On the Computational Power of Neural Nets

by Hava Siegelmann and Eduardo Sontag (1995)[3]

Presented by Jiaman Wu

Outline

- Problem Statement
- Model with I/O
- Results of Model with I/O
- Model without I/O
- Results of Model without I/O
- Model of 2-stack Turing Machine
- Highlight of Proof

Problem Statement

- Recurrent Neural Net (RNN) can theoretically simulate any Turing Machines (TM) in real time.
  - Turing complete: compute any algorithm which our computer can compute
- Many consequences of Turing completeness:
  - Undecidable ⇔ not Turing computable: no algorithm can compute it and TM loops forever.
  - Undecidable to determine if a given neuron ever assumes the value “0”.
  - Undecidable to determine if a dynamical system of the particular form x(t+1)=σ(Ax(t)+c) ever reaches an equilibrium point, from a given initial state. [2]

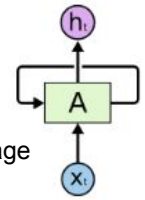

Figure 1. RNN diagram. Image from [4]

Model of RNN with I/O

- Basic structure of RNN
  - Saturated-linear function is as activation of finite neurons in the hidden layer.
  - 2 infinite binary input lines are in input layer (TBD in the later).
  - Partially defined function maps to output (TBD in the later).
  - Coefficients are rationals.

$x(t+1) = \sigma(Ax(t) + bu(t) + c)$: update rule

$x(t) \in \mathbb{Q}^N$: the state at time t

$u(t) = [u_D(t), u_V(t)]^T \in \{0,1\}^2$: inputs (TBD in the later)

$A \in \mathbb{Q}^{N \times N}$: weights

$b \in \mathbb{Q}^{N \times 2}$: weights

$c \in \mathbb{Q}^N$: bias

$y(t) = [y_D(t), y_V(t)]^T \in \{0,1\}^2$: outputs (TBD in the later)

$$\sigma(x) := \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leqslant x \leqslant 1 \\ 1 & \text{if } x > 1. \end{cases}$$

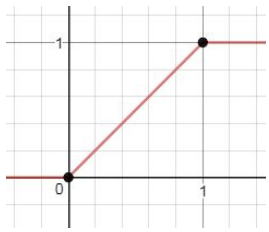

Figure 2. Saturated-linear function graph

Model of RNN with I/O

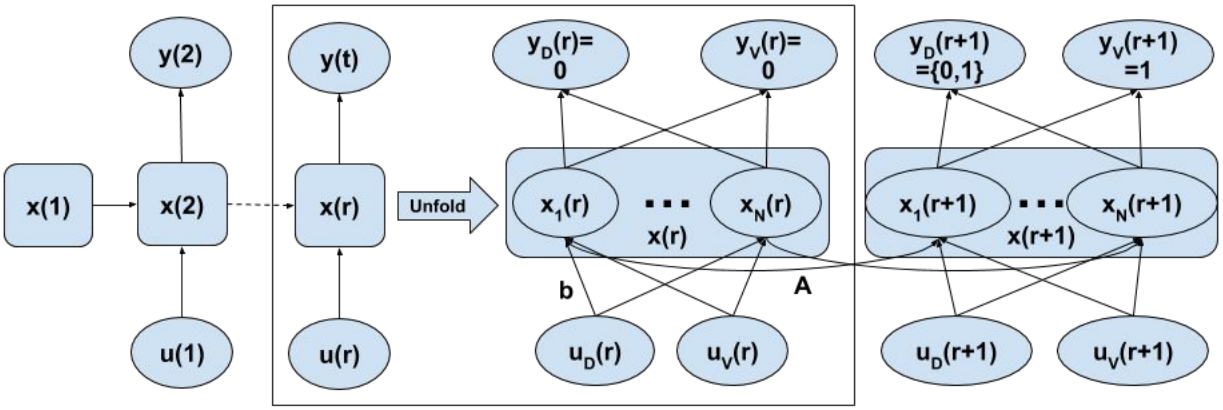

Figure 3. Diagram of RNN with I/O

Model of RNN with I/O

- Two infinite binary input lines for reading w ∈ $\{0,1\}^k$

$$u(t) = \begin{cases} [w_t \in \{0,1\}, 1]^T, & \text{if } t = 1, \ldots, k \\ [0,0]^T, & \text{otherwise} \end{cases}$$

t	1	2	3	4	5	...
data line u_D	w_1=1	w_2=0	w_3=1	0	0	...
valid line u_V	1	1	1	0	0	...
u(t)	u(1)=$[1,1]^T$	u(2)=$[0,1]^T$	u(3)=$[1,1]^T$	u(4)=$[0,0]^T$	u(5)=$[0,0]^T$	...

w

Model of RNN with I/O

- Partially defined function maps input w to output ϕ(w) ∈ $\{0,1\}^l$ or ϕ(w) undefined
  - ϕ(w) is partially defined function: some w's are not mapped and their ϕ(w) is undefined.
  - ϕ(w) is undefined ⇔ TM loops forever and RNN gives all 0's on both lines.
  - ϕ(w) is defined ⇔ TM halts and RNN gives output on data line and 1's on validation line
  - Output takes time r ∈ N, the response time.

$$y_r(t) = \begin{cases} [\phi_{t-r+1}(w), 1]^T, & \text{if } \phi(w) \text{ defined and } t = r, \ldots, r+l-1 \\ [0,0]^T, & \text{otherwise} \end{cases}$$

Results of Model with I/O

- Theorem 1. Let ϕ: $\{0,1\}^+ \rightarrow \{0,1\}^+$ be any recursively computable partial function. Then, there exists a net N with the following property: If N is started from the 0 initial state, and the input sequence u is applied, N produces the output y_r for some r. Furthermore, if a p-stack Turing Machine M computes ϕ (w) in time T(w), then one may take r(w)=O(|w|)+T(w).
- Any function computable by a Turing Machine can be computed by such a processor net.
- 886 neurons are sufficient for computing such a universal partial recursive function to be Turing complete.

Model of RNN without I/O

- A model analogous to the previous one but without input and output stream.
- It is more convenient to encode the binary input string into Cantor set and to the initial state.
  - The number ranges in [0,1) with gaps.
  - Encoding converts binary into rationals.

$$\delta(w) = \sum_{i=1}^{k} \frac{2w_i+1}{4^i}$$

$$x^1(w) = [\delta(w), 0, \ldots, 0]^T \in \mathbb{Q}^N$$

$$x^{t+1}(w) = \sigma(Ax^t(w) + c)$$

Example of encoding with Cantor set:
empty → 0
0 → 1/4
1 → 3/4
01 → 1/4 + 3/4^2 = 0.4375
10 → 3/4 + 1/4^2 = 0.8125
11 → 3/4 + 3/4^2 = 0.9375

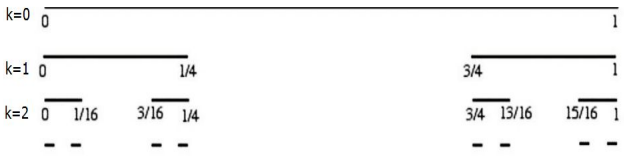

Figure 4. Graph of Cantor set [1]

Results of RNN without I/O

Theorem 2. Let M be a p-stack Turing Machine computing ϕ: $\{0,1\}^+ \rightarrow \{0,1\}^+$ in time T. Then there exists a processor net N without inputs so that properties below hold. In both cases, for each $w \in \{0,1\}^+$, we consider the corresponding initial state $x^1(w) = [\delta[w],0,...,0]^T \in Q^N$.

ϕ(w) is undefined, if and only if $x^j_2(w)$ is identically equal to 0, for all j. If instead ϕ (w) is defined and computed in time T, there exists a J=O(T) so that: $x^j_2(w)=0$, $j \in [0, J-1]$ and $x^J_2(w)=1$, $x^J_1(w)=\delta[\phi(w)]$. ($x^J(w)=[\delta[\phi(w)],1,...]^T$)

Model of 2-stack TM

- 2-tape TM
  - 2 tapes
  - Operations: read, write, move
- 2-stack TM is a common 2-tape TM, but has different operations.
  - Read the top element
  - Push 0 or 1
  - Pop
  - Judge if empty

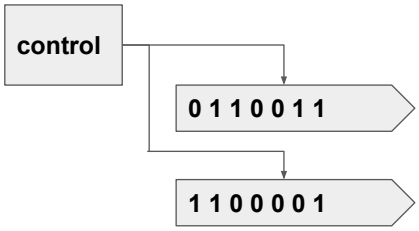

Figure 5. Diagram of 2-tape TM

Highlights of Proof

- Rational weighted RNN is proven to be computationally equivalent to TMs
- Encode binary string into Cantor set of number $\delta(w) = \sum_{i=1}^{k} \frac{2w_i+1}{4^i}$
- Simulate TM stack operation:
  - No operations: q = q
  - Read the top element: top(q) = σ(4q-2)
    - e.g. 1000 → 0.832, σ(4*0.832-2= 1.328) = 1
    - e.g. 0001 → 0.3398, σ(4*0.3398-2= -0.6408) = 0
  - Push 0: q/4+1/4; Push 1: q/4+3/4
  - Pop: q = 4q-2top(q)-1 = 4q-1(if pop 0) or 4q-3 (if pop 1)
    - e.g. pop 1000 → 4*0.832-3=0.328; 000 → 0.328
  - Judge if empty: σ(4q)=0 ⇔ empty
- 3 neurons for encoding q, top(q), empty indicator per stack + finite neurons for no operators, push, pop + finite neurons for reading and computing the next step.

Reference

[1] A little finding regarding the middle-half Cantor set. Retrieved from https://drchristiansalas.files.wordpress.com/2013/02/n04.pdf

[2] Hopfield, J. J., & Tank, D. W. (1985). “Neural” computation of decisions in optimization problems. Biological cybernetics, 52(3), 141-152.

[3] Siegelmann, H. T., & Sontag, E. D. (1995). On the computational power of neural nets. Journal of computer and system sciences, 50(1), 132-150.

[4] Wang, T. Recurrent Neural Network. Retrieved from https://www.cs.toronto.edu/~tingwuwang/rnn_tutorial.pdf