_{n-m+i} \cup \alpha_{n-m+i})} W(z, y) \geq 1 \}.
\]
We see that the right hand side matches the definition of \( N_{n-m+(i+1)} \) as given by the operational semantics. So, \( q_{i+1} \subseteq N_{n-m+(i+1)} \).

### B Proof of Lemma 3.3

Let \( M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1) \) be an automaton. We convert \( M_1 \) to an automaton \( M_2 = (Q_2, \Sigma_2, \delta_2, q_2^1, F_2) \) that recognizes the same regular language as \( M_1 \) and that (i) contains no self-loops, (ii) uses at most double the number of states of \( M_1 \), and (iii) is jump-free if \( M_1 \) is jump-free.

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. We define the set \( V \) of all states of \( M_1 \) involved in a self-loop:
\[
V = \{ q \in Q_1 \mid \exists S \in \Sigma_1 \cup \{ \epsilon \} \text{ such that } q \in \delta_1(q, S) \}.
\]
Let \( f \) be an injective function that maps each \( q \in V \) to a new state \( f(q) \) outside \( Q_1 \). We define:
- \( Q_2 = Q_1 \cup \{ f(q) \mid q \in V \} \);
- \( \Sigma_2 = \Sigma_1 \);
- for each \( q \in Q_1 \) and \( S \in \Sigma_1 \cup \{ \epsilon \} \) with \( q \notin \delta_1(q, S) \),
  \[
  \delta_2(q, S) = \delta_1(q, S);
  \]
- for each \( q \in Q_1 \) and \( S \in \Sigma_1 \cup \{ \epsilon \} \) with \( q \in \delta_1(q, S) \),
  \[
  \delta_2(q, S) = \{ f(q) \} \cup (\delta_1(q, S) \setminus \{ q \});
  \]
- for each \( q \in V \) and \( S \in \Sigma_1 \cup \{ \epsilon \} \),
  \[
  \delta_2(f(q), S) = \delta_1(q, S);
  \]
- \( q_2^1 = q_1^1 \);
- \( F_2 = F_1 \cup \{ f(q) \mid q \in V \cap F_1 \} \).
Note the following: in the definition of $\delta_2$, for $q \in V$ and $S \in \Sigma_1 \cup \{\epsilon\}$, if $q \in \delta_1(q, S)$ then $q \in \delta_2(f(q), S)$, i.e., we can go back from the newly constructed state $f(q)$ to the old state $q$ by reading symbol $S$. This demonstrates that looping over the symbol $S$ is still possible in $M_2$, but now using both $f(q)$ and $q$. An odd number of repetitions over symbol $S$ is possible because we have copied all outgoing transitions of $q$ to $f(q)$.

Also note that $M_2$ is jump-free if $M_1$ is jump-free: (i) for each $q \in V$, if $\delta_2(f(q), \epsilon) \neq \emptyset$ then $\delta_1(q, \epsilon) \neq \emptyset$; and, (ii) for each $q \in Q_1$, if $\delta_2(q, \epsilon) \neq \emptyset$ then $\delta_1(q, \epsilon) \neq \emptyset$.

We show that $M_2$ recognizes the same language as $M_1$.

B.1 Direction 1

Let $\alpha$ be a string accepted by $M_1$. We show that $\alpha$ is accepted by $M_2$.

Because $\alpha$ is accepted by $M_1$, there is an $\epsilon$-extension $\beta = (S_1, \ldots, S_n)$ of $\alpha$ and a sequence of states $q_1, \ldots, q_{n+1}$ of $M_1$ such that:

- $q_1 = q_1^1$;
- $q_i \in \delta_1(q_{i-1}, S_{i-1})$ for each $i \in \{2, \ldots, n+1\}$;
- $q_{n+1} \in F_1$.

We now strategically replace any subsequent repetitions of a state in $q_1, \ldots, q_{n+1}$ by the newly created states, as follows. First, we split $q_1, \ldots, q_{n+1}$ into maximal subsequences of repeated states. Concretely, let

$$T_1, \ldots, T_k$$

be a subdivision of the sequence $q_1, \ldots, q_{n+1}$, where each subsequence $T_j$ with $j \in \{1, \ldots, k\}$ is of the form $q, \ldots, q$ (repeating one state symbol) and such that $T_j$ and $T_{j+1}$ contain different state symbols for each $j \in \{1, \ldots, k-1\}$. Next, we define a sequence of modified subsequences

$$T'_1, \ldots, T'_k,$$

as follows: for each $j \in \{1, \ldots, k\}$, we define $T'_j$ as $T_j$ if $T_j$ has length 1, and otherwise we define $T'_j$ as the modification of $T_j$ where the repeated state $q$ is replaced by $f(q)$ on even positions inside $T_j$.

We now check that all transitions are valid for $M_2$ (i) inside each $T'_j$ with $j \in \{1, \ldots, k\}$; and, (ii) from $T'_j$ to $T'_{j+1}$ for each $j \in \{1, \ldots, k-1\}$.

First, consider some $T'_j$ with $j \in \{1, \ldots, k\}$ and with length at least 2. Let $(q_{i-1}, q_i)$ be a pair of subsequent states inside the original sequence $T_j$, where $q_{i-1} = q_i$. We abbreviate $q = q_{i-1} = q_i$. So, $q_i \in \delta_1(q_{i-1}, S_{i-1})$ implies $q \in \delta_1(q, S_{i-1})$.

- If $q_i$ occurs on an even position inside $T_j$ then we have replaced $q_i$ by $f(q)$ in $T'_j$. Since $q \in \delta_1(q, S_{i-1})$, we have $f(q) \in \delta_2(q, S_{i-1})$, implying $f(q) \in \delta_2(q_{i-1}, S_{i-1})$.

\[\text{The positions inside } T_j \text{ start counting at 1. Moreover, for the case that } T_j \text{ has length at least two, the repeated state } q \text{ clearly occurs in a self-loop; so, } f(q) \text{ is defined.}\]
• If \(q_{i-1}\) occurs on an even position inside \(T_j\) then we have replaced \(q_{i-1}\) by \(f(q)\) in \(T'_j\). By definition of \(\delta_2\), we have \(\delta_2(f(q), S_{i-1}) = \delta_1(q, S_{i-1})\). Since \(q \in \delta_1(q, S_{i-1})\), we obtain \(q \in \delta_2(f(q), S_{i-1})\). So, \(q_i \in \delta_2(f(q), S_{i-1})\).

Consider now some \(T'_j\) and \(T'_{j+1}\) with \(j \in \{1, \ldots, k-1\}\). Let \((q_{i-1}, q_i)\) be the pair of original states such that \(q_{i-1}\) is the (last) state of \(T_j\) and \(q_i\) is the (first) state of \(T_{j+1}\). Recall that \(q_i \in \delta_1(q_{i-1}, S_{i-1})\). Please note that by definition of \(T_j\) and \(T_{j+1}\), we always have \(q_{i-1} \neq q_i\). Also, the first state of \(T'_{j+1}\) is \(q_i\) because we only replace states on even positions inside the subsequences. To continue, we again distinguish between two cases, depending on whether \(q_{i-1}\) was replaced or not:

• Suppose the last state of \(T'_j\) is \(q_{i-1}\). Then, using the first and second line in the definition of \(\delta_2\), the nonequality \(q_{i-1} \neq q_i\) implies \(q_i \in \delta_2(q_{i-1}, S_{i-1})\).

• Suppose the last state of \(T'_{j}\) is \(f(q_{i-1})\). Then by definition of \(\delta_2\), we have \(\delta_2(f(q_{i-1}), S_{i-1}) = \delta_1(q_{i-1}, S_{i-1})\). So, \(q_i \in \delta_1(q_{i-1}, S_{i-1})\).

Let \(r_1, \ldots, r_{n+1}\) denote the state sequence obtained by concatenating the sequences \(T'_1, \ldots, T'_k\). We note that \(r_1 = q_1^2 = q_2^2\) because \(T'_1\) has the same first state as \(T_1\), which is \(q_1^3\). Moreover, \(r_{n+1} \in F_2\) because \(q_{n+1} \in F_1\) and for any state \(q\) that is replaced by \(f(q)\) we have \(f(q) \in F_2\) if \(q \in F_1\). So, \(r_1, \ldots, r_{n+1}\) is an accepting run of \(M_2\) on the \(\epsilon\)-extension \(\beta\) of \(\alpha\).

B.2 Direction 2

Let \(\alpha\) be a string accepted by \(M_2\). We show that \(\alpha\) is accepted by \(M_1\). Because \(\alpha\) is accepted by \(M_2\), there is an \(\epsilon\)-extension \(\beta = (S_1, \ldots, S_{n+1})\) of \(\alpha\) and a sequence of states \(q_1, \ldots, q_{n+1}\) of \(M_2\) such that:

- \(q_1 = q_2^2\);
- \(q_i \in \delta_2(q_{i-1}, S_{i-1})\) for each \(i \in \{2, \ldots, n+1\}\);
- \(q_{n+1} \in F_2\).

We now convert the state sequence \(q_1, \ldots, q_{n+1}\) to a state sequence of \(M_1\). Let \(r_1, \ldots, r_{n+1}\) be the state sequence in \(M_1\) obtained from \(q_1, \ldots, q_{n+1}\) by replacing \(q_i\) with \(f^{-1}(q_i)\) if \(q_i \in Q_2 \setminus Q_1\) for each \(i \in \{1, \ldots, n+1\}\)\(^{23}\). We show that all transitions in the new state sequence are valid in \(M_1\). Suppose we replace some \(q_i\) by \(q = f^{-1}(q_i)\) with \(i \in \{2, \ldots, n\}\). By definition of \(\delta_2\), whenever \(M_2\) reads a symbol while being in a state of \(Q_2 \setminus Q_1\) then we always arrive at a state of \(Q_1\). This implies \(q_{i-1} \in Q_1\) and \(q_{i+1} \in Q_1\), and thus \(r_{i-1} = q_{i-1}\) and \(r_{i+1} = q_{i+1}\).

Note the following.

- We see that \(q \in \delta_1(q_{i-1}, S_{i-1})\): by definition of \(\delta_2\), we know that the only way to reach \(q_i = f(q)\) is from the state \(q_i\), giving \(q_{i-1} = q\); further, \(q_i \in \delta_2(q_{i-1}, S_{i-1})\), which is equivalent to \(f(q) \in \delta_2(q, S_{i-1})\), implies \(q \in \delta_1(q, S_{i-1})\).
- We see that \(q_{i+1} \in \delta_1(q, S_i)\): by definition of \(\delta_2\), we know \(\delta_2(q_i, S_i) = \delta_1(q, S_i)\); so, \(q_{i+1} \in \delta_2(q_i, S_i)\) implies \(q_{i+1} \in \delta_1(q, S_i)\).

\(^{23}\)Note that \(f^{-1}\) is defined on \(Q_2 \setminus Q_1\).
Finally, note that (i) $r_1 = q_1^1$ because $q_1 = q_2 = q_1^1 \in Q_1$; and (ii) $r_{n+1} \in F_1$ because $q_{n+1} \in F_2$ and for each $q \in Q_2 \setminus Q_1$ if $q \in F_2$ then $f^{-1}(q) \in F_1$. We conclude that $r_1, \ldots, r_{n+1}$ is an accepting run of $M_1$ on $\alpha$.

C Proof of Lemma 3.4

Let $I$ be a set of input neurons. Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^s, F_1)$ with $\Sigma_1 = \mathcal{P}(I)$ be a jump-free automaton recognizing a founded regular language over $\mathcal{P}(I)$. We now convert $M_1$ to a jump-free automaton $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^s, F_2)$ such that $M_2$ recognizes the same language as $M_1$ and where (i) $\delta_2(q_2^s, \emptyset) = \emptyset$, and (ii) $Q_2 = Q_1$.

We define:

- $Q_2 = Q_1$;
- $\Sigma_2 = \Sigma_1$;
- $q_2^s = q_1^s$;

regarding $\delta_2$,

1. $\delta_2(q_2^s, \emptyset) = \emptyset$;
2. $\delta_2(q, S) = \delta_1(q, S)$ for each $q \in Q_1$ and $S \in \Sigma_1$ with $q \neq q_1^s$ or $S \neq \emptyset$;

- $F_2 = F_1$.

We now show that $M_2$ recognizes the same language as $M_1$.

C.1 Direction 1

Let $\alpha = (S_1, \ldots, S_n)$ be a string accepted by $M_1$. We show that $\alpha$ is also accepted by $M_2$. Because $\alpha$ is accepted by $M_1$, and $M_1$ is jump-free, there is a sequence of states $q_1, \ldots, q_{n+1}$ of $M_1$ such that

- $q_1 = q_1^s$;
- $q_i \in \delta_1(q_{i-1}, S_{i-1})$ for each $i \in \{2, \ldots, n+1\}$;
- $q_{n+1} \in F_1$.

We show that the state sequence $q_1, \ldots, q_{n+1}$ is also an accepting run of $M_2$ on $\alpha$. Towards a contradiction, suppose there is some $i \in \{2, \ldots, n+1\}$ with $q_i \in \delta_1(q_{i-1}, S_{i-1})$ but $q_i \notin \delta_2(q_{i-1}, S_{i-1})$.

By definition of $\delta_2$, this implies $q_{i-1} = q_1^s$ and $S_{i-1} = \emptyset$. Now, note that the state sequence $q_{i-1}, q_i, \ldots, q_{n+1}$ is an accepting run of $M_1$ on the suffix $\beta = (S_{i-1}, \ldots, S_n)$. But since $S_{i-1} = \emptyset$, automaton $M_1$ would not recognize a founded regular language, which is a contradiction.

31
C.2 Direction 2
Let $\alpha = (S_1, \ldots, S_n)$ be a string accepted by $M_2$. We show that $\alpha$ is also accepted by $M_1$. Because $\alpha$ is accepted by $M_2$, and $M_2$ is jump-free, there is a sequence of states $q_1,\ldots, q_{n+1}$ such that

- $q_1 = q_2^s$;
- $q_i \in \delta_2(q_{i-1}, S_{i-1})$ for each $i \in \{2, \ldots, n+1\}$; and,
- $q_{n+1} \in F_2$.

Note that $q_1,\ldots, q_{n+1}$ is also an accepting run of $M_1$ on $\alpha$, because:

- $q_{s_2} = q_{s_1}$;
- $\delta_2(q, S) \subseteq \delta_1(q, S)$ for each $(q, S) \in Q_2 \times \Sigma_2 = Q_1 \times \Sigma_1$; and,
- $F_2 = F_1$.

Hence, $\alpha$ is also accepted by $M_1$.

D Proof of Claim 3.7
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 let $x$ be activated; 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 $f \in \mathbb{N}_0$:

\[
\begin{align*}
w_1 &= 1/f, \\
w_2 &= (1 - 1/f)/n.
\end{align*}
\]

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

\[
\frac{1}{f} + n \left(\frac{(1 - 1/f)}{n}\right) = \frac{1}{f} + (1 - 1/f) = 1 \geq 1.
\]

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

\[
n \left(\frac{(1 - 1/f)}{n}\right) = 1 - 1/f < 1.
\]
For Condition 1, we solve for $f$:

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

$$m/f + (1 - 1/f) \left( \frac{n - 1}{n} \right) < 1;$$

$$m + (f - 1) \left( \frac{n - 1}{n} \right) < f;$$

$$(f - 1) \left( \frac{n - 1}{n} \right) - f < -m;$$

$$f \left( \frac{n - 1}{n} \right) - f - \left( \frac{n - 1}{n} \right) < -m;$$

$$f \left( \frac{n - 1}{n} \right) - f < \left( \frac{n - 1}{n} \right) - m;$$

$$f \left( \frac{n - 1}{n} \right) < \left( \frac{n - 1}{n} \right) - m;$$

$$f > -(n - 1) + m \cdot n;$$

$$f > m \cdot n - n + 1;$$

$$f > n(m - 1) + 1.$$  

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

E Proof of Theorem 3.6

We show that $\mathcal{N}$ implements $\mathcal{B}$ with delay 1. Let $\mathcal{L}(x)$ denote the founded regular language over $\mathcal{P}(\mathcal{I})$ such that for all input strings $\alpha$ over $\mathcal{P}(\mathcal{I})$,

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

Let $\alpha$ be an input string over $\mathcal{P}(\mathcal{I})$. If $|\alpha| = 1$ then only auxiliary neurons could have been activated after reading the single symbol, giving $\mathcal{N}(\alpha) = \emptyset$. Henceforth, suppose $|\alpha| \geq 2$. Denoting $n = |\alpha|$, we show that $\mathcal{N}(\alpha) = \mathcal{B}(\alpha \rightarrow n - 1)$.

E.1 Direction 1

Suppose $x \in \mathcal{N}(\alpha)$. We show that $x \in \mathcal{B}(\alpha \rightarrow n - 1)$. We start with a sketch. We collect a chain of auxiliary neurons that represent the activation history of neuron $x$. We also simultaneously collect a string $\beta$ inside a suffix of $\alpha \rightarrow n - 1$ containing the symbols that triggered the chain of auxiliary neurons. The chain of auxiliary neurons is subsequently converted into a sequence of states of $M$, to show that $\beta$ is accepted by $M$.  

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

\footnote{Indeed, there are no connections from the input neurons to the output neuron with nonzero weight.}
Sequence of neurons  We consider the sequence of sets of activated neurons when running $\mathcal{N}$ on $\alpha$:

$$N_1, \ldots, N_n, N_{n+1}.$$ 

By definition of $\mathcal{N}(\alpha)$, we know $x \in N_{n+1}$. It is possible to consider a maximal sequence of neurons $y_1, \ldots, y_{k+1}$ such that

- $y_{k+1} = x$;
- $y_1, \ldots, y_{k+1}$ is embedded into the last $k + 1$ sets of $N_1, \ldots, N_{n+1}$, i.e., $y_i \in N_{n-k+i}$ for each $i \in \{1, \ldots, k + 1\}$;\footnote{Here, $n - k + i$ abbreviates the expression $n + 1 - (k + 1) + i$.}
- $y_{i-1} \in \text{pre}(y_i)$ for each $i \in \{2, \ldots, k + 1\}$.

This sequence satisfies the following properties:

- $k \geq 1$ because the activation of neuron $x$ requires the activation of at least one presynaptic (auxiliary) neuron of $x$;
- $y_i \in \mathcal{A}$ for each $i \in \{1, \ldots, k\}$ because $N_j \subseteq \mathcal{O} \cup \mathcal{A}$ for each $j \in \{1, \ldots, n + 1\}$ and no neuron has $x$ as presynaptic neuron;
- $y_1 \in \mathcal{T}$ because the sequence $y_1, \ldots, y_{k+1}$ is chosen to be maximal.\footnote{Towards a contradiction, suppose $y_1 \notin \mathcal{T}$. So, $y_1 \in \mathcal{A} \setminus \mathcal{T}$. Since, $\text{con}(y_1) \neq \emptyset$, we have made $y_1$ dependent upon other auxiliary neurons $z$, one of which must have been active before $y_1$ and hence could be added at the front of the sequence $y_1, \ldots, y_{k+1}$. But then $y_1, \ldots, y_{k+1}$ is not maximal, which contradicts our assumption.}

String and accepting run  Because $y_i \in \mathcal{A}$ for each $i \in \{1, \ldots, k\}$, we may denote $y_i = (q_i, S_i)$. Now, based on the resulting sequence

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

let $\beta$ be the following string over $\mathcal{P}(\mathcal{I})$:

$$\beta = S_1, \ldots, S_k.$$ 

We argue that $\alpha_{\alpha_{n-1}}$ embeds $\beta$. Fix some $i \in \{1, \ldots, k\}$. We show that $S_i \subseteq \alpha_{n-1-k+i}$ (or equivalently $S_i \subseteq \alpha_{n-k+i-1}$). First, note that the symbol $\alpha_{n-k+i-1}$ is defined because always $N_i = \emptyset$ by the operational semantics, so $(q_i, S_i) \in N_{n-k+i}$ implies $n - k + i \geq 2$. We distinguish between the following cases:

- if $(q_i, S_i) \in \mathcal{T}$ then by definition of $\mathcal{W}$, the activation $(q_i, S_i) \in N_{n-k+i}$ implies $S_i \subseteq \alpha_{n-k+i-1}$;
- if $(q_i, S_i) \in \mathcal{A} \setminus \mathcal{T}$ with $S_i = \emptyset$ then surely $S_i \subseteq \alpha_{n-k+i-1}$;
- if $(q_i, S_i) \in \mathcal{A} \setminus \mathcal{T}$ with $S_i \neq \emptyset$ then by definition of $\mathcal{W}$ and Claim 3.7, the activation $(q_i, S_i) \in N_{n-k+i}$ implies $S_i \subseteq \alpha_{n-k+i-1}$.\footnote{Towards a contradiction, suppose $y_1 \notin \mathcal{T}$. So, $y_1 \in \mathcal{A} \setminus \mathcal{T}$. Since, $\text{con}(y_1) \neq \emptyset$, we have made $y_1$ dependent upon other auxiliary neurons $z$, one of which must have been active before $y_1$ and hence could be added at the front of the sequence $y_1, \ldots, y_{k+1}$. But then $y_1, \ldots, y_{k+1}$ is not maximal, which contradicts our assumption.}
We show that $M$ accepts $\beta$. This then gives $\beta \in \mathcal{L}(x)$, which, together with the embedding of $\beta$ into $\alpha \rightarrow n_{-1}$, implies $x \in B(\alpha \rightarrow n_{-1})$ as desired. Also based on the above sequence of auxiliary neurons, we consider the following sequence of states of $M$:

$$q^*, q_1, \ldots, q_k,$$

where $q^*$ denotes the start state of $M$. We see that $q^*, q_1, \ldots, q_k$ is an accepting run of $M$ on $\beta$:

- we have $q_1 \in \delta(q^*, S_1)$ because $(q_1, S_1) \in T$ (see above);
- 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 $(q_i, S_i)$, which, by definition of $\mathcal{W}$, implies $(q_{i-1}, S_{i-1}) \in \text{con}(q_i, S_i)$, and thus $q_i \in \delta(q_{i-1}, S_i)$;
- $q_k \in F$ because $(q_k, S_k) \in \text{pre}(x)$.

### E.2 Direction 2

Suppose $x \in B(\alpha \rightarrow n_{-1})$. We show that $x \in \mathcal{N}(\alpha)$.

**String and accepting run** Because $B$ is monotone-regular, $x \in B(\alpha \rightarrow n_{-1})$ implies there is a string $\beta \in \mathcal{L}(x)$ that is embedded into $\alpha \rightarrow n_{-1}$. Denote

$$\beta = (S_1, \ldots, S_k).$$

Next, $\beta \in \mathcal{L}(x)$ implies there is an accepting run of automaton $M$ on input $\beta$:

$$q^*, q_1, \ldots, q_k.$$

The string $\beta$ can be chosen so that $q^* \notin \{q_1, \ldots, q_k\}$, and this will henceforth be our assumption: if $q^* = q_i$ 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 would instead focus on the smaller string $\beta'$ that is also embedded into $\alpha \rightarrow n_{-1}$.

**Sequence of neurons** We convert the sequence of states $q_1, \ldots, q_k$ and the string $\beta = (S_1, \ldots, S_k)$ from above into the following sequence of auxiliary neurons of $\mathcal{N}$:

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

Note that these are valid auxiliary neurons in $\mathcal{N}^{\geq 3}$. We also consider the sequence of sets of activated neurons when running $\mathcal{N}$ on input $\alpha$:

$$N_1, \ldots, N_n, N_{n+1}.$$ We show that the sequence $(q_1, S_1), \ldots, (q_k, S_k)$ is embedded into the last $k$ sets of $N_1, \ldots, N_n$, so just before $N_{n+1}$. More formally, we show by induction on $i \in \{1, \ldots, k\}$ that $(q_i, S_i) \in N_{n-k+i}$. For $i = k$ this gives $(q_k, S_k) \in N_n$. Now, because $q^*, q_1, \ldots, q_k$ is an accepting run, which implies $q_k \in F$, we have defined $\mathcal{W}((q_k, S_k), x) = 1$. So, $(q_k, S_k) \in N_n$ implies $x \in N_{n+1}$. Hence, $x \in \mathcal{N}(\alpha)$, as desired.

Before we continue, note that the embedding of $\beta$ into $\alpha \rightarrow n_{-1}$ concretely means $S_i \subseteq \alpha_{n-1-k+i}$ for each $i \in \{1, \ldots, k\}$. We now proceed with the inductive proof.

---

23Indeed, (i) $q^* \notin \{q_1, \ldots, q_k\}$ by assumption; and, (ii) because $q^*, q_1, \ldots, q_k$ is an accepting run, we have $q_1 \in \delta(q^*, S_1)$ and $q_i \in \delta(q_{i-1}, S_i)$ for each $i \in \{2, \ldots, k\}$.
Base case We show that \((q_1, S_1) \in N_{n-k+1}\). We know \(q_1 \in \delta(q^*, S_1)\). So, \((q_1, S_1)\) is a trigger neuron. By definition of \(\mathcal{W}\), neuron \((q_1, S_1)\) is activated when all input neurons in \(S_1\) are activated. Now, since \(S_1 \subseteq \alpha_{n-1-k+1} = \alpha_{n-k}\) (see above), we obtain \((q_1, S_1) \in N_{n-k+1}\).

Inductive step We assume \((q_i, S_i) \in N_{n-k+i}\), where \(i \in \{1, \ldots, k-1\}\). We now show that \((q_{i+1}, S_{i+1}) \in N_{n-k+i+1}\). We distinguish between two cases, depending on whether \((q_{i+1}, S_{i+1})\) is a trigger neuron or not.

- Suppose \((q_{i+1}, S_{i+1})\) is a trigger neuron. By definition of \(\mathcal{W}\), neuron \((q_{i+1}, S_{i+1})\) is activated when all input neurons in \(S_{i+1}\) are activated. Since \(S_{i+1} \subseteq \alpha_{n-1-k+i+1} = \alpha_{n-k+i}\) (see above), we obtain \((q_{i+1}, S_{i+1}) \in N_{n-k+i+1}\).

- Suppose \((q_{i+1}, S_{i+1})\) is not a trigger neuron. Because \(q_{i+1} \in \delta(q_i, S_i)\) by definition of accepting run, we have \((q_i, S_i) \in \text{con}(q_{i+1}, S_{i+1})\). We consider two cases:
  - Suppose \(S_{i+1} = \emptyset\). We have defined \(\mathcal{W}((q_i, S_i), (q_{i+1}, S_{i+1})) = 1\). So, \((q_i, S_i) \in N_{n-k+i}\) from the induction hypothesis readily implies \((q_{i+1}, S_{i+1}) \in N_{n-k+i+1}\).
  - Suppose \(S_{i+1} \neq \emptyset\). Denote \(a = |\text{con}(q_{i+1}, S_{i+1})|\) and \(b = |S_{i+1}|\). Note that \(a > 0\) and \(b > 0\). We have defined
    \[
    \mathcal{W}((q_i, S_i), (q_{i+1}, S_{i+1})) = w_1(a, b),
    \]
    and for each \(z \in S_{i+1}\),
    \[
    \mathcal{W}(z, (q_{i+1}, S_{i+1})) = w_2(a, b).
    \]
    Now, we again use \(S_{i+1} \subseteq \alpha_{n-1-k+i+1} = \alpha_{n-k+i}\) (see above). Combined with the induction hypothesis \((q_i, S_i) \in N_{n-k+i}\), and letting \(Y\) denote the set of presynaptic neurons of \((q_{i+1}, S_{i+1})\), we have
    \[
    \sum_{y \in Y \cap (N_{n-k+i} \cup \alpha_{n-k+i})} \mathcal{W}(y, (q_{i+1}, S_{i+1})) \geq w_1(a, b) + b \cdot w_2(a, b).
    \]
    The sum \(w_1(a, b) + b \cdot w_2(a, b)\) is at least 1 by Claim \([3.7]\) Hence, \((q_{i+1}, S_{i+1}) \in N_{n-k+i+1}\).

\(^{29}\)Note that this is actually a special case, because the activation of trigger neuron \((q_{i+1}, S_{i+1})\) renders the activation of the previous neurons \((q_1, S_1), \ldots, (q_i, S_i)\) superfluous.