Computational Capabilities of Recurrent Neural Networks

Jérémie Cabessa

Department of Mathematical Economics
University of Paris 2
France

1 December 2014

Introduction

- Artificial neural networks have a tremendous range of applications in current artificial intelligence, mainly due to their capability to implement efficient learning algorithms.
- However, the theoretical approach to neural computation is rather limited.
- Here, we provide a review of some important theoretical results concerning the computational capabilities of various kinds of neural models.
- It hopes to shed a light on the crucial issue of information processing in the brain, and ultimately, on biological and artificial intelligences.

Introduction

- Artificial neural networks have a tremendous range of applications in current artificial intelligence, mainly due to their capability to implement efficient learning algorithms.
- However, the theoretical approach to neural computation is rather limited.
- Here, we provide a review of some important theoretical results concerning the computational capabilities of various kinds of neural models.
- It hopes to shed a light on the crucial issue of information processing in the brain, and ultimately, on biological and artificial intelligences.

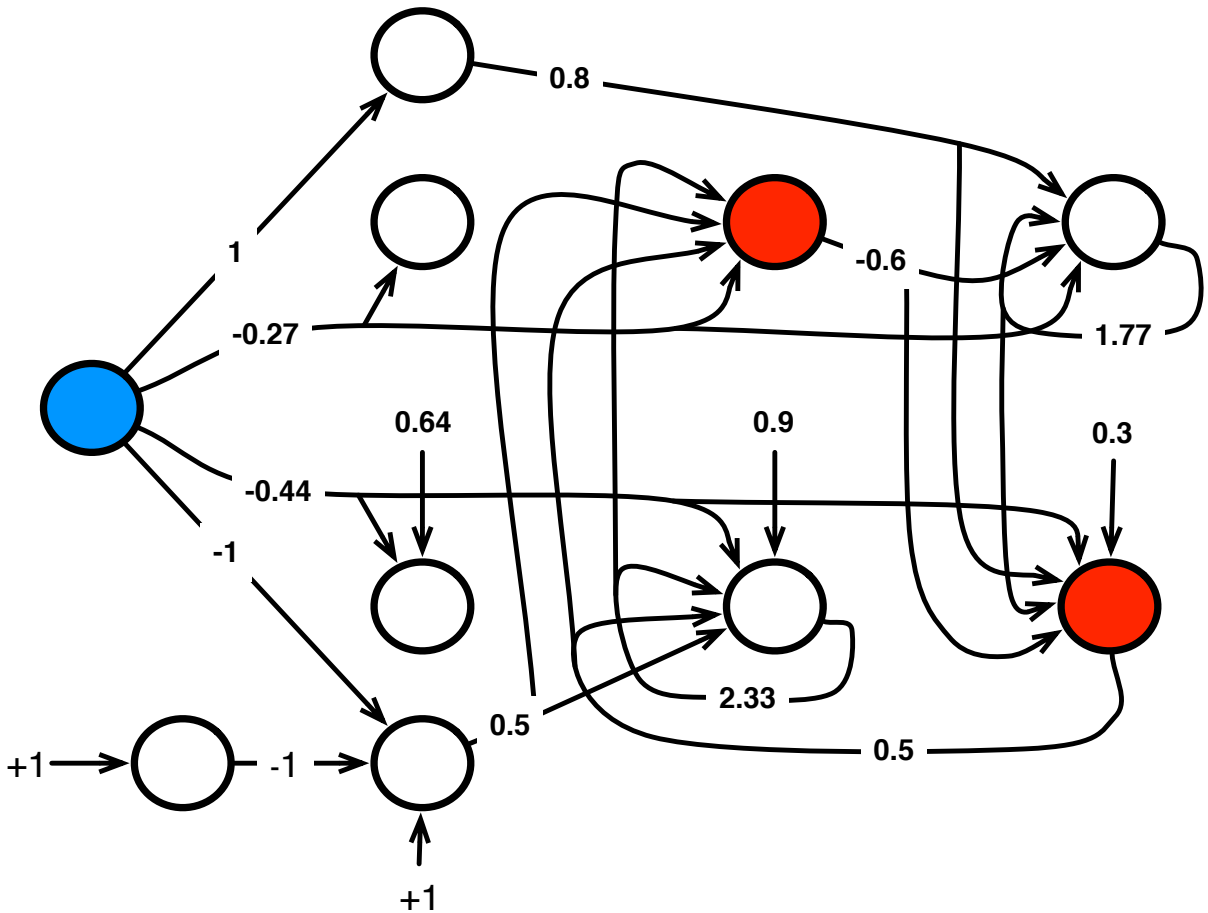
Introduction

- Artificial neural networks have a tremendous range of applications in current artificial intelligence, mainly due to their capability to implement efficient learning algorithms.
- However, the theoretical approach to neural computation is rather limited.
- Here, we provide a review of some important theoretical results concerning the computational capabilities of various kinds of neural models.
- It hopes to shed a light on the crucial issue of information processing in the brain, and ultimately, on biological and artificial intelligences.

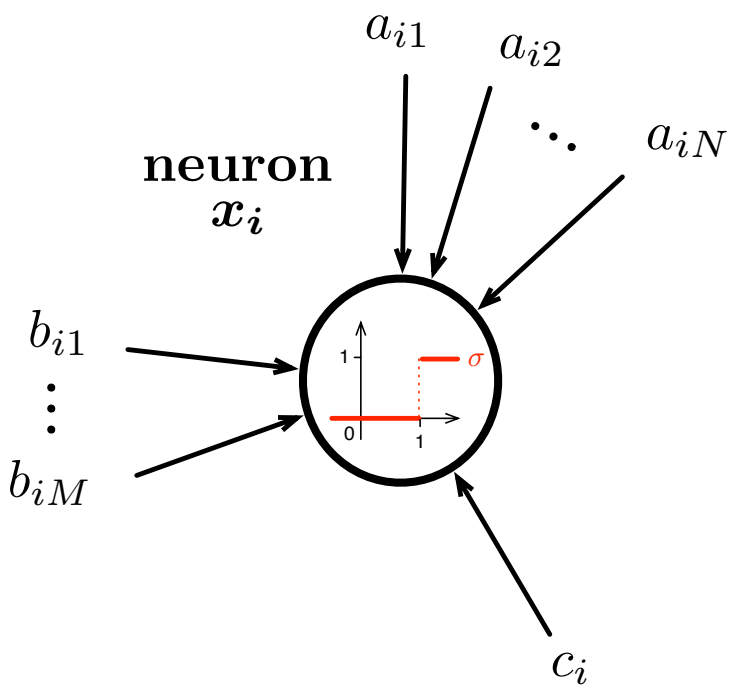
Introduction

- Artificial neural networks have a tremendous range of applications in current artificial intelligence, mainly due to their capability to implement efficient learning algorithms.
- However, the theoretical approach to neural computation is rather limited.
- Here, we provide a review of some important theoretical results concerning the computational capabilities of various kinds of neural models.
- It hopes to shed a light on the crucial issue of information processing in the brain, and ultimately, on biological and artificial intelligences.

Boolean Recurrent Neural Networks

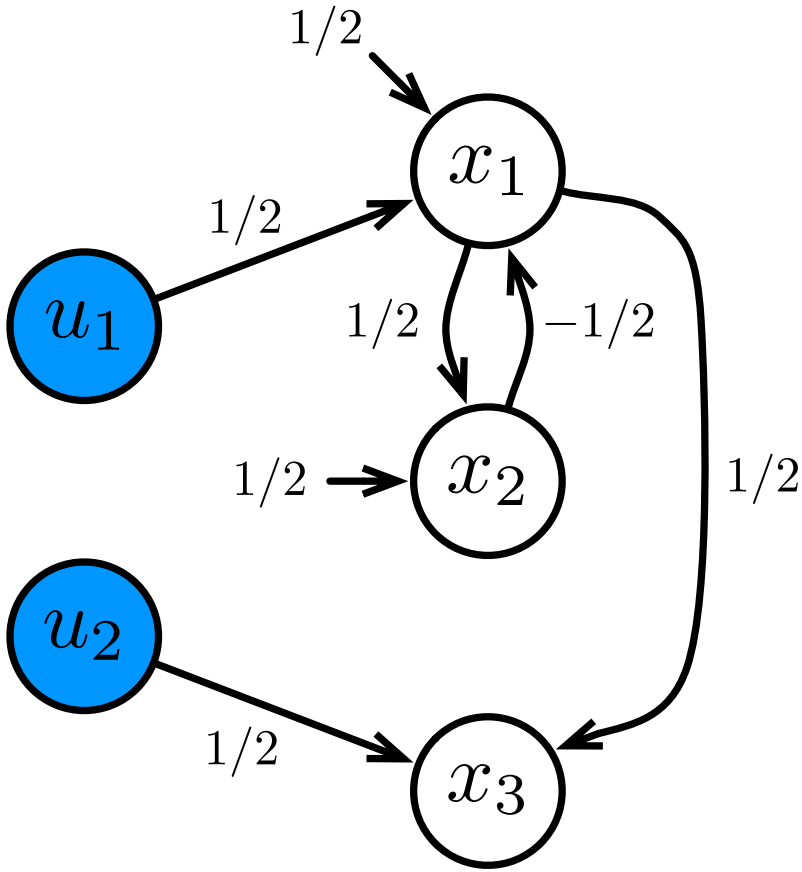


Dynamics of Boolean RNNs

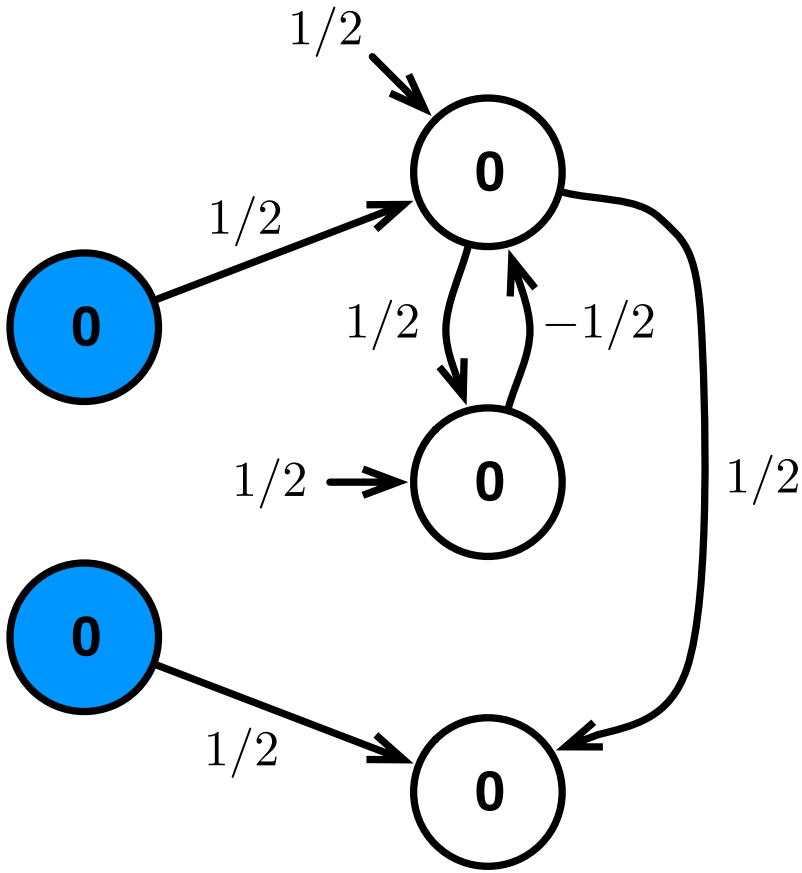


$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

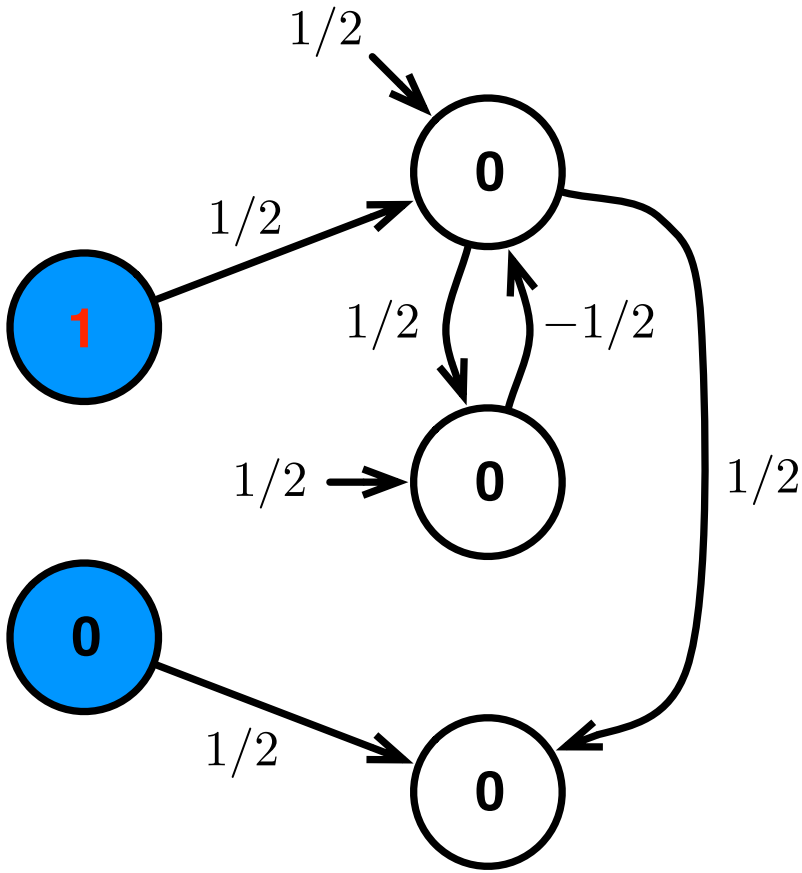
Dynamics of Boolean RNNs



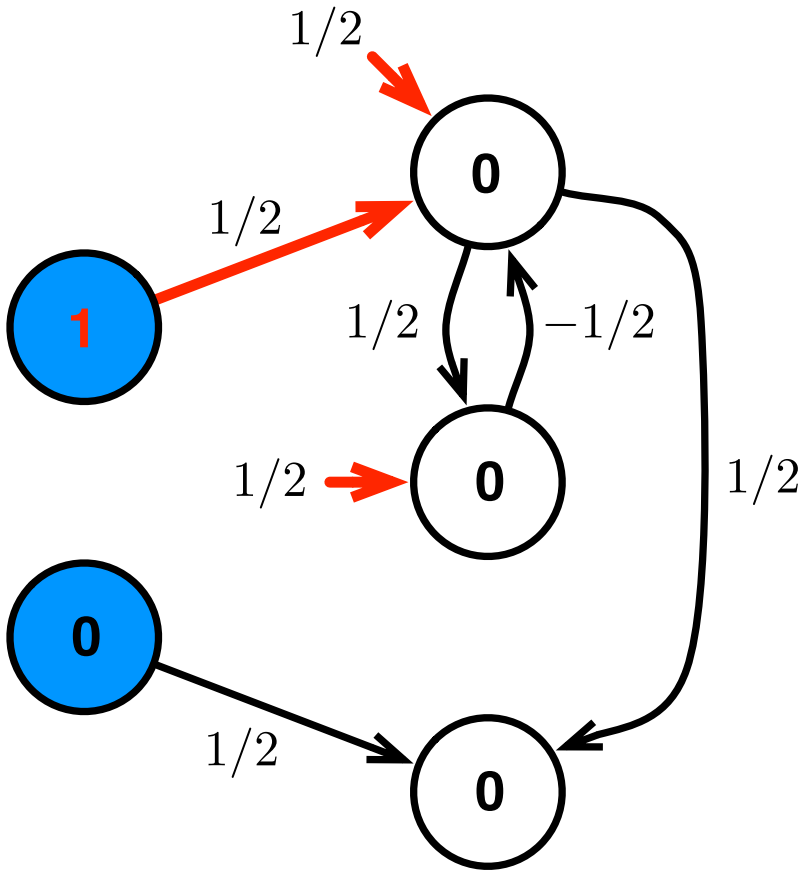
Dynamics of Boolean RNNs



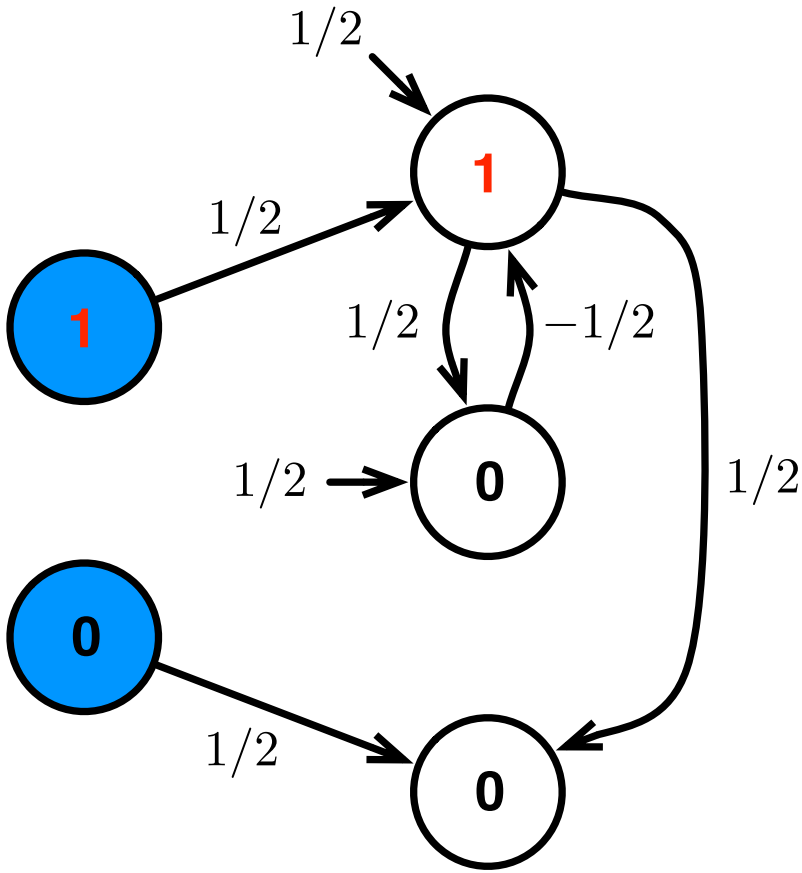
Dynamics of Boolean RNNs



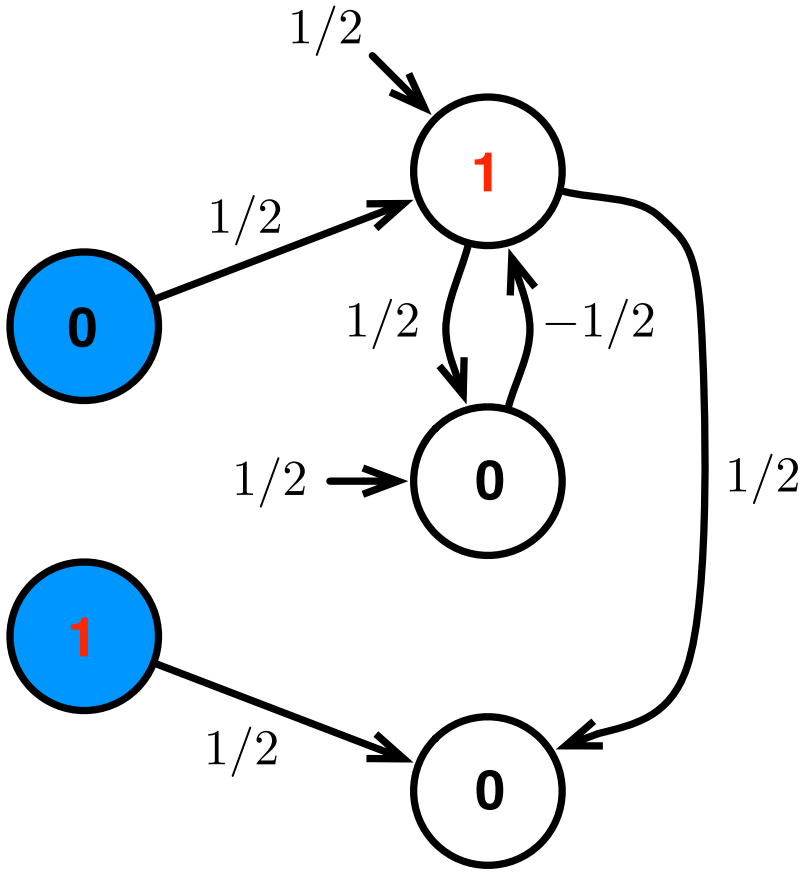
Dynamics of Boolean RNNs



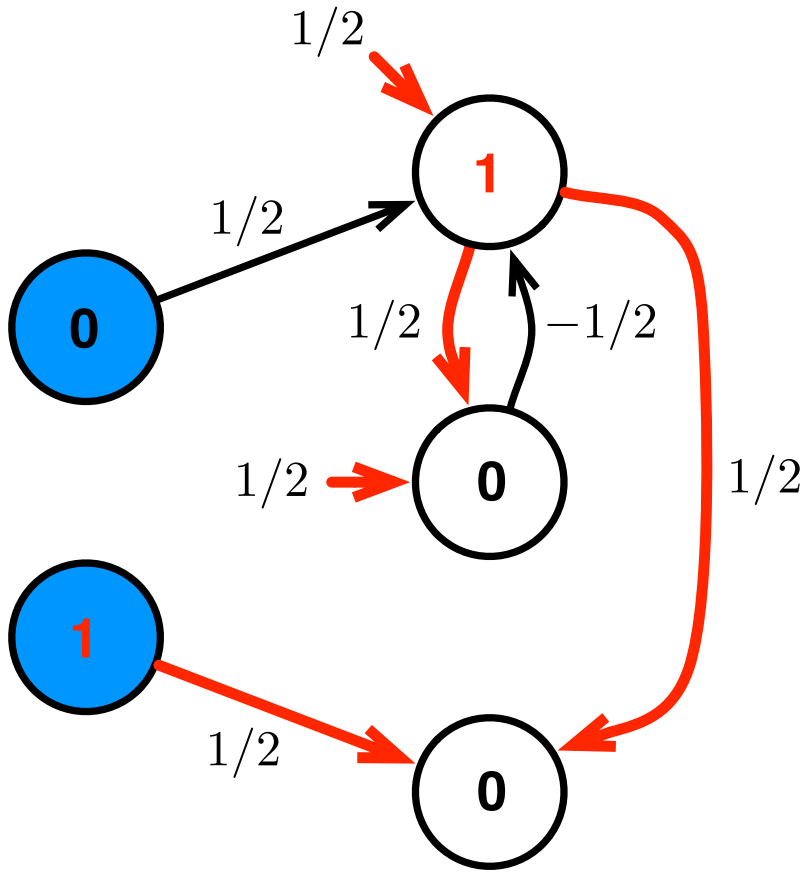
Dynamics of Boolean RNNs



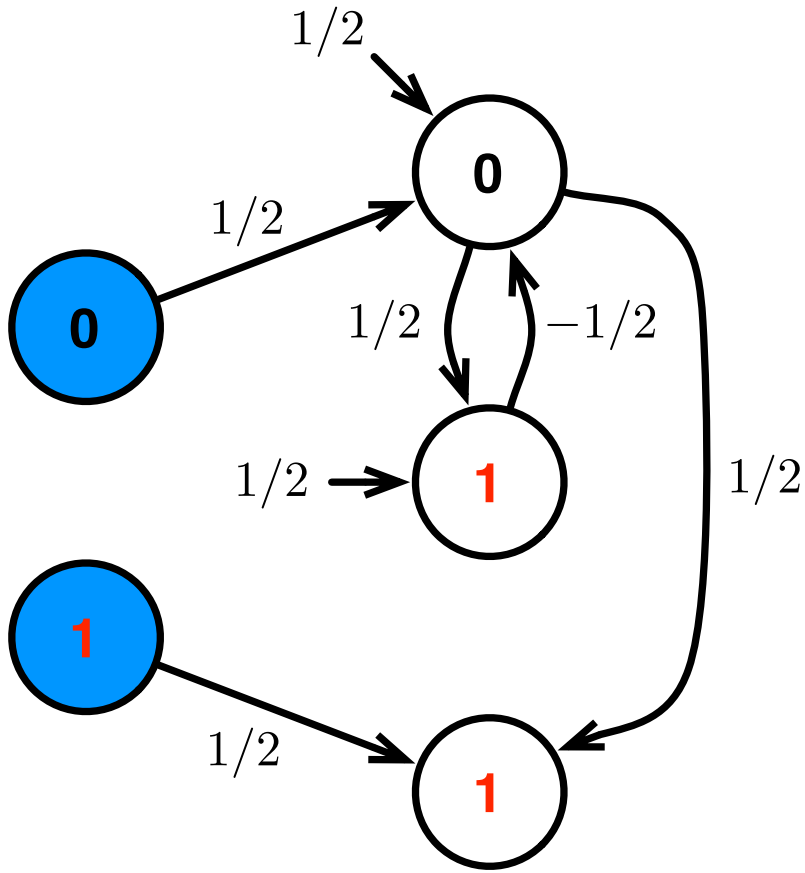
Dynamics of Boolean RNNs



Dynamics of Boolean RNNs

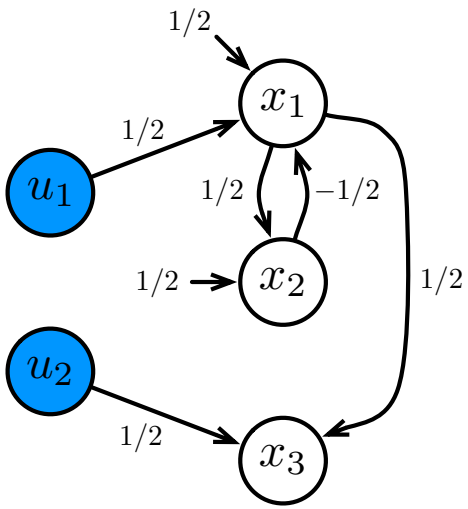


Dynamics of Boolean RNNs



From Boolean Neural Networks to Automata

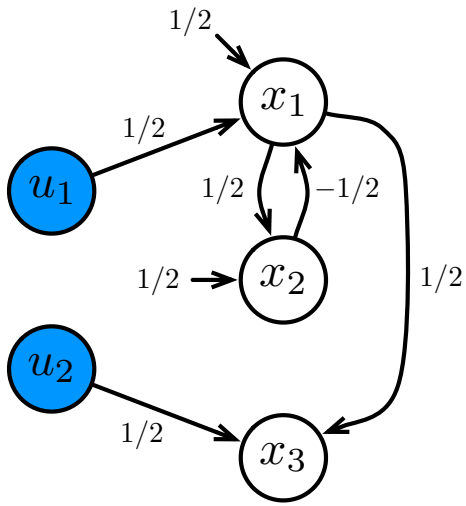
Boolean Neural Network



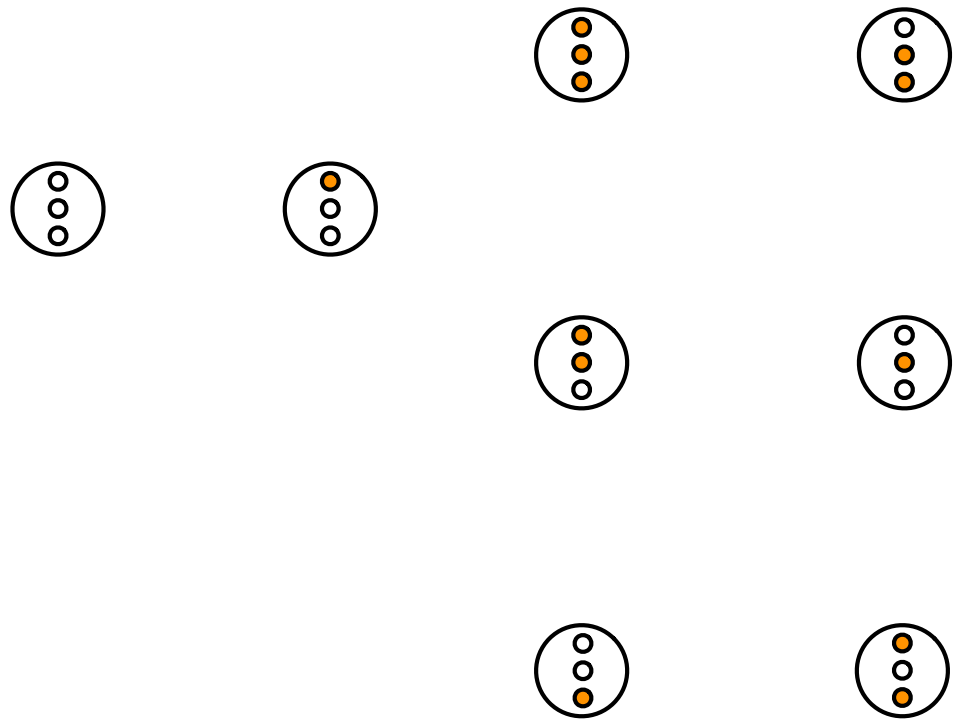
Automaton

From Boolean Neural Networks to Automata

Boolean Neural Network

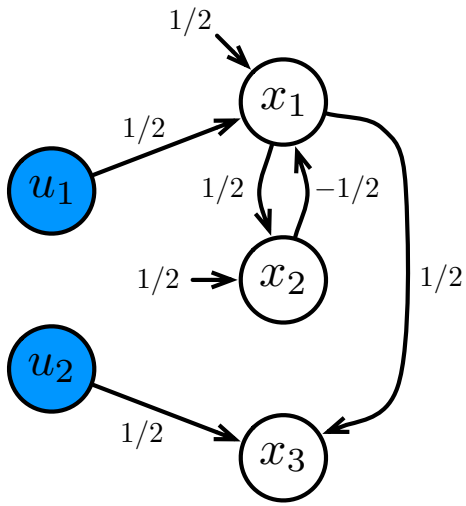


Automaton

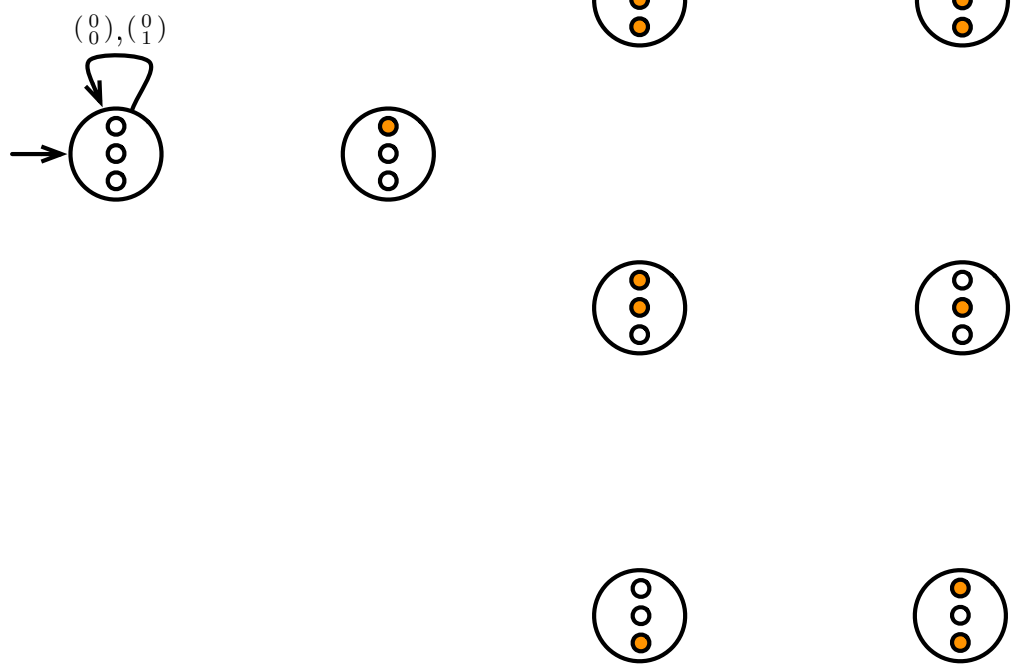


From Boolean Neural Networks to Automata

Boolean Neural Network

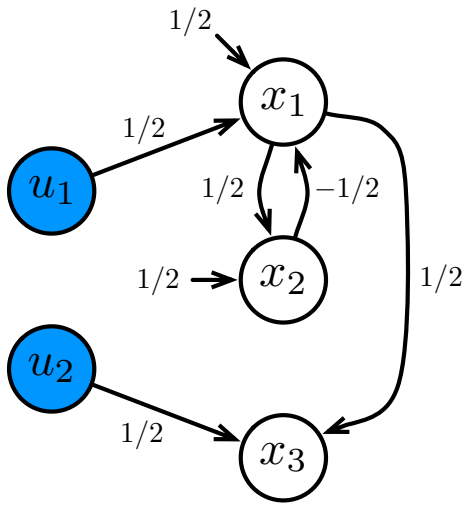


Automaton

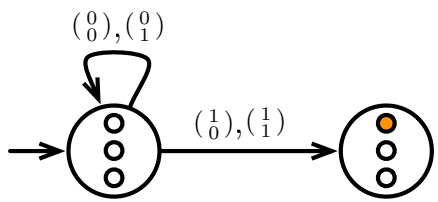


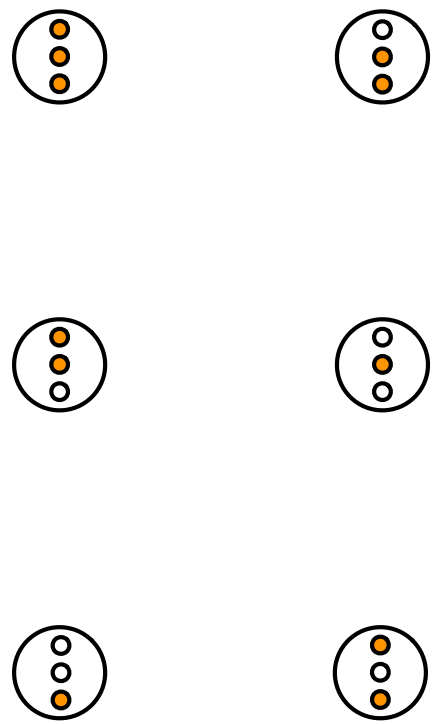
From Boolean Neural Networks to Automata

Boolean Neural Network



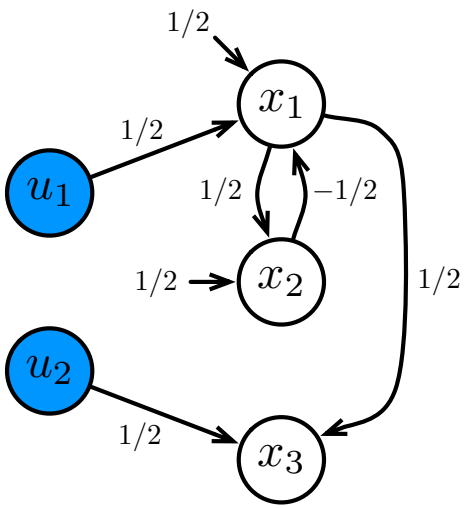
Automaton



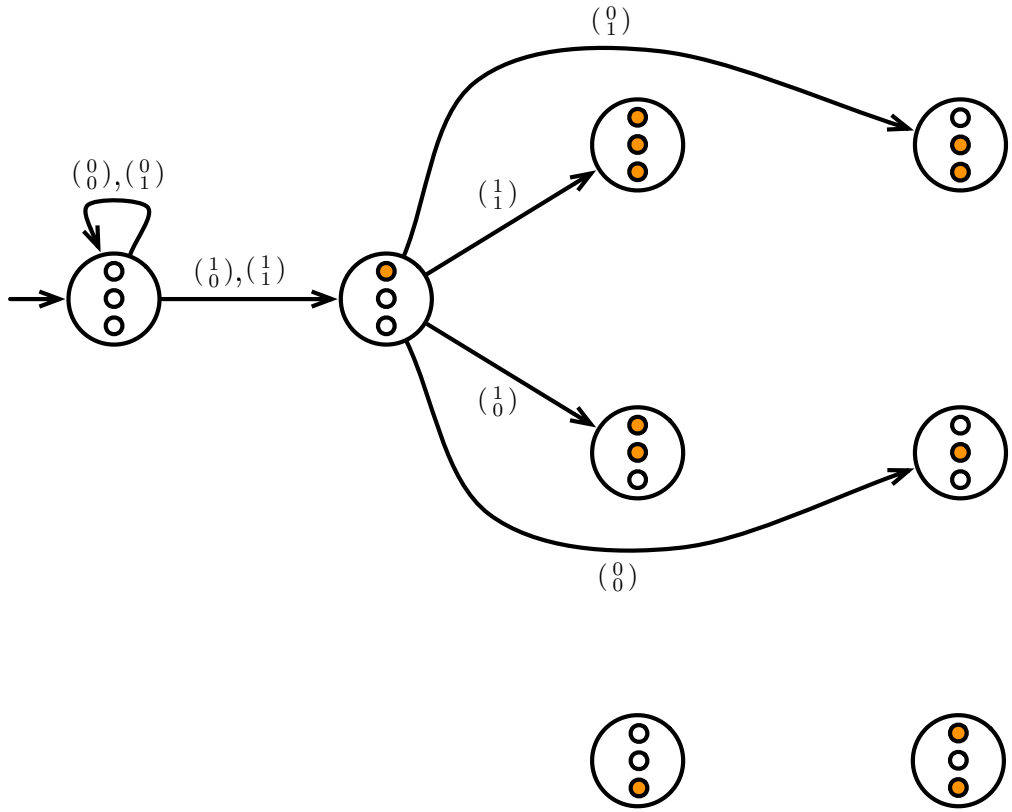


From Boolean Neural Networks to Automata

Boolean Neural Network

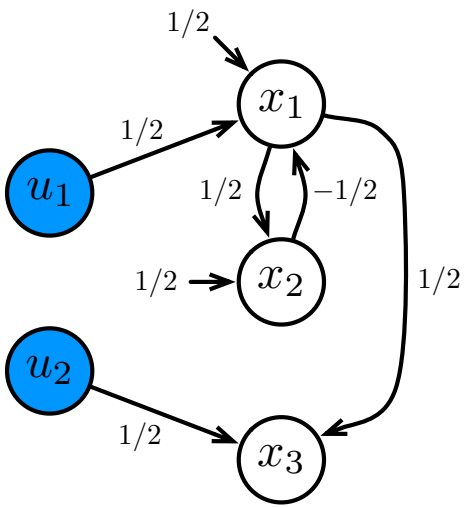


Automaton

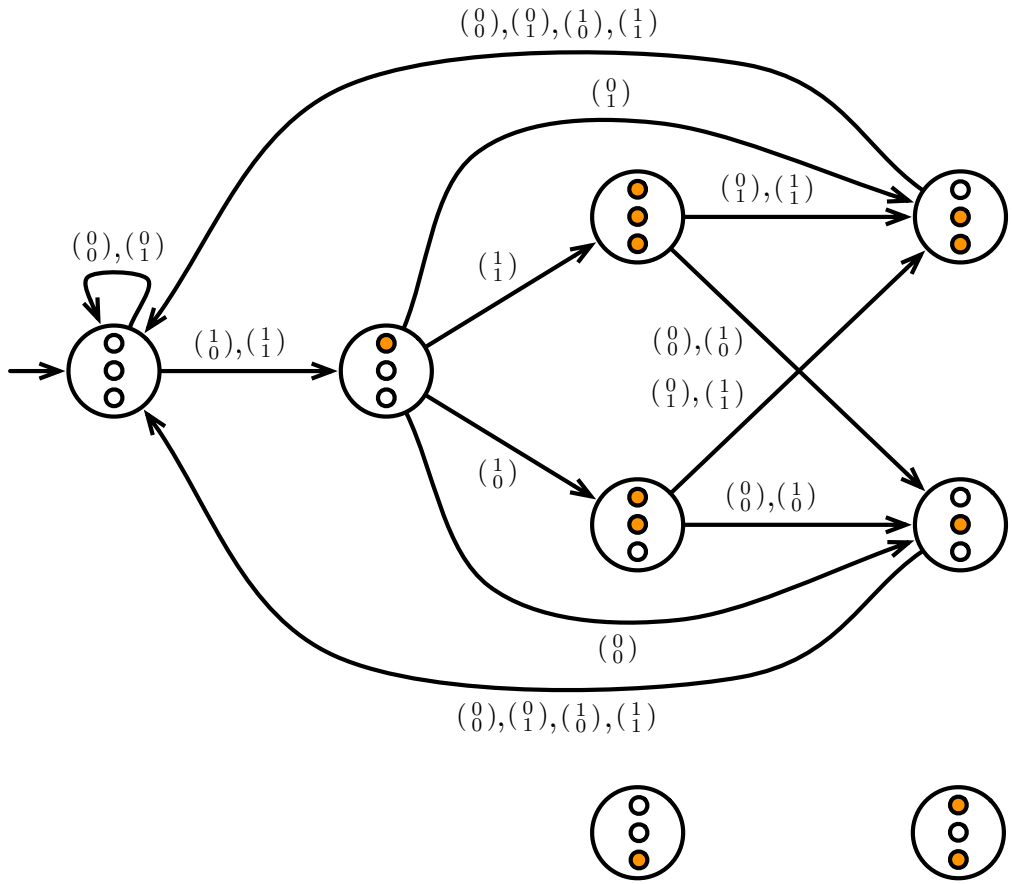


From Boolean Neural Networks to Automata

Boolean Neural Network

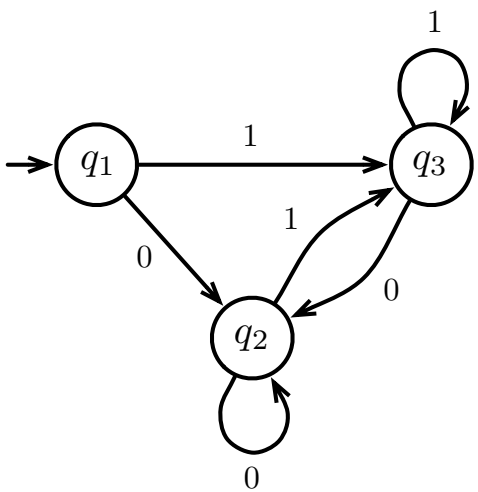


Automaton



From Automata to Boolean Neural Networks

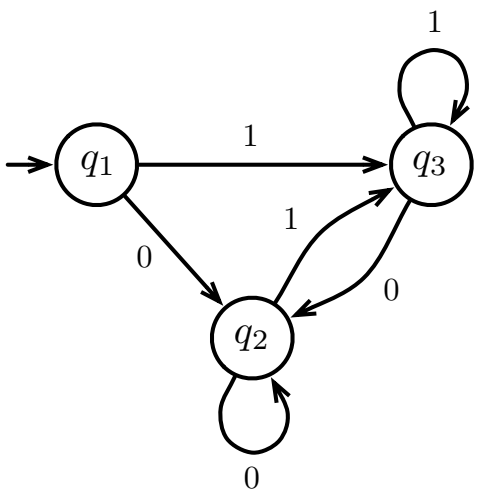
Automaton



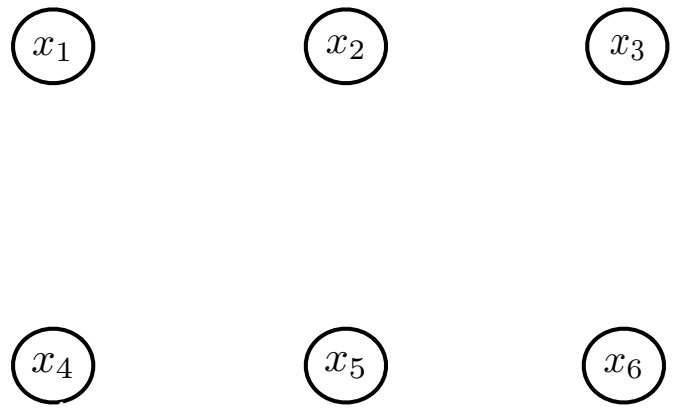
Boolean Neural Network

From Automata to Boolean Neural Networks

Automaton

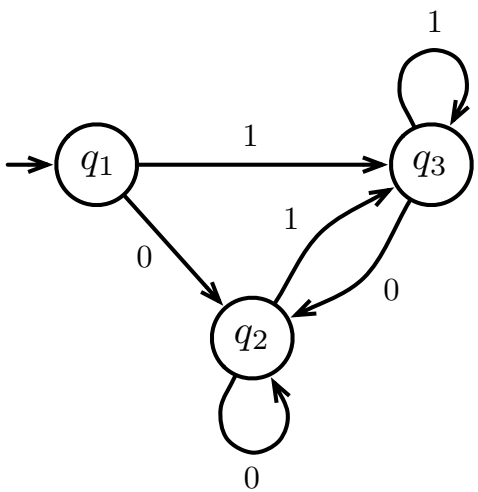


Boolean Neural Network

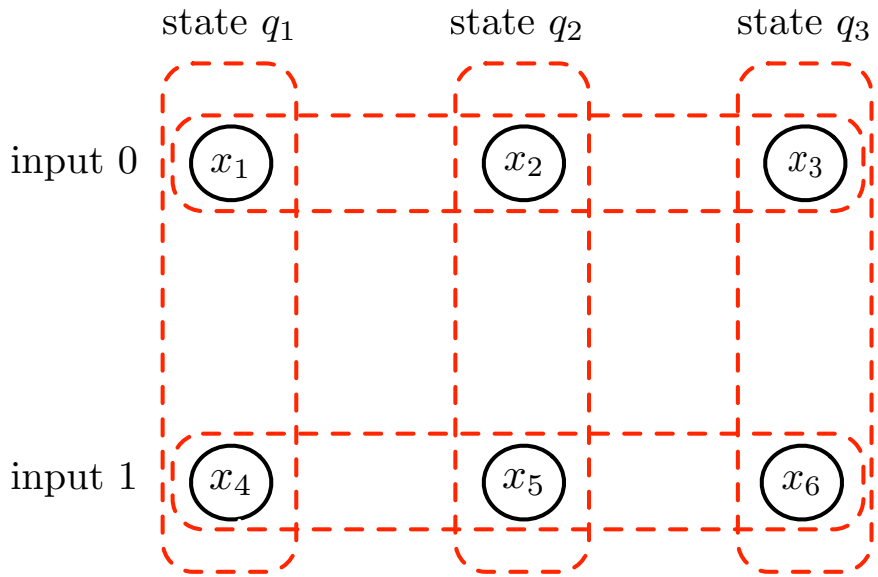


From Automata to Boolean Neural Networks

Automaton

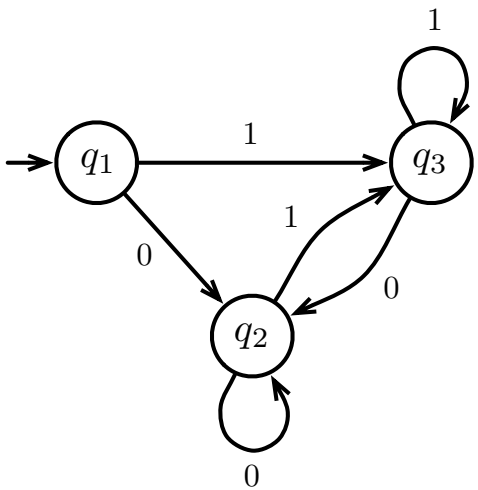


Boolean Neural Network

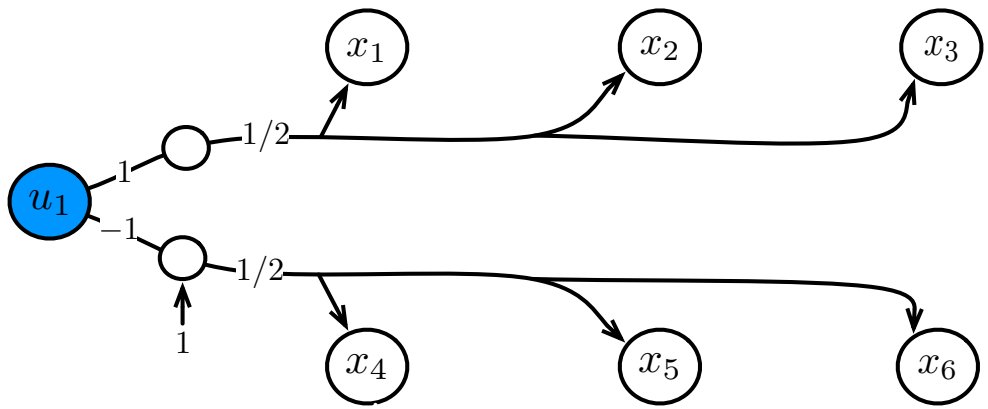


From Automata to Boolean Neural Networks

Automaton

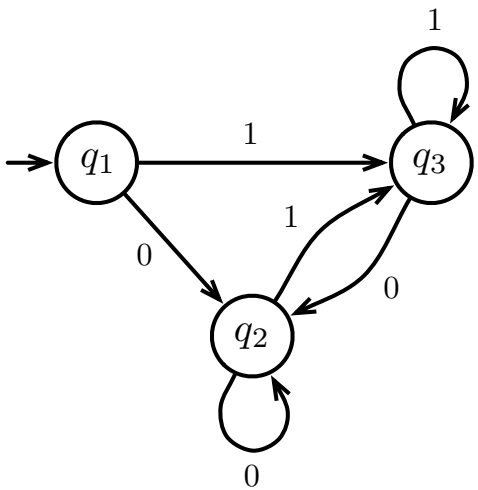


Boolean Neural Network

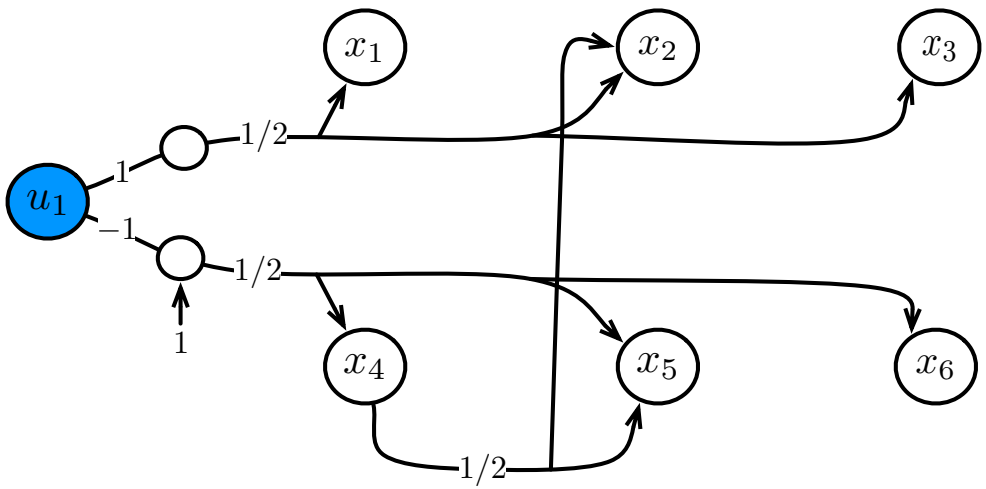


From Automata to Boolean Neural Networks

Automaton

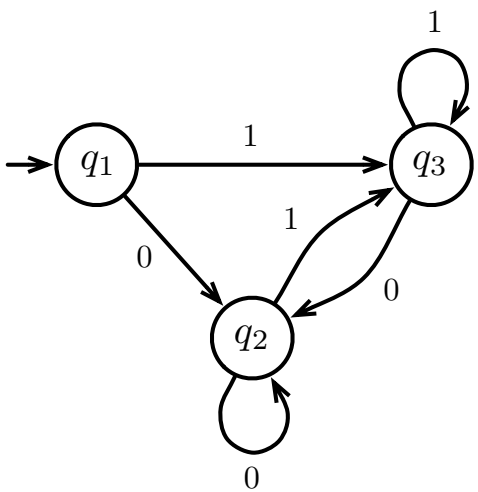


Boolean Neural Network

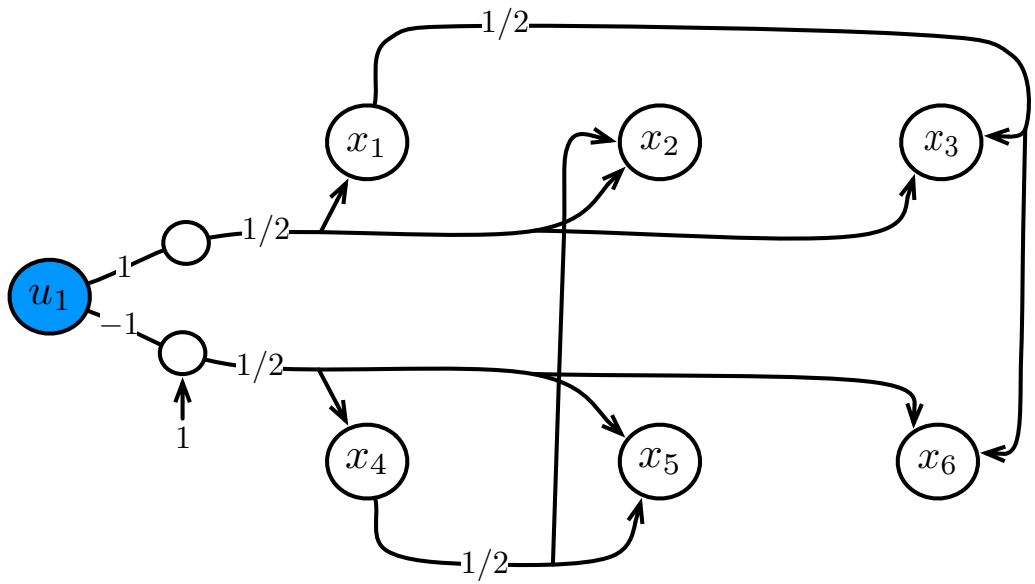


From Automata to Boolean Neural Networks

Automaton

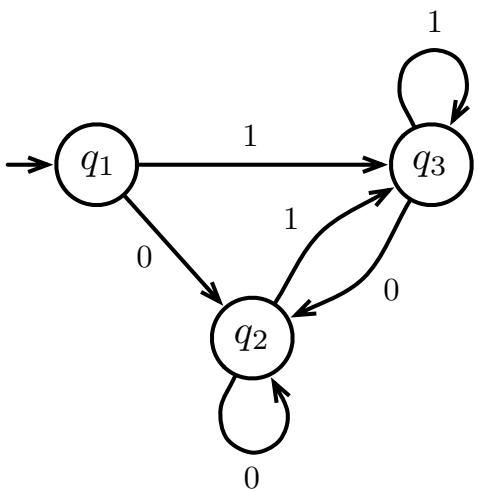


Boolean Neural Network

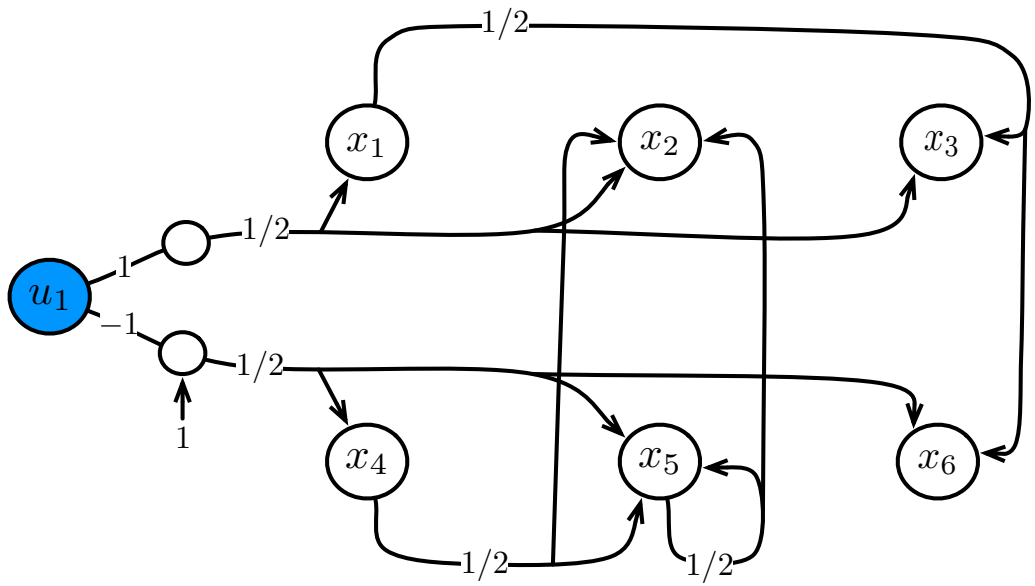


From Automata to Boolean Neural Networks

Automaton

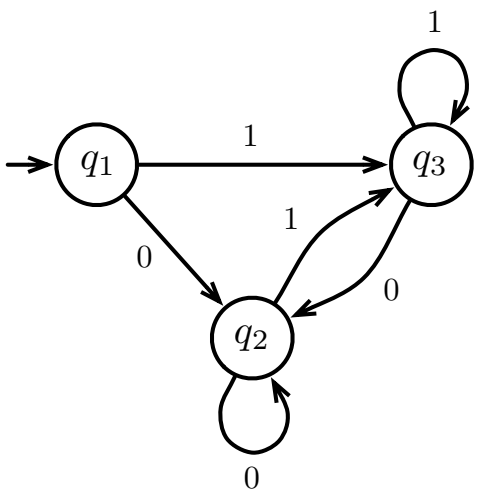


Boolean Neural Network

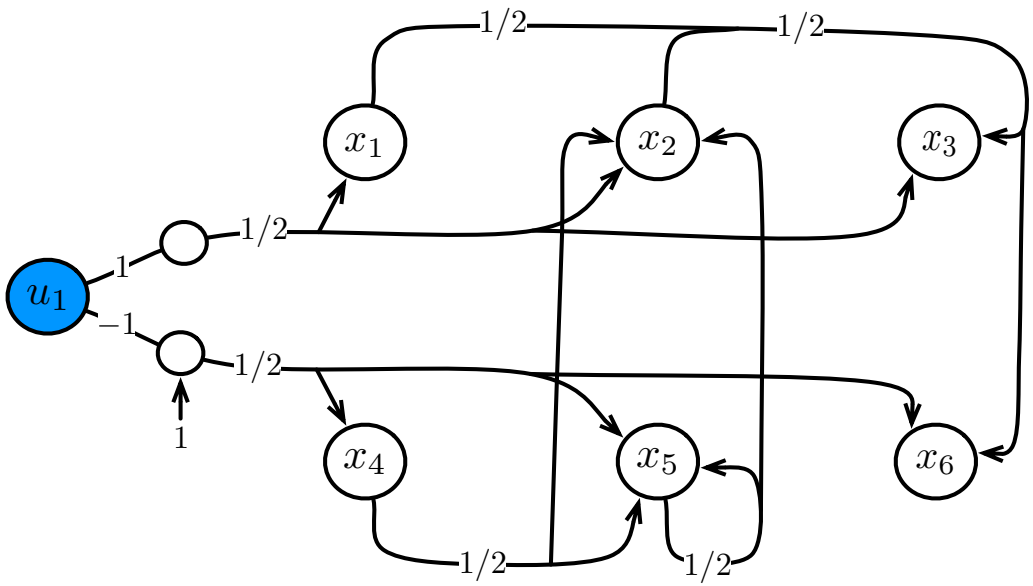


From Automata to Boolean Neural Networks

Automaton

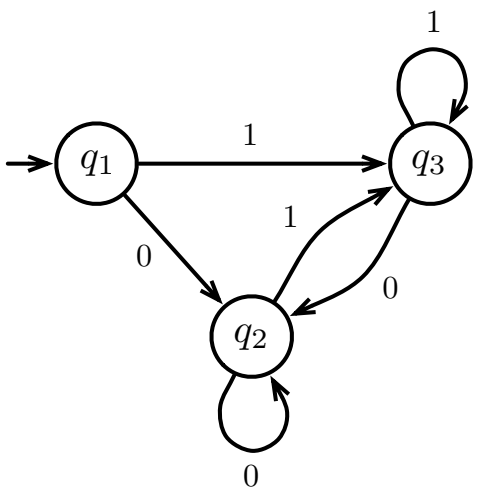


Boolean Neural Network

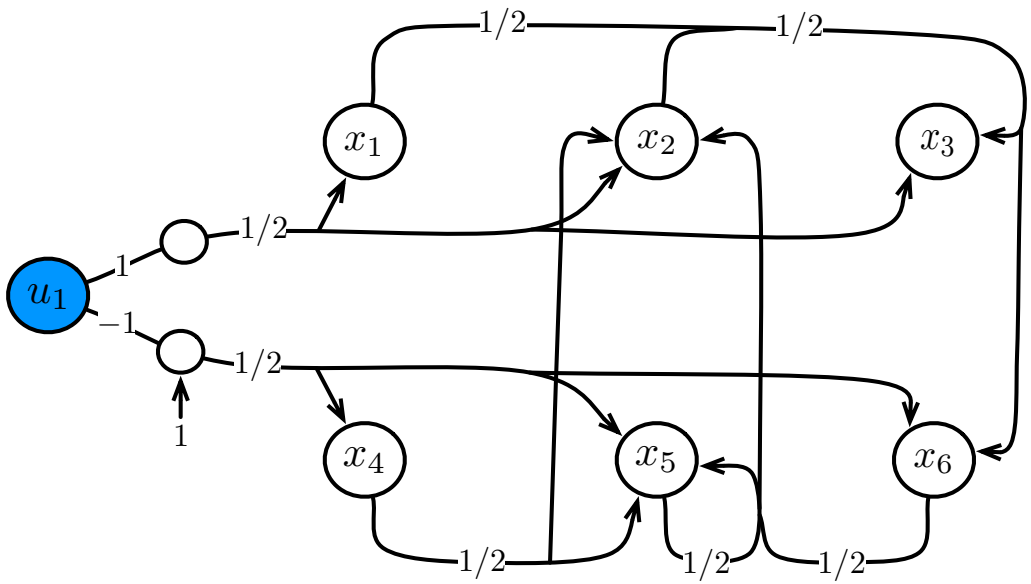


From Automata to Boolean Neural Networks

Automaton

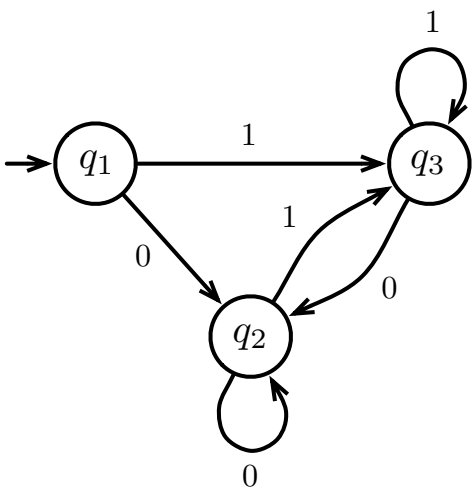


Boolean Neural Network

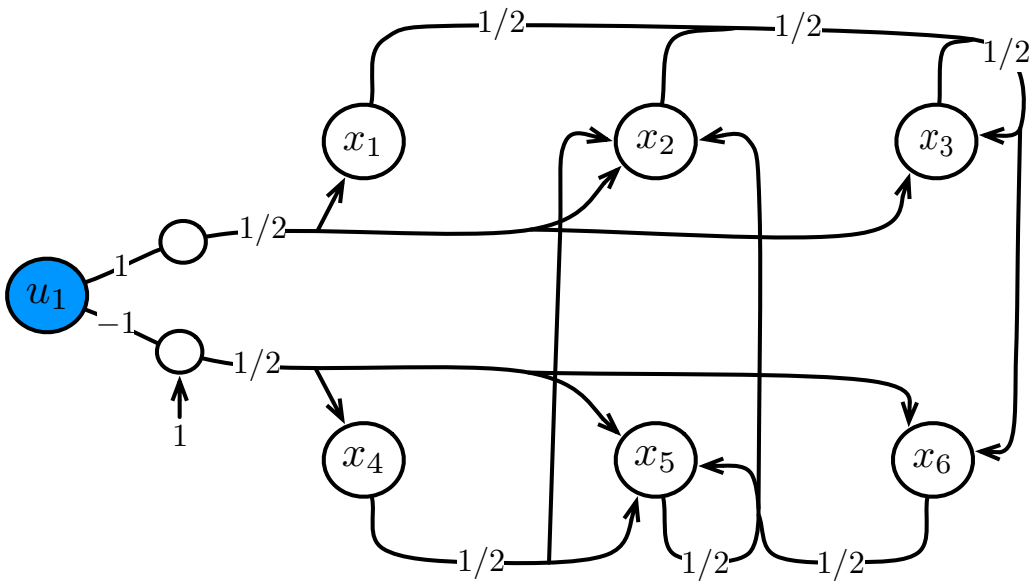


From Automata to Boolean Neural Networks

Automaton

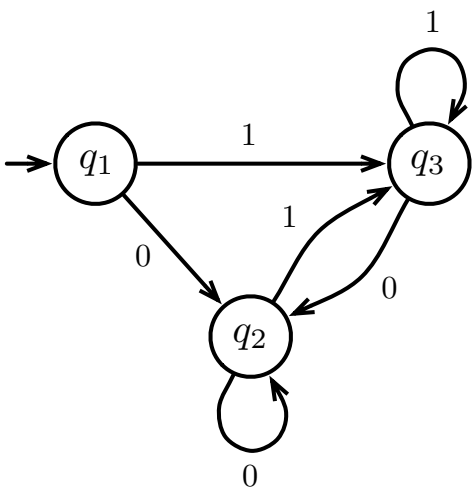


Boolean Neural Network

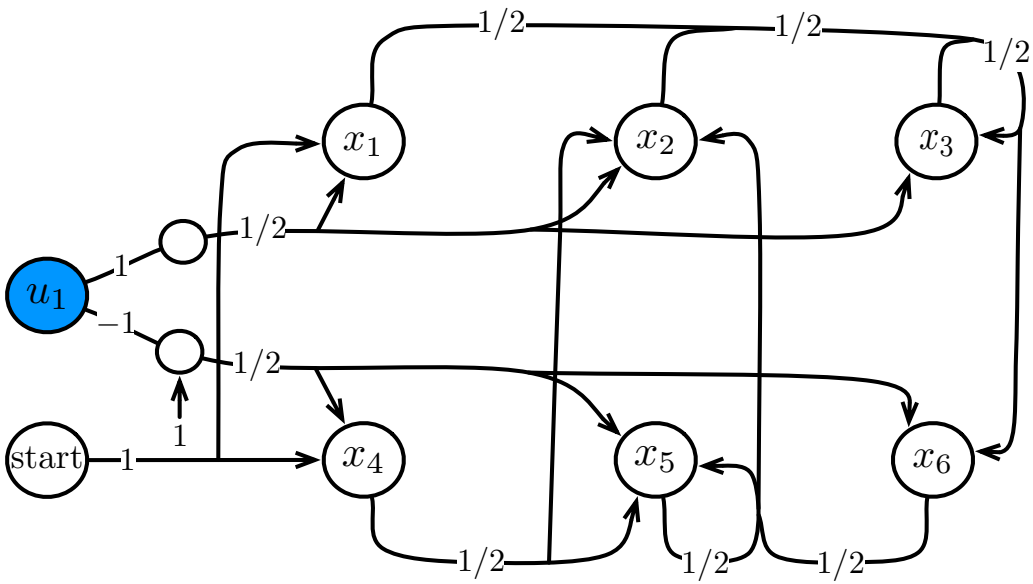


From Automata to Boolean Neural Networks

Automaton



Boolean Neural Network



Equivalence between Boolean Neural Networks and Automata

Theorem (Minsky 67)

"It is evident that each neural network of the kind we have been considering is a finite-state machine."

"[...] It is interesting and even surprising that there is a converse to this. Every finite-state machine is equivalent to, and can be "simulated" by, some neural net."

Equivalence between Boolean Neural Networks and Automata

Theorem (Minsky 67)

"It is evident that each neural network of the kind we have been considering is a finite-state machine."

"[...] It is interesting and even surprising that there is a converse to this. Every finite-state machine is equivalent to, and can be "simulated" by, some neural net."

Equivalence between Boolean Neural Networks and Automata

Theorem (Minsky 67)

"It is evident that each neural network of the kind we have been considering is a finite-state machine."

"[...] It is interesting and even surprising that there is a converse to this. Every finite-state machine is equivalent to, and can be "simulated" by, some neural net."

Introduction

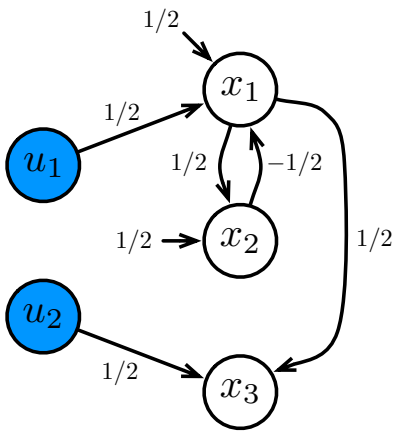
- By translating the so-called Wagner hierarchy from the automaton to the neural network context, we introduce a new attractor-based complexity measurement for Boolean recurrent neural networks.
- The measurement reflects the complexity of the attractors' structure of the networks.

Introduction

- By translating the so-called Wagner hierarchy from the automaton to the neural network context, we introduce a new attractor-based complexity measurement for Boolean recurrent neural networks.
- The measurement reflects the complexity of the attractors' structure of the networks.

Cycles and Attractors

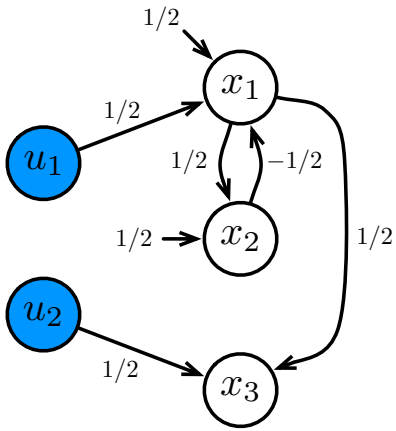
Boolean Neural Network



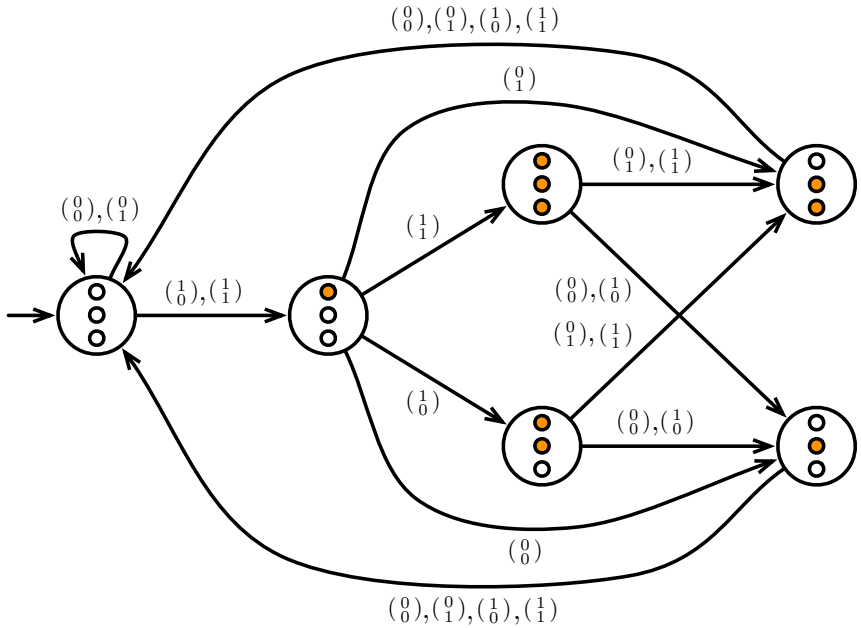
Automaton

Cycles and Attractors

Boolean Neural Network

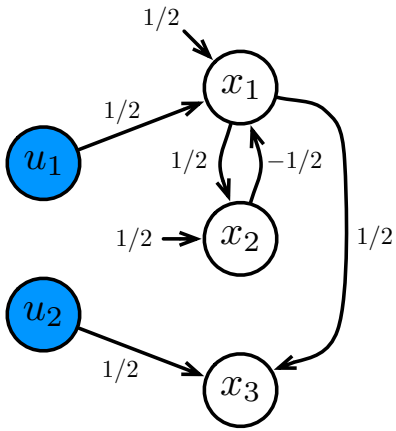


Automaton

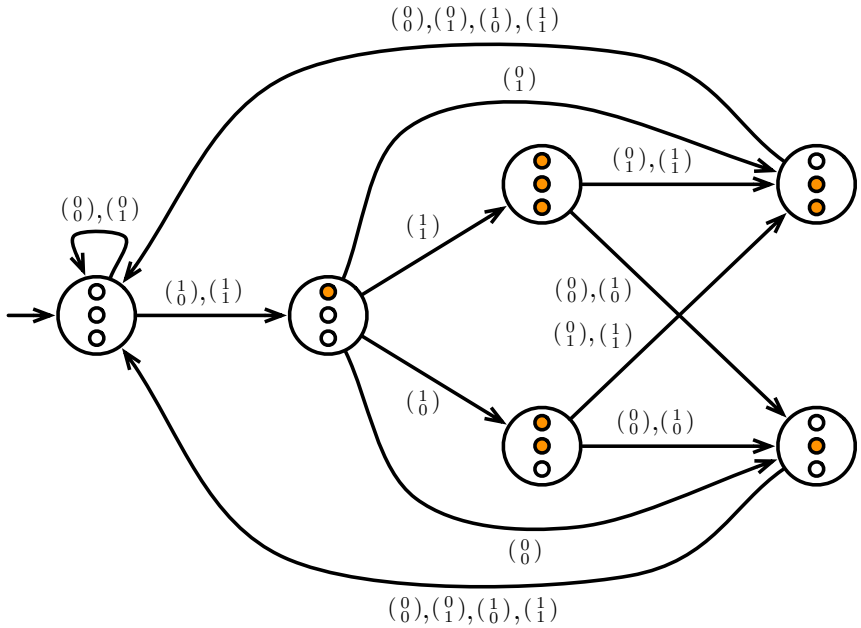


Cycles and Attractors

Boolean Neural Network



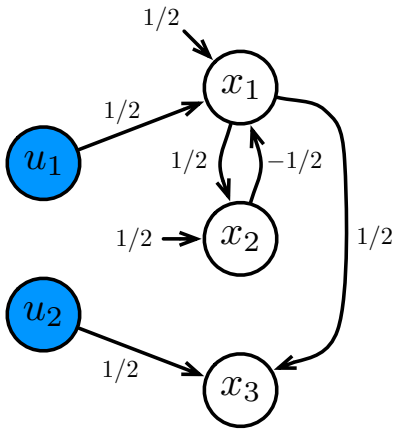
Automaton



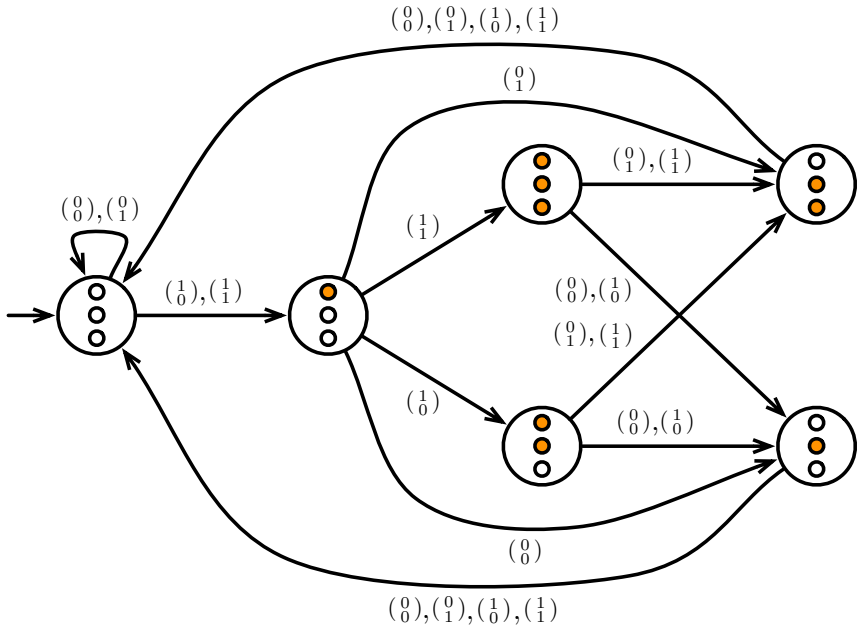
Input stream $\binom{1}{1}\binom{0}{1}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{0}{1}\binom{0}{0}\binom{0}{1}\cdots$

Cycles and Attractors

Boolean Neural Network

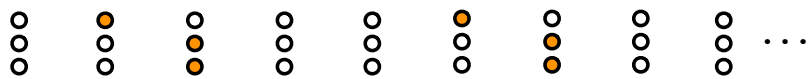


Automaton



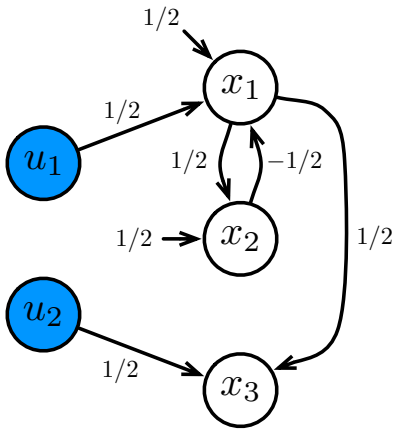
Input stream $\binom{1}{1}\binom{0}{1}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{0}{1}\binom{0}{0}\binom{0}{1}\cdots$

Sequence of states

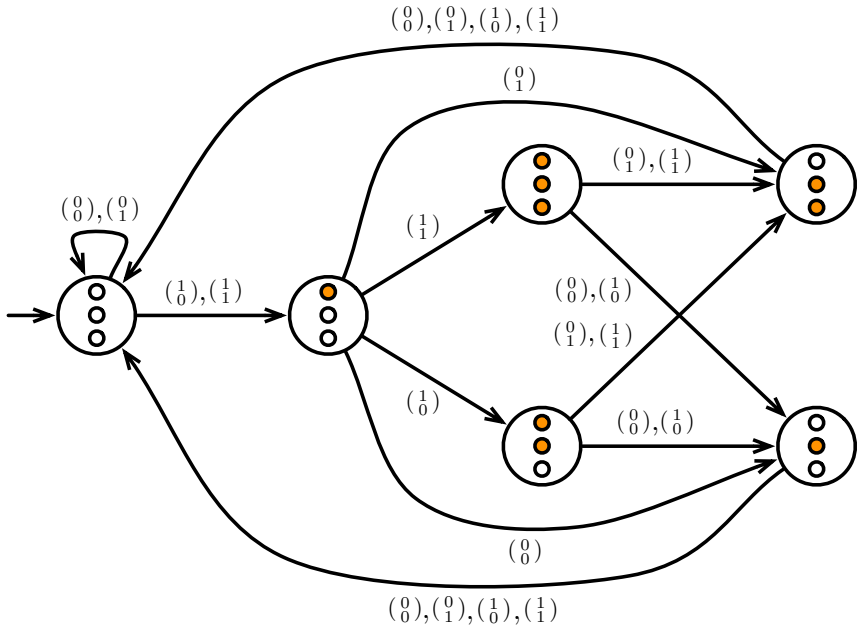


Cycles and Attractors

Boolean Neural Network

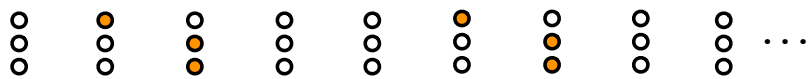


Automaton

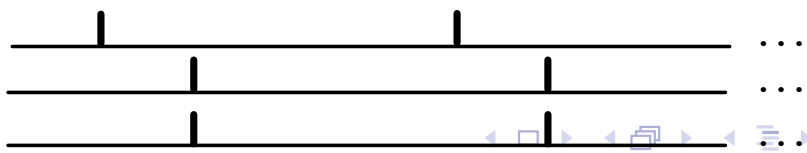


Input stream $\binom{1}{1}\binom{0}{1}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{0}{1}\binom{0}{0}\binom{0}{1}\cdots$

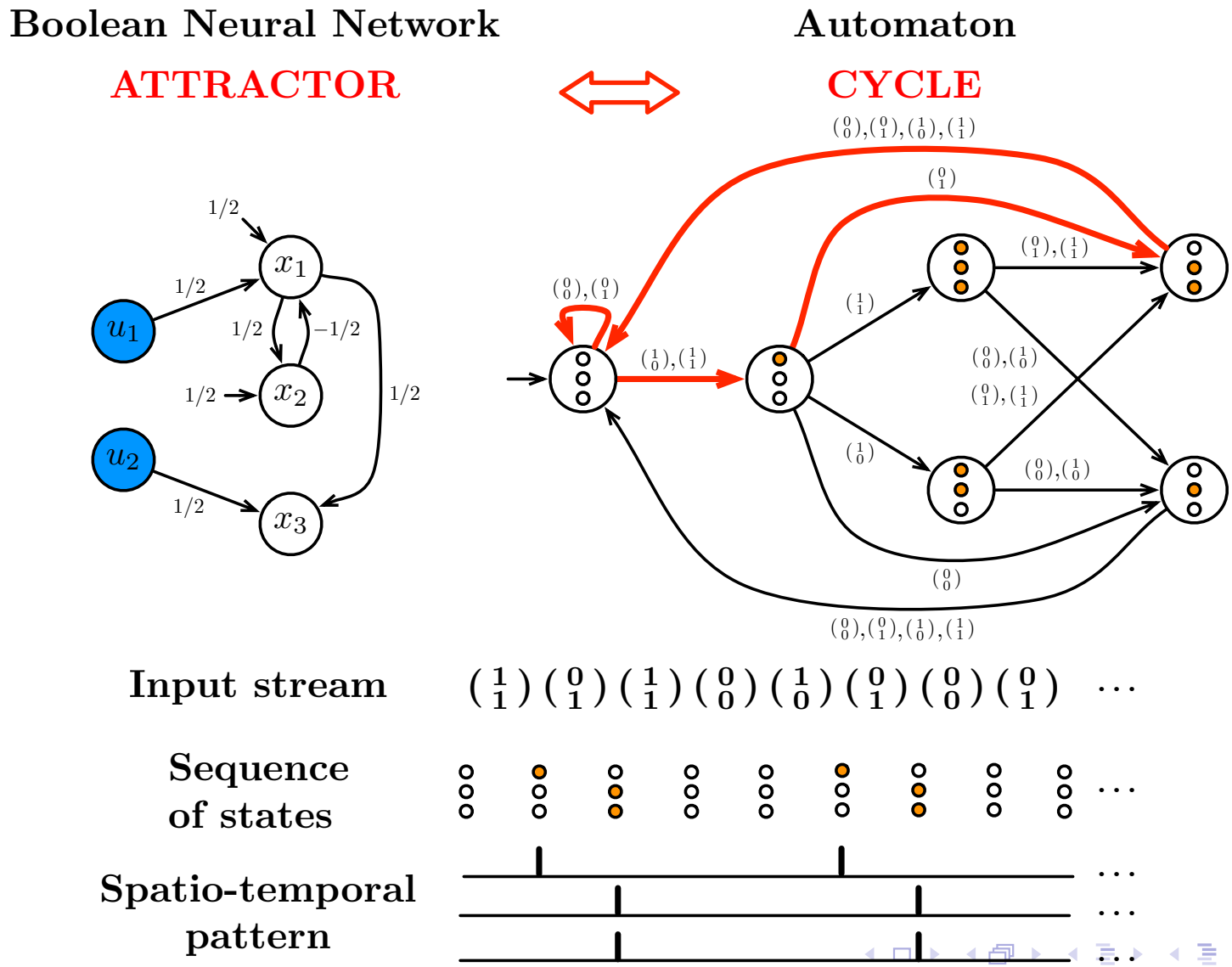
Sequence of states



Spatio-temporal pattern



Cycles and Attractors



The Wagner Hierarchy

- In ω-automata theory, there is a transfinite classification of ω-automata according to the way their cycles are intricated one into the other...
- **The Wagner hierarchy**
- By translating the Wagner hierarchy from the ω-automata to the Boolean neural network context, one obtains a transfinite classification of Boolean neural networks according to the way their attractors are intricated one into the other...
- **The Boolean RNN hierarchy**

The Wagner Hierarchy

- In ω-automata theory, there is a transfinite classification of ω-automata according to the way their cycles are intricated one into the other...
- **The Wagner hierarchy**
- By translating the Wagner hierarchy from the ω-automata to the Boolean neural network context, one obtains a transfinite classification of Boolean neural networks according to the way their attractors are intricated one into the other...
- **The Boolean RNN hierarchy**

The Wagner Hierarchy

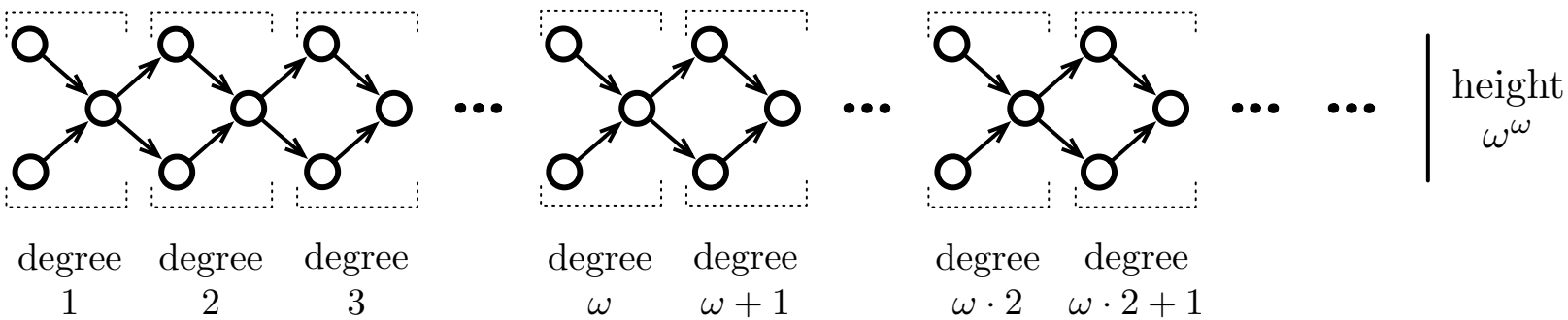
- In ω-automata theory, there is a transfinite classification of ω-automata according to the way their cycles are intricated one into the other...
- **The Wagner hierarchy**
- By translating the Wagner hierarchy from the ω-automata to the Boolean neural network context, one obtains a transfinite classification of Boolean neural networks according to the way their attractors are intricated one into the other...
- The Boolean RNN hierarchy

The Wagner Hierarchy

- In ω-automata theory, there is a transfinite classification of ω-automata according to the way their cycles are intricated one into the other...
- **The Wagner hierarchy**
- By translating the Wagner hierarchy from the ω-automata to the Boolean neural network context, one obtains a transfinite classification of Boolean neural networks according to the way their attractors are intricated one into the other...
- **The Boolean RNN hierarchy**

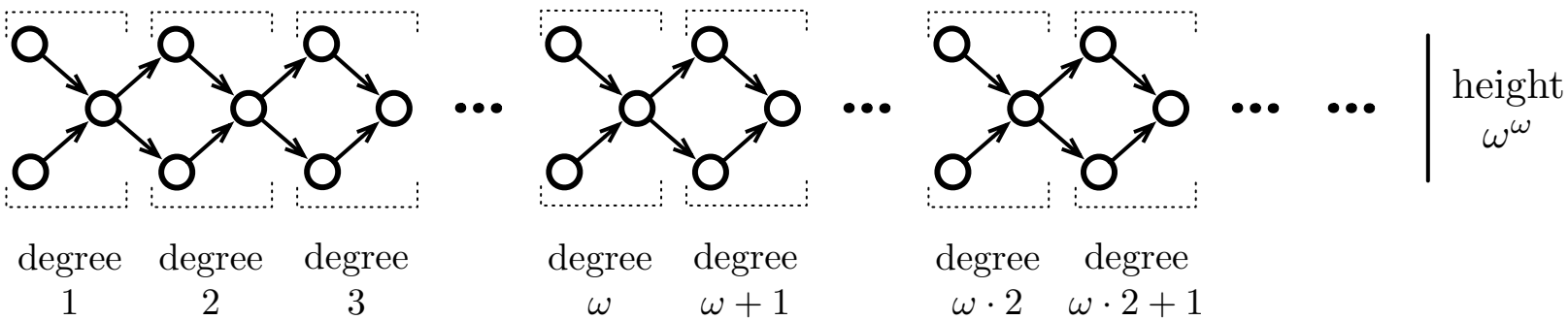
The Wagner Hierarchy

- A transfinite classification of *Muller automata* according to the topological complexity of their underlying language
- Equivalently, a transfinite classification of *Muller automata* according to the graph-theoretical complexity of their cycles
- Quasi well-ordering of transfinite height ω^ω



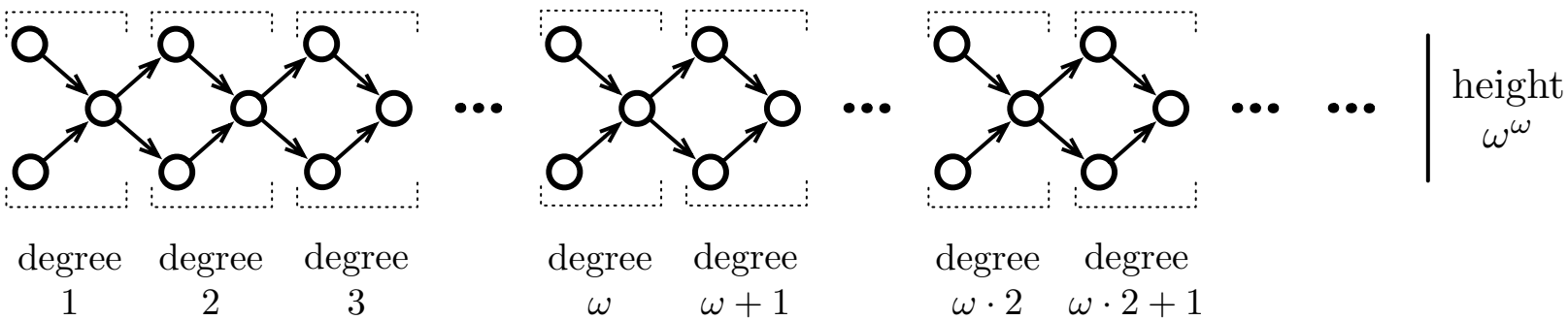
The Wagner Hierarchy

- A transfinite classification of *Muller automata* according to the topological complexity of their underlying language
- Equivalently, a transfinite classification of *Muller automata* according to the graph-theoretical complexity of their cycles
- Quasi well-ordering of transfinite height ω^ω



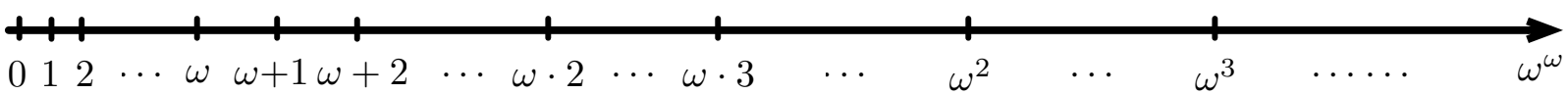
The Wagner Hierarchy

- A transfinite classification of *Muller automata* according to the topological complexity of their underlying language
- Equivalently, a transfinite classification of *Muller automata* according to the graph-theoretical complexity of their cycles
- Quasi well-ordering of transfinite height ω^ω



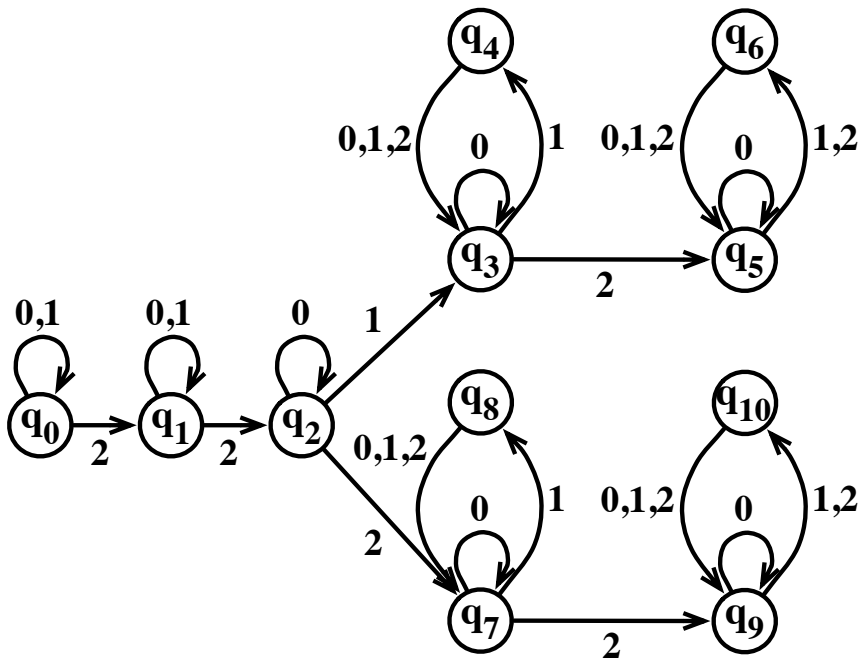
The Wagner Hierarchy

- A transfinite classification of *Muller automata* according to the topological complexity of their underlying language
- Equivalently, a transfinite classification of *Muller automata* according to the graph-theoretical complexity of their cycles
- Quasi well-ordering of transfinite height ω^ω



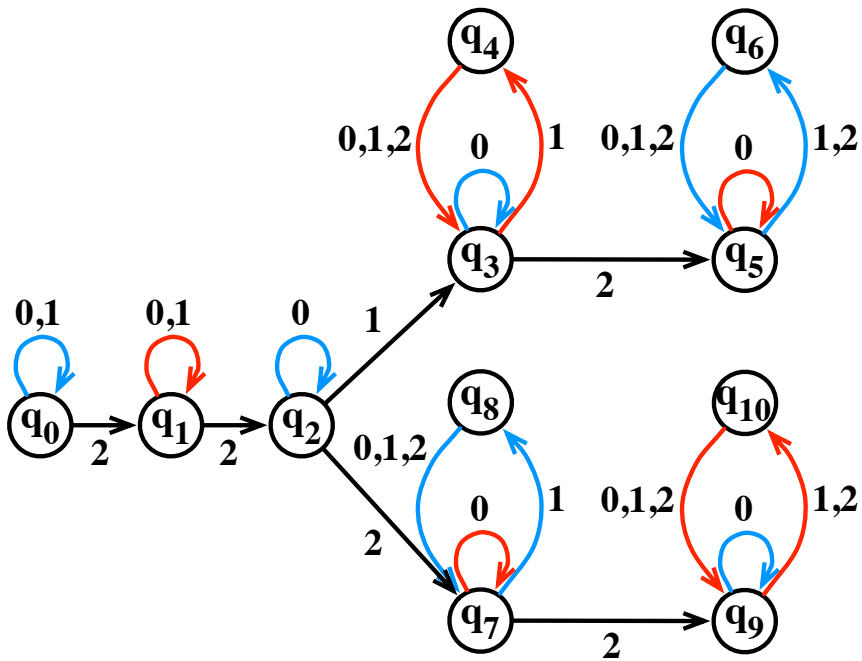
The Wagner Hierarchy – Muller Automata

- A *Muller automaton* consists of an automaton provided with an additional specification of every of its *cycles* into an accepting or a rejecting mode



The Wagner Hierarchy – Muller Automata

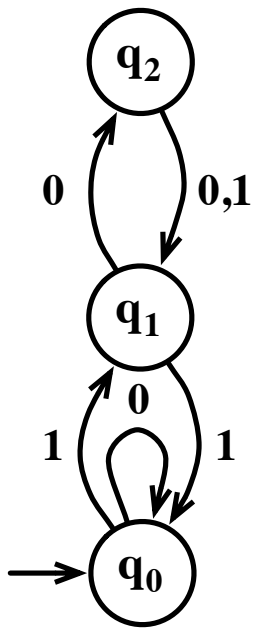
- A *Muller automaton* consists of an automaton provided with an additional specification of every of its *cycles* into an accepting or a rejecting mode



The Wagner Hierarchy – Degrees ω^n

Muller automaton

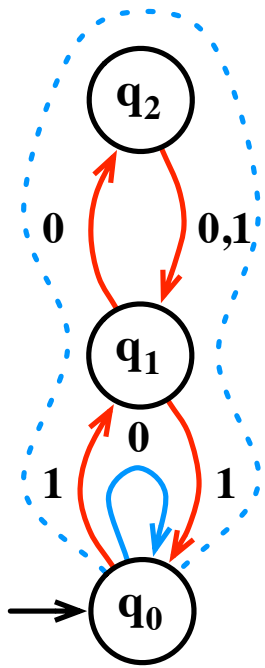
Wagner degree



The Wagner Hierarchy – Degrees ω^n

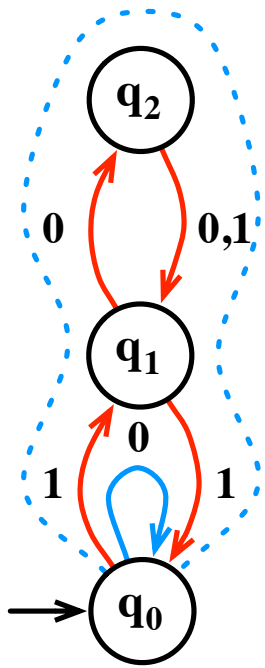
Muller automaton

Wagner degree

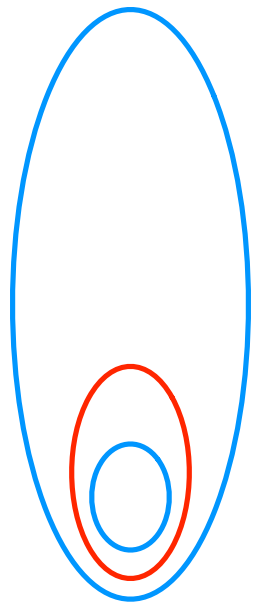


The Wagner Hierarchy – Degrees ω^n

Muller automaton



Wagner degree

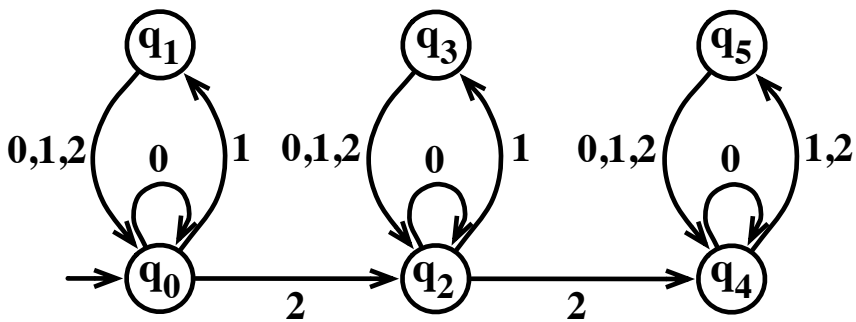


$[+]\ \omega^2$

The Wagner Hierarchy – Degrees $\omega^n \cdot k$

Muller automaton

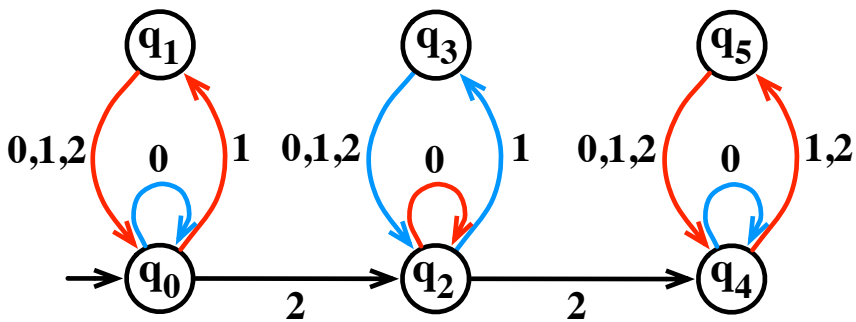
Wagner degree



The Wagner Hierarchy – Degrees $\omega^n \cdot k$

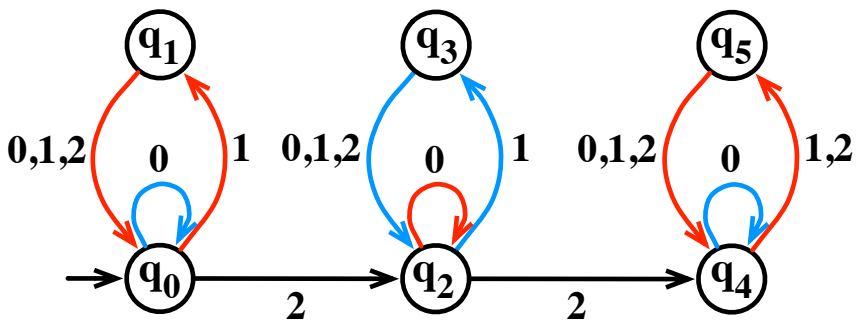
Muller automaton

Wagner degree

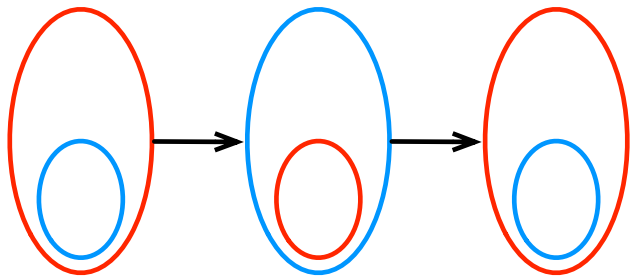


The Wagner Hierarchy – Degrees $\omega^n \cdot k$

Muller automaton



Wagner degree

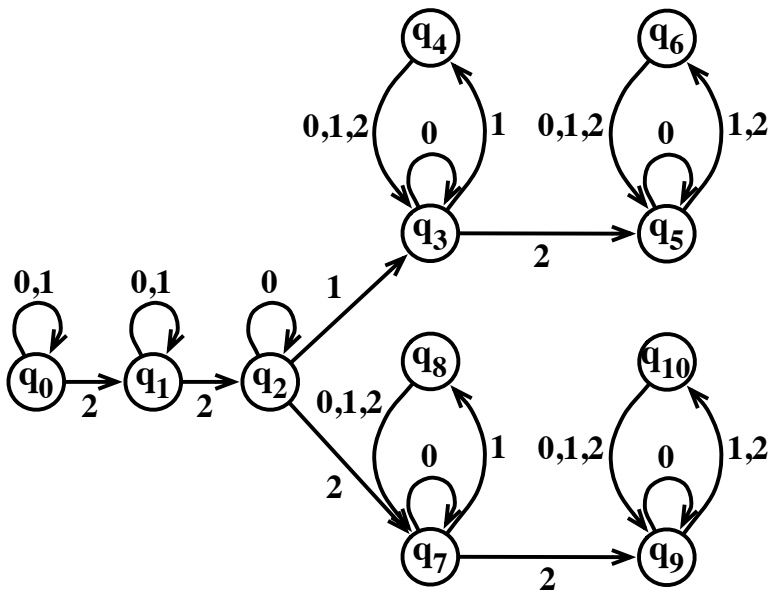


$[+]\ \omega^1 \cdot 3$

The Wagner Hierarchy – Degrees $\omega^n \cdot k + \omega^{n'} \cdot k'$

Muller automaton

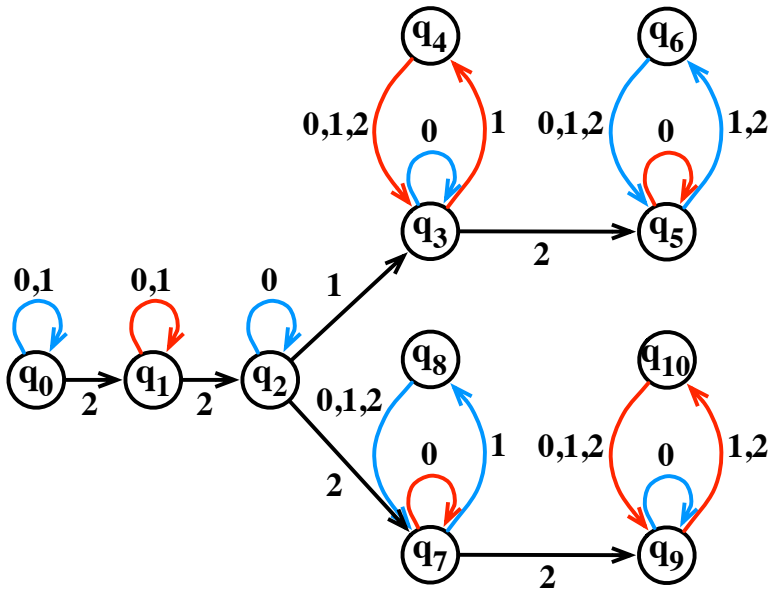
Wagner degree



The Wagner Hierarchy – Degrees $\omega^n \cdot k + \omega^{n'} \cdot k'$

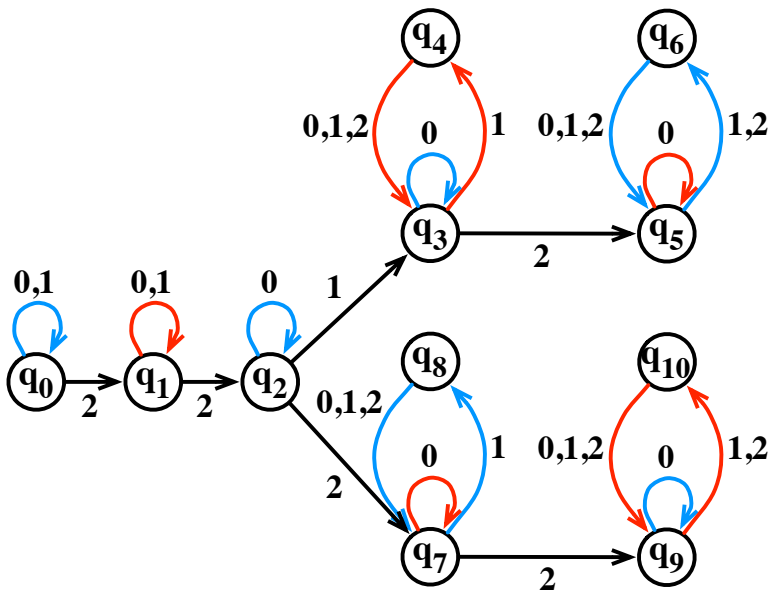
Muller automaton

Wagner degree

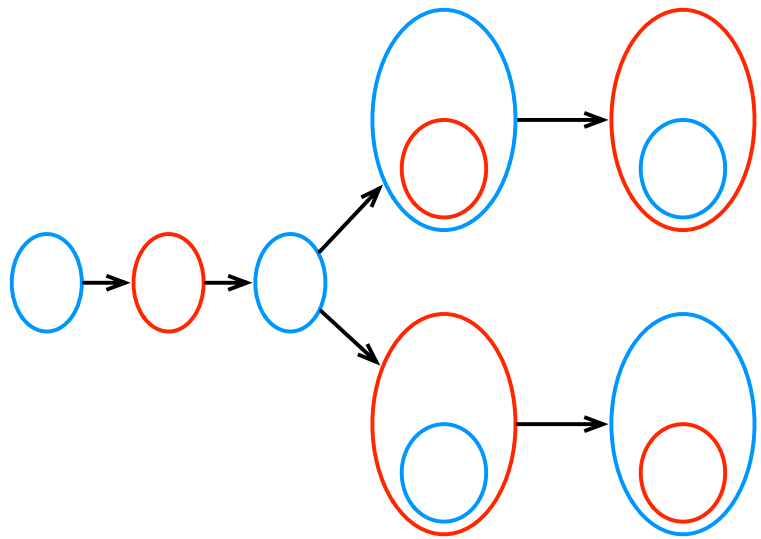


The Wagner Hierarchy – Degrees $\omega^n \cdot k + \omega^{n'} \cdot k'$

Muller automaton



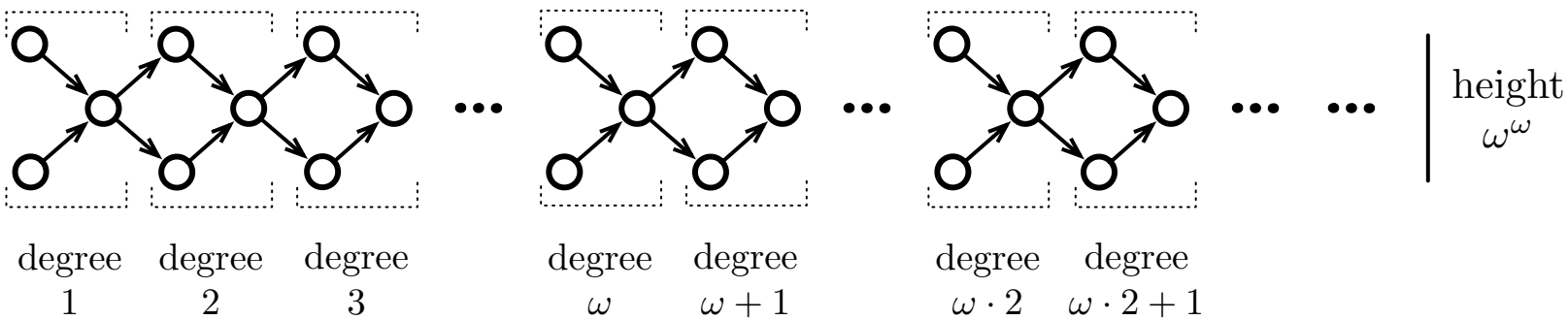
Wagner degree



$[+]\ \omega^1 \cdot 2 + \omega^0 \cdot 3$

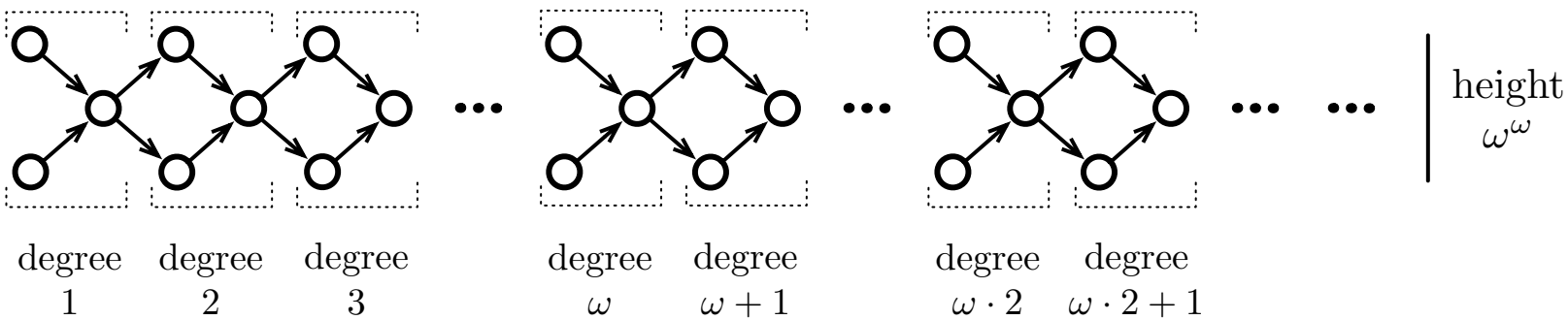
The Wagner Hierarchy – Summary

- A quasi well-ordering of transfinite height ω^ω
- Every ordinal $\alpha < \omega^\omega$ has a unique Cantor normal form $\alpha = \omega^{n_0} \cdot p_0 + \omega^{n_1} \cdot p_1 + \cdots + \omega^{n_k} \cdot p_k$, where $n_0 > n_1 > \cdots > n_k$
- The degree α of a Muller automaton $\mathcal{M}$ in the Wagner hierarchy is the maximal "tree of cycles" $\mathcal{T}_\alpha$ in $\mathcal{M}$



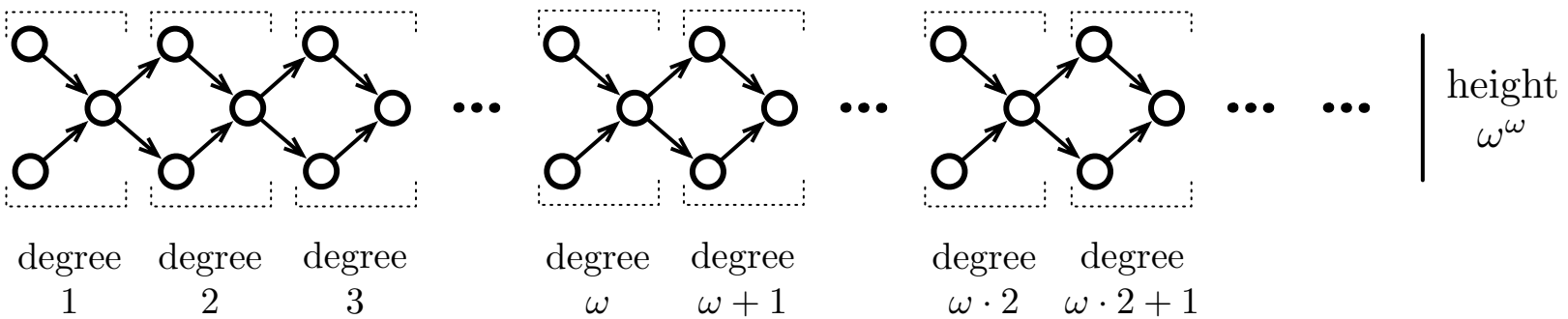
The Wagner Hierarchy – Summary

- A quasi well-ordering of transfinite height ω^ω
- Every ordinal $\alpha < \omega^\omega$ has a unique Cantor normal form $\alpha = \omega^{n_0} \cdot p_0 + \omega^{n_1} \cdot p_1 + \cdots + \omega^{n_k} \cdot p_k$, where $n_0 > n_1 > \cdots > n_k$
- The degree α of a Muller automaton $\mathcal{M}$ in the Wagner hierarchy is the maximal "tree of cycles" $\mathcal{T}_\alpha$ in $\mathcal{M}$



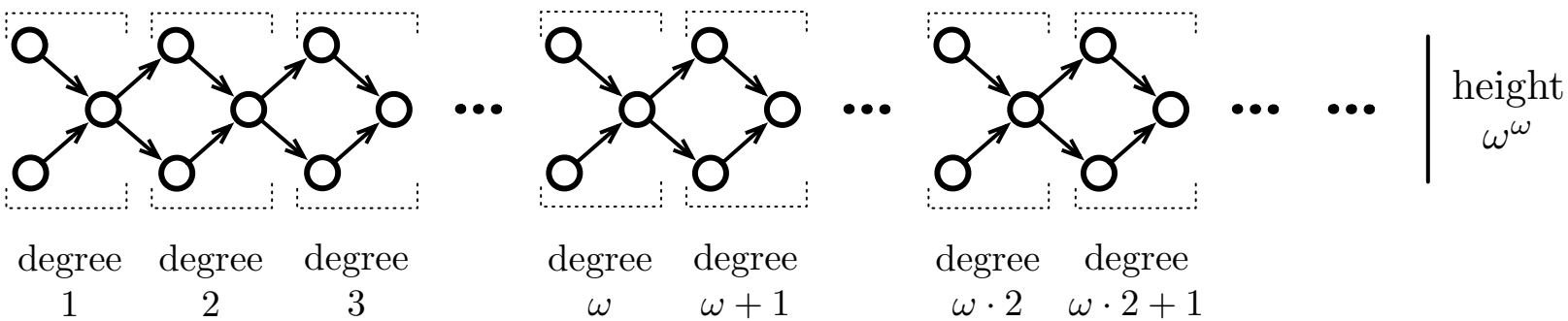
The Wagner Hierarchy – Summary

- A quasi well-ordering of transfinite height ω^ω
- Every ordinal $\alpha < \omega^\omega$ has a unique Cantor normal form $\alpha = \omega^{n_0} \cdot p_0 + \omega^{n_1} \cdot p_1 + \cdots + \omega^{n_k} \cdot p_k$, where $n_0 > n_1 > \cdots > n_k$
- The degree α of a Muller automaton $\mathcal{M}$ in the Wagner hierarchy is the maximal "tree of cycles" $\mathcal{T}_\alpha$ in $\mathcal{M}$



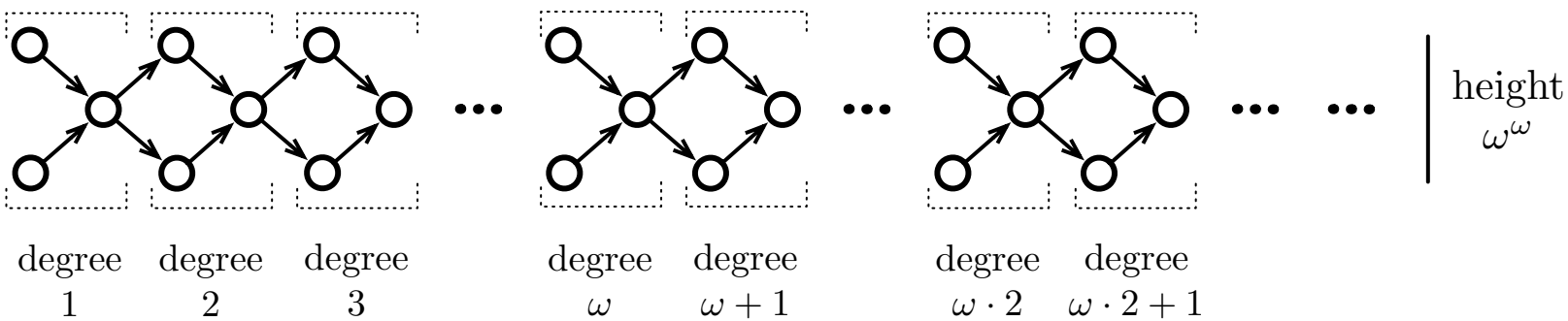
The Boolean RNNs Hierarchy

- We assume that our *Boolean RNNs* are provided with an additional specification of every of their *attractors* into a meaningful or a spurious mode
- We can transpose the Wagner hierarchy from the *Muller automata* to the *Boolean RNNs* context.
- One obtains a transfinite classification of *Boolean RNNs* according to the topological complexity of their attractors



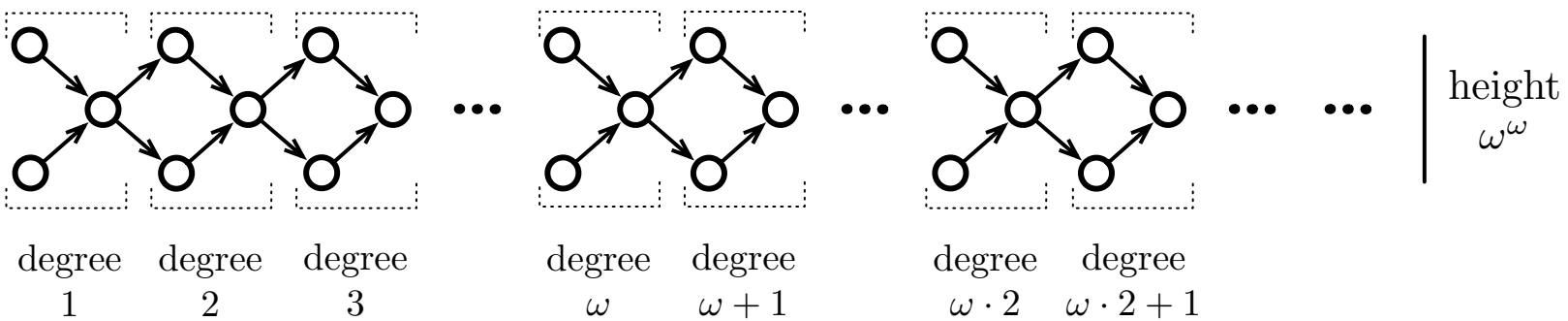
The Boolean RNNs Hierarchy

- We assume that our *Boolean RNNs* are provided with an additional specification of every of their *attractors* into a meaningful or a spurious mode
- We can transpose the Wagner hierarchy from the *Muller automata* to the *Boolean RNNs* context.
- One obtains a transfinite classification of *Boolean RNNs* according to the topological complexity of their attractors



The Boolean RNNs Hierarchy

- We assume that our *Boolean RNNs* are provided with an additional specification of every of their *attractors* into a meaningful or a spurious mode
- We can transpose the Wagner hierarchy from the *Muller automata* to the *Boolean RNNs* context.
- One obtains a transfinite classification of *Boolean RNNs* according to the topological complexity of their attractors



Introduction

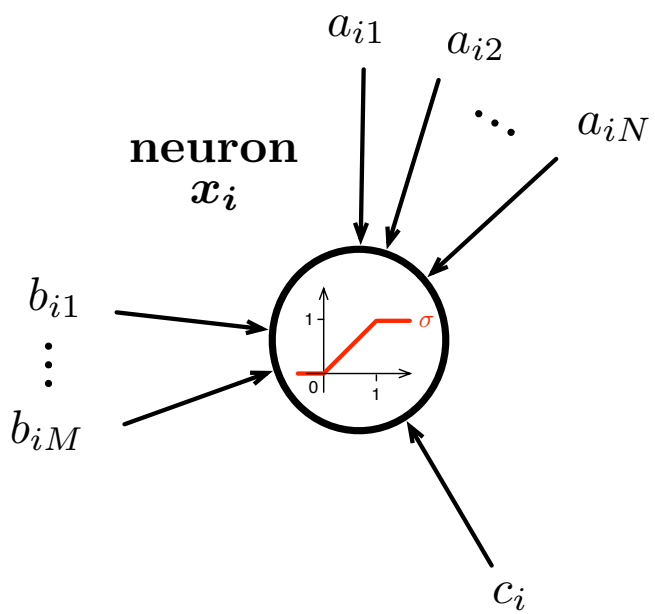
- Siegelmann and Sontag studied the computational power or RNNs whose activation function is given by a simple sigmoid function and whose synaptic weights are allowed to range over rational numbers.
- The so-called *rational-weighted RNNs* are shown to be Turing equivalent.

Introduction

- Siegelmann and Sontag studied the computational power or RNNs whose activation function is given by a simple sigmoid function and whose synaptic weights are allowed to range over rational numbers.
- The so-called *rational-weighted RNNs* are shown to be Turing equivalent.

Dynamics of rational-weighted RNNs

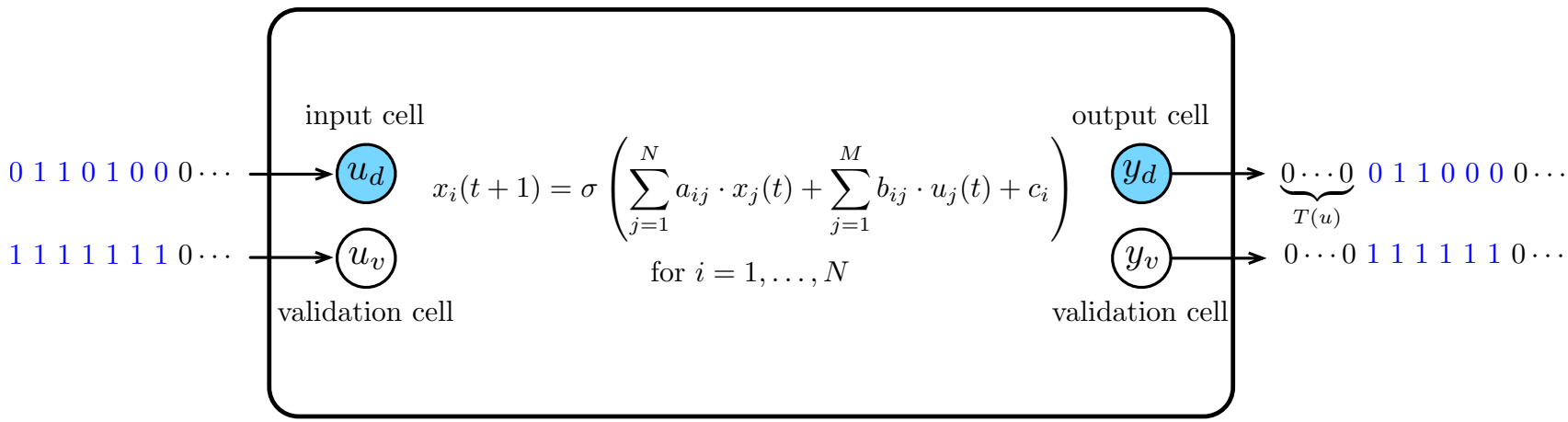
A *rational-weighted RNN* (RNN[$\mathbb{Q}$]) is a RNN whose synaptic weights are allowed to range over rational numbers and whose activation function is given by a linear sigmoid function.



$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right), \; a_{ij}, b_{ij}, c_i \in \mathbb{Q}$$

Formal RNNs

We define a *formal RNN* which can compute partial functions of the form $\varphi : \{0,1\}^+ \to \{0,1\}^+$.



Results

Rational-weighted RNNs are actually Turing equivalent.

Theorem (Siegelmann & Sontag 95)

- *Let $\varphi : \{0,1\}^+ \to \{0,1\}^+$ be a partial function. Then φ is Turing computable (i.e. partial recursive) iff φ can be computed by some RNN[$\mathbb{Q}$].*
- *Moreover, for any p-stack machine $\mathcal{M}$ ($p \geq 2$) computing φ, there exists a RNN[$\mathbb{Q}$] $\mathcal{N}$ which simulates $\mathcal{M}$ in real time.*

Results

Rational-weighted RNNs are actually Turing equivalent.

Theorem (Siegelmann & Sontag 95)

- *Let* $\varphi : \{0,1\}^+ \to \{0,1\}^+$ *be a partial function. Then* φ *is Turing computable (i.e. partial recursive) iff* φ *can be computed by some RNN*$[\mathbb{Q}]$.
- *Moreover, for any* p*-stack machine* $\mathcal{M}$ $(p \geq 2)$ *computing* φ*, there exists a RNN*$[\mathbb{Q}]$ $\mathcal{N}$ *which simulates* $\mathcal{M}$ *in real time.*

Proof – First Implication

Let $\varphi : \{0,1\}^+ \to \{0,1\}^+$ be some partial function computable by some RNN[$\mathbb{Q}$] $\mathcal{N}$.

- The networks dynamics $\mathcal{F} : \mathbb{Q}^N \to \mathbb{Q}^N$ given by the equations

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

  is clearly recursive.

Therefore, φ is obviously Turing-computable.

Proof – First Implication

Let $\varphi : \{0,1\}^+ \to \{0,1\}^+$ be some partial function computable by some RNN[$\mathbb{Q}$] $\mathcal{N}$.

- The networks dynamics $\mathcal{F} : \mathbb{Q}^N \to \mathbb{Q}^N$ given by the equations

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

  is clearly recursive.

Therefore, φ is obviously Turing-computable.

Proof – First Implication

Let $\varphi : \{0,1\}^+ \to \{0,1\}^+$ be some partial function computable by some RNN[$\mathbb{Q}$] $\mathcal{N}$.

- The networks dynamics $\mathcal{F} : \mathbb{Q}^N \to \mathbb{Q}^N$ given by the equations

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

  is clearly recursive.

Therefore, φ is obviously Turing-computable.

Proof – Second Implication

Let $\varphi : \{0,1\}^+ \to \{0,1\}^+$ be some Turing-computable partial function.

- Then φ is computable by some p-stack machine $\mathcal{M}$.
- We simulate the behaviour of $\mathcal{M}$ by some RNN[$\mathbb{Q}$] $\mathcal{N}$.
- Towards this purpose, we first show how to perform the stack operations with sigmoid rational-weighted neurons.

Proof – Second Implication

Let $\varphi : \{0,1\}^+ \to \{0,1\}^+$ be some Turing-computable partial function.

- Then φ is computable by some p-stack machine $\mathcal{M}$.
- We simulate the behaviour of $\mathcal{M}$ by some RNN[$\mathbb{Q}$] $\mathcal{N}$.
- Towards this purpose, we first show how to perform the stack operations with sigmoid rational-weighted neurons.

Proof – Second Implication

Let $\varphi : \{0,1\}^+ \to \{0,1\}^+$ be some Turing-computable partial function.

- Then φ is computable by some p-stack machine $\mathcal{M}$.
- We simulate the behaviour of $\mathcal{M}$ by some RNN[$\mathbb{Q}$] $\mathcal{N}$.
- Towards this purpose, we first show how to perform the stack operations with sigmoid rational-weighted neurons.

Proof – Second Implication

Let $\varphi : \{0,1\}^+ \to \{0,1\}^+$ be some Turing-computable partial function.

- Then φ is computable by some p-stack machine $\mathcal{M}$.
- We simulate the behaviour of $\mathcal{M}$ by some RNN[$\mathbb{Q}$] $\mathcal{N}$.
- Towards this purpose, we first show how to perform the stack operations with sigmoid rational-weighted neurons.

Proof – Second Implication

- We encode every stack content $w = w_1 \cdots w_n$ as the rational number $q_w = \sum_{i=1}^{n} \frac{2 \cdot w(i)+1}{4^i}$
  - For instance, $w = 0110$ is encoded into $q_w = \frac{1}{4} + \frac{3}{16} + \frac{3}{44} + \frac{1}{256}$
- Reading the top of the stack: $top(q) = \sigma(4q - 2)$
- Pushing 0 into the stack: $push_0(q) = \sigma(\frac{1}{4}q + \frac{1}{4})$
- Pushing 1 into the stack: $push_1(q) = \sigma(\frac{1}{4}q + \frac{3}{4})$
- Popping the stack: $pop(q) = \sigma(4q - (2top(q) + 1))$
- Emptiness of the stack: $empty(q) = \sigma(4q)$

Proof – Second Implication

- We encode every stack content $w = w_1 \cdots w_n$ as the rational number $q_w = \sum_{i=1}^{n} \frac{2 \cdot w(i)+1}{4^i}$
  - For instance, $w = 0110$ is encoded into $q_w = \frac{1}{4} + \frac{3}{16} + \frac{3}{44} + \frac{1}{256}$
- Reading the top of the stack: $top(q) = \sigma(4q - 2)$
- Pushing 0 into the stack: $push_0(q) = \sigma(\frac{1}{4}q + \frac{1}{4})$
- Pushing 1 into the stack: $push_1(q) = \sigma(\frac{1}{4}q + \frac{3}{4})$
- Popping the stack: $pop(q) = \sigma(4q - (2top(q) + 1))$
- Emptiness of the stack: $empty(q) = \sigma(4q)$

Proof – Second Implication

- We encode every stack content $w = w_1 \cdots w_n$ as the rational number $q_w = \sum_{i=1}^{n} \frac{2 \cdot w(i)+1}{4^i}$
  - For instance, $w = 0110$ is encoded into $q_w = \frac{1}{4} + \frac{3}{16} + \frac{3}{44} + \frac{1}{256}$
- Reading the top of the stack: $top(q) = \sigma(4q - 2)$
- Pushing 0 into the stack: $push_0(q) = \sigma(\frac{1}{4}q + \frac{1}{4})$
- Pushing 1 into the stack: $push_1(q) = \sigma(\frac{1}{4}q + \frac{3}{4})$
- Popping the stack: $pop(q) = \sigma(4q - (2top(q) + 1))$
- Emptiness of the stack: $empty(q) = \sigma(4q)$

Proof – Second Implication

- We encode every stack content $w = w_1 \cdots w_n$ as the rational number $q_w = \sum_{i=1}^{n} \frac{2 \cdot w(i)+1}{4^i}$
  - For instance, $w = 0110$ is encoded into $q_w = \frac{1}{4} + \frac{3}{16} + \frac{3}{44} + \frac{1}{256}$
- Reading the top of the stack: $top(q) = \sigma(4q - 2)$
- Pushing 0 into the stack: $push_0(q) = \sigma(\frac{1}{4}q + \frac{1}{4})$
- Pushing 1 into the stack: $push_1(q) = \sigma(\frac{1}{4}q + \frac{3}{4})$
- Popping the stack: $pop(q) = \sigma(4q - (2top(q) + 1))$
- Emptiness of the stack: $empty(q) = \sigma(4q)$

Proof – Second Implication

- We encode every stack content $w = w_1 \cdots w_n$ as the rational number $q_w = \sum_{i=1}^{n} \frac{2 \cdot w(i)+1}{4^i}$
  - For instance, $w = 0110$ is encoded into $q_w = \frac{1}{4} + \frac{3}{16} + \frac{3}{44} + \frac{1}{256}$
- Reading the top of the stack: $top(q) = \sigma(4q - 2)$
- Pushing 0 into the stack: $push_0(q) = \sigma(\frac{1}{4}q + \frac{1}{4})$
- Pushing 1 into the stack: $push_1(q) = \sigma(\frac{1}{4}q + \frac{3}{4})$
- Popping the stack: $pop(q) = \sigma(4q - (2top(q) + 1))$
- Emptiness of the stack: $empty(q) = \sigma(4q)$

Proof – Second Implication

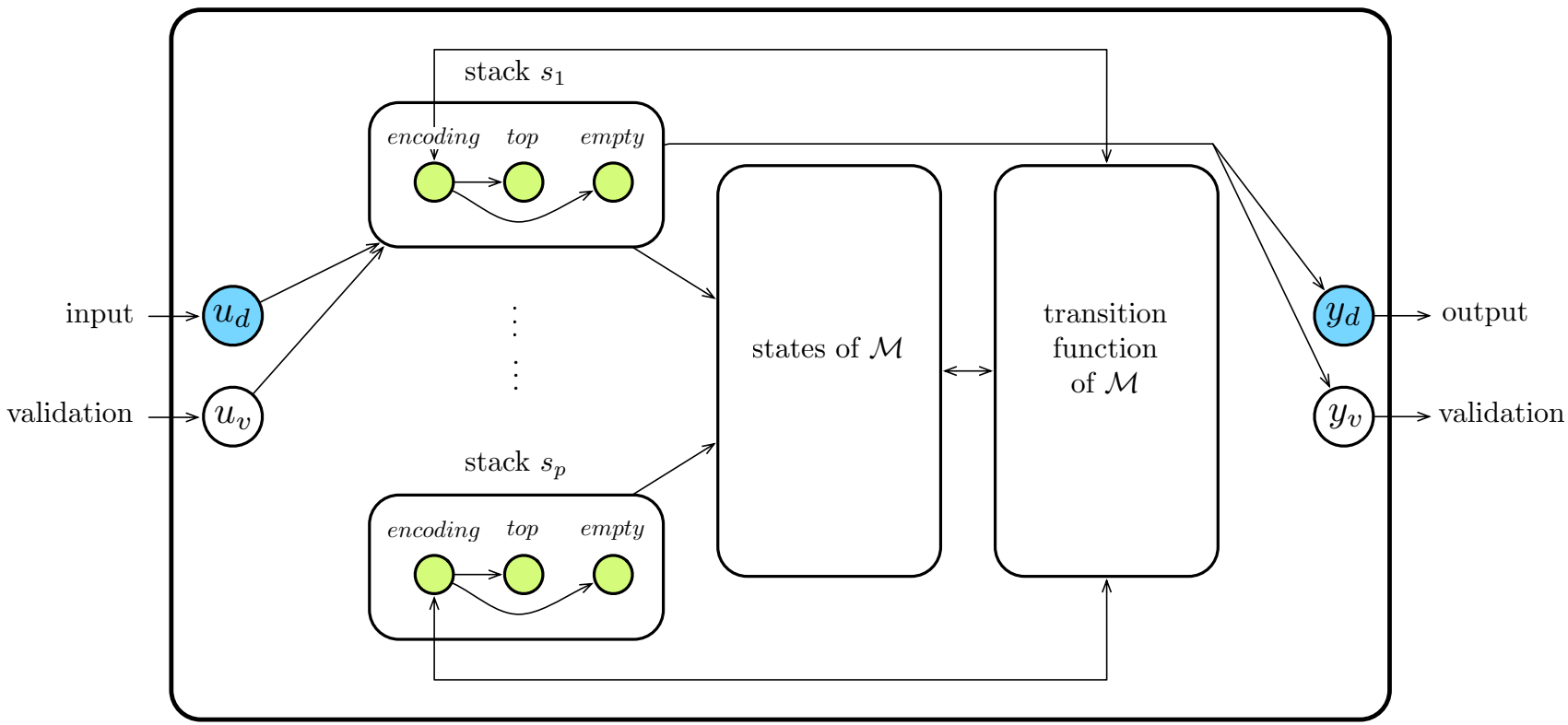
- We encode every stack content $w = w_1 \cdots w_n$ as the rational number $q_w = \sum_{i=1}^{n} \frac{2 \cdot w(i)+1}{4^i}$
  - For instance, $w = 0110$ is encoded into $q_w = \frac{1}{4} + \frac{3}{16} + \frac{3}{44} + \frac{1}{256}$
- Reading the top of the stack: $top(q) = \sigma(4q - 2)$
- Pushing 0 into the stack: $push_0(q) = \sigma(\frac{1}{4}q + \frac{1}{4})$
- Pushing 1 into the stack: $push_1(q) = \sigma(\frac{1}{4}q + \frac{3}{4})$
- Popping the stack: $pop(q) = \sigma(4q - (2top(q) + 1))$
- Emptiness of the stack: $empty(q) = \sigma(4q)$

Proof – Second Implication

- We encode every stack content $w = w_1 \cdots w_n$ as the rational number $q_w = \sum_{i=1}^{n} \frac{2 \cdot w(i)+1}{4^i}$
  - For instance, $w = 0110$ is encoded into $q_w = \frac{1}{4} + \frac{3}{16} + \frac{3}{44} + \frac{1}{256}$
- Reading the top of the stack: $top(q) = \sigma(4q - 2)$
- Pushing 0 into the stack: $push_0(q) = \sigma(\frac{1}{4}q + \frac{1}{4})$
- Pushing 1 into the stack: $push_1(q) = \sigma(\frac{1}{4}q + \frac{3}{4})$
- Popping the stack: $pop(q) = \sigma(4q - (2top(q) + 1))$
- Emptiness of the stack: $empty(q) = \sigma(4q)$

Proof – Second Implication

The q-stack machine $\mathcal{M}$ is then simulated by the network below, showing that φ is RNN[$\mathbb{Q}$]-computable.



Introduction

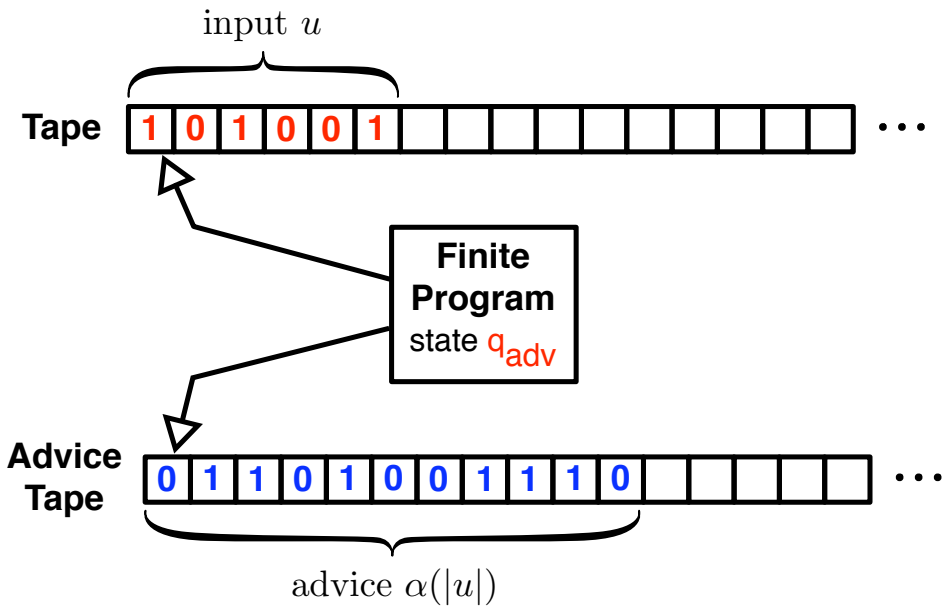
- Siegelmann and Sontag assumed that the variables appearing in the underlying chemical and physical phenomena could be modelled by continuous rather than discrete (rational) numbers.
- They proposed an approach to the computational power of recurrent neural networks from the perspective of *analog computation*.

Introduction

- Siegelmann and Sontag assumed that the variables appearing in the underlying chemical and physical phenomena could be modelled by continuous rather than discrete (rational) numbers.
- They proposed an approach to the computational power of recurrent neural networks from the perspective of *analog computation*.

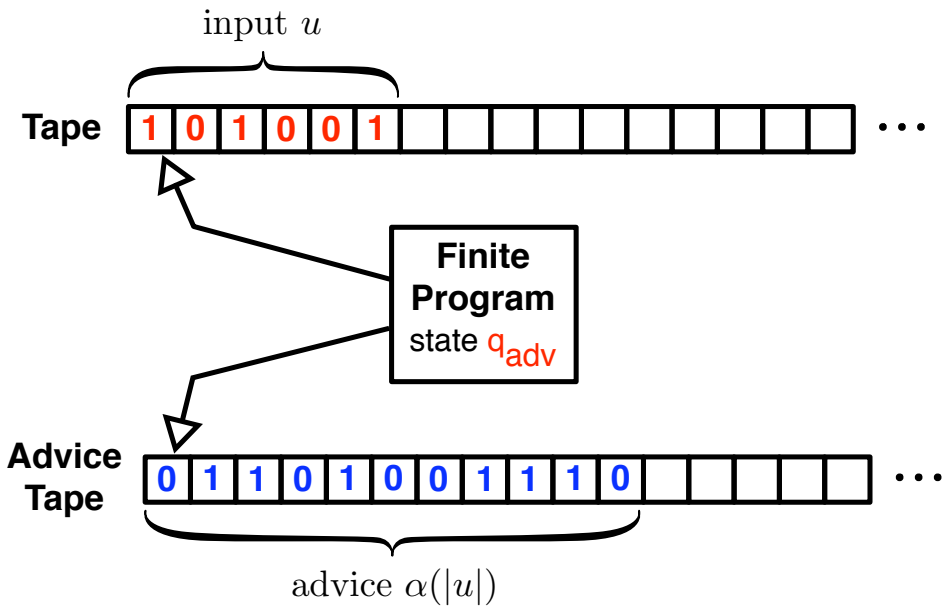
Turing machine with advice

- A Turing machine with advice (TM/A) is a Turing machine provided with an additional advice tape and advice function $\alpha : \mathbb{N} \longrightarrow \{0,1\}^*$.
- $P/poly$ is the class of languages recognized in polynomial time by Turing machines with polynomial advices (TM/poly(A)).



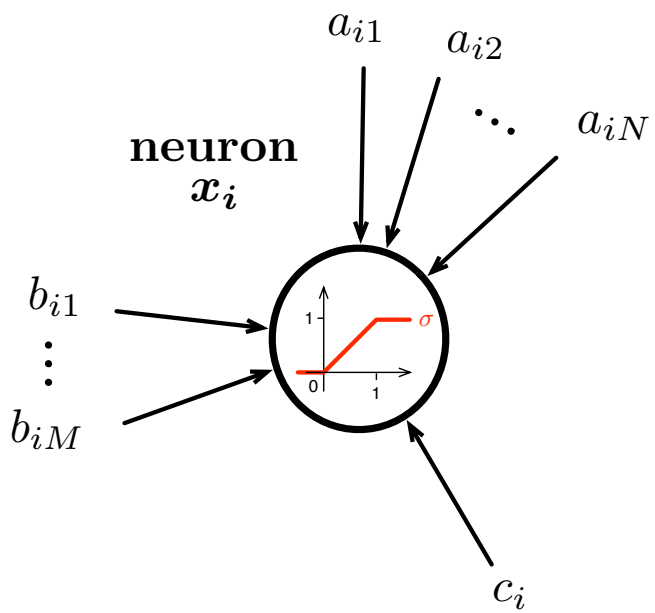
Turing machine with advice

- A Turing machine with advice (TM/A) is a Turing machine provided with an additional advice tape and advice function $\alpha : \mathbb{N} \longrightarrow \{0,1\}^*$.
- $P/poly$ is the class of languages recognized in polynomial time by Turing machines with polynomial advices (TM/poly(A)).



Dynamics of real-weighted RNNs

A *real-weighted or analog RNN* (RNN[$\mathbb{R}$]) is a RNN whose synaptic weights are allowed to range over real numbers.



$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right), \; a_{ij}, b_{ij}, c_i \in \mathbb{R}$$

Results

In this case, the computational power of RNNs is drastically increased.

Theorem (Siegelmann & Sontag 94)

RNN[$\mathbb{R}$]s are super-Turing:

- *They can decide any possible language in exponential time.*
- *They compute exactly the class $P/poly$ in polynomial time.*

Proof – First Implication

Let $L \in P/poly$.

- By some alternative characterization of $P/poly$, there exists a polynomial size circuits family $\mathcal{C} = \{C_n : n \geq 0\}$ such that each circuit C_n decides the language $L \cap \{0,1\}^n$.
- We provide a suitable encoding of the circuit family $\mathcal{C}$ into some real number $r(\mathcal{C})$: first $\mathcal{C}$ is represented by some infinite word $w_{\mathcal{C}} \in \{0,2,4,6,8\}^\omega$, and then $r(\mathcal{C}) = \sum_{i=0}^{\infty} \frac{w_{\mathcal{C}}(i)}{9^i}$.

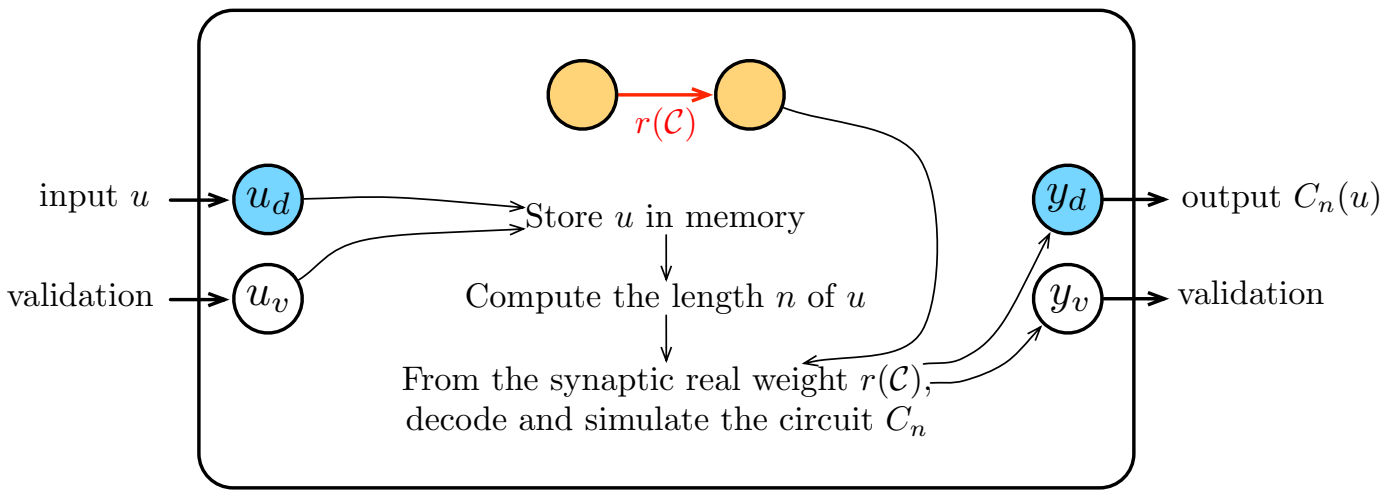
Proof – First Implication

Let $L \in P/poly$.

- By some alternative characterization of $P/poly$, there exists a polynomial size circuits family $\mathcal{C} = \{C_n : n \geq 0\}$ such that each circuit C_n decides the language $L \cap \{0,1\}^n$.
- We provide a suitable encoding of the circuit family $\mathcal{C}$ into some real number $r(\mathcal{C})$: first $\mathcal{C}$ is represented by some infinite word $w_{\mathcal{C}} \in \{0,2,4,6,8\}^\omega$, and then $r(\mathcal{C}) = \sum_{i=0}^{\infty} \frac{w_{\mathcal{C}}(i)}{9^i}$.

Proof – First Implication

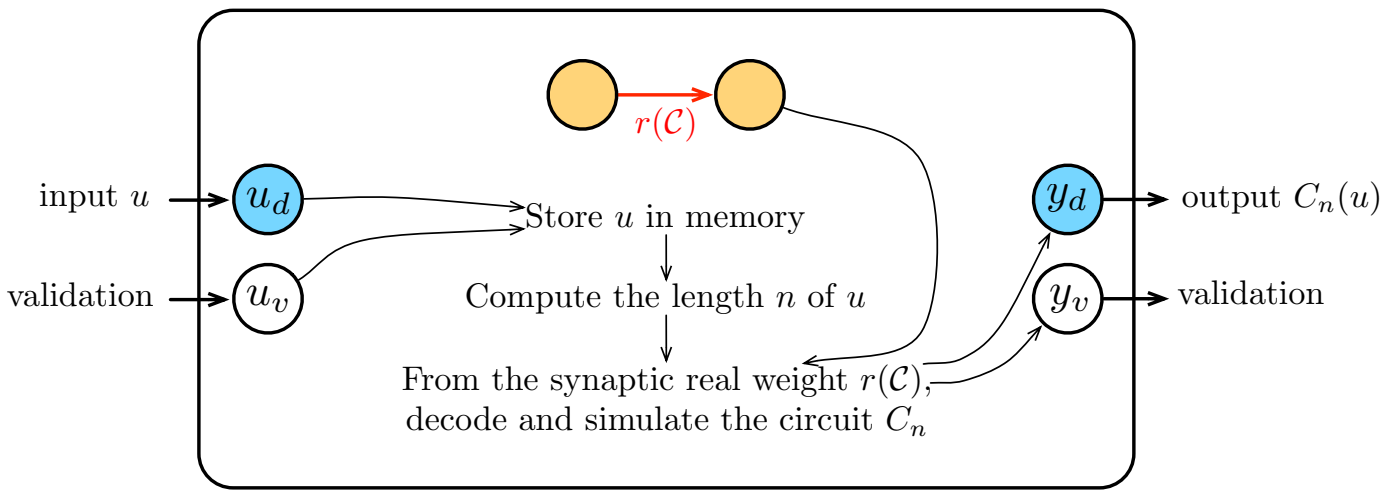
- Then, one can build some RNN[$\mathbb{R}$] $\mathcal{N}$ which contains the real $r(\mathcal{C})$ as a synaptic weight, and which, given some input u of length n, is able to retrieve the circuit C_n of the family $\mathcal{C}$, simulate it, and output its result in polynomial time.



Since the circuits family $\mathcal{C}$ decides L, so does $\mathcal{N}$ in polynomial time, i.e. $L(\mathcal{N}) = L$.

Proof – First Implication

- Then, one can build some RNN[$\mathbb{R}$] $\mathcal{N}$ which contains the real $r(\mathcal{C})$ as a synaptic weight, and which, given some input u of length n, is able to retrieve the circuit C_n of the family $\mathcal{C}$, simulate it, and output its result in polynomial time.



Since the circuits family $\mathcal{C}$ decides L, so does $\mathcal{N}$ in polynomial time, i.e. $L(\mathcal{N}) = L$.

Proof – Second Implication

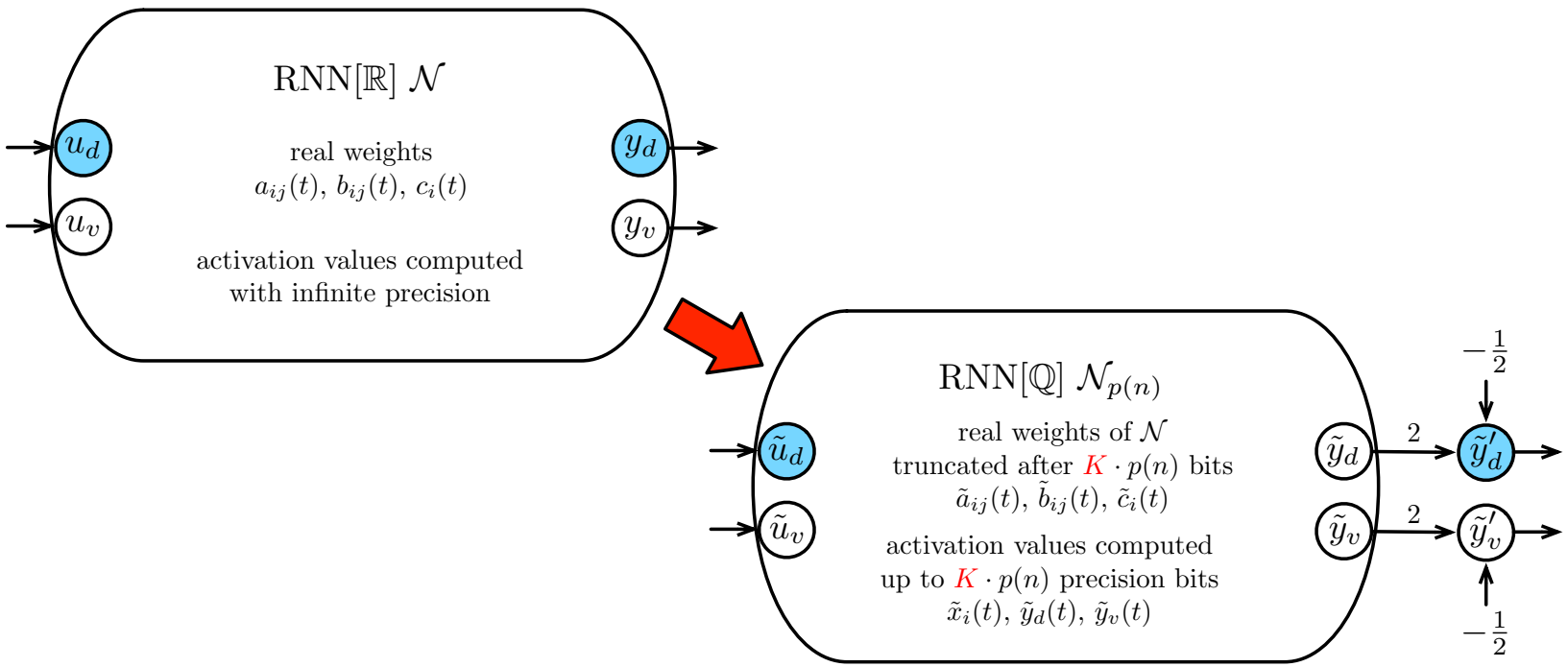
Let L be decidable in polynomial time p by some RNN[$\mathbb{R}$] $\mathcal{N}$.

- Then, by some technical lemma, there exists a so-called *p-truncated family* of Ev-RNN[$\mathbb{Q}$]s $\{\mathcal{N}_{p(n)} : n \geq 0\}$ such that each network $\mathcal{N}_{p(n)}$ computes exactly like $\mathcal{N}$ up to time $p(n)$.

Proof – Second Implication

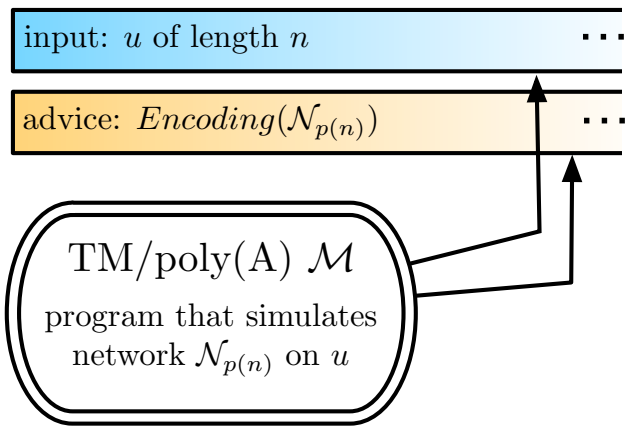
Let L be decidable in polynomial time p by some RNN[$\mathbb{R}$] $\mathcal{N}$.

- Then, by some technical lemma, there exists a so-called p-*truncated family* of Ev-RNN[$\mathbb{Q}$]s $\{\mathcal{N}_{p(n)} : n \geq 0\}$ such that each network $\mathcal{N}_{p(n)}$ computes exactly like $\mathcal{N}$ up to time $p(n)$.



Proof – Second Implication

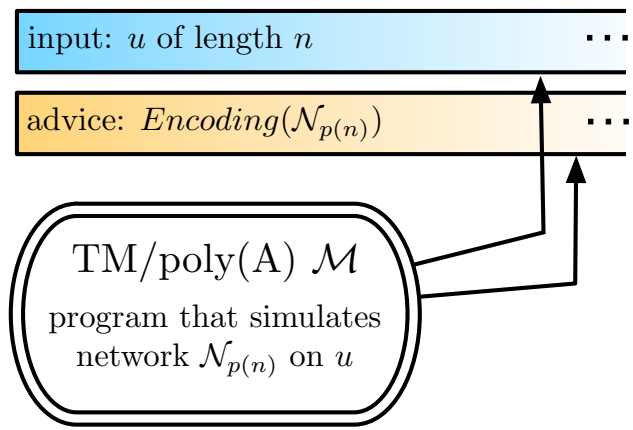
- We build a TM/poly(A) $\mathcal{M}$ with the advice $\alpha(n) = \ulcorner \mathcal{N}_{p(n)} \urcorner$ for each $n \geq 0$, and which, on every input u of length n:
  1. calls the polynomial-bounded advice value $\ulcorner \mathcal{N}_{p(n)} \urcorner$
  2. simulates the behaviour of $\mathcal{N}_{p(n)}$ on u in polynomial time



The machine $\mathcal{M}$ answers precisely like the RNN[$\mathbb{R}$] $\mathcal{N}$, hence decides L in polynomial time. Therefore $L \in P/poly$.

Proof – Second Implication

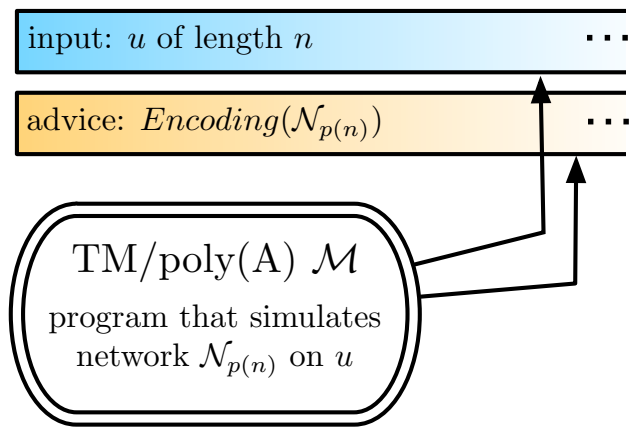
- We build a TM/poly(A) $\mathcal{M}$ with the advice $\alpha(n) = \ulcorner \mathcal{N}_{p(n)} \urcorner$ for each $n \geq 0$, and which, on every input u of length n:
  1. calls the polynomial-bounded advice value $\ulcorner \mathcal{N}_{p(n)} \urcorner$
  2. simulates the behaviour of $\mathcal{N}_{p(n)}$ on u in polynomial time



The machine $\mathcal{M}$ answers precisely like the RNN[$\mathbb{R}$] $\mathcal{N}$, hence decides L in polynomial time. Therefore $L \in P/poly$.

Proof – Second Implication

- We build a TM/poly(A) $\mathcal{M}$ with the advice $\alpha(n) = \ulcorner \mathcal{N}_{p(n)} \urcorner$ for each $n \geq 0$, and which, on every input u of length n:
  1. calls the polynomial-bounded advice value $\ulcorner \mathcal{N}_{p(n)} \urcorner$
  2. simulates the behaviour of $\mathcal{N}_{p(n)}$ on u in polynomial time



The machine $\mathcal{M}$ answers precisely like the RNN[$\mathbb{R}$] $\mathcal{N}$, hence decides L in polynomial time. Therefore $L \in P/poly$.

Result – Summary

Computational power of recurrent neural networks in polynomial time of computation.

	Recurrent Neural Networks
$\mathbb{Q}$	P
$\mathbb{R}$	P/poly

Analog RNNs over infinite words

We now extend the study of the computational power of analog recurrent neural networks to the context of infinite word reading machines.

Topology over the Cantor Space

Let $\mathcal{C} = \{0,1\}^\omega$ be the *Cantor space*.

- The *basic open sets* of $\mathcal{C}$ are of the form $p \cdot \{0,1\}^\omega$, for some $p \in \{0,1\}^+$.
- The class Δ_1^1 of Borel sets of $\mathcal{C}$ consists of the σ-algebra generated by the open sets.
- The levels of the *Borel hierarchy* are defined by induction on $\alpha < \omega_1$ as follows:

Topology over the Cantor Space

Let $\mathcal{C} = \{0,1\}^\omega$ be the *Cantor space*.

- The *basic open sets* of $\mathcal{C}$ are of the form $p \cdot \{0,1\}^\omega$, for some $p \in \{0,1\}^+$.
- The class $\mathbf{\Delta}^1_1$ of Borel sets of $\mathcal{C}$ consists of the σ-algebra generated by the open sets.
- The levels of the *Borel hierarchy* are defined by induction on $\alpha < \omega_1$ as follows:
  - $\mathbf{\Sigma}^0_1 = \{U \in \mathcal{C} \mid U \text{ is open}\}$
  - $\mathbf{\Sigma}^0_\alpha = \{\bigcup_{n \in \mathbb{N}} A_n \mid A_n \in \mathbf{\Pi}^0_{\beta_n} \text{ for } \beta_n < \alpha\}$
  - $\mathbf{\Pi}^0_\alpha = \{A \mid A^c \in \mathbf{\Sigma}^0_\alpha\}$
  - $\mathbf{\Delta}^0_\alpha = \mathbf{\Sigma}^0_\alpha \cap \mathbf{\Pi}^0_\alpha$

Topology over the Cantor Space

Let $\mathcal{C} = \{0,1\}^\omega$ be the *Cantor space*.

- The *basic open sets* of $\mathcal{C}$ are of the form $p \cdot \{0,1\}^\omega$, for some $p \in \{0,1\}^+$.
- The class $\mathbf{\Delta}_1^1$ of Borel sets of $\mathcal{C}$ consists of the σ-algebra generated by the open sets.
- The levels of the *Borel hierarchy* are defined by induction on $\alpha < \omega_1$ as follows:
  - $\mathbf{\Sigma}_1^0 = \{A \subseteq \mathcal{C} : A \text{ is open}\}$
  - $\mathbf{\Sigma}_\alpha^0 = \{\bigcup_{n \in \mathbb{N}} A_n : A_n \in \mathbf{\Pi}_{\beta_n}^0 \text{ for } \beta_n < \alpha\}$
  - $\mathbf{\Pi}_\alpha^0 = \{A : A^c \in \mathbf{\Sigma}_\alpha^0\}$
  - $\mathbf{\Delta}_\alpha^0 = \{A : A \in \mathbf{\Sigma}_\alpha^0 \cap \mathbf{\Pi}_\alpha^0\}$

Topology over the Cantor Space

Let $\mathcal{C} = \{0,1\}^\omega$ be the *Cantor space*.

- The *basic open sets* of $\mathcal{C}$ are of the form $p \cdot \{0,1\}^\omega$, for some $p \in \{0,1\}^+$.
- The class $\mathbf{\Delta}_1^1$ of Borel sets of $\mathcal{C}$ consists of the σ-algebra generated by the open sets.
- The levels of the *Borel hierarchy* are defined by induction on $\alpha < \omega_1$ as follows:
  - $\mathbf{\Sigma}_1^0 = \{A \subseteq \mathcal{C} : A \text{ is open}\}$
  - $\mathbf{\Sigma}_\alpha^0 = \{\bigcup_{n\in\mathbb{N}} A_n : A_n \in \mathbf{\Pi}_\beta^0 \text{ for } \beta < \alpha\}$
  - $\mathbf{\Pi}_\alpha^0 = \{A : A^\complement \in \mathbf{\Sigma}_\alpha^0\}$
  - $\mathbf{\Delta}_\alpha^0 = \{A : A \in \mathbf{\Sigma}_\alpha^0 \cap \mathbf{\Pi}_\alpha^0\}$

Topology over the Cantor Space

Let $\mathcal{C} = \{0,1\}^\omega$ be the *Cantor space*.

- The *basic open sets* of $\mathcal{C}$ are of the form $p \cdot \{0,1\}^\omega$, for some $p \in \{0,1\}^+$.
- The class $\mathbf{\Delta_1^1}$ of Borel sets of $\mathcal{C}$ consists of the σ-algebra generated by the open sets.
- The levels of the *Borel hierarchy* are defined by induction on $\alpha < \omega_1$ as follows:
  - $\mathbf{\Sigma_1^0} = \{A \subseteq \mathcal{C} : A \text{ is open}\}$
  - $\mathbf{\Sigma_\alpha^0} = \{\bigcup_{n\in\mathbb{N}} A_n : A_n \in \mathbf{\Pi_\beta^0} \text{ for } \beta < \alpha\}$
  - $\mathbf{\Pi_\alpha^0} = \{A : A^\complement \in \mathbf{\Sigma_\alpha^0}\}$
  - $\mathbf{\Delta_\alpha^0} = \{A : A \in \mathbf{\Sigma_\alpha^0} \cap \mathbf{\Pi_\alpha^0}\}$

Topology over the Cantor Space

Let $\mathcal{C} = \{0,1\}^\omega$ be the *Cantor space*.

- The *basic open sets* of $\mathcal{C}$ are of the form $p \cdot \{0,1\}^\omega$, for some $p \in \{0,1\}^+$.
- The class $\mathbf{\Delta_1^1}$ of Borel sets of $\mathcal{C}$ consists of the σ-algebra generated by the open sets.
- The levels of the *Borel hierarchy* are defined by induction on $\alpha < \omega_1$ as follows:
  - $\mathbf{\Sigma_1^0} = \{A \subseteq \mathcal{C} : A \text{ is open}\}$
  - $\mathbf{\Sigma_\alpha^0} = \{\bigcup_{n\in\mathbb{N}} A_n : A_n \in \mathbf{\Pi_\beta^0} \text{ for } \boldsymbol{\beta < \alpha}\}$
  - $\mathbf{\Pi_\alpha^0} = \{A : A^\complement \in \mathbf{\Sigma_\alpha^0}\}$
  - $\mathbf{\Delta_\alpha^0} = \{A : A \in \mathbf{\Sigma_\alpha^0} \cap \mathbf{\Pi_\alpha^0}\}$

Topology over the Cantor Space

Let $\mathcal{C} = \{0,1\}^\omega$ be the *Cantor space*.

- The *basic open sets* of $\mathcal{C}$ are of the form $p \cdot \{0,1\}^\omega$, for some $p \in \{0,1\}^+$.
- The class $\mathbf{\Delta_1^1}$ of Borel sets of $\mathcal{C}$ consists of the σ-algebra generated by the open sets.
- The levels of the *Borel hierarchy* are defined by induction on $\alpha < \omega_1$ as follows:
  - $\mathbf{\Sigma_1^0} = \{A \subseteq \mathcal{C} : A \text{ is open}\}$
  - $\mathbf{\Sigma_\alpha^0} = \{\bigcup_{n\in\mathbb{N}} A_n : A_n \in \mathbf{\Pi_\beta^0} \text{ for } \boldsymbol{\beta < \alpha}\}$
  - $\mathbf{\Pi_\alpha^0} = \{A : A^\complement \in \mathbf{\Sigma_\alpha^0}\}$
  - $\mathbf{\Delta_\alpha^0} = \{A : A \in \mathbf{\Sigma_\alpha^0} \cap \mathbf{\Pi_\alpha^0}\}$

Topology over the Cantor Space

Let $\mathcal{C} = \{0,1\}^\omega$ be the *Cantor space*.

- The *basic open sets* of $\mathcal{C}$ are of the form $p \cdot \{0,1\}^\omega$, for some $p \in \{0,1\}^+$.
- The class $\mathbf{\Delta_1^1}$ of Borel sets of $\mathcal{C}$ consists of the σ-algebra generated by the open sets.
- The levels of the *Borel hierarchy* are defined by induction on $\alpha < \omega_1$ as follows:
  - $\mathbf{\Sigma_1^0} = \{A \subseteq \mathcal{C} : A \text{ is open}\}$
  - $\mathbf{\Sigma_\alpha^0} = \{\bigcup_{n\in\mathbb{N}} A_n : A_n \in \mathbf{\Pi_\beta^0} \text{ for } \boldsymbol{\beta < \alpha}\}$
  - $\mathbf{\Pi_\alpha^0} = \{A : A^\complement \in \mathbf{\Sigma_\alpha^0}\}$
  - $\mathbf{\Delta_\alpha^0} = \{A : A \in \mathbf{\Sigma_\alpha^0} \cap \mathbf{\Pi_\alpha^0}\}$

Topology over the Cantor Space

Let $\mathcal{C} \times \mathcal{C}$ be equipped with the product topology of $\mathcal{C}$.

- A set $X \subseteq \mathcal{C}$ is *analytic* (Σ^1_1) iff it is the projection of some Π^0_2-set, or more generally, of some Borel set $Y \subseteq \mathcal{C} \times \mathcal{C}$, i.e.

$$X = \pi_1(Y) = \{x \in \mathcal{C} : (x, y) \in Y \text{ for some } y \in \mathcal{C}\}$$

- A set $X \subseteq \mathcal{C}$ is *effectively analytic* (Σ^1_1) iff it is recognized by some non-deterministic Büchi Turing machine.

- The class of analytic sets strictly contains that of Borel sets and that of effectively analytic sets, i.e. $\Sigma^0_2 \subsetneq \Sigma^1_1$ and $\Sigma^0_2 \subsetneq \Sigma^1_1$.

Topology over the Cantor Space

Let $\mathcal{C} \times \mathcal{C}$ be equipped with the product topology of $\mathcal{C}$.

- A set $X \subseteq \mathcal{C}$ is *analytic* ($\mathbf{\Sigma^1_1}$) iff it is the projection of some $\mathbf{\Pi^0_2}$-set, or more generally, of some Borel set $Y \subseteq \mathcal{C} \times \mathcal{C}$, i.e.

$$X = \pi_1(Y) = \{x \in \mathcal{C} : (x, y) \in Y \text{ for some } y \in \mathcal{C}\}$$

- A set $X \subseteq \mathcal{C}$ is *effectively analytic* (Σ^1_1) iff it is recognized by some non-deterministic Büchi Turing machine.
- The class of analytic sets strictly contains that of Borel sets and that of effectively analytic sets, i.e. $\mathbf{\Sigma^1_1} \supsetneq \mathbf{\Delta^1_1}$ and $\mathbf{\Sigma^1_1} \supsetneq \Sigma^1_1$.

Topology over the Cantor Space

Let $\mathcal{C} \times \mathcal{C}$ be equipped with the product topology of $\mathcal{C}$.

- A set $X \subseteq \mathcal{C}$ is *analytic* ($\mathbf{\Sigma_1^1}$) iff it is the projection of some $\mathbf{\Pi_2^0}$-set, or more generally, of some Borel set $Y \subseteq \mathcal{C} \times \mathcal{C}$, i.e.

$$X = \pi_1(Y) = \{x \in \mathcal{C} : (x, y) \in Y \text{ for some } y \in \mathcal{C}\}$$

- A set $X \subseteq \mathcal{C}$ is *effectively analytic* (Σ_1^1) iff it is recognized by some non-deterministic Büchi Turing machine.
- The class of analytic sets strictly contains that of Borel sets and that of effectively analytic sets, i.e. $\mathbf{\Sigma_1^1} \supsetneq \mathbf{\Delta_1^1}$ and $\mathbf{\Sigma_1^1} \supsetneq \Sigma_1^1$.

Topology over the Cantor Space

Let $\mathcal{C} \times \mathcal{C}$ be equipped with the product topology of $\mathcal{C}$.

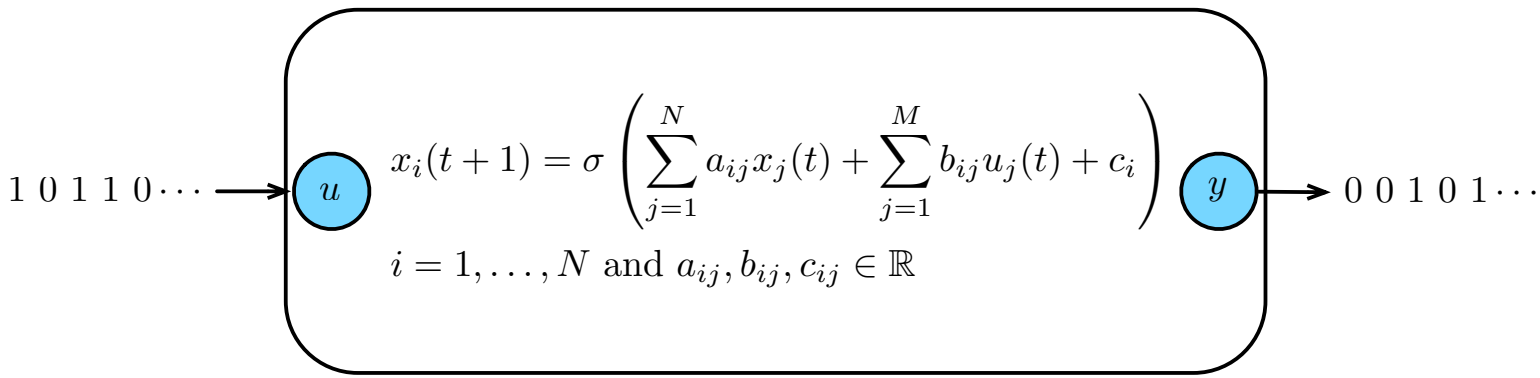
- A set $X \subseteq \mathcal{C}$ is *analytic* ($\mathbf{\Sigma^1_1}$) iff it is the projection of some $\mathbf{\Pi^0_2}$-set, or more generally, of some Borel set $Y \subseteq \mathcal{C} \times \mathcal{C}$, i.e.

$$X = \pi_1(Y) = \{x \in \mathcal{C} : (x, y) \in Y \text{ for some } y \in \mathcal{C}\}$$

- A set $X \subseteq \mathcal{C}$ is *effectively analytic* (Σ^1_1) iff it is recognized by some non-deterministic Büchi Turing machine.
- The class of analytic sets strictly contains that of Borel sets and that of effectively analytic sets, i.e. $\mathbf{\Sigma^1_1} \supsetneq \mathbf{\Delta^1_1}$ and $\mathbf{\Sigma^1_1} \supsetneq \Sigma^1_1$.

Deterministic Analog RNN on Infinite Words

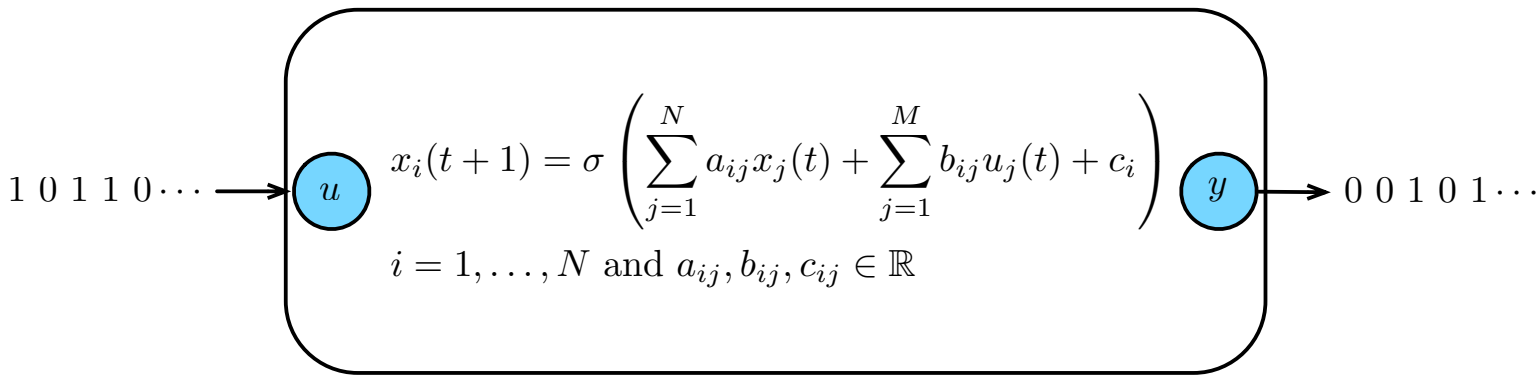
A *deterministic real-weighted (or analog) RNN over infinite words* (ω-Det-RNN[$\mathbb{R}$]) is a real-weighted RNN equipped with one binary input cell u and one binary output cell y.



- Any ω-Det-RNN[$\mathbb{R}$] can naturally be identified with some function $f_{\mathcal{N}} : \{0,1\}^\omega \to \{0,1\}^\omega$.
- By its sequential nature, $f_{\mathcal{N}}$ is Lipschitz, thus continuous.

Deterministic Analog RNN on Infinite Words

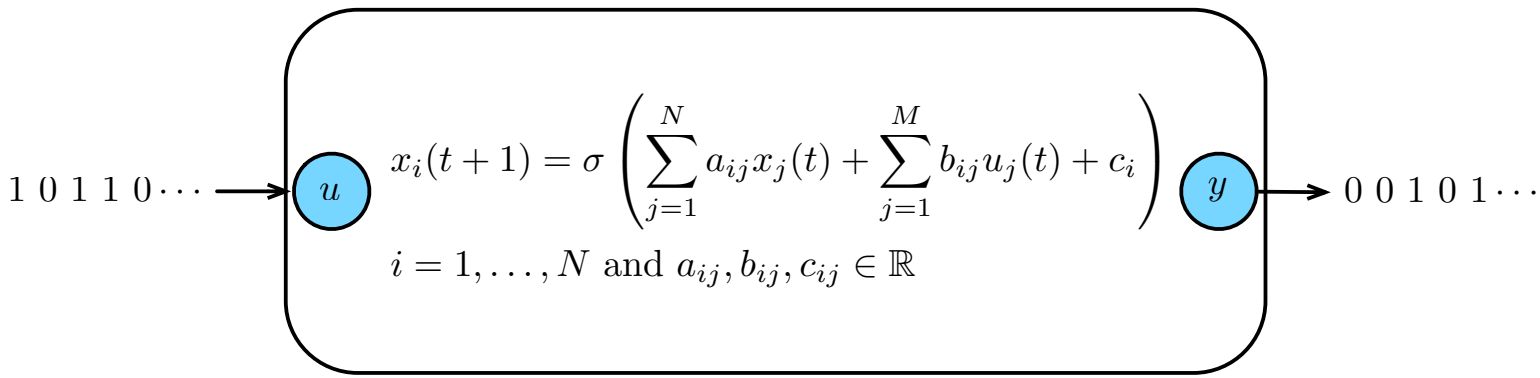
A *deterministic real-weighted (or analog) RNN over infinite words* (ω-Det-RNN[$\mathbb{R}$]) is a real-weighted RNN equipped with one binary input cell u and one binary output cell y.



- Any ω-Det-RNN[$\mathbb{R}$] can naturally be identified with some function $f_{\mathcal{N}} : \{0,1\}^\omega \to \{0,1\}^\omega$.
- By its sequential nature, $f_{\mathcal{N}}$ is Lipschitz, thus continuous.

Deterministic Analog RNN on Infinite Words

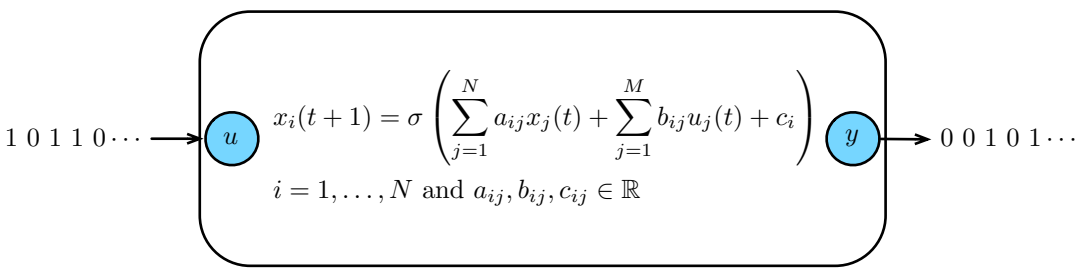
A *deterministic real-weighted (or analog) RNN over infinite words* (ω-Det-RNN[$\mathbb{R}$]) is a real-weighted RNN equipped with one binary input cell u and one binary output cell y.



- Any ω-Det-RNN[$\mathbb{R}$] can naturally be identified with some function $f_{\mathcal{N}} : \{0,1\}^\omega \to \{0,1\}^\omega$.
- By its sequential nature, $f_{\mathcal{N}}$ is Lipschitz, thus continuous.

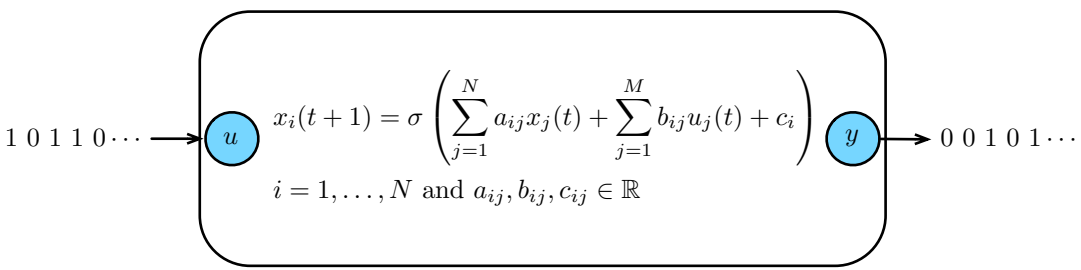
Deterministic Analog RNN on Infinite Words

- An infinite word $w \in \{0,1\}^\omega$ is *accepted* by $\mathcal{N}$ if it induces infinitely many output responses (i.e. 1's) to the output cell y (Büchi-like accepting condition).
- The *neural language* $L(\mathcal{N})$ *recognized* by $\mathcal{N}$ consists of the set of infinite words accepted by $\mathcal{N}$.
- **Remark:** Let $1_\infty = \{w \in \{0,1\}^\omega : w \text{ contains } \infty\text{-many } 1\text{'s}\}$. One has by definition that $L(\mathcal{N}) = f_{\mathcal{N}}^{-1}(1_\infty)$.



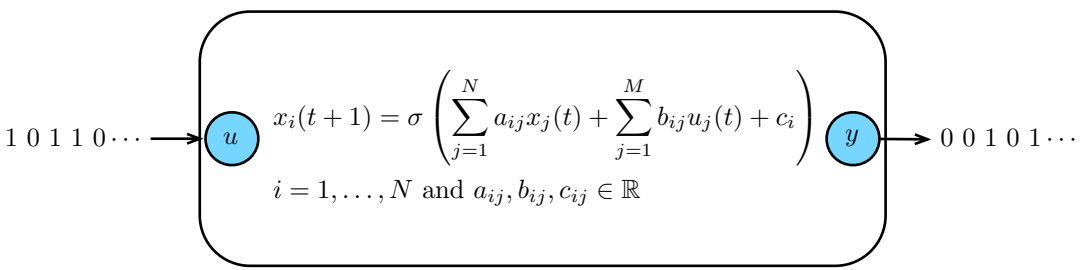
Deterministic Analog RNN on Infinite Words

- An infinite word $w \in \{0,1\}^\omega$ is *accepted* by $\mathcal{N}$ if it induces infinitely many output responses (i.e. 1's) to the output cell y (Büchi-like accepting condition).
- The *neural language* $L(\mathcal{N})$ *recognized* by $\mathcal{N}$ consists of the set of infinite words accepted by $\mathcal{N}$.
- **Remark:** Let $1_\infty = \{w \in \{0,1\}^\omega : w \text{ contains } \infty\text{-many } 1\text{'s}\}$. One has by definition that $L(\mathcal{N}) = f_{\mathcal{N}}^{-1}(1_\infty)$.



Deterministic Analog RNN on Infinite Words

- An infinite word $w \in \{0,1\}^\omega$ is *accepted* by $\mathcal{N}$ if it induces infinitely many output responses (i.e. 1's) to the output cell y (Büchi-like accepting condition).
- The *neural language* $L(\mathcal{N})$ *recognized* by $\mathcal{N}$ consists of the set of infinite words accepted by $\mathcal{N}$.
- **Remark:** Let $1_\infty = \{w \in \{0,1\}^\omega : w \text{ contains } \infty\text{-many } 1\text{'s}\}$. One has by definition that $L(\mathcal{N}) = f_{\mathcal{N}}^{-1}(1_\infty)$.



Result

Deterministic analog RNNs on infinite words recognize precisely the $\mathbf{\Pi_2^0}$-sets.

Theorem (Cabessa & Villa 12)

Let $L \subseteq \{0,1\}^\omega$. The following conditions are equivalent:

- *L is recognizable by some ω-Det-RNN[$\mathbb{R}$]*
- $L \in \mathbf{\Pi_2^0}$

Proof – First Implication

Let $L \subseteq \{0,1\}^\omega$ be recognizable by some ω-Det-RNN[$\mathbb{R}$] $\mathcal{N}$. One has:

- $L(\mathcal{N}) = f_{\mathcal{N}}^{-1}(1_\infty)$
- $f_{\mathcal{N}}$ is continuous
- $1_\infty = \bigcap_{n\geq 0} \bigcup_{m\geq 0} \{0,1\}^{n+m} 1 \{0,1\}^\omega \in \mathbf{\Pi_2^0}$

It follows that $L(\mathcal{N}) \in \mathbf{\Pi_2^0}$.

Proof – Second Implication

Let $L \in \mathbf{\Pi_2^0}$. Then L is of the form $L = \bigcap_{i \geq 0} \bigcup_{j \geq 0} p_{i,j} \cdot \{0,1\}^\omega$, where each $p_{i,j} \in \{0,1\}^+$.

- We provide a suitable encoding of the infinite sequence $(p_{i,j})_{i,j \geq 0}$ into some real number $r(L)$.
- (technical lemma) There exists a RNN$[\mathbb{R}]$ $\mathcal{N}_{r(L)}$ which contains the real $r(L)$ as a synaptic weight, and which, given some encoding of (u, p) as input, is able to output some suitable encoding of pu.

Proof – Second Implication

Let $L \in \mathbf{\Pi_2^0}$. Then L is of the form $L = \bigcap_{i \geq 0} \bigcup_{j \geq 0} p_{i,j} \cdot \{0,1\}^\omega$, where each $p_{i,j} \in \{0,1\}^+$.

- We provide a suitable encoding of the infinite sequence $(p_{i,j})_{i,j \geq 0}$ into some real number $r(L)$.
- (technical lemma) There exists a RNN$[\mathbb{R}]$ $\mathcal{N}_{r(L)}$, which contains the real $r(L)$ as a synaptic weight, and which, given some encoding of (i,j) as input, is able to output some suitable encoding of $p_{i,j}$.

Proof – Second Implication

Let $L \in \mathbf{\Pi_2^0}$. Then L is of the form $L = \bigcap_{i \geq 0} \bigcup_{j \geq 0} p_{i,j} \cdot \{0,1\}^\omega$, where each $p_{i,j} \in \{0,1\}^+$.

- We provide a suitable encoding of the infinite sequence $(p_{i,j})_{i,j \geq 0}$ into some real number $r(L)$.
- (technical lemma) There exists a RNN$[\mathbb{R}]$ $\mathcal{N}_{r(L)}$, which contains the real $r(L)$ as a synaptic weight, and which, given some encoding of (i,j) as input, is able to output some suitable encoding of $p_{i,j}$.

Proof – Second Implication

Let $L \in \mathbf{\Pi_2^0}$. Then L is of the form $L = \bigcap_{i \geq 0} \bigcup_{j \geq 0} p_{i,j} \cdot \{0,1\}^\omega$, where each $p_{i,j} \in \{0,1\}^+$.

- We provide a suitable encoding of the infinite sequence $(p_{i,j})_{i,j \geq 0}$ into some real number $r(L)$.
- (technical lemma) There exists a RNN[$\mathbb{R}$] $\mathcal{N}_{r(L)}$, which contains the real $r(L)$ as a synaptic weight, and which, given some encoding of (i,j) as input, is able to output some suitable encoding of $p_{i,j}$.

Proof – Second Implication

- We build an ω-Det-RNN[$\mathbb{R}$] $\mathcal{N}$ which contains $\mathcal{N}_{r(L)}$ as a sub-network and which performs the following algorithm:

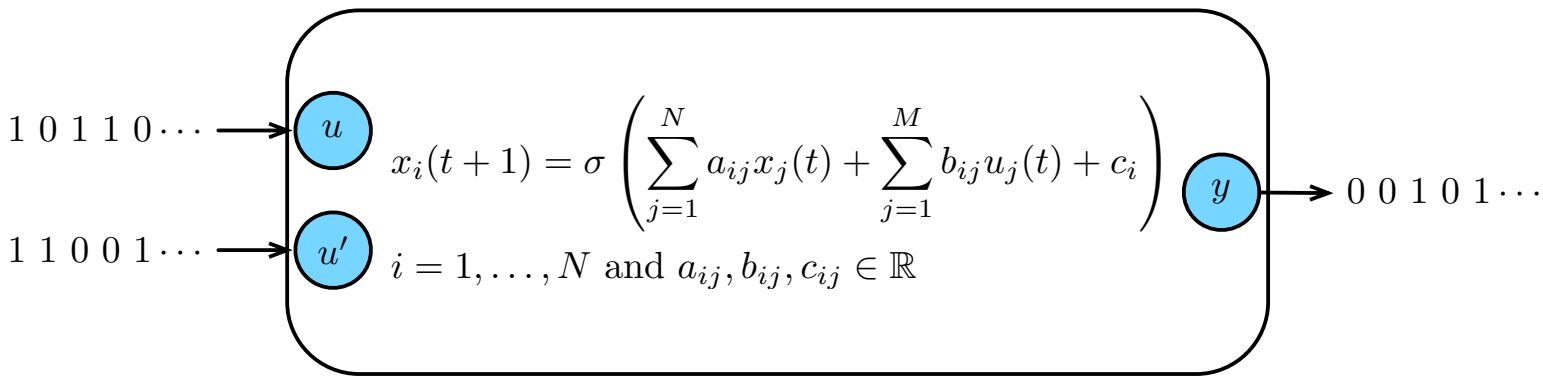
Algorithm 1

Input s is provided bit by bit at successive time steps
$i \leftarrow 0$, $j \leftarrow 0$
loop
  Submit input (i, j) to $\mathcal{N}_{r(L)}$ and get $p_{i,j}$ as output
  Wait until $|s| \geq |p_{i,j}|$
  if $p_{i,j} \subseteq s$ **then** # $s \in \bigcup_{j \geq 0} p_{i,j}\{0,1\}^\omega$
    return 1 and **do** $i \leftarrow i+1$, $j \leftarrow 0$ # test if $s \in p_{i+1,j}\{0,1\}^\omega$
  else # $s \notin \bigcup_{k \leq j} p_{i,k}\{0,1\}^\omega$
    return 0 and **do** $i \leftarrow i$, $j \leftarrow j+1$ # test if $s \in p_{i,j+1}\{0,1\}^\omega$
  end if
end loop

In this way, $\mathcal{N}$ outputs infinitely many 1's iff $s \in L$, i.e $L(\mathcal{N}) = L$.

Non-Deterministic Analog RNN on Infinite Words

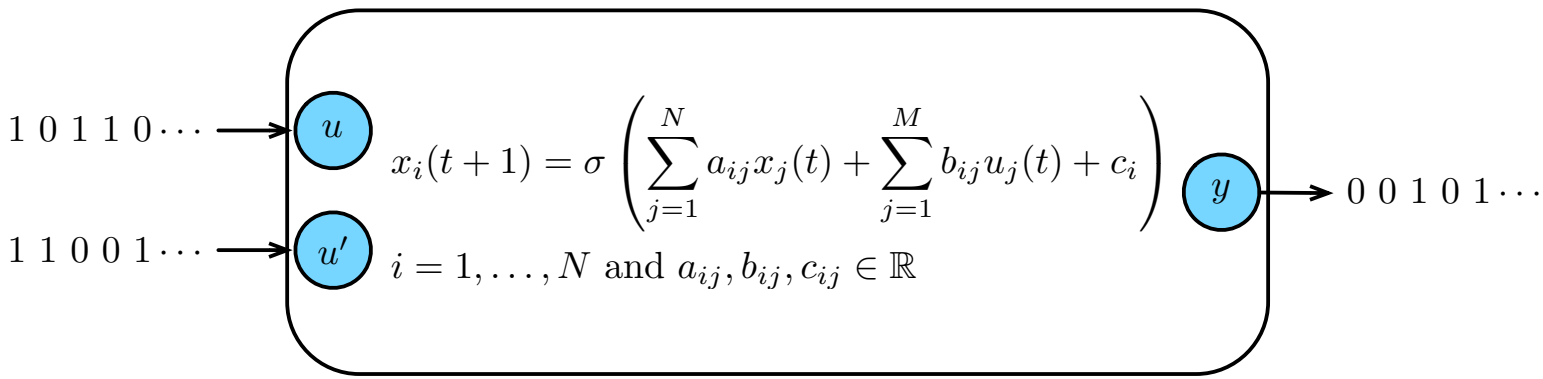
A *non-deterministic real-weighted (or analog) RNN over infinite words* (ω-NDet-RNN[$\mathbb{R}$]) is a real-weighted RNN equipped with two binary input cells u and u' and one binary output cell y.



- Any ω-NDet-RNN[$\mathbb{R}$] can naturally be identified with some function $f_{\mathcal{N}} : \{0,1\}^\omega \times \{0,1\}^\omega \to \{0,1\}^\omega$.
- By its sequential nature, $f_{\mathcal{N}}$ is Lipschitz, thus continuous.

Non-Deterministic Analog RNN on Infinite Words

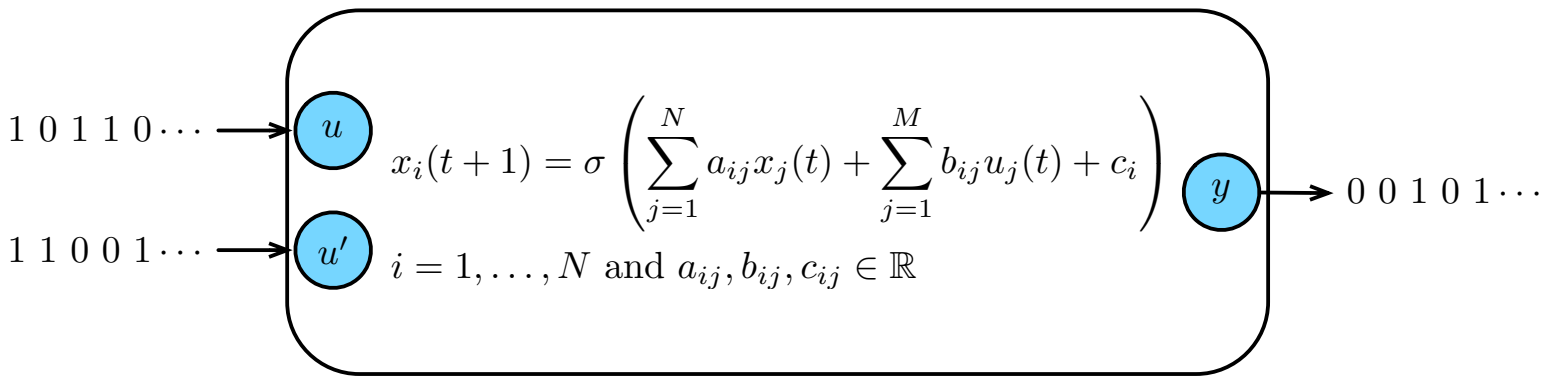
A *non-deterministic real-weighted (or analog) RNN over infinite words* (ω-NDet-RNN[$\mathbb{R}$]) is a real-weighted RNN equipped with two binary input cells u and u' and one binary output cell y.



- Any ω-NDet-RNN[$\mathbb{R}$] can naturally be identified with some function $f_{\mathcal{N}} : \{0,1\}^\omega \times \{0,1\}^\omega \to \{0,1\}^\omega$.
- By its sequential nature, $f_{\mathcal{N}}$ is Lipschitz, thus continuous.

Non-Deterministic Analog RNN on Infinite Words

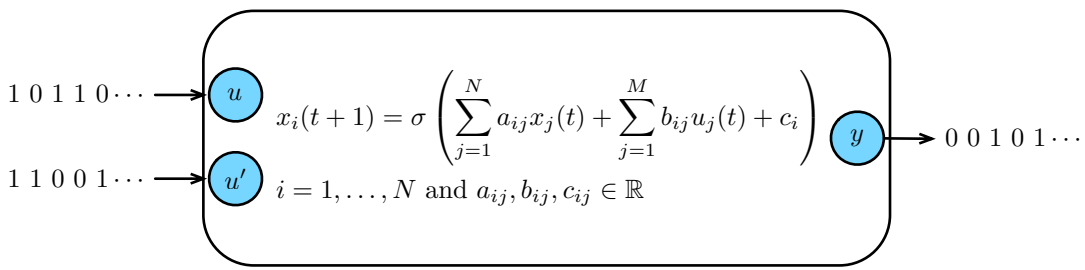
A *non-deterministic real-weighted (or analog) RNN over infinite words* (ω-NDet-RNN[$\mathbb{R}$]) is a real-weighted RNN equipped with two binary input cells u and u' and one binary output cell y.



- Any ω-NDet-RNN[$\mathbb{R}$] can naturally be identified with some function $f_{\mathcal{N}} : \{0,1\}^\omega \times \{0,1\}^\omega \to \{0,1\}^\omega$.
- By its sequential nature, $f_{\mathcal{N}}$ is Lipschitz, thus continuous.

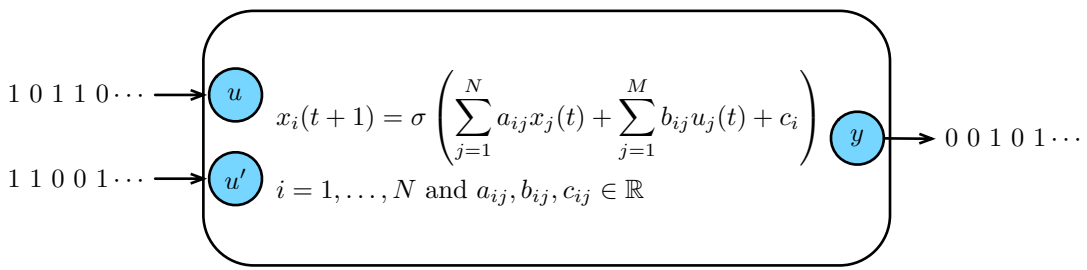
Non-Deterministic Analog RNN on Infinite Words

- An infinite word $w \in \{0,1\}^\omega$ is *accepted* by $\mathcal{N}$ if there exists a guess stream w' such that w and w' induces infinitely many output responses to the output cell y (Büchi-like condition).
- The *neural language* $L(\mathcal{N})$ *recognized* by $\mathcal{N}$ consists of the set of infinite words accepted by $\mathcal{N}$.
- **Remark:** $L(\mathcal{N}) = \{w \in \{0,1\}^\omega : \exists w' \in \{0,1\}^\omega \quad (w, w') \in f_{\mathcal{N}}^{-1}(1_\infty)\} = \pi_1(f_{\mathcal{N}}^{-1}(1_\infty))$.



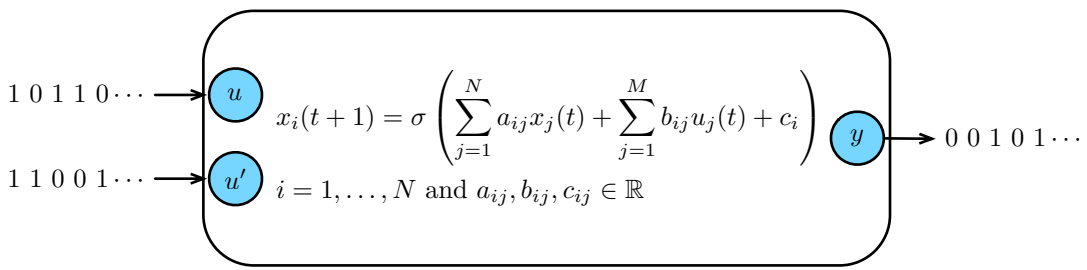
Non-Deterministic Analog RNN on Infinite Words

- An infinite word $w \in \{0,1\}^\omega$ is *accepted* by $\mathcal{N}$ if there exists a guess stream w' such that w and w' induces infinitely many output responses to the output cell y (Büchi-like condition).
- The *neural language* $L(\mathcal{N})$ *recognized* by $\mathcal{N}$ consists of the set of infinite words accepted by $\mathcal{N}$.
- **Remark:** $L(\mathcal{N}) = \{w \in \{0,1\}^\omega : \exists w' \in \{0,1\}^\omega \quad (w, w') \in f_{\mathcal{N}}^{-1}(1_\infty)\} = \pi_1(f_{\mathcal{N}}^{-1}(1_\infty))$.



Non-Deterministic Analog RNN on Infinite Words

- An infinite word $w \in \{0,1\}^\omega$ is *accepted* by $\mathcal{N}$ if there exists a guess stream w' such that w and w' induces infinitely many output responses to the output cell y (Büchi-like condition).
- The *neural language* $L(\mathcal{N})$ *recognized* by $\mathcal{N}$ consists of the set of infinite words accepted by $\mathcal{N}$.
- **Remark:** $L(\mathcal{N}) = \{w \in \{0,1\}^\omega : \exists w' \in \{0,1\}^\omega \quad (w, w') \in f_{\mathcal{N}}^{-1}(1_\infty)\} = \pi_1(f_{\mathcal{N}}^{-1}(1_\infty))$.



Result

Non-deterministic analog RNNs on infinite words recognize precisely the $\mathbf{\Sigma}_1^1$-sets.

Theorem (Cabessa & Villa 12)

Let $L \subseteq \{0,1\}^\omega$. The following conditions are equivalent:

- *L is recognizable by some ω-NDet-RNN[$\mathbb{R}$]*
- $L \in \mathbf{\Sigma}_1^1$

Proof – First Implication

Let $L \subseteq \{0,1\}^\omega$ be recognizable by some ω-NDet-RNN[$\mathbb{R}$] $\mathcal{N}$. One has:

- $L(\mathcal{N}) = \pi_1(f_{\mathcal{N}}^{-1}(1_\infty))$
- $f_{\mathcal{N}}$ is continuous
- $1_\infty \in \mathbf{\Pi_2^0}$

It follows that $L(\mathcal{N}) \in \mathbf{\Sigma_1^1}$.

Proof – Second Implication

Let $L \in \mathbf{\Sigma_1^1}$. Then $L = \pi_1(X)$, for some $\mathbf{\Pi_2^0}$-set X of the form

$$X = \bigcap_{i \geq 0} \bigcup_{j \geq 0} (p_{i,j} \cdot \{0,1\}^\omega \times q_{i,j} \cdot \{0,1\}^\omega),$$

where each $p_{i,j}$ and $q_{i,j} \in \{0,1\}^+$.

- We provide a suitable encoding of the infinite sequence of pairs $((p_{i,j}, q_{i,j}))_{i,j \geq 0}$ into some real number $r(X)$.
- (technical lemma) There exists a RNN[$\mathbb{R}$] $\mathcal{N}_{r(X)}$, which contains the real $r(X)$ as a synaptic weight, and which, given some encoding of (i, j) as input, is able to output some suitable encoding of $(p_{i,j}, q_{i,j})$.

Proof – Second Implication

Let $L \in \mathbf{\Sigma_1^1}$. Then $L = \pi_1(X)$, for some $\mathbf{\Pi}_2^0$-set X of the form

$$X = \bigcap_{i \geq 0} \bigcup_{j \geq 0} (p_{i,j} \cdot \{0,1\}^\omega \times q_{i,j} \cdot \{0,1\}^\omega),$$

where each $p_{i,j}$ and $q_{i,j} \in \{0,1\}^+$.

- We provide a suitable encoding of the infinite sequence of pairs $(p_{i,j}, q_{i,j})_{i,j \geq 0}$ into some real number $r(X)$.
- (technical lemma) There exists a RNN[$\mathbb{R}$] $\mathcal{N}_{r(X)}$, which contains the real $r(X)$ as a synaptic weight, and which, given some encoding of (i,j) as input, is able to output some suitable encoding of $(p_{i,j}, q_{i,j})$.

Proof – Second Implication

Let $L \in \mathbf{\Sigma_1^1}$. Then $L = \pi_1(X)$, for some $\mathbf{\Pi}_2^0$-set X of the form

$$X = \bigcap_{i \geq 0} \bigcup_{j \geq 0} (p_{i,j} \cdot \{0,1\}^\omega \times q_{i,j} \cdot \{0,1\}^\omega),$$

where each $p_{i,j}$ and $q_{i,j} \in \{0,1\}^+$.

- We provide a suitable encoding of the infinite sequence of pairs $((p_{i,j}, q_{i,j}))_{i,j \geq 0}$ into some real number $r(X)$.
- (technical lemma) There exists a RNN$[\mathbb{R}]$ $\mathcal{N}_{r(X)}$, which contains the real $r(X)$ as a synaptic weight, and which, given some encoding of (i,j) as input, is able to output some suitable encoding of $(p_{i,j}, q_{i,j})$.

Proof – Second Implication

Let $L \in \mathbf{\Sigma_1^1}$. Then $L = \pi_1(X)$, for some $\mathbf{\Pi_2^0}$-set X of the form

$$X = \bigcap_{i \geq 0} \bigcup_{j \geq 0} (p_{i,j} \cdot \{0,1\}^\omega \times q_{i,j} \cdot \{0,1\}^\omega),$$

where each $p_{i,j}$ and $q_{i,j} \in \{0,1\}^+$.

- We provide a suitable encoding of the infinite sequence of pairs $((p_{i,j}, q_{i,j}))_{i,j \geq 0}$ into some real number $r(X)$.
- (technical lemma) There exists a RNN[$\mathbb{R}$] $\mathcal{N}_{r(X)}$, which contains the real $r(X)$ as a synaptic weight, and which, given some encoding of (i,j) as input, is able to output some suitable encoding of $(p_{i,j}, q_{i,j})$.

Proof – Second Implication

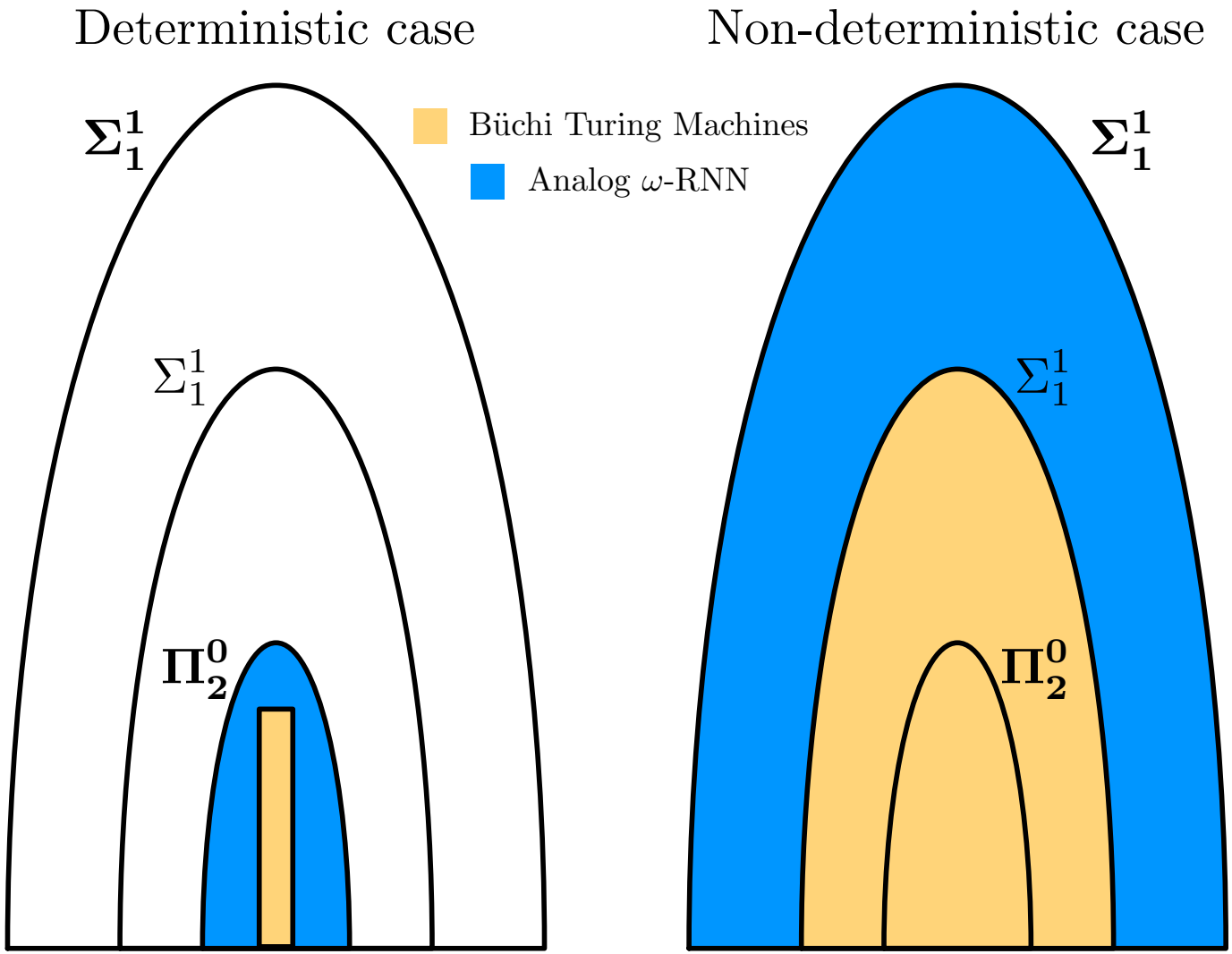
- We build an ω-NDet-RNN[$\mathbb{R}$] $\mathcal{N}$ which contains $\mathcal{N}_{r(L)}$ as a sub-network and which performs the following algorithm:

Algorithm 2

Input s is provided bit by bit at successive time steps
Guess g is provided bit by bit at successive time steps
$i \leftarrow 0$, $j \leftarrow 0$
loop
    Submit input (i, j) to $\mathcal{N}_{r(X)}$ and get $(p_{i,j}, q_{i,j})$ as output
    Wait until $|s| \geq p_{i,j}, q_{i,j}$
    if $p_{i,j} \subseteq s$ and $q_{i,j} \subseteq g$ **then**
        return 1 and **do** $i \leftarrow i + 1$, $j \leftarrow 0$
    else
        return 0 and **do** $i \leftarrow i$, $j \leftarrow j + 1$
    end if
end loop

$\mathcal{N}$ outputs ∞-many 1's iff $(s, g) \in X$, i.e $L(\mathcal{N}) = \pi_1(X) = L$.

Results – Summary



Introduction

- We study the computational capabilities of recurrent neural networks whose synaptic weights might evolve over time.
- These considerations are related to the biological concept of *plasticity* – in particular *synaptic plasticity* – which is crucially involved in the processing of information in the brain.
- These considerations are also related to the key concept of *learning* for artificial neural networks.

Introduction

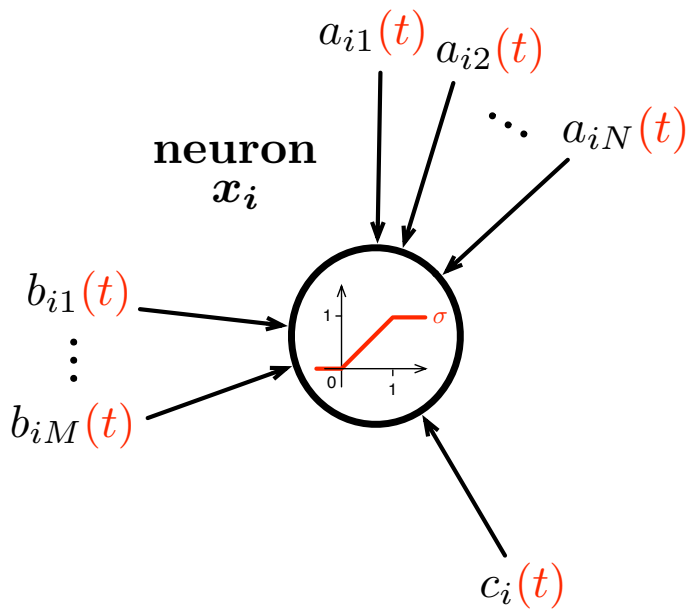
- We study the computational capabilities of recurrent neural networks whose synaptic weights might evolve over time.
- These considerations are related to the biological concept of *plasticity* – in particular *synaptic plasticity* – which is crucially involved in the processing of information in the brain.
- These considerations are also related to the key concept of *learning* for artificial neural networks.

Introduction

- We study the computational capabilities of recurrent neural networks whose synaptic weights might evolve over time.
- These considerations are related to the biological concept of *plasticity* – in particular *synaptic plasticity* – which is crucially involved in the processing of information in the brain.
- These considerations are also related to the key concept of *learning* for artificial neural networks.

Dynamics of Evolving RNNs

An *evolving RNN* (Ev-RNN) consists of a RNN whose synaptic weights might evolve over time (between bounded values)

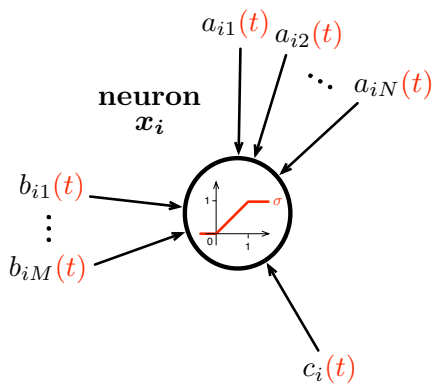


$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t) \cdot u_j(t) + c_i(t)\right)$$

Models of Evolving RNNs

We consider four models of evolving recurrent neural networks:

1. rational-weighted bi-valued evolving RNNs: $\text{Ev}_2\text{-RNN}[\mathbb{Q}]$s
2. rational-weighted (general) evolving RNNs: $\text{Ev-RNN}[\mathbb{Q}]$s
3. real-weighted bi-valued evolving RNNs: $\text{Ev}_2\text{-RNN}[\mathbb{R}]$s
4. real-weighted (general) evolving RNNs: $\text{Ev-RNN}[\mathbb{R}]$s

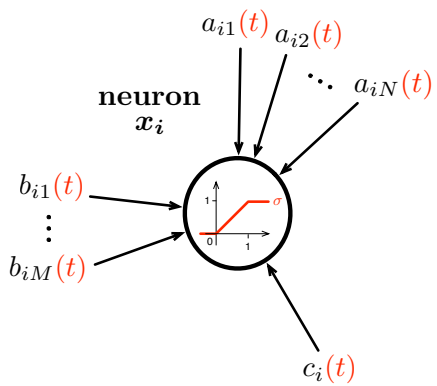


$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t) \cdot u_j(t) + c_i(t)\right)$$

Models of Evolving RNNs

We consider four models of evolving recurrent neural networks:

1. rational-weighted bi-valued evolving RNNs: Ev$_2$-RNN[$\mathbb{Q}$]s
2. rational-weighted (general) evolving RNNs: Ev-RNN[$\mathbb{Q}$]s
3. real-weighted bi-valued evolving RNNs: Ev$_2$-RNN[$\mathbb{R}$]s
4. real-weighted (general) evolving RNNs: Ev-RNN[$\mathbb{R}$]s

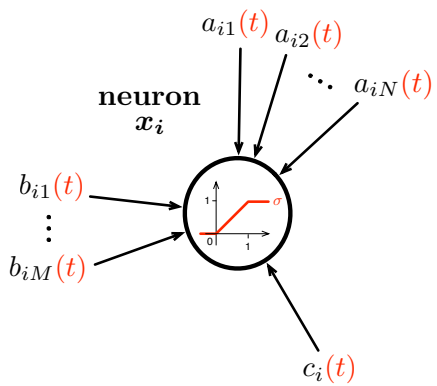


$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t) \cdot u_j(t) + c_i(t)\right)$$

Models of Evolving RNNs

We consider four models of evolving recurrent neural networks:

1. rational-weighted bi-valued evolving RNNs: Ev$_2$-RNN[$\mathbb{Q}$]s
2. rational-weighted (general) evolving RNNs: Ev-RNN[$\mathbb{Q}$]s
3. real-weighted bi-valued evolving RNNs: Ev$_2$-RNN[$\mathbb{R}$]s
4. real-weighted (general) evolving RNNs: Ev-RNN[$\mathbb{R}$]s

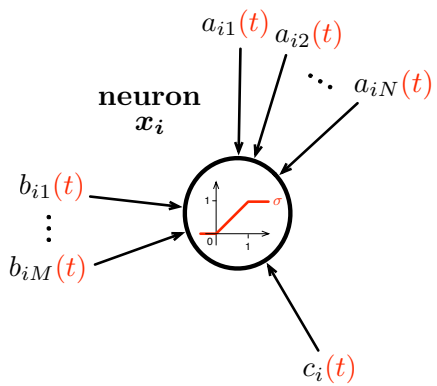


$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t) \cdot u_j(t) + c_i(t)\right)$$

Models of Evolving RNNs

We consider four models of evolving recurrent neural networks:

1. rational-weighted bi-valued evolving RNNs: $\text{Ev}_2\text{-RNN}[\mathbb{Q}]$s
2. rational-weighted (general) evolving RNNs: $\text{Ev-RNN}[\mathbb{Q}]$s
3. real-weighted bi-valued evolving RNNs: $\text{Ev}_2\text{-RNN}[\mathbb{R}]$s
4. real-weighted (general) evolving RNNs: $\text{Ev-RNN}[\mathbb{R}]$s

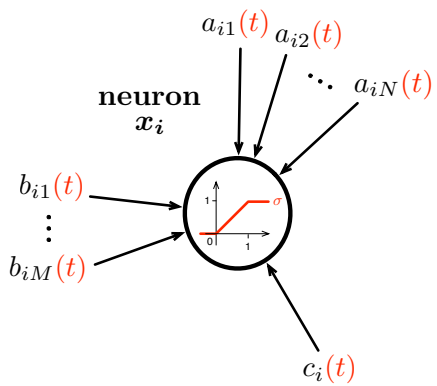


$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t) \cdot u_j(t) + c_i(t)\right)$$

Models of Evolving RNNs

We consider four models of evolving recurrent neural networks:

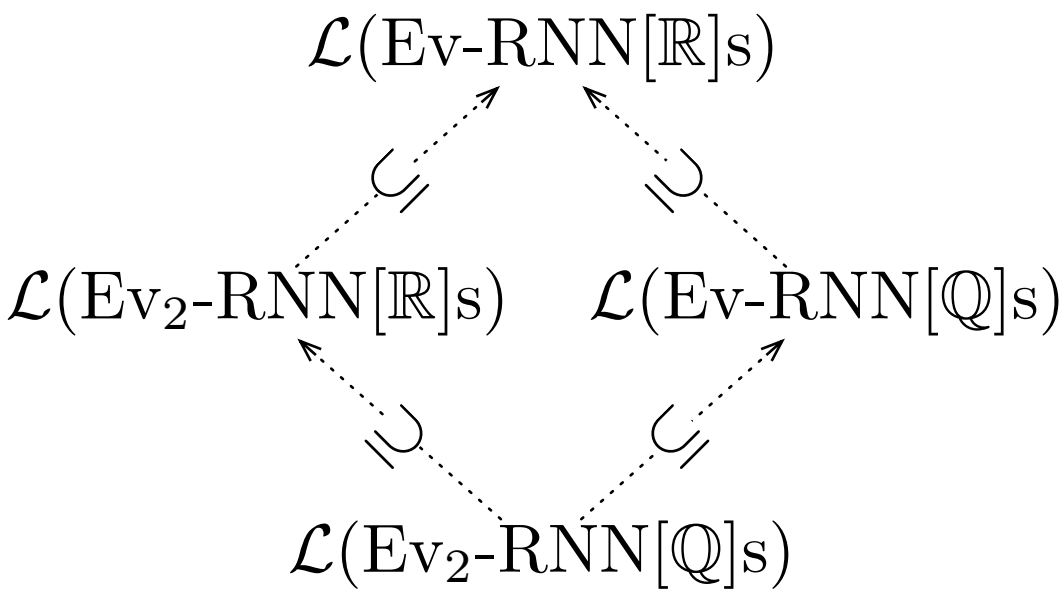
1. rational-weighted bi-valued evolving RNNs: $\text{Ev}_2\text{-RNN}[\mathbb{Q}]$s
2. rational-weighted (general) evolving RNNs: $\text{Ev-RNN}[\mathbb{Q}]$s
3. real-weighted bi-valued evolving RNNs: $\text{Ev}_2\text{-RNN}[\mathbb{R}]$s
4. real-weighted (general) evolving RNNs: $\text{Ev-RNN}[\mathbb{R}]$s



$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t) \cdot u_j(t) + c_i(t)\right)$$

Models of Evolving RNNs

One has the following relations between those models



Results

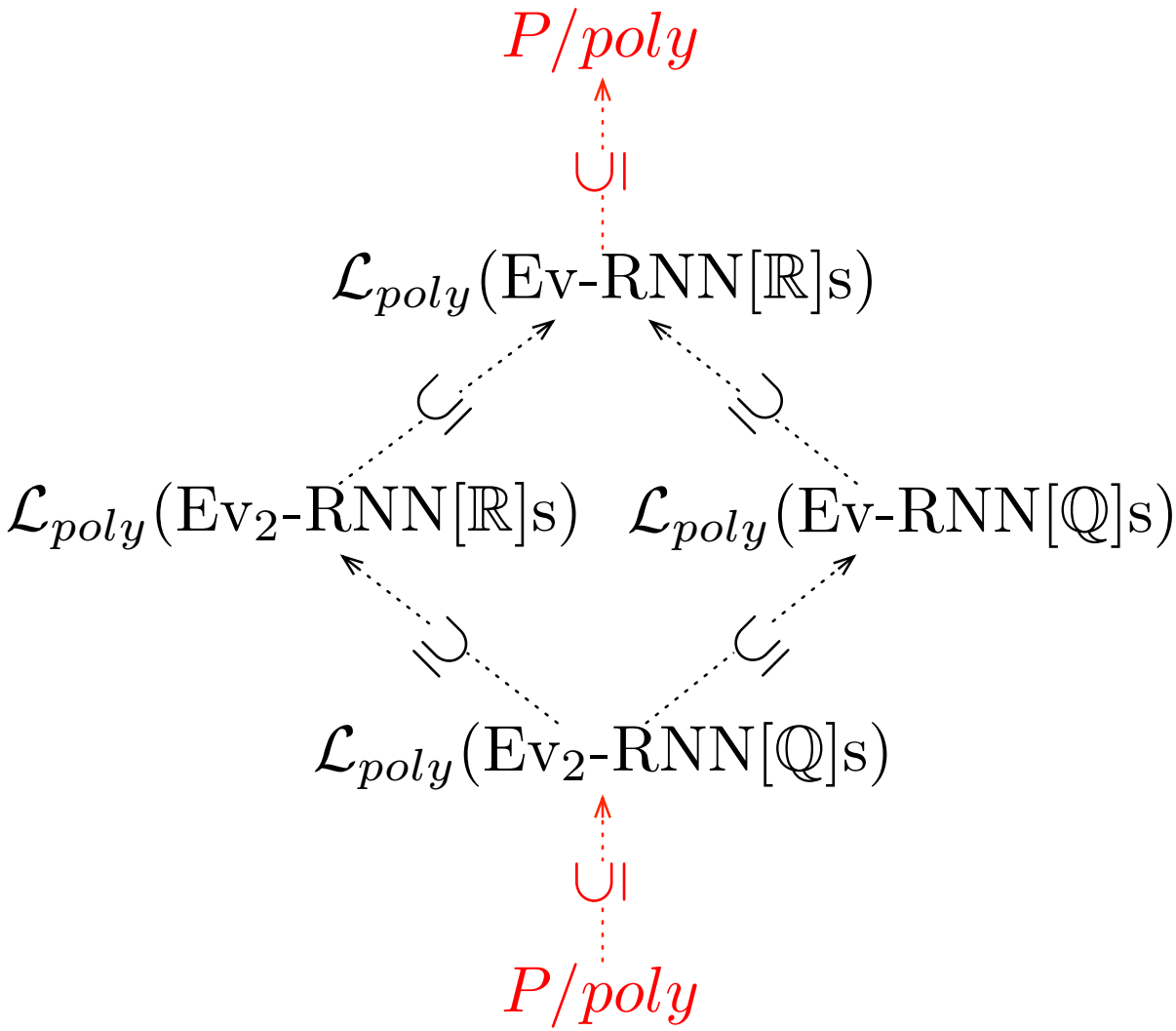
Theorem (Cabessa & Siegelmann 11, 14)

The four models of Evolving RNNs are super-Turing equivalent:

- *They can decide any possible language in exponential time.*
- *They compute exactly the class $P/poly$ in polynomial time.*

	Bi-valued	General
$\mathbb{Q}$	super-Turing	super-Turing
$\mathbb{R}$	super-Turing	super-Turing

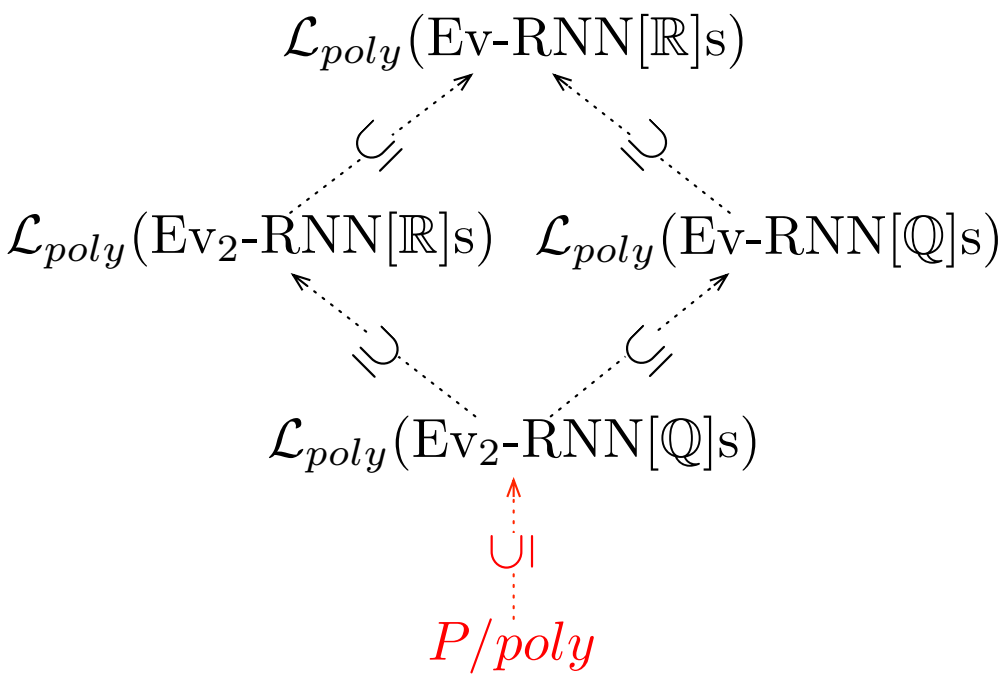
Proof – General Idea



Proof – The Lower Bound

Lemma

Let $L \subseteq \{0,1\}^*$. If $L \in P/poly$, then L is decidable in polynomial time by some Ev$_2$-RNN[$\mathbb{Q}$].



Proof – The Lower Bound

Let $L \in P/poly$. Then there exists some TM/poly(A) $\mathcal{M}$ with advice function α that decides L in polynomial time.

- We build an Ev_2-RNN[$\mathbb{Q}$] $\mathcal{N}$ as described below which decides L in polynomial time.

Proof – The Lower Bound

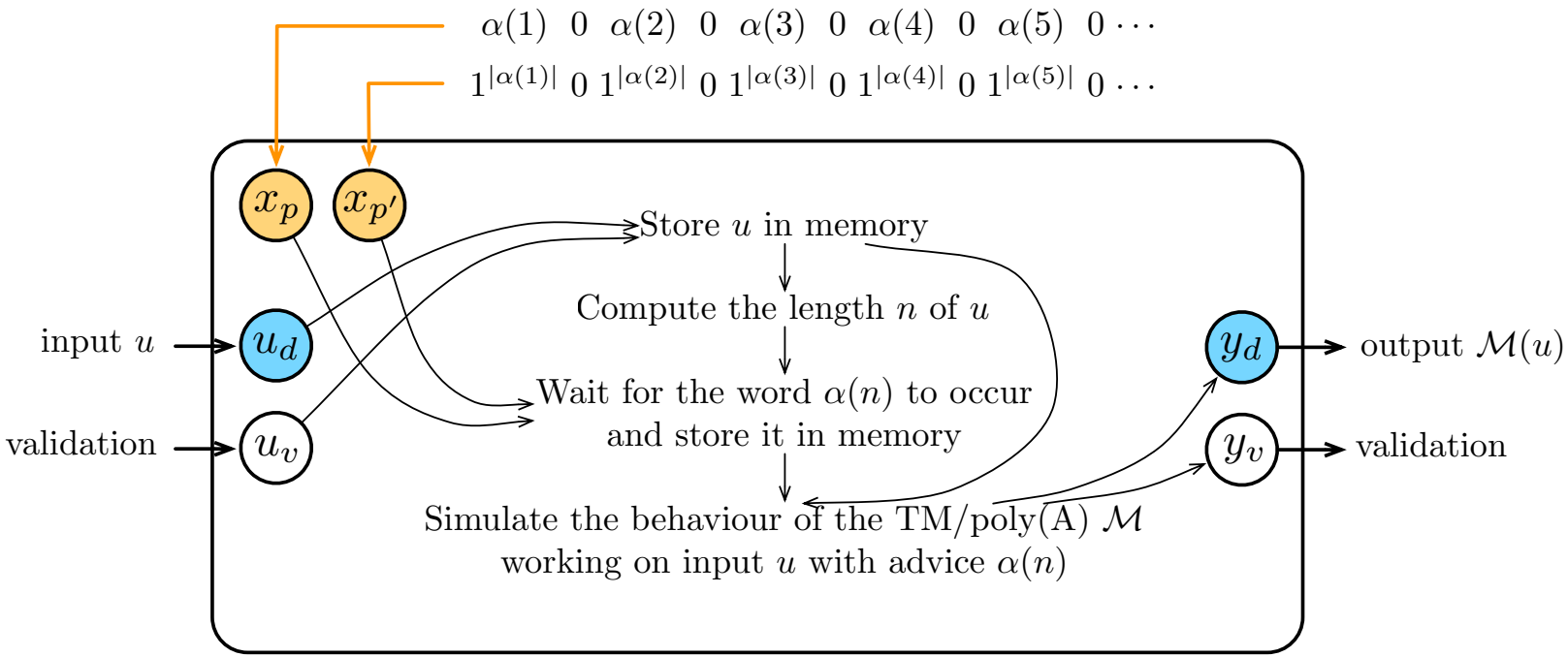
Let $L \in P/poly$. Then there exists some TM/poly(A) $\mathcal{M}$ with advice function α that decides L in polynomial time.

- We build an Ev_2-RNN[$\mathbb{Q}$] $\mathcal{N}$ as described below which decides L in polynomial time.

Proof – The Lower Bound

Let $L \in P/poly$. Then there exists some TM/poly(A) $\mathcal{M}$ with advice function α that decides L in polynomial time.

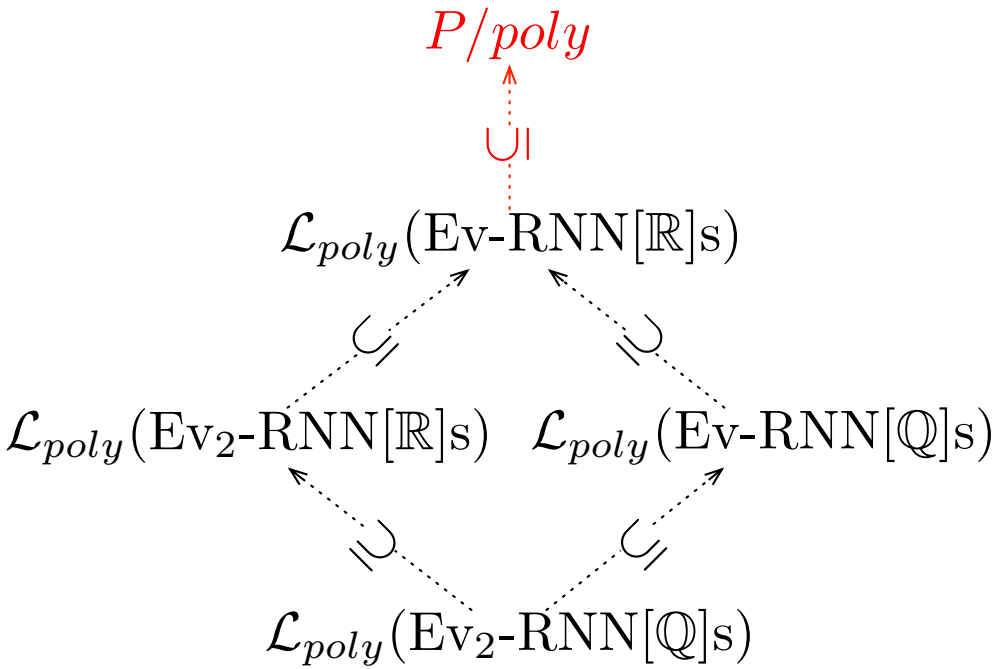
- We build an Ev_2-RNN[$\mathbb{Q}$] $\mathcal{N}$ as described below which decides L in polynomial time.



Proof – The Upper Bound

Lemma

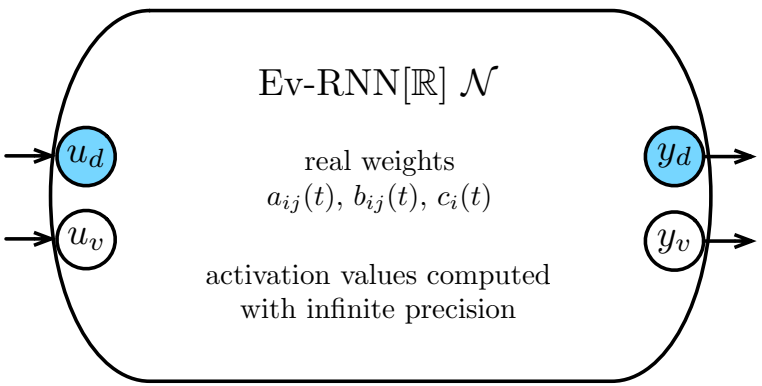
Let $L \subseteq \{0,1\}^*$. If L is decidable in polynomial time by some Ev-RNN[$\mathbb{R}$], then $L \in P/poly$.



Proof – The Upper Bound

Let L be decidable in polynomial time p by some Ev-RNN[$\mathbb{R}$] $\mathcal{N}$.

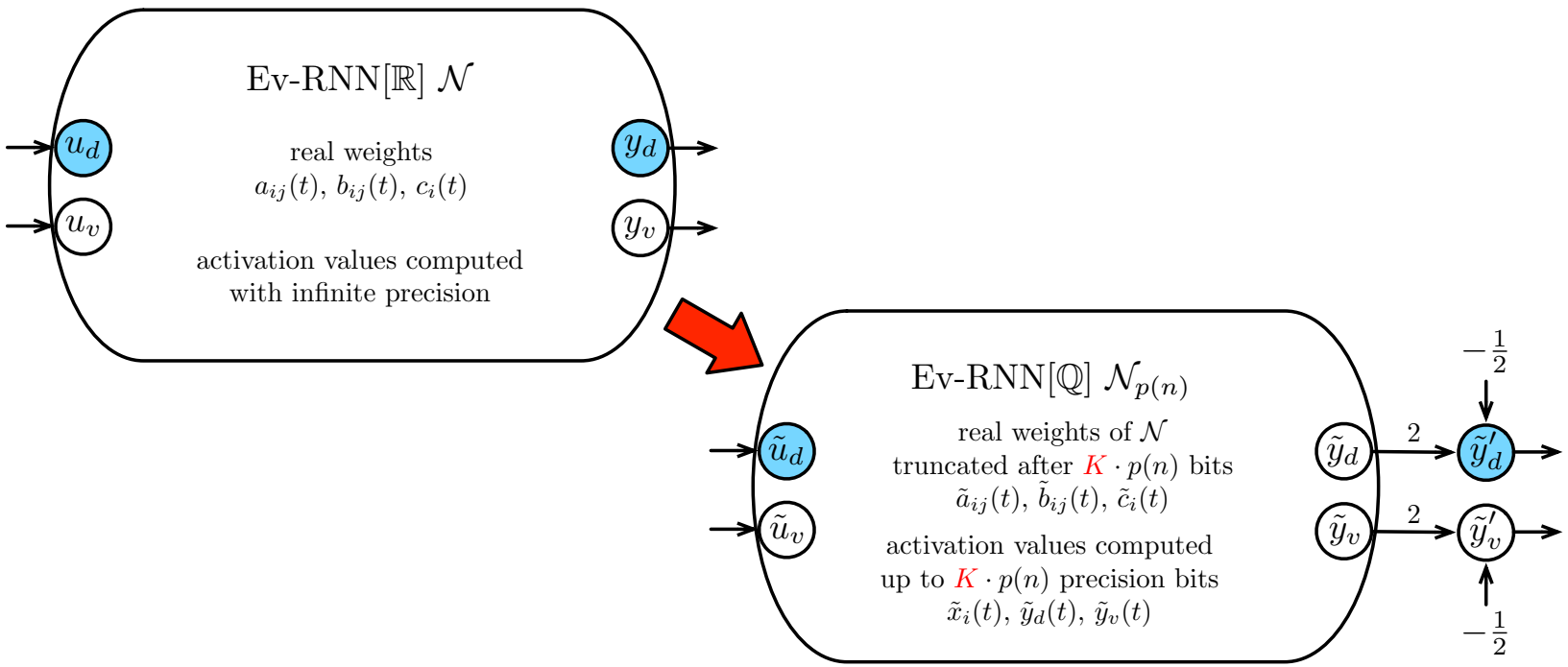
- Then, by some technical lemma, there exists a so-called p-*truncated family* of Ev-RNN[$\mathbb{Q}$]s $\{\mathcal{N}_{p(n)} : n \geq 0\}$ such that each network $\mathcal{N}_{p(n)}$ computes exactly like $\mathcal{N}$ up to time $p(n)$.



Proof – The Upper Bound

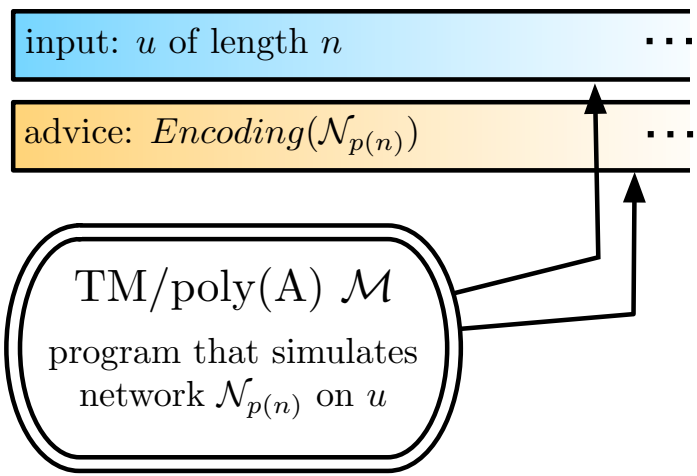
Let L be decidable in polynomial time p by some Ev-RNN[$\mathbb{R}$] $\mathcal{N}$.

- Then, by some technical lemma, there exists a so-called p-*truncated family* of Ev-RNN[$\mathbb{Q}$]s $\{\mathcal{N}_{p(n)} : n \geq 0\}$ such that each network $\mathcal{N}_{p(n)}$ computes exactly like $\mathcal{N}$ up to time $p(n)$.



Proof – The Upper Bound

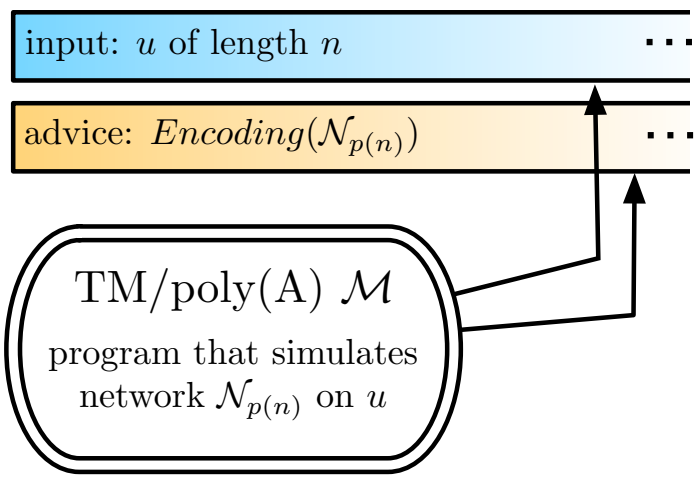
- We build a TM/poly(A) $\mathcal{M}$ with the advice $\alpha(n) = \ulcorner \mathcal{N}_{p(n)} \urcorner$ for each $n \geq 0$, and which, on every input u of length n:
  1. calls the polynomial-bounded advice value $\ulcorner \mathcal{N}_{p(n)} \urcorner$
  2. simulates the behaviour of $\mathcal{N}_{p(n)}$ on u in polynomial time



The machine $\mathcal{M}$ answers precise like the Ev-RNN[$\mathbb{R}$] $\mathcal{N}$, hence decides L in polynomial time. Therefore $L \in P/poly$.

Proof – The Upper Bound

- We build a TM/poly(A) $\mathcal{M}$ with the advice $\alpha(n) = \ulcorner \mathcal{N}_{p(n)} \urcorner$ for each $n \geq 0$, and which, on every input u of length n:
  1. calls the polynomial-bounded advice value $\ulcorner \mathcal{N}_{p(n)} \urcorner$
  2. simulates the behaviour of $\mathcal{N}_{p(n)}$ on u in polynomial time



The machine $\mathcal{M}$ answers precise like the Ev-RNN[$\mathbb{R}$] $\mathcal{N}$, hence decides L in polynomial time. Therefore $L \in P/poly$.

Results – Summary

Computational power of recurrent neural networks in polynomial time of computation.

	Static	Evolving (bi-valued)	Evolving (general)
$\mathbb{Q}$	P	P/poly	P/poly
$\mathbb{R}$	P/poly	P/poly	P/poly

Introduction

- We study the computational capabilities of recurrent neural networks provided with some discrete source of stochasticity.
- The discrete stochasticity might add to the computational power of the networks, but only up to the level P/poly.

Introduction

- We study the computational capabilities of recurrent neural networks provided with some discrete source of stochasticity.
- The discrete stochasticity might add to the computational power of the networks, but only up to the level P/poly.

Probabilistic Turing Machines

- A *probabilistic Turing machine* (PTM) is a non-deterministic Turing machine such that:
  - Every step of the computation can be made in exactly two possible ways chosen randomly with probability $1/2$
  - Every computational path ends up in a final state
  - All computational paths are of the same length
- A word w is *accepted* by a PTM if strictly more than half of the computational paths on w end up in an ACCEPT state (i.e. if the probability of acceptance is above $1/2$).
- The *error probability* of a PTM $\mathcal{M}$ is the function

$$e_{\mathcal{M}} : x \mapsto \frac{\#\text{ wrong answers for } x}{\#\text{ total computations}}.$$

Probabilistic Turing Machines

- A *probabilistic Turing machine* (PTM) is a non-deterministic Turing machine such that:
  - Every step of the computation can be made in exactly two possible ways chosen randomly with probability $1/2$
  - Every computational path ends up in a final state
  - All computational paths are of the same length
- A word w is *accepted* by a PTM if strictly more than half of the computational paths on w end up in an ACCEPT state (i.e. if the probability of acceptance is above $1/2$).
- The *error probability* of a PTM $\mathcal{M}$ is the function

$$e_{\mathcal{M}} : x \mapsto \frac{\#\text{ wrong answers for } x}{\#\text{ total computations}}.$$

Probabilistic Turing Machines

- A *probabilistic Turing machine* (PTM) is a non-deterministic Turing machine such that:
  - Every step of the computation can be made in exactly two possible ways chosen randomly with probability $1/2$
  - Every computational path ends up in a final state
  - All computational paths are of the same length
- A word w is *accepted* by a PTM if strictly more than half of the computational paths on w end up in an ACCEPT state (i.e. if the probability of acceptance is above $1/2$).
- The *error probability* of a PTM $\mathcal{M}$ is the function

$$e_{\mathcal{M}} : x \mapsto \frac{\#\text{ wrong answers for } x}{\#\text{ total computations}}.$$

Probabilistic Turing Machines

- A *probabilistic Turing machine* (PTM) is a non-deterministic Turing machine such that:
  - Every step of the computation can be made in exactly two possible ways chosen randomly with probability $1/2$
  - Every computational path ends up in a final state
  - All computational paths are of the same length
- A word w is *accepted* by a PTM if strictly more than half of the computational paths on w end up in an ACCEPT state (i.e. if the probability of acceptance is above $1/2$).
- The *error probability* of a PTM $\mathcal{M}$ is the function

$$e_{\mathcal{M}} : x \mapsto \frac{\#\text{ wrong answers for } x}{\#\text{ total computations}}.$$

Probabilistic Turing Machines

- A *probabilistic Turing machine* (PTM) is a non-deterministic Turing machine such that:
  - Every step of the computation can be made in exactly two possible ways chosen randomly with probability $1/2$
  - Every computational path ends up in a final state
  - All computational paths are of the same length
- A word w is *accepted* by a PTM if strictly more than half of the computational paths on w end up in an ACCEPT state (i.e. if the probability of acceptance is above $1/2$).
- The *error probability* of a PTM $\mathcal{M}$ is the function

$$e_{\mathcal{M}} : x \mapsto \frac{\#\text{ wrong answers for } x}{\#\text{ total computations}}.$$

Probabilistic Turing Machines

- A *probabilistic Turing machine* (PTM) is a non-deterministic Turing machine such that:
  - Every step of the computation can be made in exactly two possible ways chosen randomly with probability $1/2$
  - Every computational path ends up in a final state
  - All computational paths are of the same length
- A word w is *accepted* by a PTM if strictly more than half of the computational paths on w end up in an ACCEPT state (i.e. if the probability of acceptance is above $1/2$).
- The *error probability* of a PTM $\mathcal{M}$ is the function

$$e_{\mathcal{M}} : x \mapsto \frac{\#\text{ wrong answers for } x}{\#\text{ total computations}}.$$

Probabilistic Turing Machines

- BPP is the class of languages recognized by polynomial time PTMs whose error probabilities are bounded above by some constant $\varepsilon < 1/2$.
- BPP/log^* is the class of languages recognized by such PTMs which further use logarithmic monotone advice functions.
- **Remark:** $P \subsetneq BPP/log^* \subsetneq BPP/log$, whereas $P/poly* = P/poly$.

Probabilistic Turing Machines

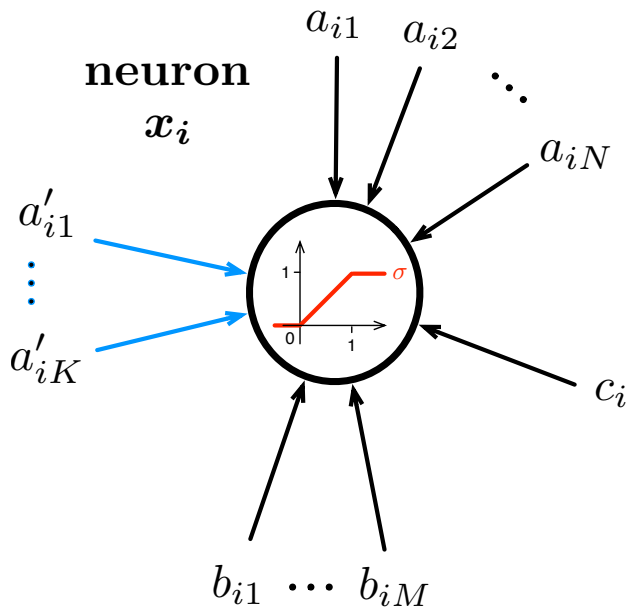
- BPP is the class of languages recognized by polynomial time PTMs whose error probabilities are bounded above by some constant $\varepsilon < 1/2$.
- BPP/log^* is the class of languages recognized by such PTMs which further use logarithmic monotone advice functions.
- **Remark:** $P \subsetneq BPP/log^* \subsetneq BPP/log$, whereas $P/poly* = P/poly$.

Probabilistic Turing Machines

- BPP is the class of languages recognized by polynomial time PTMs whose error probabilities are bounded above by some constant $\varepsilon < 1/2$.
- BPP/log^* is the class of languages recognized by such PTMs which further use logarithmic monotone advice functions.
- **Remark:** $P \subsetneq BPP/log^* \subsetneq BPP/log$, whereas $P/poly* = P/poly$.

Dynamics of Stochastic RNNs

An *stochastic RNN* (S-RNN) consists of a RNN including a set of binary input neurons x'_j's taking activation values 1 with probabilities p_j's.



$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{K} a'_{ij} \cdot x'_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

Dynamics of Stochastic RNNs

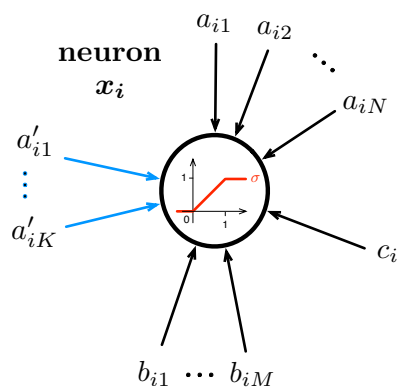
Let $L \subsetneq \{0,1\}^*$ and $0 < \varepsilon < 1/2$. The language L is *ε-recognized in time T* by some S-RNN $\mathcal{N}$ if for every input w of length n

- all computation paths of $\mathcal{N}$ classify w in time $T(n)$
- $e_{\mathcal{N}}(w) < \varepsilon < 1/2$

Models of Stochastic RNNs

We consider the following models of stochastic (or probabilistic) recurrent neural networks:

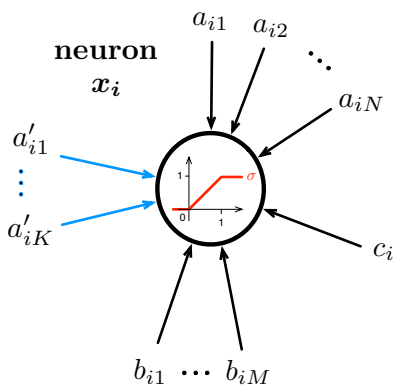
1. rational-weighted $\mathbb{Q}$-stochastic RNNs: $\mathrm{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$]s
2. rational-weighted $\mathbb{R}$-stochastic RNNs: $\mathrm{S}_{\mathbb{R}}$-RNN[$\mathbb{Q}$]s
3. real-weighted stochastic RNNs: S-RNN[$\mathbb{R}$]s
4. Rational-weighted stochastic evolving RNNs: S-Ev-RNN[$\mathbb{Q}$]s
5. Real-weighted stochastic evolving RNNs: S-Ev-RNN[$\mathbb{R}$]s



Models of Stochastic RNNs

We consider the following models of stochastic (or probabilistic) recurrent neural networks:

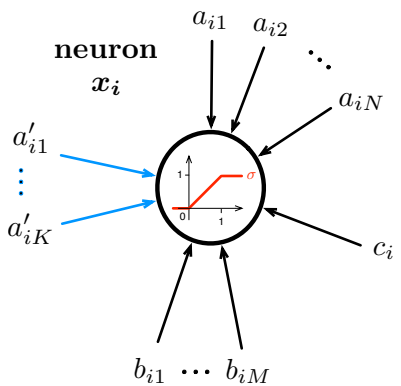
1. rational-weighted $\mathbb{Q}$-stochastic RNNs: $\mathrm{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$]s
2. rational-weighted $\mathbb{R}$-stochastic RNNs: $\mathrm{S}_{\mathbb{R}}$-RNN[$\mathbb{Q}$]s
3. real-weighted stochastic RNNs: S-RNN[$\mathbb{R}$]s
4. Rational-weighted stochastic evolving RNNs: S-Ev-RNN[$\mathbb{Q}$]s
5. Real-weighted stochastic evolving RNNs: S-Ev-RNN[$\mathbb{R}$]s



Models of Stochastic RNNs

We consider the following models of stochastic (or probabilistic) recurrent neural networks:

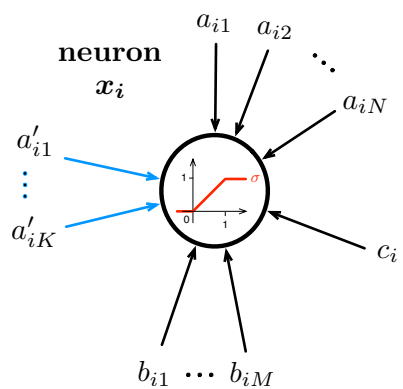
1. rational-weighted $\mathbb{Q}$-stochastic RNNs: $\mathrm{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$]s
2. rational-weighted $\mathbb{R}$-stochastic RNNs: $\mathrm{S}_{\mathbb{R}}$-RNN[$\mathbb{Q}$]s
3. real-weighted stochastic RNNs: S-RNN[$\mathbb{R}$]s
4. Rational-weighted stochastic evolving RNNs: S-Ev-RNN[$\mathbb{Q}$]s
5. Real-weighted stochastic evolving RNNs: S-Ev-RNN[$\mathbb{R}$]s



Models of Stochastic RNNs

We consider the following models of stochastic (or probabilistic) recurrent neural networks:

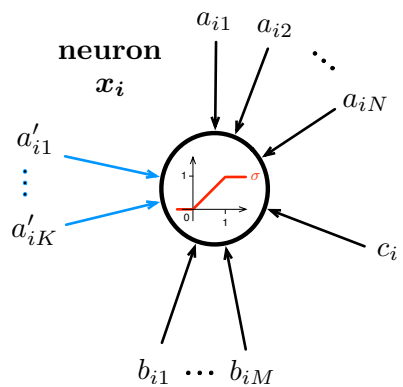
1. rational-weighted $\mathbb{Q}$-stochastic RNNs: $\mathrm{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$]s
2. rational-weighted $\mathbb{R}$-stochastic RNNs: $\mathrm{S}_{\mathbb{R}}$-RNN[$\mathbb{Q}$]s
3. real-weighted stochastic RNNs: S-RNN[$\mathbb{R}$]s
4. Rational-weighted stochastic evolving RNNs: S-Ev-RNN[$\mathbb{Q}$]s
5. Real-weighted stochastic evolving RNNs: S-Ev-RNN[$\mathbb{R}$]s



Models of Stochastic RNNs

We consider the following models of stochastic (or probabilistic) recurrent neural networks:

1. rational-weighted $\mathbb{Q}$-stochastic RNNs: $\mathrm{S}_{\mathbb{Q}}$-RNN$[\mathbb{Q}]$s
2. rational-weighted $\mathbb{R}$-stochastic RNNs: $\mathrm{S}_{\mathbb{R}}$-RNN$[\mathbb{Q}]$s
3. real-weighted stochastic RNNs: S-RNN$[\mathbb{R}]$s
4. Rational-weighted stochastic evolving RNNs: S-Ev-RNN$[\mathbb{Q}]$s
5. Real-weighted stochastic evolving RNNs: S-Ev-RNN$[\mathbb{R}]$s



Result

We first consider rational-weighted stochastic networks whose probabilities are also rational.

Theorem (Siegelmann 99)

Let $L \subseteq \{0,1\}^*$ be some language. Then L is ε-recognizable in polynomial time by some $S_{\mathbb{Q}}$-RNN[$\mathbb{Q}$] iff $L \in BPP$.

Proof – First Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time by some $\mathrm{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$] $\mathcal{N}$.

- The networks dynamics $\mathcal{F} : \mathbb{Q}^N \to \mathbb{Q}^N$ given by

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{K} a'_{ij} \cdot x'_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

  with rational underlying probabilities can easily be simulated by some PTM with rational probabilities, and hence also by some fair PTM, up to some polynomial time increase.

Therefore, $L \in BPP$.

Proof – First Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time by some $\mathsf{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$] $\mathcal{N}$.

- The networks dynamics $\mathcal{F} : \mathbb{Q}^N \to \mathbb{Q}^N$ given by

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{K} a'_{ij} \cdot x'_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

with rational underlying probabilities can easily be simulated by some PTM with rational probabilities, and hence also by some fair PTM, up to some polynomial time increase.

Therefore, $L \in BPP$.

Proof – First Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time by some $\mathsf{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$] $\mathcal{N}$.

- The networks dynamics $\mathcal{F} : \mathbb{Q}^N \to \mathbb{Q}^N$ given by

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{K} a'_{ij} \cdot x'_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

  with rational underlying probabilities can easily be simulated by some PTM with rational probabilities, and hence also by some fair PTM, up to some polynomial time increase.

Therefore, $L \in BPP$.

Proof – Second Implication

Let $L \in BPP$.

- Then there exists a PTM $\mathcal{M}$ that ε-recognizes L.
- By the real time equivalence between TMs and RNN[$\mathbb{Q}$]s (Theorem 2), one can build a $S_{\mathbb{Q}}$-RNN[$\mathbb{Q}$] $\mathcal{N}$ that simulates $\mathcal{M}$.

Therefore, L is ε-recognizable in polynomial time by some $S_{\mathbb{Q}}$-RNN[$\mathbb{Q}$] $\mathcal{N}$.

Proof – Second Implication

Let $L \in BPP$.

- Then there exists a PTM $\mathcal{M}$ that ε-recognizes L.
- By the real time equivalence between TMs and RNN[$\mathbb{Q}$]s (Theorem 2), one can build a $\mathrm{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$] $\mathcal{N}$ that simulates $\mathcal{M}$.

Therefore, L is ε-recognizable in polynomial time by some $\mathrm{S}_{\mathbb{Q}}$-RNN[$\mathbb{Q}$] $\mathcal{N}$.

Result

We now consider rational-weighted stochastic networks whose probabilities are real.

Theorem (Siegelmann 99)

Let $L \subseteq \{0,1\}^*$ be some language. Then L is ε-recognizable in polynomial time by some $S_{\mathbb{R}}$-RNN[$\mathbb{Q}$] iff $L \in BPP/log^*$.

Proof – First Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time by some $\mathrm{S}_{\mathbb{R}}$-RNN[$\mathbb{Q}$] $\mathcal{N}$.

- The networks dynamics $\mathcal{F} : \mathbb{Q}^N \to \mathbb{Q}^N$ given by

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{K} a'_{ij} \cdot x'_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

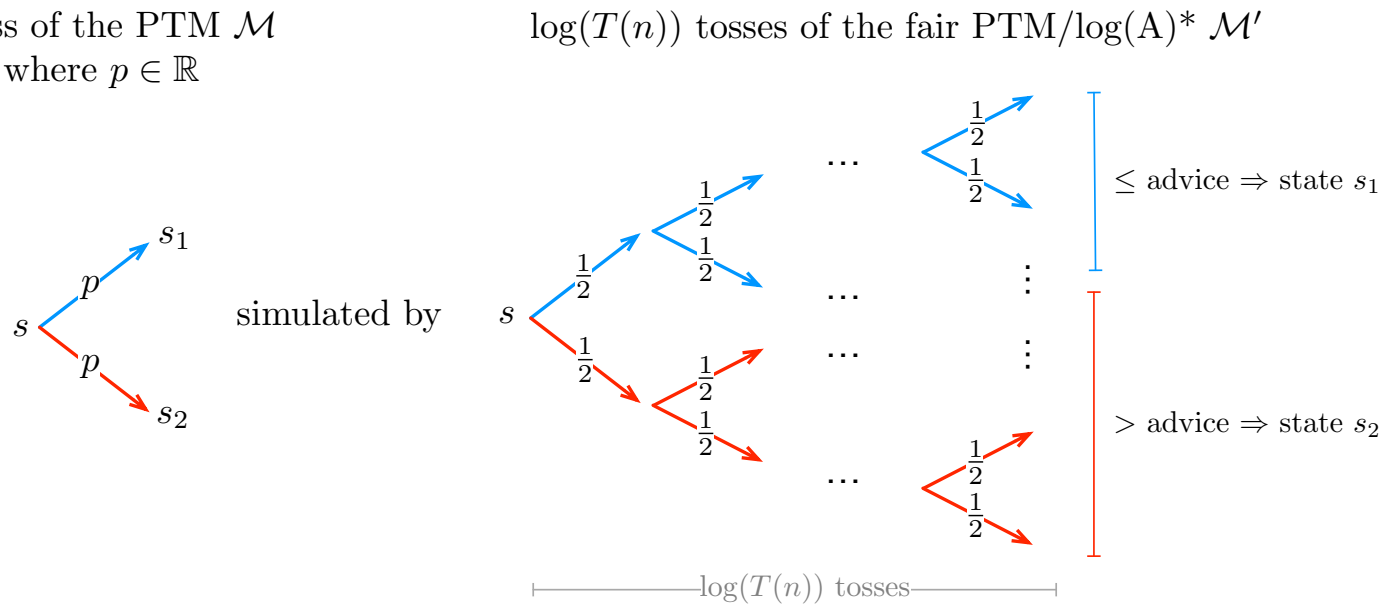
with *real* underlying probabilities can easily be simulated by some PTM with *real* probabilities $\mathcal{M}$.

Proof – First Implication

- Now, if $\mathcal{M}$ is a PTM with *real* probability p computing in time T, we consider the *fair* PTM/log(A)* $\mathcal{M}'$ with advice function

$$n \longmapsto [p]_{\log(T(n))}$$

and which simulates every binary choice of $\mathcal{M}$ as follows:



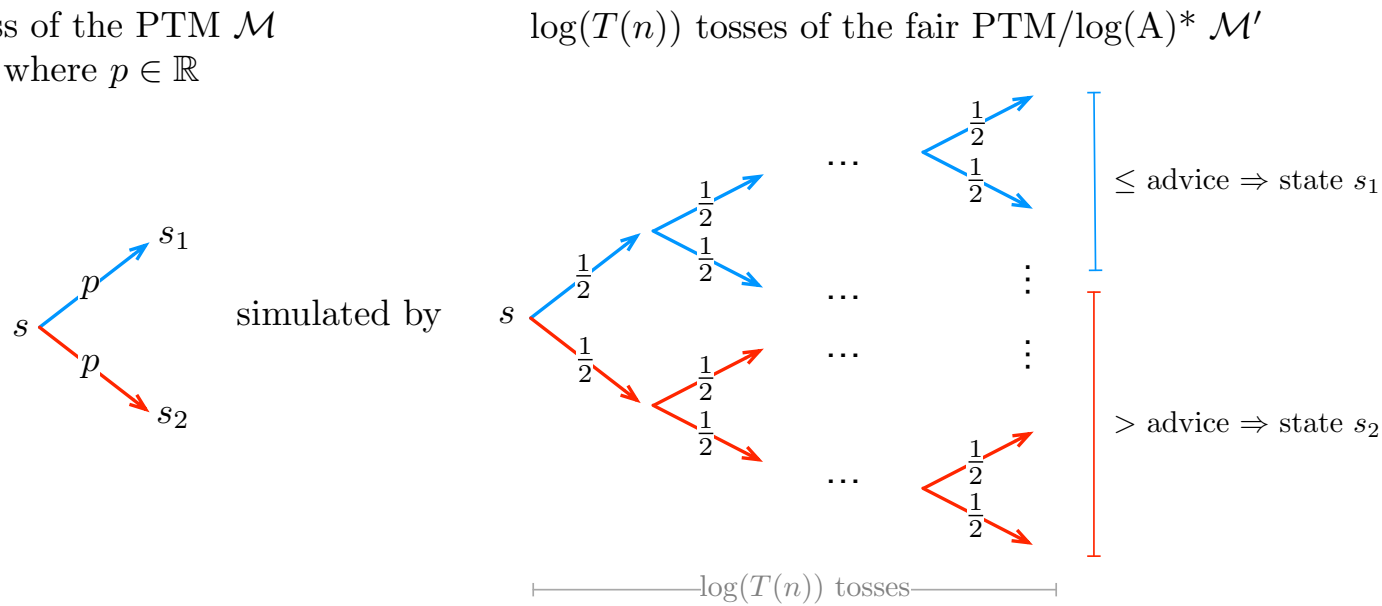
$\mathcal{M}'$ ε-recognizes L in time $O(T \log(T))$, and thus $L \in BPP/log^*$.

Proof – First Implication

- Now, if $\mathcal{M}$ is a PTM with *real* probability p computing in time T, we consider the *fair* PTM/log(A)* $\mathcal{M}'$ with advice function

$$n \longmapsto [p]_{\log(T(n))}$$

and which simulates every binary choice of $\mathcal{M}$ as follows:



$\mathcal{M}'$ ε-recognizes L in time $O(T\log(T))$, and thus $L \in BPP/log^*$.

Proof – Second Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time T by some fair PTM/log(A)* $\mathcal{M}$.

- We consider the PTM $\mathcal{M}'$ with *real* probability given by the binary expansion $p = 0.01 advice \cdots$ and which works on every input of length n as follows:
  - *Advice reconstruction:* $\mathcal{M}'$ tosses its fair coin $cT^2(n)$ times, writes the resulting binary sequence on a tape, and uses it as $\mathcal{M}$ would have done with its advice string
  - *Fair toss simulation:* $\mathcal{M}'$ simulates every fair toss of $\mathcal{M}$ by at most $2T(n)$ p-tosses following a specific algorithm
- We can show that the PTM $\mathcal{M}'$ ε-recognizes L in time $O(T^2)$.

Then, there exists a $\mathrm{S}_{\mathbb{R}}$-RNN$[\mathbb{Q}]$ $\mathcal{N}$ – using real probability p also – which can simulate $\mathcal{M}'$, and thus recognizes L in poly time.

Proof – Second Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time T by some fair PTM/log(A)* $\mathcal{M}$.

- We consider the PTM $\mathcal{M}'$ with *real* probability given by the binary expansion $p = 0.01advice\cdots$ and which works on every input of length n as follows:
  - *Advice reconstruction:* $\mathcal{M}'$ tosses its fair coin $cT^2(n)$ times, writes the resulting binary sequence on a tape, and uses it as $\mathcal{M}$ would have done with its advice string
  - *Fair toss simulation:* $\mathcal{M}'$ simulates every fair toss of $\mathcal{M}$ by at most $2T(n)$ p-tosses following a specific algorithm
- We can show that the PTM $\mathcal{M}'$ ε-recognizes L in time $O(T^2)$.

Then, there exists a $\mathrm{S}_{\mathbb{R}}$-RNN$[\mathbb{Q}]$ $\mathcal{N}$ – using real probability p also – which can simulate $\mathcal{M}'$, and thus recognizes L in poly time.

Proof – Second Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time T by some fair PTM/log(A)* $\mathcal{M}$.

- We consider the PTM $\mathcal{M}'$ with *real* probability given by the binary expansion $p = 0.01 advice \cdots$ and which works on every input of length n as follows:
  - *Advice reconstruction:* $\mathcal{M}'$ tosses its fair coin $cT^2(n)$ times, writes the resulting binary sequence on a tape, and uses it as $\mathcal{M}$ would have done with its advice string
  - *Fair toss simulation:* $\mathcal{M}'$ simulates every fair toss of $\mathcal{M}$ by at most $2T(n)$ p-tosses following a specific algorithm
- We can show that the PTM $\mathcal{M}'$ ε-recognizes L in time $O(T^2)$.

Then, there exists a $\mathrm{S}_{\mathbb{R}}$-RNN$[\mathbb{Q}]$ $\mathcal{N}$ – using real probability p also – which can simulate $\mathcal{M}'$, and thus recognizes L in poly time.

Proof – Second Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time T by some fair PTM/log(A)* $\mathcal{M}$.

- We consider the PTM $\mathcal{M}'$ with *real* probability given by the binary expansion $p = 0.01 advice \cdots$ and which works on every input of length n as follows:
  - *Advice reconstruction:* $\mathcal{M}'$ tosses its fair coin $cT^2(n)$ times, writes the resulting binary sequence on a tape, and uses it as $\mathcal{M}$ would have done with its advice string
  - *Fair toss simulation:* $\mathcal{M}'$ simulates every fair toss of $\mathcal{M}$ by at most $2T(n)$ p-tosses following a specific algorithm
- We can show that the PTM $\mathcal{M}'$ ε-recognizes L in time $O(T^2)$.

Then, there exists a $\mathrm{S}_{\mathbb{R}}$-RNN$[\mathbb{Q}]$ $\mathcal{N}$ – using real probability p also – which can simulate $\mathcal{M}'$, and thus recognizes L in poly time.

Proof – Second Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time T by some fair PTM/log(A)* $\mathcal{M}$.

- We consider the PTM $\mathcal{M}'$ with *real* probability given by the binary expansion $p = 0.01 advice \cdots$ and which works on every input of length n as follows:
  - *Advice reconstruction:* $\mathcal{M}'$ tosses its fair coin $cT^2(n)$ times, writes the resulting binary sequence on a tape, and uses it as $\mathcal{M}$ would have done with its advice string
  - *Fair toss simulation:* $\mathcal{M}'$ simulates every fair toss of $\mathcal{M}$ by at most $2T(n)$ p-tosses following a specific algorithm
- We can show that the PTM $\mathcal{M}'$ ε-recognizes L in time $O(T^2)$.

Then, there exists a $\mathrm{S}_{\mathbb{R}}$-RNN$[\mathbb{Q}]$ $\mathcal{N}$ – using real probability p also – which can simulate $\mathcal{M}'$, and thus recognizes L in poly time.

Proof – Second Implication

Let $L \subseteq \{0,1\}^*$ be ε-recognizable in polynomial time T by some fair PTM/log(A)* $\mathcal{M}$.

- We consider the PTM $\mathcal{M}'$ with *real* probability given by the binary expansion $p = 0.01 advice \cdots$ and which works on every input of length n as follows:
  - *Advice reconstruction:* $\mathcal{M}'$ tosses its fair coin $cT^2(n)$ times, writes the resulting binary sequence on a tape, and uses it as $\mathcal{M}$ would have done with its advice string
  - *Fair toss simulation:* $\mathcal{M}'$ simulates every fair toss of $\mathcal{M}$ by at most $2T(n)$ p-tosses following a specific algorithm
- We can show that the PTM $\mathcal{M}'$ ε-recognizes L in time $O(T^2)$.

Then, there exists a $\mathrm{S}_{\mathbb{R}}$-RNN$[\mathbb{Q}]$ $\mathcal{N}$ – using real probability p also – which can simulate $\mathcal{M}'$, and thus recognizes L in poly time.

Results

We finally consider the cases of real-weighted as well as all kinds of evolving networks. In this case, the addition of rational or real stochasticity does not further increase the computational power of the networks.

Theorem (Siegelmann 99, Cabessa & Siegelmann 14)

S-RNN[$\mathbb{R}$]s, S-Ev-RNN[$\mathbb{Q}$]s, and S-Ev-RNN[$\mathbb{R}$]s are all super-Turing equivalent:

- *They can decide any possible language in exponential time.*
- *They compute exactly the class $P/poly$ in polynomial time.*

Proof – First Implication

We prove the case of S-RNN[$\mathbb{R}$]s. The other cases are similar...

Let $L \in P/poly$

- By Theorem 3, L is recognizable by some RNN[$\mathbb{R}$].
- Note that the addition of stochasticity does obviously not decrease the computational power of the networks.

Hence, L is also ε-recognizable by some S-RNN[$\mathbb{R}$].

Proof – First Implication

We prove the case of S-RNN[$\mathbb{R}$]s. The other cases are similar...

Let $L \in P/poly$

- By Theorem 3, L is recognizable by some RNN[$\mathbb{R}$].
- Note that the addition of stochasticity does obviously not decrease the computational power of the networks.

Hence, L is also ε-recognizable by some S-RNN[$\mathbb{R}$].

Proof – Second Implication

Let L be ε-recognizable by some S-RNN[$\mathbb{R}$] $\mathcal{N}$ of size N in polynomial time T.

- Then there exists a family of feedforward RNN[$\mathbb{R}$] $\{\mathcal{F}_n\}_{n\geq 0}$ of depths $T(n)+1$ and sizes $cnNT(n)+1$ such that each $\mathcal{F}_n$ recognizes $L \cap \{0,1\}^n$ (for some constant c).
- Each network $\mathcal{F}_n$ consists of cn copies of a $T(n)$-layer unfolding of $\mathcal{N}$ and related by a majority cell. We can prove that there exists a constant c and a suitable fixed choice of the random input cells (orange) such that $\mathcal{F}_n$ recognizes $L \cap \{0,1\}^n$.

Proof – Second Implication

Let L be ε-recognizable by some S-RNN[$\mathbb{R}$] $\mathcal{N}$ of size N in polynomial time T.

- Then there exists a family of feedforward RNN[$\mathbb{R}$] $\{\mathcal{F}_n\}_{n\geq 0}$ of depths $T(n)+1$ and sizes $cnNT(n)+1$ such that each $\mathcal{F}_n$ recognizes $L \cap \{0,1\}^n$ (for some constant c).
- Each network $\mathcal{F}_n$ consists of cn copies of a $T(n)$-layer unfolding of $\mathcal{N}$ and related by a majority cell. We can prove that there exists a constant c and a suitable fixed choice of the random input cells (orange) such that $\mathcal{F}_n$ recognizes $L \cap \{0,1\}^n$.

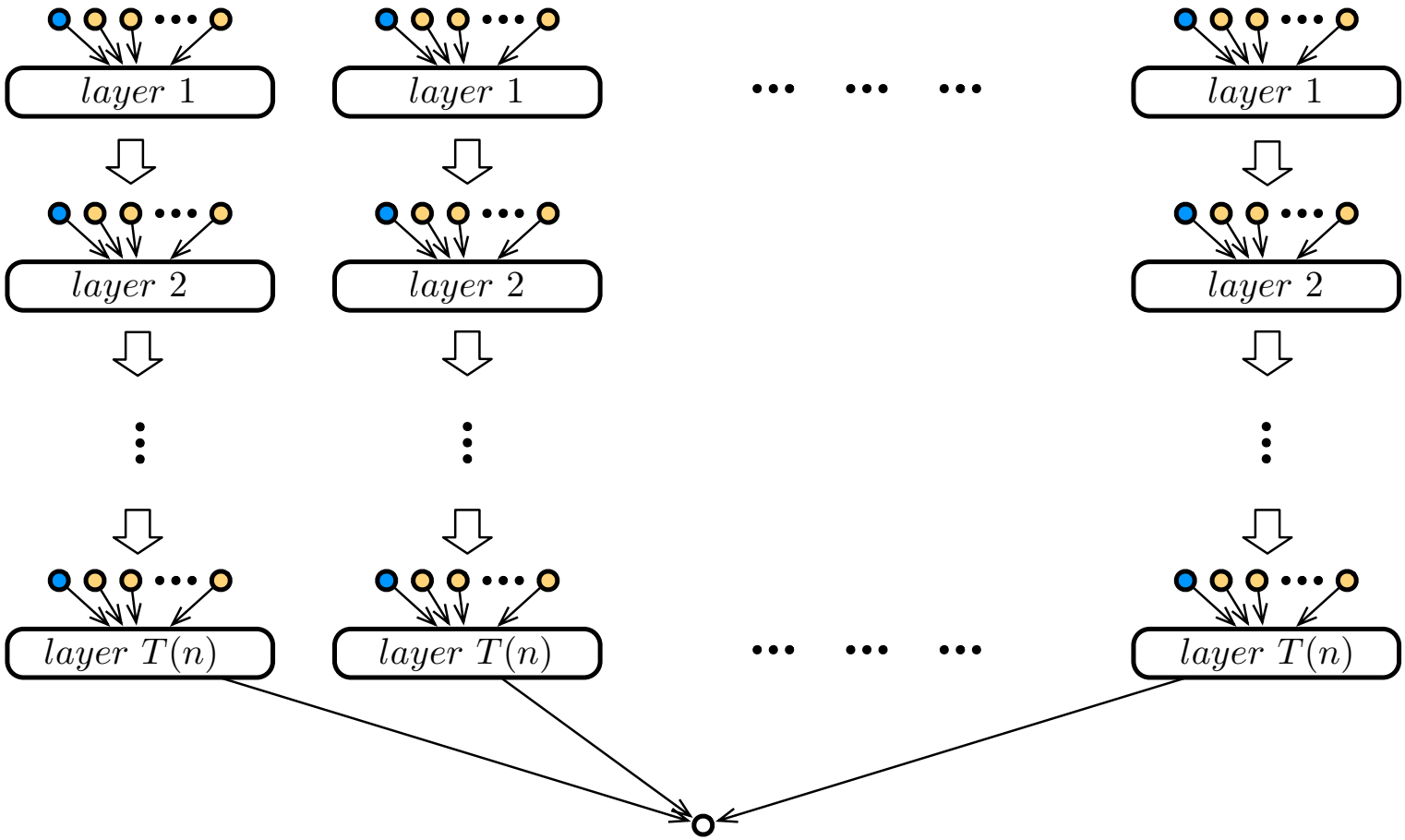
Proof – Second Implication

Let L be ε-recognizable by some S-RNN[$\mathbb{R}$] $\mathcal{N}$ of size N in polynomial time T.

- Then there exists a family of feedforward RNN[$\mathbb{R}$] $\{\mathcal{F}_n\}_{n\geq 0}$ of depths $T(n)+1$ and sizes $cnNT(n)+1$ such that each $\mathcal{F}_n$ recognizes $L \cap \{0,1\}^n$ (for some constant c).
- Each network $\mathcal{F}_n$ consists of cn copies of a $T(n)$-layer unfolding of $\mathcal{N}$ and related by a majority cell. We can prove that there exists a constant c and a suitable fixed choice of the random input cells (orange) such that $\mathcal{F}_n$ recognizes $L \cap \{0,1\}^n$.

Proof – Second Implication

A network $\mathcal{F}_n$ of the family $\{\mathcal{F}_n\}_{n\geq 0}$



Proof – Second Implication

- Then, there exists some (static) RNN[$\mathbb{R}$] $\mathcal{N}'$ which contains a suitable encoding of the family $\{\mathcal{F}_n\}_{n\geq 0}$ as one of its real synaptic weight.
- On every u input of length n, $\mathcal{N}'$ is able to decode the network $\mathcal{F}_n$ from its real weight, simulate it, and provide the answer whether $u \in L$ or not.
- $\mathcal{N}'$ works with a polynomial slowdown.

By Theorem 3, $L \in P/poly$.

Proof – Second Implication

- Then, there exists some (static) RNN[$\mathbb{R}$] $\mathcal{N}'$ which contains a suitable encoding of the family $\{\mathcal{F}_n\}_{n\geq 0}$ as one of its real synaptic weight.
- On every u input of length n, $\mathcal{N}'$ is able to decode the network $\mathcal{F}_n$ from its real weight, simulate it, and provide the answer whether $u \in L$ or not.
- $\mathcal{N}'$ works with a polynomial slowdown.

By Theorem 3, $L \in P/poly$.

Proof – Second Implication

- Then, there exists some (static) RNN[$\mathbb{R}$] $\mathcal{N}'$ which contains a suitable encoding of the family $\{\mathcal{F}_n\}_{n\geq 0}$ as one of its real synaptic weight.
- On every u input of length n, $\mathcal{N}'$ is able to decode the network $\mathcal{F}_n$ from its real weight, simulate it, and provide the answer whether $u \in L$ or not.
- $\mathcal{N}'$ works with a polynomial slowdown.

By Theorem 3, $L \in P/poly$.

Proof – Second Implication

- Then, there exists some (static) RNN[$\mathbb{R}$] $\mathcal{N}'$ which contains a suitable encoding of the family $\{\mathcal{F}_n\}_{n\geq 0}$ as one of its real synaptic weight.
- On every u input of length n, $\mathcal{N}'$ is able to decode the network $\mathcal{F}_n$ from its real weight, simulate it, and provide the answer whether $u \in L$ or not.
- $\mathcal{N}'$ works with a polynomial slowdown.

By Theorem 3, $L \in P/poly$.

Proof – Second Implication

- Then, there exists some (static) RNN[$\mathbb{R}$] $\mathcal{N}'$ which contains a suitable encoding of the family $\{\mathcal{F}_n\}_{n\geq 0}$ as one of its real synaptic weight.
- On every u input of length n, $\mathcal{N}'$ is able to decode the network $\mathcal{F}_n$ from its real weight, simulate it, and provide the answer whether $u \in L$ or not.
- $\mathcal{N}'$ works with a polynomial slowdown.

By Theorem 3, $L \in P/poly$.

Results – Summary

Computational power of recurrent neural networks in polynomial time of computation.

	Static	Sto[$\mathbb{Q}$] Static	Sto[$\mathbb{R}$] Static	Sto. Evolving
$\mathbb{Q}$	P	BPP	BPP/log*	P/poly
$\mathbb{R}$	P/poly	P/poly	P/poly	P/poly

Results – Summary

Computational power of recurrent neural networks in polynomial time of computation.

	Static	Evolving (bi-valued)	Evolving (general)
$\mathbb{Q}$	P	P/poly	P/poly
$\mathbb{R}$	P/poly	P/poly	P/poly

	Sto[$\mathbb{Q}$] Static	Sto[$\mathbb{R}$] Static	Sto. Evolving
$\mathbb{Q}$	BPP	BPP/log*	P/poly
$\mathbb{R}$	P/poly	P/poly	P/poly

Thesis of Natural Computation

- The fact that the complexity class P/poly seems to cannot be overcome led Siegelmann and Sontag to propose the following Thesis of Natural Computation:

  "Every natural computational phenomenon can be captured within the TM/poly(A) model."

Hypercomputation

The fact P/poly contains P as well as non-recursive languages raises the issue of the co-called *hypercomputation*.

- Some physical theories allow for the theoretical possibility of hypercomputational systems (quantum, relativistic, etc.).
- No such system is actually feasible.
- Martin Davis is strongly opposed to the relevance of the "hypercomputational issue".
- Philosophical and scientific literature about hypercomputation is however more and more flourishing.

Hypercomputation

The fact P/poly contains P as well as non-recursive languages raises the issue of the co-called *hypercomputation*.

- Some physical theories allow for the theoretical possibility of hypercomputational systems (quantum, relativistic, etc.).
- No such system is actually feasible.
- Martin Davis is strongly opposed to the relevance of the "hypercomputational issue".
- Philosophical and scientific literature about hypercomputation is however more and more flourishing.

Hypercomputation

The fact P/poly contains P as well as non-recursive languages raises the issue of the co-called *hypercomputation*.

- Some physical theories allow for the theoretical possibility of hypercomputational systems (quantum, relativistic, etc.).
- No such system is actually feasible.
- Martin Davis is strongly opposed to the relevance of the "hypercomputational issue".
- Philosophical and scientific literature about hypercomputation is however more and more flourishing.

Hypercomputation

The fact P/poly contains P as well as non-recursive languages raises the issue of the co-called *hypercomputation*.

- Some physical theories allow for the theoretical possibility of hypercomputational systems (quantum, relativistic, etc.).
- No such system is actually feasible.
- Martin Davis is strongly opposed to the relevance of the "hypercomputational issue".
- Philosophical and scientific literature about hypercomputation is however more and more flourishing.