# Notes on Turing Computability of Neural Nets

Discussion on Hava T. Siegelmann and Eduardo Sontag’s research

Compiled by D. Gueorguiev 3/1/2024

## Introductory Notes

The arguments in Hava T. Siegelmann’s paper rely on the notion of *threshold (binary valued) neuron* introduced by [McCulloch and Pits in 1943](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/A_Logical_Caclculus_of_the_Ideas_Immanent_in_Nervous_Activity_McCulloch_and_Pitts_1943.pdf) and a network composed by the latter.

Further

**Discussion of RNN** in Hava T. Siegelmann’s [paper from 1991](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/computability/TuringComputabilityWithNeuralNets_Siegelman1991.pdf).

It has been shown in the literature (JB Pollack in his PhD dissertation for instance) that certain RNN model is Turning complete. In the JB Pollack’s model all neurons synchronously update their states according to a quadratic combination of past activation values. This is an example of *a higher-order net* in which activations are combined using multiplications. So the question is if these higher order connections are really necessary to obtain Turing completeness. The answer to this question is provided in Siegelmann’s paper showing that standard linear connections are enough to construct Turing complete RNN networks.

In Siegelmann’s paper in the context of Turing computability the individual neurons are denoted as “*evolving processors”*. Around the same time, other researchers in the context of dynamic stability use the term network of “*saturable amplifiers*” in their analysis of the dynamic behavior of neural networks ([see paper](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/dynamics_and_stability/NIPS-1988-dynamics-of-analog-neural-networks-with-time-delay-Marcus.pdf)).

A recursive net is an arbitrary interconnection of synchronously evolving processors. One of the processors serves as an *output node* of the neural net. There is an external input signal that feeds into every processor.

We know that finitely many threshold neurons cannot simulate more than finite automata behavior ([paper here](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/computability/Efficient_simulations_of_finite_automata_by_neural_nets_Alon_1991.pdf)). In the Siegelmann’s paper are studied continuous-valued neurons.

Each neural net represents a dynamical system with scalar inputs. At each instant the state of the system is a vector of rational numbers, where the th coordinate keeps track of the activation value of the -th processor. The external input scalar signal feeds into every processor and its weight into each processor will be modeled by the vector .

**Definition**: Processor Net

*Processor Net* is described by a system of equations in the form:

(1)

Here the *vector sigmoid function* is defined as follows:

where the vector input is given as and the scalar function is the regular *saturated-linear* sigmoid function given with

(2)

Given an infinite sequence of rational numbers serving as inputs to the network, then the state of the network is defined as the solution of (1) for each integer with initial condition .

The reason why the scalar domain in the definition of the processor net was chosen to be the set of rational numbers is because this allows to establish easily that a Turing machine can simulate any output from processor net defined as above.

Hava T. Siegelmann establishes the converse result – that any function computable by Turing machine can be computed by processor net. In order to precisely formulate the statement the notion of a *unary input signal* is introduced.

**Definition** unary input signal of Processor Net

Any input sequence which consists of a string of ’s () followed by an infinite string of ’s.

Theorem:

Let

## Literature

[Turing Computability with Neural Nets, Hava T. Siegelmann, Eduardo Sontag, 1991](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/computability/TuringComputabilityWithNeuralNets_Siegelman1991.pdf)

[On the computational power of Neural Nets, Hava T. Siegelmann, 1992 (earlier version)](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/computability/OnTheComputationalPowerOfNeuralNets_1992_Siegelmann.pdf)

[On The Computation Power of Neural Nets, Hava T. Siegelmann, Eduardo Sontag, 1995](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/computability/OnTheComputationalPowerOfNeuralNets_1995_Siegelmann_JComSysSci.pdf)

[Computation beyond Turing limit, Hava T. Siegelmann, 1995](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/computability/ComputationBeyondtheTuringLimit_1995_Siegelmann_Science.pdf)

[Neural Networks and Analog Computation beyond Turing limit, Hava T. Siegelmann, 1999](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/computability/NeuralNetworksandAnalogComputationBeyondTheTuringLimit.pdf)

[Efficient Simulation of Finite Automata by Neural Nets, Noga Alon et al, JACM, Vol. 38, No. 2, 1991](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/computability/Efficient_simulations_of_finite_automata_by_neural_nets_Alon_1991.pdf)

[The Induction of Dynamical Recognizers, JB Pollack, Ohio State University, 1991](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/dynamics_and_stability/the_induction_of_dynamical_recognizers_pollack_1991.pdf)

[Dynamics of Analog Neural Networks with Time Delay, CM Marcus and RM Westervelt, NIPS, 1988](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/dynamics_and_stability/NIPS-1988-dynamics-of-analog-neural-networks-with-time-delay-Marcus.pdf)

## Appendix

### Computable functions and partial-recursive functions

**Definition**: Computable function

A function is said to be *computable* if it is manufactured from a specific set of *basic functions* and some *construction rules*

**Definition** *primitive function* (aka *axiom*) are:

* , the zero-ary constant
* , the unary successor function
* set of -ary projection functions

**Definition** construction rules are:

* composition rule (): if and are computable functions, then is a computable function. We use the shorthand notation:
* recursion rule (): if and are computable functions, then the unique function defined by and is a computable function. We use the shorthand notation: .

**Definition**: *primitive recursive* functions

functions obtainable from the primitive function by applying the construction rules of composition and recursion are denoted as *primitive recursive functions*.

**Definition**: *minimization* construction rule ():

**Theorem**: If is effectively computable total or partial function, then will also be effectively computable total or partial function.

In case f is a total function, we can remove the second of the two ways in which may fail to be defined . Thus the last Definition becomes:

### Mealy Machines

**Definition**: -state binary *Mealy Machine*

-state binary *Mealy Machine*  is deterministic finite automaton which at each point in the discrete time () is in a state . is *the state space*. At each point in time , receives an input where is *the input alphabet*. At each point in time t, generates an output where is *the output alphabet*.

The output depends on the state and the input through *the state-output maps*:

(1)

where

(2)

The set of all state-output maps will be denoted with . Thus .

The new state depends on the old state and the input through the *state update maps*

(3)

where

(4)

The set of all state-update maps will be denoted with . Thus .

A Mealy Machine is entirely determined by its space state , the input alphabet , the output alphabet , state-output maps , and state-update maps . Thus .

Mealy Machine as a labeled directed graph with cycles

The -state Mealy Machine can be thought of as a directed graph with nodes, where every node has two outgoing arcs: one which is depicted with dashed line and represents low input and the other one with solid line representing high input . Each of the 2m arcs in the graph is labeled either with representing low output or with representing high output .

An example Mealy Machine for represented with such graph is shown below:

3

1

2

Figure 1: Mealy Machine for

**Question 1**: how to decide whether two Mealy Machines are the same or not.

Let the first machine is defined with the triplet while the second machine is defined with the triplet . Here denotes the size of the state space, is the set of the two state-update maps , , is the set of the two state-output maps , for each of the machines accordingly.

Understandably, one may argue that the following relations establish equivalence between the two machines:

(5)

, ,, , (6)

This simple argument leads to the conclusion that there are different -state Mealy Machines. This clearly is not a tight estimate of the upper bound of the number of distinct -state Mealy Machines.

There are different ways to relabel the states which still will result in equivalent machines. So a tighter upper bound for the number of distinct -state Mealy machines would be .

For more rigorous treatment, we ask ourselves another question :

**Question 2**: when is Mealy Machine implementation of Mealy Machine ?

In other words, when any copy of can be replaced by a copy ?

**Definition**. Mealy Machine , defined with the triplet is an implementation of , defined with the triplet

Whenever there exists a canonical map with the property that whenever starts in any state (that is, ), and starts in state with , then for every common input sequence as long as both machines receive the input sequence it follows that both machines generate the same output sequence .