

COMPUTATIONAL CAPABILITIES OF RECURRENT NEURAL NETWORKS BASED ON THEIR ATTRACTOR DYNAMICS

Jérémie Cabessa

Joint work with Alessandro E.P. Villa

Department of Mathematical Economics
University of Paris 2
France

14 July 2015

INTRODUCTION

- ▶ We consider that some aspect of information processing in the brain can be approached from the perspective of computability theory.
- ▶ The computational capabilities of recurrent neural networks (RNN) have mainly been studied in the context of classical computation (McCulloch & Pitts, Turing, Kleene, von Neumann, Minsky, Papert,..., Siegelmann & Sontag,...).
- ▶ Here, we provide a theoretical characterization of the computational power of recurrent neural networks, in terms of their attractor dynamics (infinite input stream computation).

INTRODUCTION

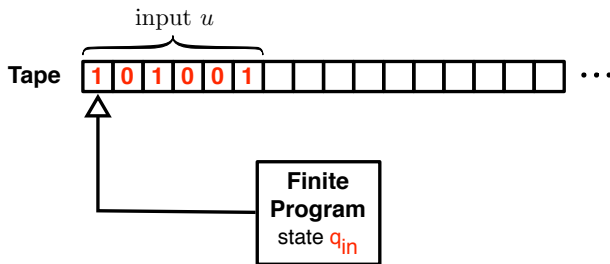
- ▶ We consider that some aspect of information processing in the brain can be approached from the perspective of computability theory.
- ▶ The computational capabilities of recurrent neural networks (RNN) have mainly been studied in the context of classical computation (McCulloch & Pitts, Turing, Kleene, von Neumann, Minsky, Papert,..., Siegelmann & Sontag,...).
- ▶ Here, we provide a theoretical characterization of the computational power of recurrent neural networks, in terms of their attractor dynamics (infinite input stream computation).

INTRODUCTION

- ▶ We consider that some aspect of information processing in the brain can be approached from the perspective of computability theory.
- ▶ The computational capabilities of recurrent neural networks (RNN) have mainly been studied in the context of classical computation (McCulloch & Pitts, Turing, Kleene, von Neumann, Minsky, Papert,..., Siegelmann & Sontag,...).
- ▶ Here, we provide a theoretical characterization of the computational power of recurrent neural networks, in terms of their attractor dynamics (infinite input stream computation).

TURING MACHINE

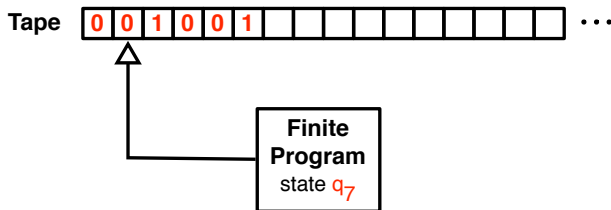
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input u is *accepted* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{acc}
- ▶ input u is *rejected* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{rej}

TURING MACHINE

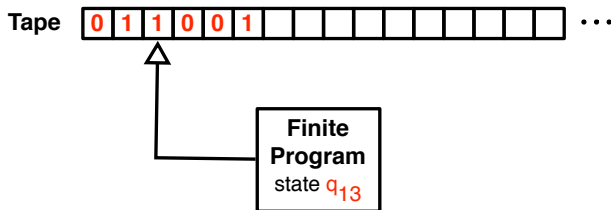
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input u is *accepted* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{acc}
- ▶ input u is *rejected* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{rej}

TURING MACHINE

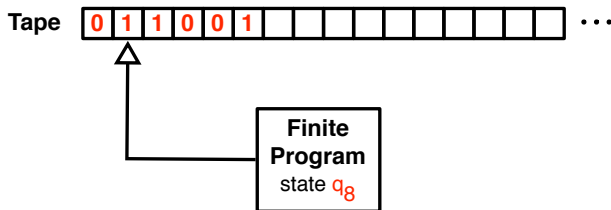
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input u is *accepted* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{acc}
- ▶ input u is *rejected* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{rej}

TURING MACHINE

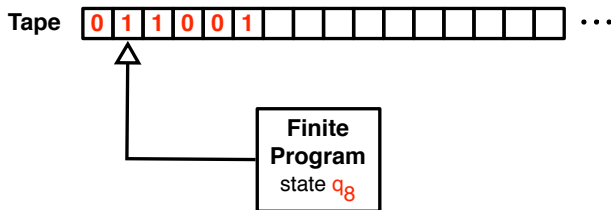
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input u is *accepted* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{acc}
- ▶ input u is *rejected* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{rej}

TURING MACHINE

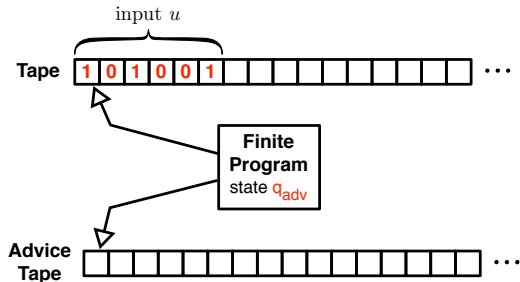
A *Turing machine* (TM) consists of an infinite tape, a read-write head, and a finite program.



- ▶ input u is *accepted* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{acc}
- ▶ input u is *rejected* by \mathcal{M} if $\mathcal{M}(u)$ reaches the state q_{rej}

TURING MACHINE WITH ADVICE

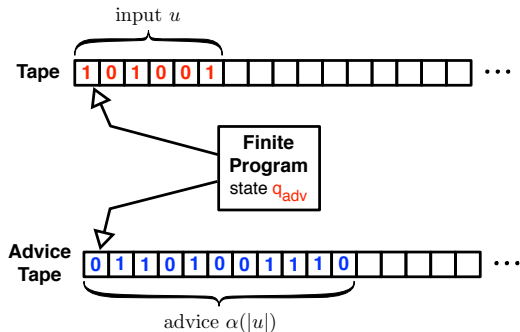
A *Turing machine with advice* (TM/A) is a TM provided with an additional advice tape and advice function $\alpha : \mathbb{N} \rightarrow \{0, 1\}^*$.



- $P/poly$ is the class of languages recognized in polynomial time by Turing machines with polynomial advices (TM/poly(A)).

TURING MACHINE WITH ADVICE

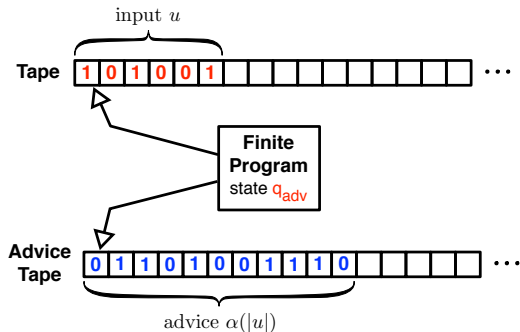
A *Turing machine with advice* (TM/A) is a TM provided with an additional advice tape and advice function $\alpha : \mathbb{N} \rightarrow \{0, 1\}^*$.



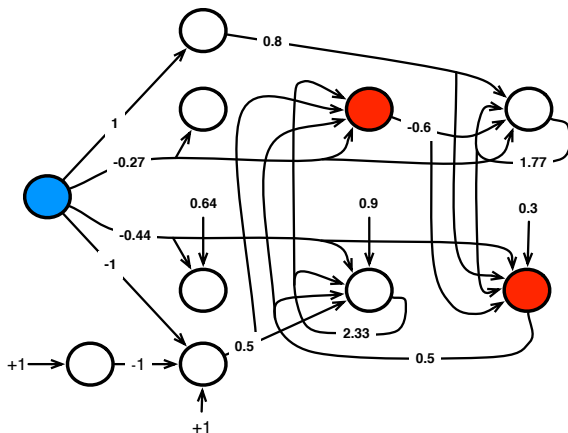
- $P/poly$ is the class of languages recognized in polynomial time by Turing machines with polynomial advices (TM/poly(A)).

TURING MACHINE WITH ADVICE

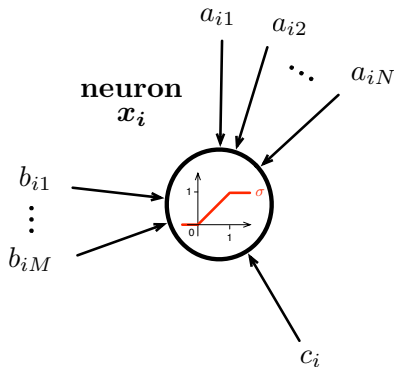
A *Turing machine with advice* (TM/A) is a TM provided with an additional advice tape and advice function $\alpha : \mathbb{N} \rightarrow \{0, 1\}^*$.



- $P/poly$ is the class of languages recognized in polynomial time by Turing machines with polynomial advices (TM/poly(A)).

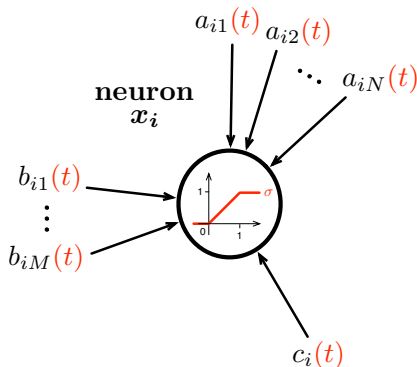


RECURRENT NEURAL NETWORKS – DYNAMICS



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

RECURRENT NEURAL NETWORKS – DYNAMICS

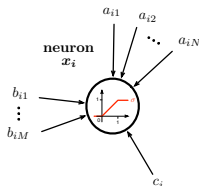


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

RECURRENT NEURAL NETWORKS – MODELS

We consider six models of RNNs:

1. static rational-weighted RNNs
2. static real-weighted (or analog) RNNs
3. evolved evolving rational-weighted RNNs
4. evolved evolving real-weighted RNNs
5. evolved evolving rational-weighted RNNs with evolution
6. evolved evolving real-weighted RNNs with evolution

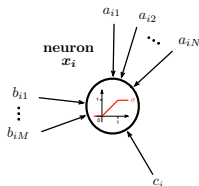


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij} \cdot x_j(t) + \sum_{j=1}^M b_{ij} \cdot u_j(t) + c_i \right)$$

RECURRENT NEURAL NETWORKS — MODELS

We consider six models of RNNs:

1. static rational-weighted RNNs
2. static real-weighted (or analog) RNNs
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

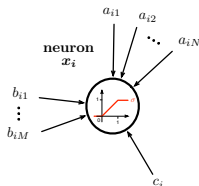


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij} \cdot x_j(t) + \sum_{j=1}^M b_{ij} \cdot u_j(t) + c_i \right)$$

RECURRENT NEURAL NETWORKS — MODELS

We consider six models of RNNs:

1. static rational-weighted RNNs
2. static real-weighted (or analog) RNNs
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

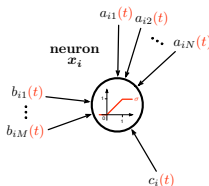


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij} \cdot x_j(t) + \sum_{j=1}^M b_{ij} \cdot u_j(t) + c_i \right)$$

RECURRENT NEURAL NETWORKS — MODELS

We consider six models of RNNs:

1. static rational-weighted RNNs
2. static real-weighted (or analog) RNNs
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

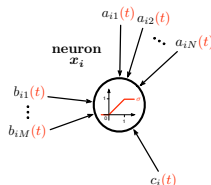


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

RECURRENT NEURAL NETWORKS — MODELS

We consider six models of RNNs:

1. static rational-weighted RNNs
2. static real-weighted (or analog) RNNs
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

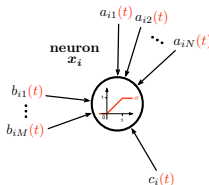


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

RECURRENT NEURAL NETWORKS — MODELS

We consider six models of RNNs:

1. static rational-weighted RNNs
2. static real-weighted (or analog) RNNs
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

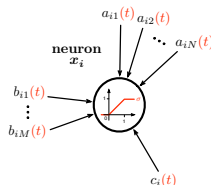


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

RECURRENT NEURAL NETWORKS — MODELS

We consider six models of RNNs:

1. static rational-weighted RNNs
2. static real-weighted (or analog) RNNs
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs



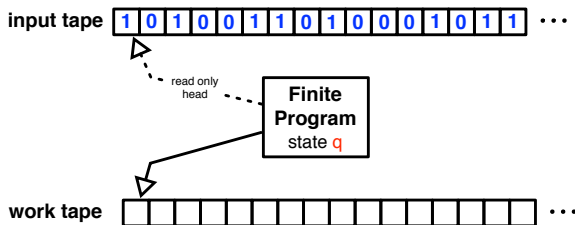
$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

RESULTS – CLASSICAL COMPUTATION

	STATIC	BI-VALUED EVOLVING	EVOLVING
\mathbb{Q}	Turing P Sieg. & Sont. 95	super-Turing P/poly Cab. & Sieg. 11,14	super-Turing P/poly Cab. & Sieg. 11,14
\mathbb{R}	super-Turing P/poly Sieg. & Sont. 94	super-Turing P/poly Cab. & Sieg. 11,14	super-Turing P/poly Cab. & Sieg. 11,14

MULLER TURING MACHINES

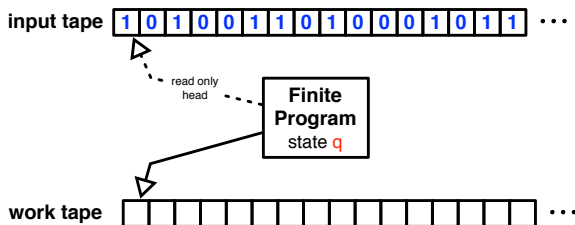
A *Muller Turing machine* is a Turing machine working over infinite input streams (reactive systems, non-terminating processes).



Muller table $\mathcal{T} = \{T_1, \dots, T_k\}$

MULLER TURING MACHINES

A *Muller Turing machine* is a Turing machine working over infinite input streams (reactive systems, non-terminating processes).

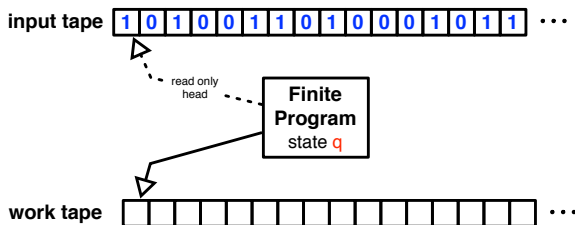


Muller table $\mathcal{T} = \{T_1, \dots, T_k\}$

- When processing an infinite input stream w , the machine necessarily visits some states infinitely often $\{q_{i_1}, \dots, q_{i_n}\}$.

MULLER TURING MACHINES

A *Muller Turing machine* is a Turing machine working over infinite input streams (reactive systems, non-terminating processes).

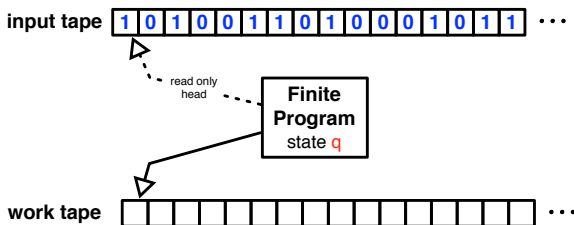


Muller table $\mathcal{T} = \{T_1, \dots, T_k\}$

- The infinite input stream w is *accepted* by \mathcal{M} if $\{q_{i_1}, \dots, q_{i_n}\} \in \mathcal{T}$. It is *rejected* by \mathcal{M} otherwise.

MULLER TURING MACHINES

A *Muller Turing machine* is a Turing machine working over infinite input streams (reactive systems, non-terminating processes).



$$\text{Muller table } \mathcal{T} = \{T_1, \dots, T_k\}$$

- The set of all input streams that are accepted by \mathcal{M} is the ω -language recognized by \mathcal{M} .

TOPOLOGICAL COMPLEXITY

Let $\{0, 1\}^\omega$ be equipped with the product topology of the discrete topology. The Borel hierarchy of $\{0, 1\}^\omega$ is as follows:

height ω_1

⋮

TOPOLOGICAL COMPLEXITY

Let $\{0, 1\}^\omega$ be equipped with the product topology of the discrete topology. The Borel hierarchy of $\{0, 1\}^\omega$ is as follows:

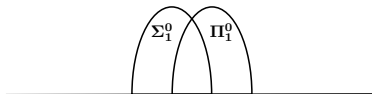
height ω_1

⋮

Σ_1^0

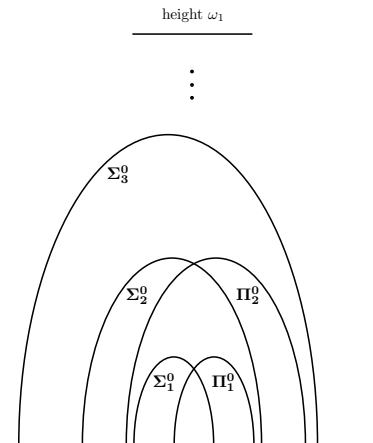
Let $\{0, 1\}^\omega$ be equipped with the product topology of the discrete topology. The Borel hierarchy of $\{0, 1\}^\omega$ is as follows:

•
•
•



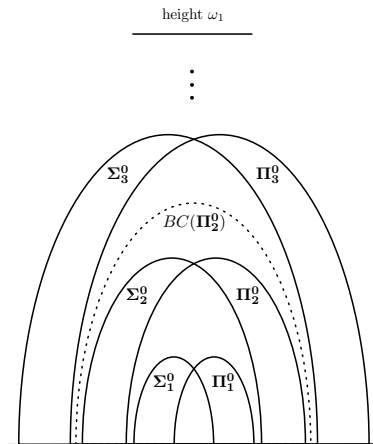
TOPOLOGICAL COMPLEXITY

Let $\{0, 1\}^\omega$ be equipped with the product topology of the discrete topology. The Borel hierarchy of $\{0, 1\}^\omega$ is as follows:



TOPOLOGICAL COMPLEXITY

Let $\{0, 1\}^\omega$ be equipped with the product topology of the discrete topology. The Borel hierarchy of $\{0, 1\}^\omega$ is as follows:

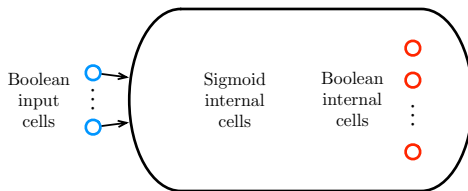


- ▶ The set of ω -languages recognizable by Muller Turing machines (depicted in red) is a countable subset of $BC(\Pi_2^0)$.



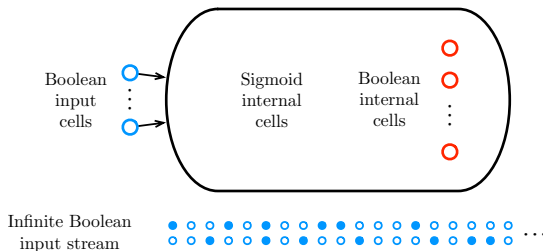
ω -HYBRID RECURRENT NEURAL NETWORKS

We consider RNNs with Boolean input cells, sigmoid and Boolean internal cells, and working on infinite input streams.



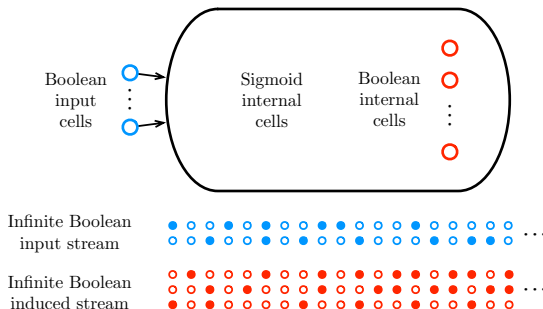
ω -HYBRID RECURRENT NEURAL NETWORKS

We consider RNNs with Boolean input cells, sigmoid and Boolean internal cells, and working on infinite input streams.



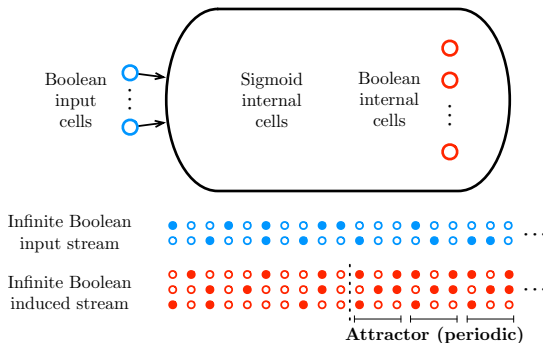
ω -HYBRID RECURRENT NEURAL NETWORKS

We consider RNNs with Boolean input cells, sigmoid and Boolean internal cells, and working on infinite input streams.



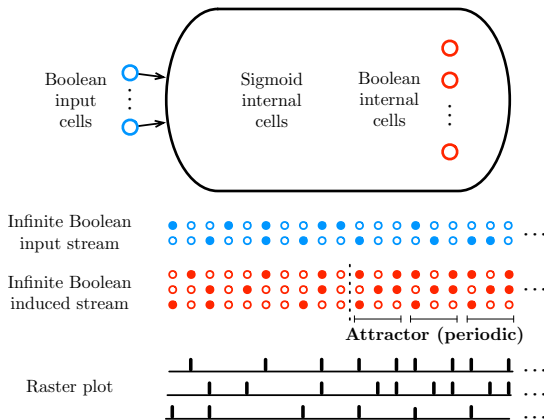
ω -HYBRID RECURRENT NEURAL NETWORKS

We consider RNNs with Boolean input cells, sigmoid and Boolean internal cells, and working on infinite input streams.



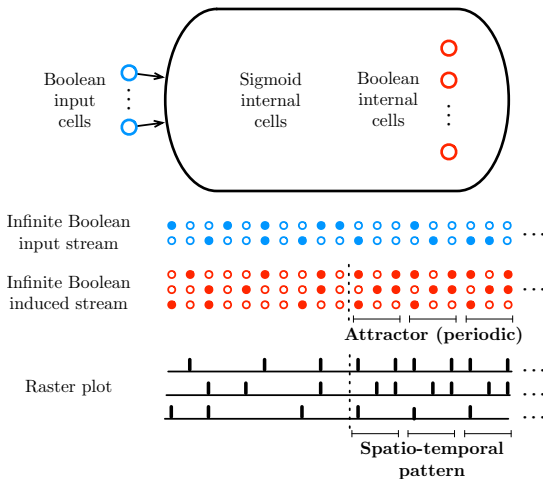
ω -HYBRID RECURRENT NEURAL NETWORKS

We consider RNNs with Boolean input cells, sigmoid and Boolean internal cells, and working on infinite input streams.

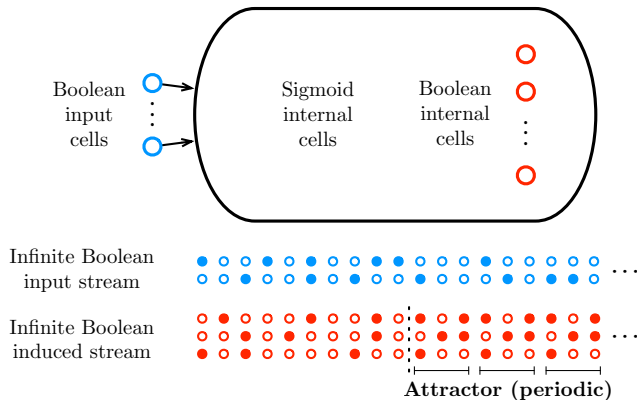


ω -HYBRID RECURRENT NEURAL NETWORKS

We consider RNNs with Boolean input cells, sigmoid and Boolean internal cells, and working on infinite input streams.

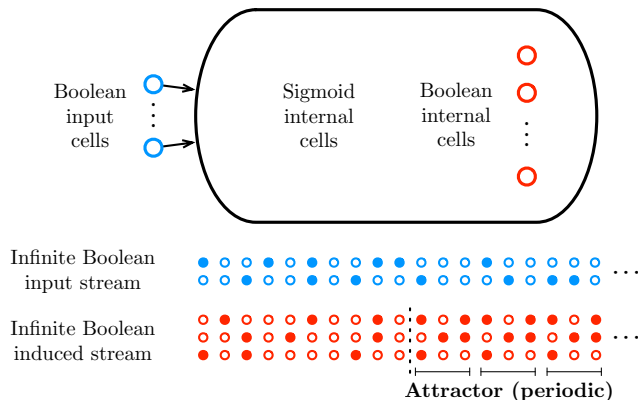


ω -HYBRID RECURRENT NEURAL NETWORKS



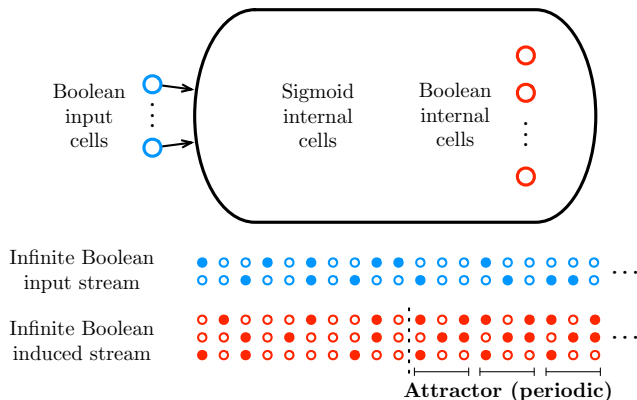
- The attractors are assumed to be classified into two possible kinds: *meaningful* or *spurious*.

ω -HYBRID RECURRENT NEURAL NETWORKS



- An infinite Boolean input stream is *accepted* by \mathcal{N} if the corresponding Boolean internal stream visits a *meaningful* attractor.

ω -HYBRID RECURRENT NEURAL NETWORKS

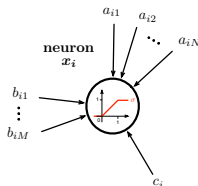


- The set of all input streams that are accepted by \mathcal{N} is the ω -language recognized by \mathcal{N} .

ω -HYBRID RECURRENT NEURAL NETWORKS

As before, we consider six models of ω -hybrid RNNs:

1. static rational-weighted RNNs.
2. static real-weighted RNNs.
3. bounded evolving rational-weighted RNNs.
4. bounded evolving real-weighted RNNs.
5. unbounded evolving rational-weighted RNNs.
6. unbounded evolving real-weighted RNNs.

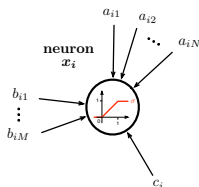


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij} \cdot x_j(t) + \sum_{j=1}^M b_{ij} \cdot u_j(t) + c_i \right)$$

ω -HYBRID RECURRENT NEURAL NETWORKS

As before, we consider six models of ω -hybrid RNNs:

1. static rational-weighted RNNs.
2. static real-weighted RNNs.
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

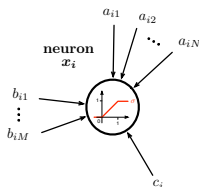


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij} \cdot x_j(t) + \sum_{j=1}^M b_{ij} \cdot u_j(t) + c_i \right)$$

ω -HYBRID RECURRENT NEURAL NETWORKS

As before, we consider six models of ω -hybrid RNNs:

1. static rational-weighted RNNs.
2. static real-weighted RNNs.
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

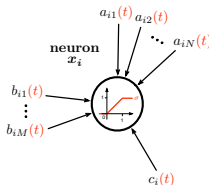


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij} \cdot x_j(t) + \sum_{j=1}^M b_{ij} \cdot u_j(t) + c_i \right)$$

ω -HYBRID RECURRENT NEURAL NETWORKS

As before, we consider six models of ω -hybrid RNNs:

1. static rational-weighted RNNs.
2. static real-weighted RNNs.
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

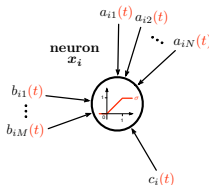


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

ω -HYBRID RECURRENT NEURAL NETWORKS

As before, we consider six models of ω -hybrid RNNs:

1. static rational-weighted RNNs.
2. static real-weighted RNNs.
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

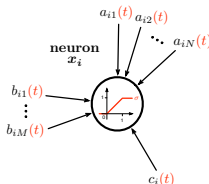


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

ω -HYBRID RECURRENT NEURAL NETWORKS

As before, we consider six models of ω -hybrid RNNs:

1. static rational-weighted RNNs.
2. static real-weighted RNNs.
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

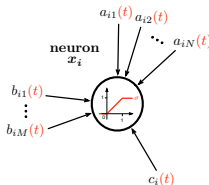


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

ω -HYBRID RECURRENT NEURAL NETWORKS

As before, we consider six models of ω -hybrid RNNs:

1. static rational-weighted RNNs.
2. static real-weighted RNNs.
3. bi-valued evolving rational-weighted RNNs
4. general evolving rational-weighted RNNs
5. bi-valued evolving real-weighted RNNs
6. general evolving real-weighted RNNs

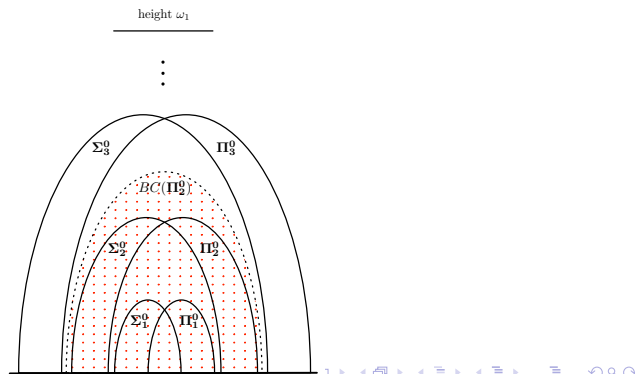


$$x_i(t+1) = \sigma \left(\sum_{j=1}^N a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^M b_{ij}(t) \cdot u_j(t) + c_i(t) \right)$$

STATIC RATIONAL ω -HYBRID RNNs

THEOREM

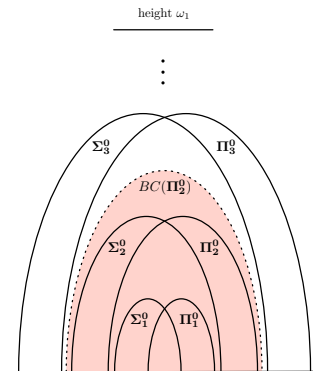
The static rational-weighted ω -hybrid RNNs are equivalent to Muller Turing machines; their underlying ω -languages belong to the class $BC(\Pi_2^0)$.



STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

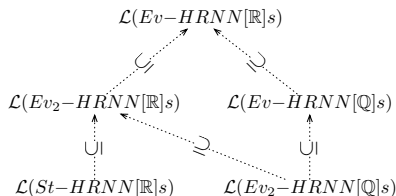
All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.



STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

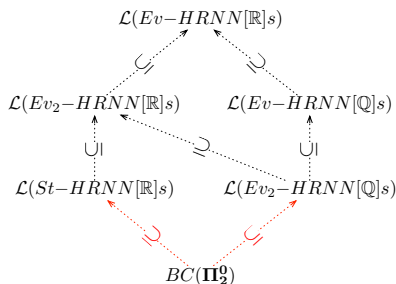
All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.



STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

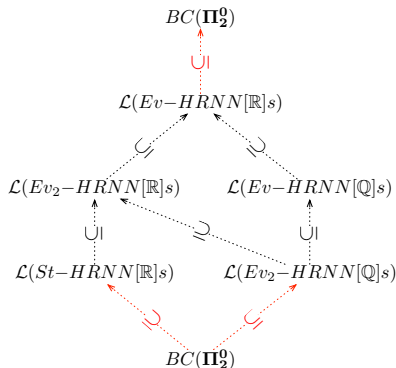
All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.



STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.



STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.

PROOF IDEA (1):

- Let $L = BC\left(\bigcap_{i \geq 0} \bigcup_{j \geq 0} p_{i,j} \cdot (\mathbb{B}^M)^\omega\right)$ be a $BC(\Pi_2^0)$ ω -language.
- We encode L into some evolving binary sequence $w_L \in \{0, 1\}^\omega$ or by some real number $r_L \in \mathbb{R}$.

Then, we construct an evolving ω -hybrid RNN that recognizes L by reading w_L or r_L and by computing the corresponding ω -language.

STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.

PROOF IDEA (1):

- ▶ Let $L = BC\left(\bigcap_{i \geq 0} \bigcup_{j \geq 0} p_{i,j} \cdot (\mathbb{B}^M)^\omega\right)$ be a $BC(\Pi_2^0)$ ω -language.
- ▶ We encode L into some evolving binary sequence $w_L \in \{0, 1\}^\omega$ or by some real number $r_L \in \mathbb{R}$.
- ▶ We can design an $\text{Ev}_2\text{-HRNN}[\mathbb{Q}]$ or an $\text{St-HRNN}[\mathbb{R}]$ which decodes w_L or r_L and recognize L , respectively.

STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.

PROOF IDEA (1):

- ▶ Let $L = BC\left(\bigcap_{i \geq 0} \bigcup_{j \geq 0} p_{i,j} \cdot (\mathbb{B}^M)^\omega\right)$ be a $BC(\Pi_2^0)$ ω -language.
- ▶ We encode L into some evolving binary sequence $w_L \in \{0, 1\}^\omega$ or by some real number $r_L \in \mathbb{R}$.
- ▶ We can design an $\text{Ev}_2\text{-HRNN}[\mathbb{Q}]$ or an $\text{St-HRNN}[\mathbb{R}]$ which decodes w_L or r_L and recognize L , respectively.

STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.

PROOF IDEA (1):

- ▶ Let $L = BC\left(\bigcap_{i \geq 0} \bigcup_{j \geq 0} p_{i,j} \cdot (\mathbb{B}^M)^\omega\right)$ be a $BC(\Pi_2^0)$ ω -language.
- ▶ We encode L into some evolving binary sequence $w_L \in \{0, 1\}^\omega$ or by some real number $r_L \in \mathbb{R}$.
- ▶ We can design an $\text{Ev}_2\text{-HRNN}[\mathbb{Q}]$ or an $\text{St-HRNN}[\mathbb{R}]$ which decodes w_L or r_L and recognize L , respectively.

STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.

PROOF IDEA (2):

- ▶ Let $L(\mathcal{N}) \subseteq (\mathbb{B}^M)^\omega$ be recognized by some Ev-HRNN[\mathbb{R}] \mathcal{N} .
- ▶ We can prove that $L(\mathcal{N})$ is the preimage of $BC(\Pi_2^0)$ -set by some continuous function, and therefore, $L(\mathcal{N}) \in BC(\Pi_2^0)$.

STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.

PROOF IDEA (2):

- ▶ Let $L(\mathcal{N}) \subseteq (\mathbb{B}^M)^\omega$ be recognized by some $\text{Ev-HRNN}[\mathbb{R}] \mathcal{N}$.
- ▶ We can prove that $L(\mathcal{N})$ is the preimage of $BC(\Pi_2^0)$ -set by some continuous function, and therefore, $L(\mathcal{N}) \in BC(\Pi_2^0)$.

STATIC ANALOG AND EVOLVING ω -HYBRID RNNs

THEOREM

All models of static real-weighted and evolving ω -hybrid RNNs are all super-Turing equivalent; they recognize the class of all $BC(\Pi_2^0)$ ω -languages.

PROOF IDEA (2):

- ▶ Let $L(\mathcal{N}) \subseteq (\mathbb{B}^M)^\omega$ be recognized by some $\text{Ev-HRNN}[\mathbb{R}] \mathcal{N}$.
- ▶ We can prove that $L(\mathcal{N})$ is the preimage of $BC(\Pi_2^0)$ -set by some continuous function, and therefore, $L(\mathcal{N}) \in BC(\Pi_2^0)$.

SUMMARY

	STATIC	BI-VALUED EVOLVING	EVOLVING
\mathbb{Q}	Turing (Muller) $\in BC(\Pi_2^0)$	super-Turing $= BC(\Pi_2^0)$	super-Turing $= BC(\Pi_2^0)$
\mathbb{R}	super-Turing $= BC(\Pi_2^0)$	super-Turing $= BC(\Pi_2^0)$	super-Turing $= BC(\Pi_2^0)$

CONCLUSION

- ▶ We provided a characterization of the super-Turing computational power of recurrent neural networks in terms of their attractor dynamics.
- ▶ In general, the super-Turing computational capabilities of neural models raises the question of *hypercomputation*.
- ▶ Current physical theories are consistent with the possibility of hypercomputational systems (quantum, relativistic, etc.). No such systems are currently feasible or harnessable.
- ▶ Philosophical and scientific literature about hypercomputation is however flourishing.

CONCLUSION

- ▶ We provided a characterization of the super-Turing computational power of recurrent neural networks in terms of their attractor dynamics.
- ▶ In general, the super-Turing computational capabilities of neural models raises the question of *hypercomputation*.
- ▶ Current physical theories are consistent with the possibility of hypercomputational systems (quantum, relativistic, etc.). No such systems are currently feasible or harnessable.
- ▶ Philosophical and scientific literature about hypercomputation is however flourishing.

CONCLUSION

- ▶ We provided a characterization of the super-Turing computational power of recurrent neural networks in terms of their attractor dynamics.
- ▶ In general, the super-Turing computational capabilities of neural models raises the question of *hypercomputation*.
- ▶ Current physical theories are consistent with the possibility of hypercomputational systems (quantum, relativistic, etc.). No such systems are currently feasible or harnessable.
- ▶ Philosophical and scientific literature about hypercomputation is however flourishing.

CONCLUSION

- ▶ We provided a characterization of the super-Turing computational power of recurrent neural networks in terms of their attractor dynamics.
- ▶ In general, the super-Turing computational capabilities of neural models raises the question of *hypercomputation*.
- ▶ Current physical theories are consistent with the possibility of hypercomputational systems (quantum, relativistic, etc.). No such systems are currently feasible or harnessable.
- ▶ Philosophical and scientific literature about hypercomputation is however flourishing.