ICONIP'2008 Tutorial on

Computational Resources in Neural Network Models

Jiří Šíma

Institute of Computer Science
Academy of Sciences of the Czech Republic

(Artificial) Neural Networks (NNs):

1. mathematical models of biological neural systems

- constantly refined to reflect current neurophysiological knowledge
- modeling the cognitive functions of human brain

already first computer designers sought their inspiration in human brain (neurocomputer, `Minsky,1951`) $\longrightarrow$

2. computational tools

- alternative to conventional computers
- learning from training data
- one of the standard tools in machine learning and data mining
- used for solving artificial intelligence tasks: pattern recognition, control, prediction, decision, signal analysis, fault detection, diagnostics, etc.
- professional software implementations (e.g. `Matlab`, `Statistica` modules)
- successful commercial applications

Neural Networks as Abstract (Formal) Computational Models

- mathematically defined computational machines
- idealized models of practical NNs from engineering applications, e.g. analog numerical parameters are true real numbers, the potential number of computational units is unbounded, etc.
- investigation of computational potential and limits of neurocomputing

special-purpose NNs: implementations of neural circuits for specific practical problems (e.g. robot control, stock predictions, etc.)

$\times$ general-purpose computation with NNs:

- the study of classes of functions (problems) that can be computed (solved) with various NN models (e.g. XOR cannot be computed by a single perceptron)
- what is ultimately or efficiently computable by particular NN models?

analogy to computability and complexity theory of conventional computers (PCs) with classical abstract computational models such as Turing machines (recursive functions), finite automata (regular languages), etc.

$\longrightarrow$ **Computability and Complexity Theory of Neural Networks**

- methodology: the computational power end efficiency of NNs is investigated by comparing their various formal models with each other and with more traditional computational models such as finite automata, grammars, Turing machines, Boolean circuits, etc.

- NNs introduce new sources of efficient computation: energy, analog state, continuous time, temporal coding, etc. (in addition to usual complexity measures such as computation time and memory space)

  $\longrightarrow$ NNs may serve as reference models for analyzing these non-standard computational resources

- NN models cover basic characteristics of biological neural systems: plenty of densely interconnected simple computational units

  $\longrightarrow$ computational principles of mental processes

Three Main Directions of Research:

1. learning and generalization complexity:
   effective creation and adaptation of NN representation
   e.g. how much time is needed for training? how many training data is required for successful generalization?

2. descriptive complexity:
   memory demands of NN representation
   e.g. how many bits are needed for weights?

3. computational power:
   effective response of NNs to their inputs
   e.g. which functions are computable by particular NN models?

Tutorial Assumptions:

- no learning issues, this would deserve a separate survey on computational learning theory, e.g. Probably Approximately Correct (PAC) framework, etc.

- only digital computation: binary (discrete) I/O values, may be encoded as analog neuron states and intermediate computation may operate with real numbers

  × NNs with real I/O values are studied in the approximation theory (functional analysis)

Technical Tools (5-slide discursion)

1. Formal Languages and Automata Theory

formal language = set of words (strings) over an alphabet, for simplicity assume binary alphabet $L \subseteq \{0,1\}^*$

L corresponds to a decision problem: L contains all positive input instances of this problem,

e.g. for the problem of deciding whether a given natural number is a prime, the corresponding language $PRIME$ contains exactly all the binary expressions of primes

(deterministic) finite automaton (FA) A recognizing a language $L = L(A)$:

- reads an input string $\mathbf{x} \in \{0,1\}^*$ bit after bit
- a finite set of internal states (including a start state and accepting states)
- transition function (finite control):

  $$\mathbf{q_{current}}\,,\ \mathbf{x_i} \longmapsto \mathbf{q_{new}}$$

  given a current internal state and the next input bit, produces a new internal state
- $\mathbf{x}$ belongs to L if A terminates in an accepting state

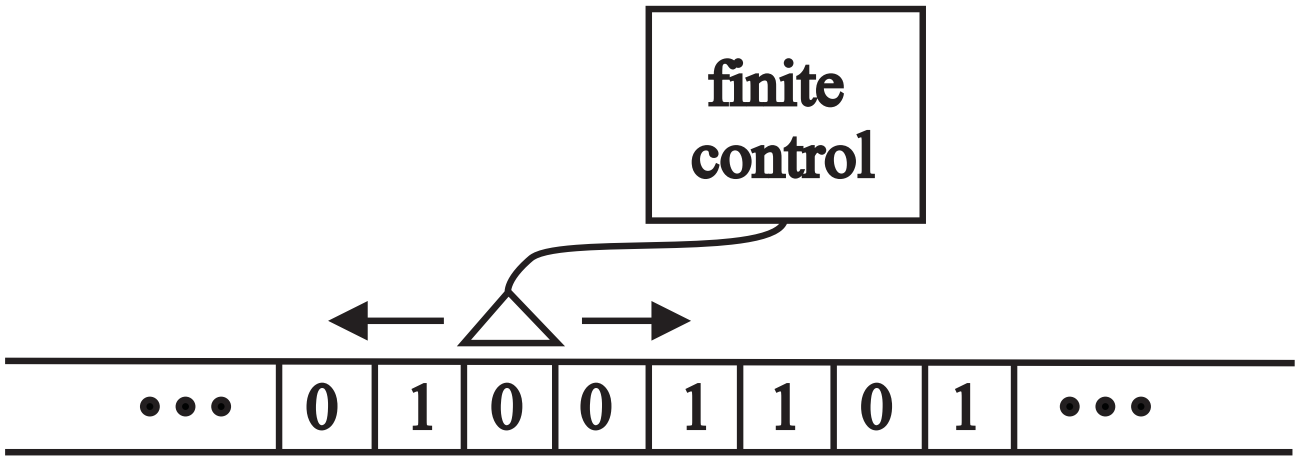
FA recognize exactly regular languages described by regular expressions (e.g. $(0+1)^*000$; $\times$ $\{0^n1^n\,;\, n \geq 1\}$ is not regular)

Turing machine (TM) = finite automaton (finite control)
+ external unbounded memory tape

- the tape initially contains an input string
- the tape is accessible via a read/write head which can move by one cell left or right
- transition function (finite control):

$$q_{\mathbf{current}}\,,\ x_{\mathbf{read}} \longmapsto q_{\mathbf{new}}\,,\ x_{\mathbf{write}}\,,\ \text{head\_move}$$

given a current internal state and a bit on the tape under head, produces a new internal state, a rewriting bit, and the head move (left or right)



e.g. TM (in contrast to FA) can read the input repeatedly and store intermediate results on the tape

TMs compute all functions that are ultimately computable e.g. on PCs (recursive functions)

$\longrightarrow$ widely accepted mathematical definition of "algorithm" (finite description)

2. Complexity Theory

- *what is computable using bounded computational resources*, e.g. within bounded time and memory
  $\longrightarrow$ the time and space complexity

- the complexity is measured in terms of input length (potentially unbounded)

- TM working in time $t(n)$ for inputs of length n performs at most $t(n)$ actions (computational steps)

  worst case complexity: also the longest computation over all inputs of length n must end within time $t(n)$
  ($\times$ average case analysis)

- TM working in space $s(n)$ for inputs of length n uses at most $s(n)$ cells of its tape

"big-O" notation:
e.g. $t(n) = O(n^2)$ (asymptotic quadratic upper bound): there is a constant r such that

$$t(n) \leq r \cdot n^2$$

for sufficiently large n, i.e. the computation time grows at most quadratically with the increasing length of inputs

similarly lower bound $t(n) = \Omega(n^2)$ ($t(n) \geq r \cdot n^2$): the computation time grows at least quadratically

$t(n) = \Theta(n^2)$ iff $t(n) = O(n^2)$ and $t(n) = \Omega(n^2)$

Famous Complexity Classes:

P is the class of decision problems (languages) that are solved (accepted) by TMs within polynomial time, i.e. $t(n) = O(n^c)$ for some constant c

$\longrightarrow$ *considered computationally feasible*

NP is the class of problems solvable by nondeterministic TMs within polynomial time

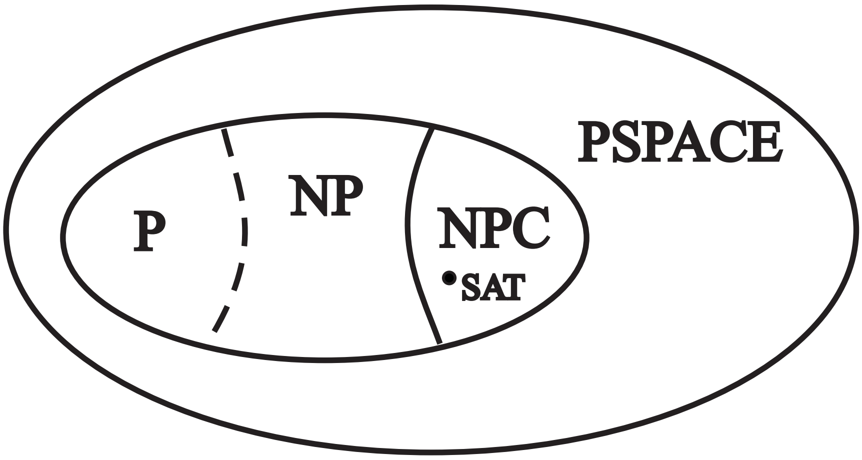
nondeterministic TM (program) can choose from e.g. two possible actions at each computational step

$\longrightarrow$ an exponential number of possible computational paths (tree) on a given input ($\times$ a single deterministic computational path)

definition: an input is accepted iff there is at least one accepting computation

example: the class of satisfiable Boolean formulas SAT is in NP: a nondeterministic algorithm "guesses" an assignment for each occurring variable and checks in polynomial time whether this assignment satisfies the formula

$\longrightarrow$ NP contains all problems whose solutions (once nondeterministically guessed) can be checked in polynomial time



NPC is the class of NP-complete problems which are the hardest problems in NP:

if A from NPC (e.g. SAT) proves to be in P then P=NP

i.e. by solving only one NP-complete problem in polynomial time one would obtain polynomial-time solutions for all problems in NP

$\longrightarrow$ *believed to be computationally infeasible*

i.e. NP $\neq$ P (finding the solutions is more difficult than their checking)

coNP contains the complements of NP languages

PSPACE is the class of problems that are solved by TMs within polynomial space; similarly PSPACE-complete problems are the harderst problems in PSPACE

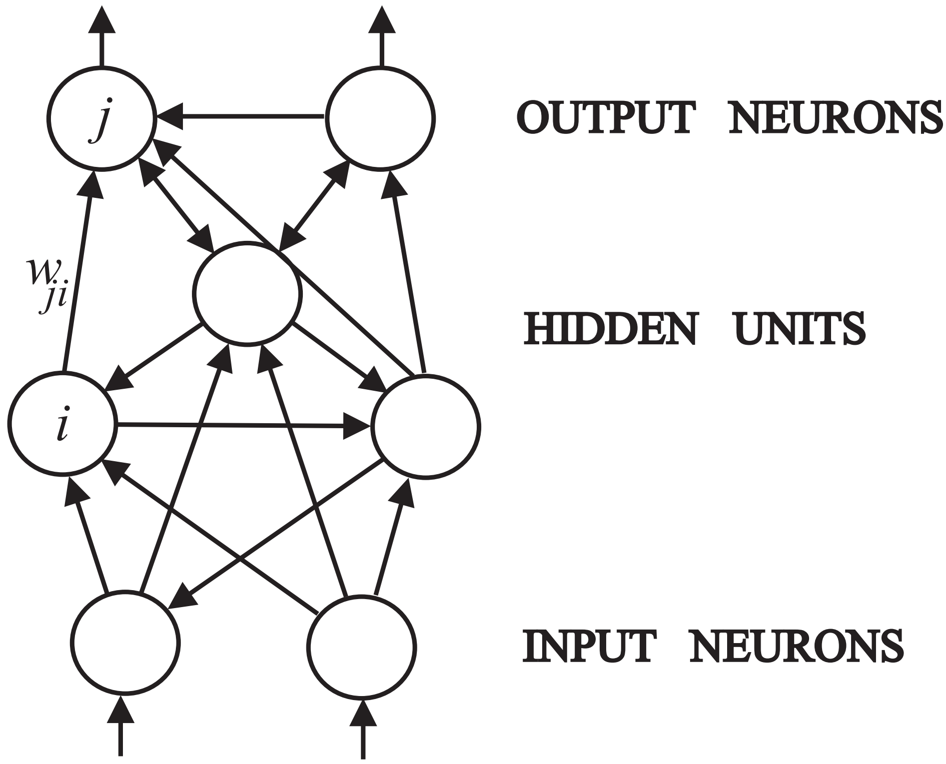
$$\text{P} \subset \text{NP} \subset \text{PSPACE}, \quad \text{P} \subset \text{coNP} \subset \text{PSPACE}$$

the main open problem in the theory of computation (mathematics) is to prove that these inclusions are proper

(end of discursion)

Definition of a Formal Neural Network Model

(sufficiently general to cover almost all practical NNs, will later be narrowed to specific NNs)



- Architecture: s computational *units (neurons)* $V = \{1, \ldots, s\}$ connected into a directed graph

  $\longrightarrow s = |V|$ is the size of the network

- Interface: n *input* and m *output* units, the remaining ones are called *hidden* neurons

- each edge from i to j is labeled with a real *weight* w_{ji} ($w_{ji} = 0$ iff there is no edge (i, j))

- Computational Dynamics: the evolution of *network state*

$$\mathbf{y}^{(t)} = (y_1^{(t)}, \ldots, y_s^{(t)}) \in \Re^s$$

at each time instant $t \geq 0$

1. *initial state* $\mathbf{y}^{(0)}$ (includes an external input)

2. network state updates: neurons from a subset $\alpha_t \subseteq V$ collect their inputs from incident units via weighted connections and transform them into their new outputs (states)

3. a global output is read from the output neurons at the end (or even in the course) of computation

Criteria of NN Classification

restrictions imposed on NN parameters and/or computational dynamics

- Unit Types: perceptrons, RBF, WTA gates, spiking neurons, etc.
- Dynamics: discrete $\times$ continuous time
- Control: deterministic $\times$ probabilistic
- Architecture: feedforward $\times$ recurrent
- State Domain: binary (discrete) $\times$ analog
- Size (Input Protocol): finite $\times$ infinite families of networks
- Weights: symmetric (antisymmetric) $\times$ asymmetric
- Mode: sequential $\times$ parallel

A Taxonomy of Neural Network Models

1. Perceptron

    Discrete Time

    Deterministic Computation

    (a) Single Perceptron

    (b) Feedforward Architecture

    i. Binary State

    ii. Analog State

    (c) Recurrent Architecture

    i. Finite Size

    A. Asymmetric Weights

    B. Symmetric Weights

    ii. Infinite Families of Networks

    2. Probabilistic Computation

    (a) Feedforward Architecture

    (b) Recurrent Architecture

    3. Continuous Time

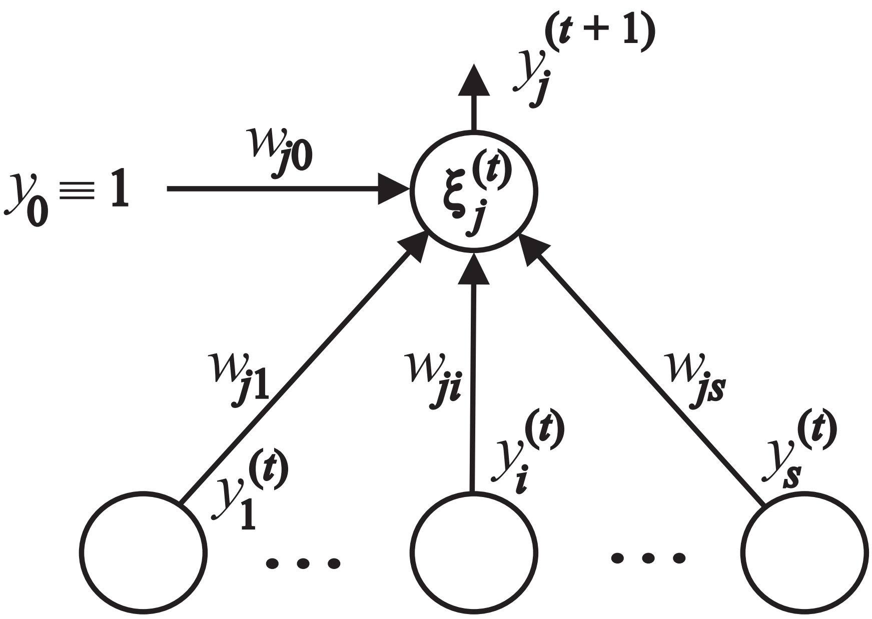
4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

Discrete-Time Perceptron Networks

("perceptron" in a wider sense including sigmoid units)



network updates at discrete time instants $t = 1, 2, \ldots$

at time $t \geq 0$, each perceptron $j \in V$ computes its *excitation*

$$\xi_j^{(t)} = \sum_{i=0}^{s} w_{ji} y_i^{(t)} \qquad j = 1, \ldots, s$$

where w_{j0} is a *bias* $(y_0 \equiv 1)$

at the next time instant $t + 1$, a subset of perceptrons $\alpha_{t+1} \subseteq V$ update their *states (outputs)*

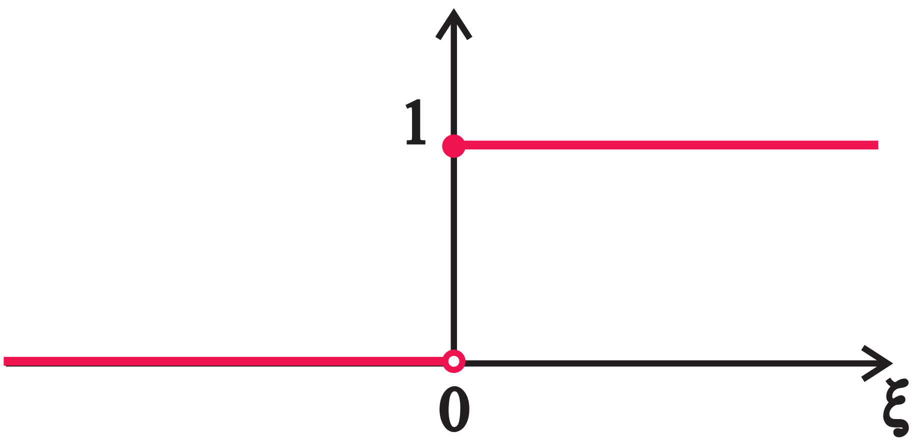
$$y_j^{(t+1)} = \begin{cases} \sigma\left(\xi_j^{(t)}\right) & \text{for } j \in \alpha_{t+1} \\ y_j^{(t)} & \text{for } j \notin \alpha_{t+1} \end{cases}$$

where $\sigma : \Re \longrightarrow \Re$ is an *activation function*

1. **Binary States** $y_j \in \{0, 1\}$ (shortly binary networks)

the threshold gates employ the *Heaviside* activation function

$$\sigma(\xi) = \begin{cases} 1 & \text{for } \xi \geq 0 \\ 0 & \text{for } \xi < 0 \end{cases}$$

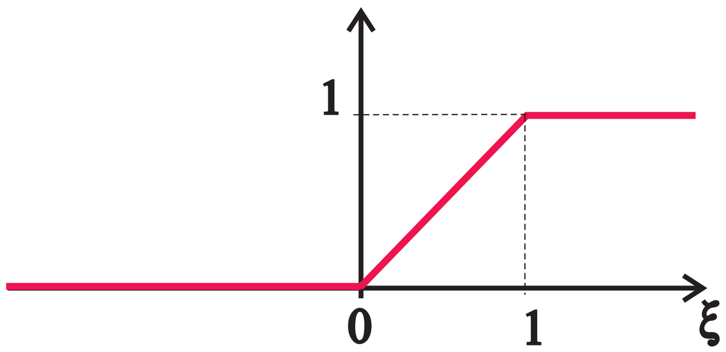


more general discrete domains (e.g. bipolar values $\{-1, 1\}$) can replace the binary values while preserving the size of weights (`Parberry,1990`)

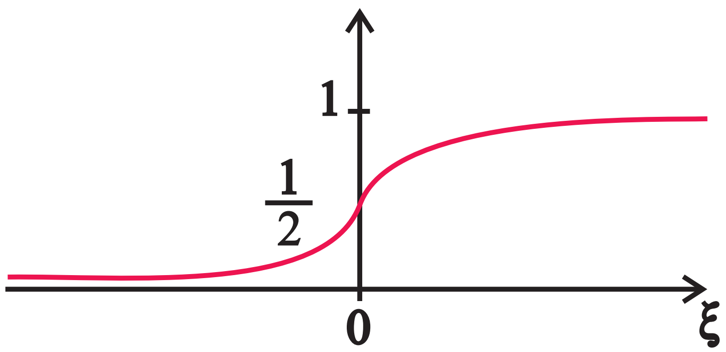
2. **Analog States** $y_j \in [0,1]$ (shortly analog networks)

the sigmoidal gates employ e.g. the *saturated-linear* activation function

$$\sigma(\xi) = \begin{cases} 1 & \text{for } \xi \geq 1 \\ \xi & \text{for } 0 < \xi < 1 \\ 0 & \text{for } \xi \leq 0 \end{cases}$$



or the *logistic sigmoid*: $\sigma(\xi) = \dfrac{1}{1+e^{-\xi}}$



Boolean interpretation of the analog states of output unit j

$$\xi_j = \begin{cases} \leq h - \varepsilon & \text{outputs } 0 \\ \geq h + \varepsilon & \text{outputs } 1 \end{cases}$$

with separation $\varepsilon > 0$, for some fixed threshold $h \in \Re$

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) **Single Perceptron** $\longleftarrow$

   (b) Feedforward Architecture

      i. Binary State

      ii. Analog State

   (c) Recurrent Architecture

      i. Finite Size

         A. Asymmetric Weights

         B. Symmetric Weights

      ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.a Single Perceptron

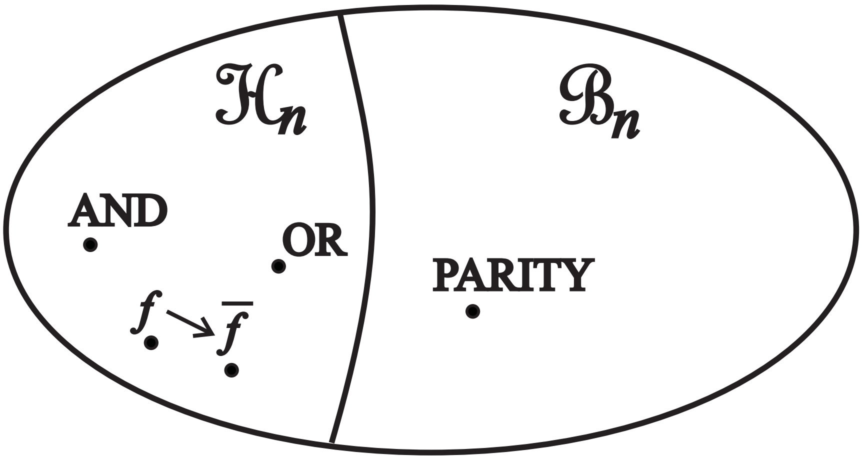
computes an n-variable Boolean function

$$f : \{0,1\}^n \longrightarrow \{0,1\}$$

called *linearly separable* or *linear threshold* function

$\mathcal{H}_n$ is the class of Boolean linear threshold functions over n variables, parametrized by real weights $(w_0, w_1, \ldots, w_n)$

$\mathcal{B}_n$ is the class of all Boolean functions over n variables



- $\mathcal{H}_n$ contains basic logical functions such as AND, OR excluding XOR (PARITY)

- $\mathcal{H}_n$ is closed under negation of both the input variables and/or the output value:

$$f \in \mathcal{H}_n \longrightarrow \bar{f} \in \mathcal{H}_n$$

$$f(x_1, \ldots, x_n) \in \mathcal{H}_n \longrightarrow f(x_1, \ldots, \bar{x}_i, \ldots, x_n) \in \mathcal{H}_n$$

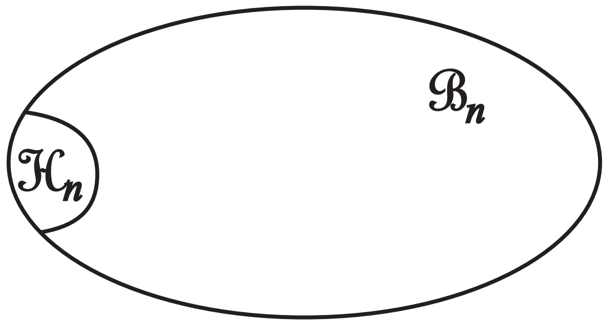
$\longrightarrow$ "De Morgan's law:"

for integer weights: $(w_0, w_1, \ldots, w_n)$ defines $\bar{f}$ iff $(w_0 - 1 - \sum_{i=1}^n w_i \,;\, w_1, \ldots, w_n)$ defines $f(\bar{x}_1, \ldots, \bar{x}_n)$

- the number of n-variable linearly separable functions

  $|\mathcal{H}_n| = 2^{\Theta(n^2)} \ll 2^{2^n} = |\mathcal{B}_n|$ (Cover,1965; Zuev,1989)

  $\longrightarrow \lim_{n\to\infty} \frac{|\mathcal{H}_n|}{|\mathcal{B}_n|} = 0$

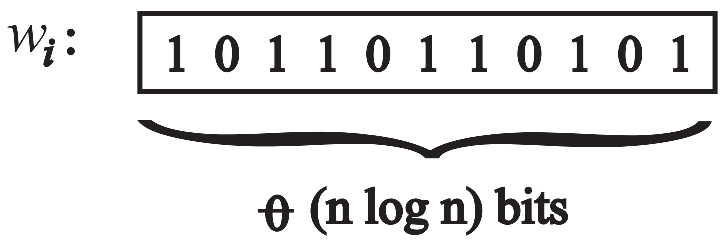


- to decide whether a Boolean function given in (disjunctive or conjunctive normal form) is linearly separable, is coNP-complete problem (Hegedüs,Megiddo,1996)

- any n-variable linearly separable function can be implemented using only *integer* weights (Minsky, Papert,1969), each within the length of

  $$\Theta(n \log n) \text{ bits}$$

  (Muroga et al.,1965; Håstad,1994)



A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) **Feedforward Architecture** $\longleftarrow$

      i. Binary State

      ii. Analog State

   (c) Recurrent Architecture

      i. Finite Size

         A. Asymmetric Weights

         B. Symmetric Weights

      ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

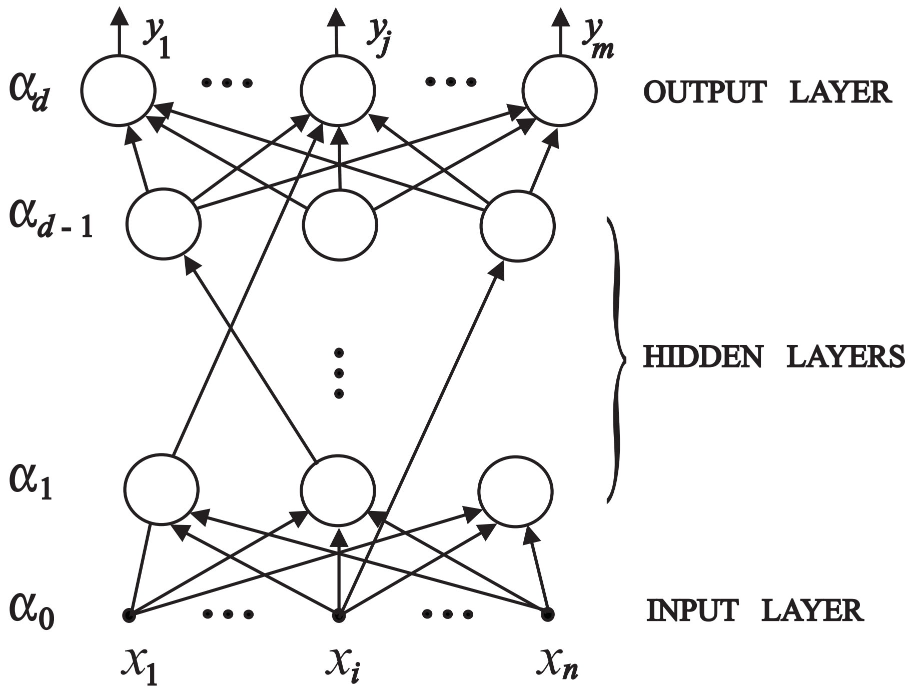
   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.b Feedforward Perceptron Networks:



- *acyclic architecture* $\longrightarrow$ minimal sequence of $d+1$ pairwise disjoint layers $\alpha_0 \cup \alpha_1 \cup \ldots \cup \alpha_d = V$ so that connections from α_t lead only to α_u for $u > t$

  where d is the *depth* of network

  1. *input layer* α_0 contains n external inputs
     (we assume $\alpha_0 \cap V = \emptyset$)
  2. $\alpha_1, \ldots, \alpha_{d-1}$ are *hidden layers*
  3. *output layer* α_d consists of m output units

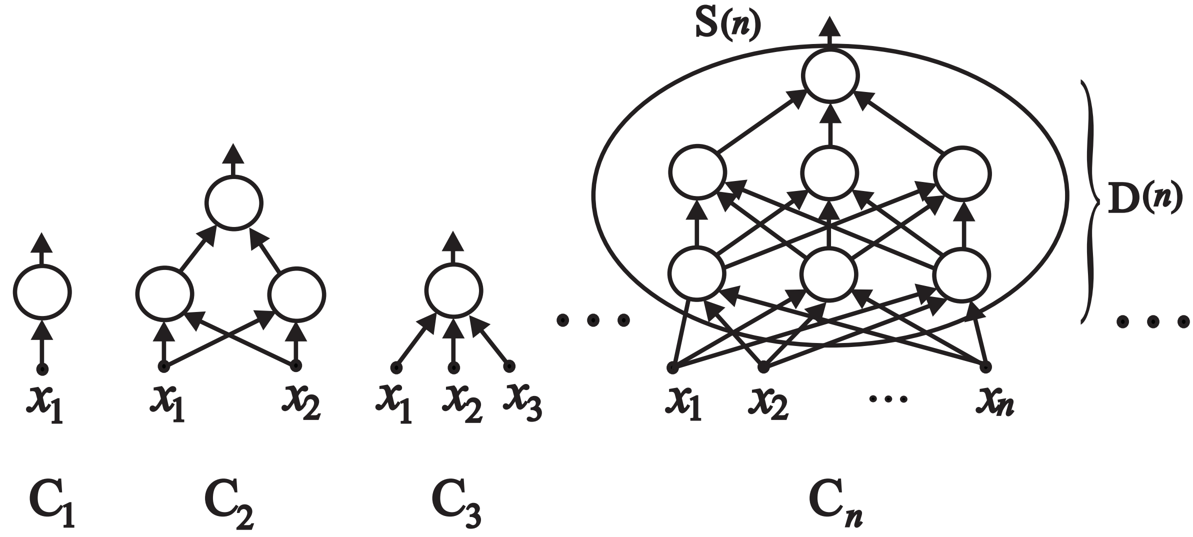
- computation:

  1. initially the states of α_0 represent an external input
  2. computation proceeds layer by layer
  3. the states of α_d represent the result of computation

  $\longrightarrow$ the *network function* $\quad \mathbf{f} : \{0,1\}^n \longrightarrow \{0,1\}^m$
  is evaluated in parallel time d

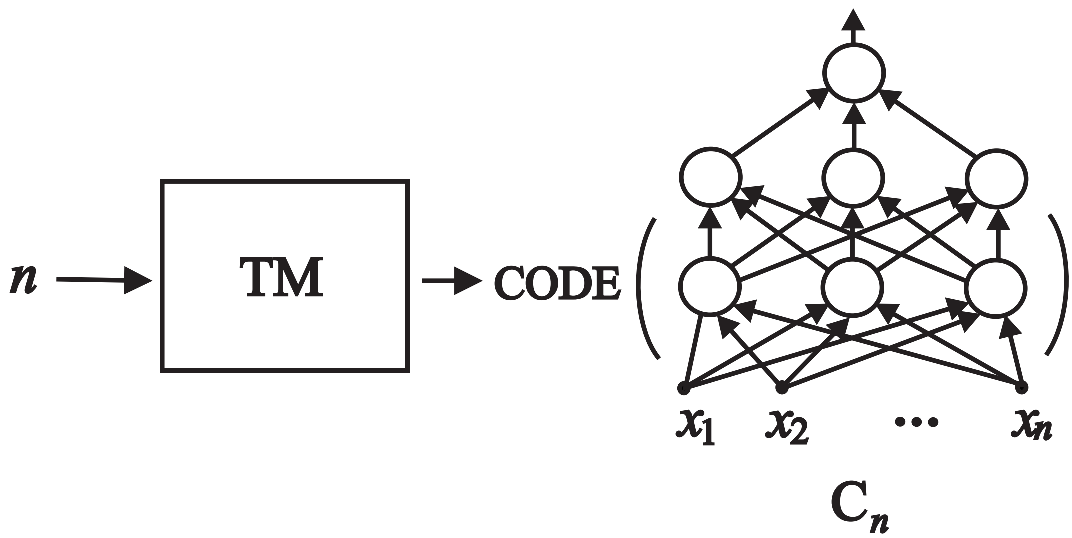
Boolean Threshold Circuits

(*units are called* gates, α_0 *may also include the negations of inputs*)



- for universal computation *infinite families* $\{C_n\}$ of circuits, each C_n for one input length $n \geq 0$
- the size $S(n)$ and depth $D(n)$ are expressed in terms of input length n

uniform $\times$ nonuniform circuit families



A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) **Feedforward Architecture**

   i. **Binary State** $\longleftarrow$

   ii. Analog State

   (c) Recurrent Architecture

   i. Finite Size

   A. Asymmetric Weights

   B. Symmetric Weights

   ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.b.i Binary-State Feedforward Networks:

computational universality: any vector Boolean function

$$\mathbf{f}:\{0,1\}^n \longrightarrow \{0,1\}^m$$

can be implemented using 4-layer perceptron network with

$$\Theta\left(\sqrt{\frac{m2^n}{n-\log m}}\right) \quad \text{neurons} \quad (\texttt{Lupanov,1961})$$

lower bound: most functions require this network size (Horne,Hush,1994) and $\Omega(m2^n/(n-\log m))$ connections (Cover,1968) even for unbounded depth

$\longrightarrow$ *(nonuniform) infinite families of threshold circuits with exponentially many gates are capable of computing any I/O mapping in constant parallel time*

polynomial weights: (each weight only $O(\log n)$ bits) exponential weights can constructively be replaced with polynomial weights (in terms of input length) by increasing the depth by one layer while only a polynomial depth-independent increase in the network size is needed (Goldmann,Karpinski,1998)

$$N(s,d,w_i=O(n^n)) \longmapsto N'(O(s^{c_1}),d{+}1,w_i=O(n^{c_2}))$$

$\longrightarrow$ *polynomial weights can be assumed in multi-layered perceptron networks if one extra parallel computational step is granted*

positive weights: at the cost of doubling the size (Hajnal et al.,1993)

$$N(s,d) \quad \longmapsto \quad N'(2s, d, w_i \geq 0)$$

unbounded fan-in:

(fan-in is the maximum number of inputs to a single unit)

conventional circuit technology with bounded fan-in

$\times$ the dense interconnection of neurons

in feedforward networks yields a speed-up factor of $O(\log\log n)$ (i.e. the depth is reduced by this factor) at the cost of squaring the network size as compared to the classical circuits with fan-in 2 (Chandra et al.,1984)

$$N(s, d, \text{fan-in} \leq 2) \longmapsto N'\left(O(s^2)\,,\ \frac{d}{\log\log n}\right)$$

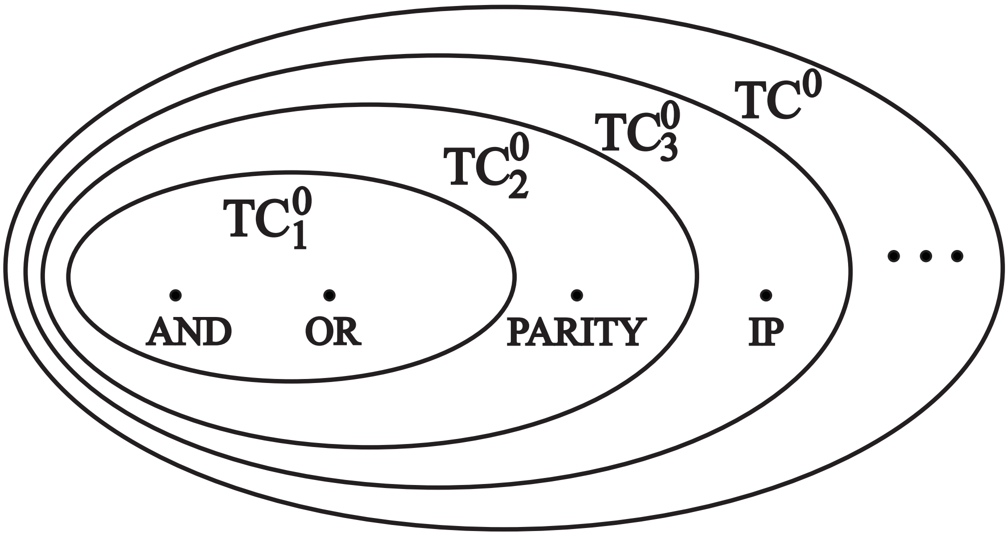
polynomial size & constant depth:

$\mathbf{TC^0_d}$ $(d \geq 1)$ is the class of all functions computable by polynomial-size and polynomial-weight threshold circuits of depth d $\quad(\widehat{LT}_d \times LT_d$ for unbounded weights)

$\longrightarrow$ a possible TC^0 *hierarchy*, $TC^0 = \bigcup_{d\geq 1} TC^0_d$

$$TC^0_1 \subseteq TC^0_2 \subseteq TC^0_3 \subseteq \cdots$$

TC^0 hierarchy



- $TC_1^0 \subsetneqq TC_2^0$: $PARITY(XOR) \in TC_2^0 \setminus TC_1^0$
- $TC_2^0 \subsetneqq TC_3^0$: $IP \in TC_3^0 \setminus TC_2^0$ (Hajnal et al.,1993)

  Boolean inner product $IP : \{0,1\}^{2k} \longrightarrow \{0,1\}, \quad k \geq 1$

  $$IP(x_1, \ldots, x_k, x'_1, \ldots, x'_k) = \bigoplus_{i=1}^{k} AND(x_i, x'_i)$$

  where $\bigoplus$ stands for the k-bit parity function

  $\longrightarrow$ *polynomial-size and polynomial-weight three-layer perceptron networks are computationally more powerful than two-layer ones*

- the separation of the TC^0 hierarchy above depth 3 is *unknown* $\times$ $TC^0 \stackrel{?}{\subseteq} TC_3^0$

  it is still conceivable that e.g. NP-complete problems could be solved by depth-3 threshold circuits with a linear number of gates

symmetric Boolean functions:

- the output value depends only on the number of 1s within the input, e.g. AND, OR, PARITY

- any symmetric function $f : \{0,1\}^n \longrightarrow \{0,1\}$ can be implemented by a polynomial-weight depth-3 threshold circuit of size $O(\sqrt{n})$ (`Siu et al.,1991`)

  lower bound: $\Omega(\sqrt{n/\log n})$ gates even for unbounded depth and weights; $\Omega(\sqrt{n})$ for depth 2 (`O'Neil,1971`)

  $\times$ $PARITY \notin AC^0$, i.e. the parity cannot be computed by polynomial-size constant-depth AND-OR circuits (`Furst et al.,1984`)

  $\longrightarrow$ *perceptron networks are more efficient than AND-OR circuits*

- conjecture $AC^0 \overset{?}{\subseteq} TC^0_3$: AC^0 functions are computable by depth-3 threshold circuits of subexponential size $n^{\log^c n}$, for some constant c (`Allender,1989`)

arithmetic functions:
can be implemented by polynomial-size and polynomial-weight feedforward perceptron networks within small constant depths:

Function	Lower bound	Upper bound
Comparison	2	2
Addition	2	2
Multiple Addition	2	2
Multiplication	3	3
Division	3	3
Powering	2	3
Sorting	3	3
Multiple Multiplication	3	4

any *analytic function* with its real argument represented as an n-bit binary input can be implemented to high precision by a perceptron network of polynomial size and weights, using only a small constant number of layers (`Reif,Tate,1992`)

$\longrightarrow$ *feedforward networks of polynomial size and weights with few layers appear to be very powerful computational devices*

cf. the neurophysiological data indicate that quite complicated functions are computed using only a few layers of brain structure

VLSI implementation model:

- the gates are placed at the intersection points of a 2-dimensional grid (unit distance between adjacent intersection points)
- the gates can be arbitrarily connected in the plane by wires, which may cross
- k-input (threshold) gates as microcircuits with unit evaluation time, each occupying a set of k intersection points of the grid which are connected by an undirected wire in some arbitrary fashion

total wire length: (Legenstein,Maass,2001)

the minimal value of the sum of wire lengths taken over all possible placements of the gates

- different approach to an optimal circuit design, e.g. complete connectivity between two linear-size layers requires a total wire length of $\Omega(n^{2.5})$
- example: simple pattern detection prototype

  $P_{LR}^k : \{0,1\}^{2k} \longrightarrow \{0,1\}, \quad k \geq 2$

  $$P_{LR}^k(x_1, \ldots, x_k, x'_1, \ldots, x'_k) = 1 \text{ iff}$$
  $$\exists 1 \leq i < j \leq k : x_i = x'_j = 1$$

  can be computed by a 2-layer network with $2\log_2 k + 1$ threshold gates and total wire length $O(k \log k)$

Threshold Circuits with Sparse Activity & Energy Complexity:

(Uchizawa,Douglas,Maass,2006)

in artificially designed threshold circuits usually 50% units fire on average during a computation

$\times$ sparse activity in the brain with only about 1% neurons firing

$\longrightarrow$ energy complexity, e.g. *maximal* energy consumption of threshold circuit C

$$EC_{max}(C) = \max \left\{ \sum_{j=1}^{s} y_j(\mathbf{x}) \, ; \, \mathbf{x} \in \{0,1\}^n \right\}$$

the entropy of circuit C:

$$H_Q = - \sum_{\mathbf{a} \in \{0,1\}^s} P\{\mathbf{y} = \mathbf{a}\} \cdot \log P\{\mathbf{y} = \mathbf{a}\}$$

for some given distribution Q of circuit inputs

$$\longrightarrow \qquad H_{max}(C) = \max_Q H_Q(C)$$

- any function computable by polynomial-size threshold circuit C with $H_{max}(C) = O(\log n)$ can be computed by polynomial-size threshold circuit C' of depth 3:

$$C(s = O(n^{c_1}), H_{max}(C) = O(\log n))$$
$$\longmapsto C'(s = O(n^{c_2}), d = 3)$$

- any polynomial-size threshold circuit C with

$$H_{max}(C) = O(\log n)$$

(i.e. satisfied by all common functions) can be replaced by equivalent polynomial-size threshold circuit C' with low energy:

$$C(s = O(n^c), d, H_{max}(C) = O(\log n))$$
$$\longmapsto C'(2^{H_{max}(C)}, s+1, EC_{max}(C') \leq H_{max}(C) + 1)$$

- the construction of low-energy threshold circuits is reminiscent of cortical circuits of biological neurons selecting different pathways in dependence of the stimulus

- low-energy circuits can possibly be useful for future VLSI implementations where energy consumption and heat dissipation become critical factor

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) **Feedforward Architecture**

      i. Binary State

      ii. **Analog State** $\longleftarrow$

   (c) Recurrent Architecture

      i. Finite Size

         A. Asymmetric Weights

         B. Symmetric Weights

      ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.b.ii Analog-State Feedforward Networks:

e.g. the standard sigmoid (backpropagation learning)

can faithfully simulate the binary networks with the same network architecture (Maass et al.,1991)

constant size: analog states may increase efficiency

e.g. the unary squaring function:

$$SQ_k : \{0,1\}^{k^2+k} \longrightarrow \{0,1\}, \quad k \geq 1$$

$$SQ_k(x_1,\ldots,x_k,z_1,\ldots,z_{k^2}) = 1 \text{ iff } (\sum_{i=1}^{k} x_i)^2 \geq \sum_{i=1}^{k^2} z_i$$

- can be computed using only 2 analog units with polynomial weights and separation $\varepsilon = \Omega(1)$
- any binary feedforward networks computing SQ_k requires $\Omega(\log k)$ units even for unbounded depth and weights (DasGupta,Schnitger,1996)

$\longrightarrow$ *the size of feedforward networks can sometimes be reduced by a logarithmic factor when the binary units are replaced by analog ones*

polynomial size:

$\mathbf{TC^0_d}(\sigma)$ $(d \geq 1)$ contains all the functions computable by polynomial-size and polynomial-weight, analog-state feedforward networks with d layers and separation $\varepsilon = \Omega(1)$ employing activation function σ (e.g. the standard sigmoid)

- $TC^0_d(\sigma) = TC^0_d$ for all $d \geq 1$ (`Maass et al.,1991`)
- this computational equivalence of polynomial-size binary and analog networks is valid even for unbounded depth and exponential weights if the depth of the simulating binary network can increase by a constant factor (`DasGupta,Schnitger,1993`)

$$N_{\text{analog}}(s = O(n^{c_1}), d) \longmapsto N_{\text{binary}}(O(s^{c_2}), O(d))$$

- the Boolean functions computable with arbitrary small separation ε by analog feedforward networks of constant depth and polynomial size, having *arbitrary* real weights and employing the *saturated-linear* activation function, belong to TC^0 (`Maass, 1997`)

$$N_{\text{analog-sat-lin}}(s = O(n^{c_1}), d = O(1), w_i \in \Re)$$
$$\longmapsto N_{\text{binary}}(O(n^{c_2}), O(1), w_i = O(n^{c_3}))$$

$\longrightarrow$ *for digital computations, the analog polynomial-size feedforward networks are equivalent to binary ones*

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

   i. Binary State

   ii. Analog State

   (c) **Recurrent Architecture** ⟵

   i. Finite Size

   A. Asymmetric Weights

   B. Symmetric Weights

   ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.c Recurrent Perceptron Networks:

the architecture is a *cyclic* graph

symmetric (Hopfield) networks

$$w_{ij} = w_{ji} \text{ for all } i, j \in V$$

$\longrightarrow$ *undirected* architectures

computational modes:

according to the choice of sets α_t of updated units

- *sequential* mode: $(\forall\, t \geq 1)\ |\alpha_t| \leq 1$
- *parallel* mode: $(\exists\, t \geq 1)\ |\alpha_t| \geq 2$
- *fully parallel* mode: $(\forall\, t \geq 1)\ \alpha_t = V$
- *productive* computation of length $t^\star$ updates:

$$(\forall\, 1 \leq t \leq t^\star)\ (\exists\, j \in \alpha_t)\ y_j^{(t)} \neq y_j^{(t-1)}$$

- *systematic* computation:

$$\text{e.g.} \quad \alpha_{\tau s + j} = \{j\} \quad \text{for } j = 1, \ldots, s$$

  $\tau = 0, 1, 2, \ldots$ is a *macroscopic time* during which all the units in the network are updated at least once

- *synchronous* computation: α_t are predestined deterministically and centrally for each $t \geq 1$

- *asynchronous* computation: a random choice of α_t, i.e. each unit decides independently when its state is updated

  asynchronous binary (asymmetric or symmetric) networks can always be (systematically) synchronized in sequential or parallel mode (`Orponen,1997`)

convergence

a productive computation *terminates, converges, reaches a stable state* $\mathbf{y}^{(t^\star)}$ at time $t^\star \geq 0$ if

$$\mathbf{y}^{(t^\star)} = \mathbf{y}^{(t^\star+k)} \quad \text{for all } k \geq 1$$

(or for analog networks, at least $\|\mathbf{y}^{(t^\star)} - \mathbf{y}^{(t^\star+k)}\| \leq \varepsilon$ holds for some small constant $0 \leq \varepsilon < 1$)

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

      i. Binary State

      ii. Analog State

   (c) **Recurrent Architecture**

      i. **Finite Size** $\longleftarrow$

         A. Asymmetric Weights

         B. Symmetric Weights

      ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.c.i Finite Recurrent Networks:

as *neural acceptors* of languages $L \subseteq \{0,1\}^\star$ working under fully parallel updates

an input string $\mathbf{x} = x_1 \ldots x_n \in \{0,1\}^n$ of any length $n \geq 0$ is presented bit by bit via an input unit $inp \in V$

$\longrightarrow$ the output unit $out \in V$ signals whether $\mathbf{x} \overset{?}{\in} L$

1. **Binary Networks:**

   $$y_{inp}^{(p(i-1))} = x_i \quad \text{for } i = 1, \ldots, n \text{ with a } \textit{period } p \geq 1$$

   $$\longrightarrow y_{out}^{(p(i-1)+k+1)} = 1 \text{ iff } x_1 \ldots x_i \in L$$

   with some *time delay* $k \geq 1$

   constant time delays k can be reduced to 1 with only a linear-size increase (`Šíma,Wiedermann,1998`)

2. **Analog Networks:** (`Siegelmann,Sontag,1995`)

   (a) *Validation* $ival, oval \in V \quad (p = 1,\ t^\star = T(n))$

   $$y_{inp}^{(t-1)} = x_t \qquad y_{ival}^{(t)} = \begin{cases} 1 & \text{for } t = 0, \ldots, n-1 \\ 0 & \text{for } t \geq n \end{cases}$$

   $$y_{out}^{(t^\star)} = 1 \text{ iff } \mathbf{x} \in L \qquad y_{oval}^{(t)} = \begin{cases} 1 & \text{for } t = t^\star \\ 0 & \text{for } t \neq t^\star \end{cases}$$

   (b) *Initial State*, e.g.

   $$y_{inp}^{(0)} = \sum_{i=1}^{n} \frac{2x_i + 1}{4^i}$$

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

      i. Binary State

      ii. Analog State

   (c) **Recurrent Architecture**

      i. **Finite Size**

         A. **Asymmetric Weights** $\longleftarrow$

         B. Symmetric Weights

      ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.c.i.A Finite Asymmetric Networks:

assume the saturated-linear activation function
(unless explicitly stated otherwise)

the computational power depends on the information contents (Kolmogorov complexity) of real weights

1. **Integer Weights:**

   binary networks $\equiv$ finite automata (Kleene, 1956)

the size of neural network implementation:

- a deterministic finite automaton with q states
  $\longrightarrow O(\sqrt{q})$ units with a period of $p = 4$ of presenting the input bits

  lower bound: $\Omega(\sqrt{q})$ for polynomial weights (Indyk, 1995) or for $p = O(\log q)$ (Horne, Hush, 1996)

- regular expression of length ℓ
  $\longrightarrow \Theta(\ell)$ units (Šíma, Wiedermann, 1998)

2. **Rational Weights:**

analog networks $\equiv$ Turing machine

(step by step simulation)

$\longrightarrow$ any function computable by a Turing machine in time $T(n)$ can be computed by a fixed universal analog network of size:

- 886 units in time $O(T(n))$ (`Siegelmann,Sontag,1995`)
- 25 units in time $O(n^2T(n))$ (`Indyk,1995`)

$\longrightarrow$ polynomial-time computations by analog networks correspond to the complexity class P

Turing universality for more general classes of sigmoid activation functions (`Koiran,1996`) including the standard sigmoid (`Kilian,Siegelmann,1996`) but the known simulations require exponential time overhead per each computational step

3. **Arbitrary Real Weights:**

super-Turing computational capabilities
(Siegelmann,Sontag,1994)

finite analog neural networks working within time $T(n)$
$$\equiv$$
infinite nonuniform families of threshold circuits of
size $S(n) = O(poly(T(n)))$

- *polynomial-time* computations: the nonuniform complexity class P/poly

  P/poly: polynomial-size (nonrecursive) advice (the same for one input length) is granted to TMs working in polynomial time

  (which is the threshold circuit for a given input length)

- *exponential-time* computations: implement any I/O mapping

- polynomial time + increasing Kolmogorov complexity of real weights: a proper *hierarchy* of nonuniform complexity classes between P and P/poly
(Balcázar et al.,1997)

Analog Noise:

$\times$ the preceding results for analog computations assume arbitrary-precision real number calculations

analog noise reduces the computational power of analog networks to at most that of finite automata

- bounded noise: faithful simulation of binary networks $\equiv$ finite automata (`Siegelmann,1996`)

- unbounded noise: unable to recognize all regular languages (*definite languages*) (`Maass,Orponen,1998`)

The Complexity of Related Problems:

- the issue of deciding whether there exists a stable state in a given binary network is NP-complete (Alon,1987)

- **Halting Problem** of deciding whether a recurrent network terminates its computation over a given input

  - PSPACE-complete for binary networks (Floréen, Orponen,1994)

  - algorithmically undecidable for analog nets with rational weights and only 25 units (Indyk,1995)

- the computations of recurrent networks of size s that terminate within time $t^\star$ can be "unwound" into feed-forward networks of size $st^\star$ and depth $t^\star$ (Savage,1972)

$$N_{\text{recurrent}}(s, t^\star) \longmapsto N_{\text{feedforward}}(s \cdot t^\star, d = t^\star)$$

$\longrightarrow$ *feedforward and* convergent *recurrent networks are computationally equivalent up to a factor of $t^\star$ in size* (Goldschlager,Parberry,1986)

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

      i. Binary State

      ii. Analog State

   (c) **Recurrent Architecture**

      i. **Finite Size**

         A. Asymmetric Weights

         B. **Symmetric Weights** $\longleftarrow$

      ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.c.i.B Finite Symmetric (Hopfield) Networks:

Convergence:

a bounded *energy (Liapunov)* function E defined on the state space of the *symmetric* network decreases along any productive computation

$\longrightarrow$ the Hopfield network converges towards some stable state corresponding to a local minimum of E

1. *Binary Symmetric Networks:*

- Sequential Mode: (Hopfield,1982)
  semisimple networks $w_{jj} \geq 0$ for all $j \in V$

$$E(\mathbf{y}) = -\sum_{j=1}^{s} y_j \left(w_{j0} + \frac{1}{2} \sum_{i=1;i\neq j}^{s} w_{ji} y_i + w_{jj} y_j \right)$$

- Parallel Mode: the networks either converge (e.g. when E is negative definite, Goles-Chacc et al.,1985), or eventually alternate between two different states (Poljak,Sůra,1983)

2. *Analog Symmetric Networks:* converge to a fixed point or to a limit cycle of length at most 2 for parallel updates (Fogelman-Soulié et al.,1989; Koiran,1994)

$$E'(\mathbf{y}) = E(\mathbf{y}) + \sum_{j=1}^{s} \int_0^{y_j} \sigma^{-1}(y)dy$$

Convergence Time: the number of discrete-time updates before the (binary) network converges

- trivial *upper bound:* 2^s different network states

- *lower bound:* a symmetric *binary counter* converging after $\Omega(2^{s/8})$ asynchronous seq. updates (`Haken,1989`) or $\Omega(2^{s/3})$ fully parallel steps (`Goles,Martínez,1989`)

- *average-case:* convergence of only $O(\log \log s)$ parallel update steps under reasonable assumptions (`Komlós,Paturi,1988`)

- obvious upper bound of $O(W)$ in terms of the *total weight* $W = \sum_{j,i \in V} |w_{ji}|$

  $\longrightarrow$ *polynomial-time convergence for binary symmetric networks with polynomial weights*

- $2^{\Omega(M^{1/3})}$-lower and $2^{O(M^{1/2})}$-upper bounds where M is the number of bits in the *binary representation of weights* (`Šíma et al.,2000`)

- lower bound of $2^{\Omega(g(M))}$for *analog Hopfield nets* where $g(M)$ is an arbitrary continuous function such that $g(M) = \Omega(M^{2/3})$, $g(M) = o(M)$ (`Šíma et al.,2000`)

  $\longrightarrow$ *an example of the analog Hopfield net converging later than any other binary symmetric network of the same representation size*

Stable States $=$ patterns stored in associative memory

- the average number of stable states: a binary Hopfield net of size s whose weights are independent identically distributed zero-mean Gaussian random variables has on the average asymptotically

  $$1.05 \times 2^{0.2874s} \text{ many stable states}$$

  (`McEliece et al.,1987; Tanaka,Edwards,1980`)

- counting the number of stable states: the issue of deciding whether there are at least one ($w_{jj} < 0$), two, or three stable states in a given Hopfield network, is NP-complete

  the problem of determining the exact number of stable states for a given Hopfield net is $\#$P-complete (`Floréen,Orponen,1989`)

- *attraction radius:* the issue of how many bits may be flipped in a given *stable* state so that the Hopfield net still converges back to it, is NP-hard (`Floréen,Orponen,1993`)

MIN ENERGY Problem:

the issue of finding a state of a given Hopfield net with energy less than a prescribed value

$\longrightarrow$ the fast approximate solution of combinatorial optimization problems, e.g. Traveling Salesman Problem (Hopfield,Tank,1985)

- *NP-complete* for both *binary* (Barahona,1982) and *analog* (Šíma et al.,2000) Hopfield nets

- *polynomial-time solvable* for binary Hopfield nets whose architectures are *planar lattices* (Bieche et al.,1980) or *planar graphs* (Barahona,1982)

- *polynomial-time approximation* to within absolute error of less than $0.243W$ in binary Hopfield nets of total weight W (Šíma et al.,2000)

  for $W = O(s^2)$ (e.g. constant weights), this matches the *lower bound* $\Omega(s^{2-\varepsilon})$ which is not guaranteed by any approximate polynomial-time MIN ENERGY algorithm for every $\varepsilon > 0$ unless P=NP (Bertoni,Campadelli,1994)

The Computational Power of Hopfield Nets:

- tight converse to Hopfield's convergence theorem for binary networks: *symmetric* networks can simulate arbitrary *convergent asymmetric* networks with only a linear overhead in time and size (`Šíma et al.,2000`)

$$N_{\text{convergent}}(s,t^*) \longmapsto N_{\text{symmetric}}(O(s),O(t^*))$$
$$\longrightarrow \textit{convergence} \equiv \textit{symmetry}$$

- *binary* symmetric neural acceptors recognize a strict subclass of the regular languages called *Hopfield languages* (`Šíma,1995`)

- *analog* symmetric neural acceptors faithfully recognize Hopfield languages (`Šíma,1997`)

- Turing machines $\equiv$ analog asymmetric networks
  $\equiv$ analog symmetric networks $+$ external oscillator

  external oscillator: produces an arbitrary infinite binary sequence containing infinitely many 3-bit substrings of the form $bx\bar{b} \in \{0,1\}^3$ where $b \neq \bar{b}$
  (`Šíma et al.,2000`)

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

      i. Binary State

      ii. Analog State

   (c) **Recurrent Architecture**

      i. Finite Size

         A. Asymmetric Weights

         B. Symmetric Weights

      ii. **Infinite Families of Networks** $\longleftarrow$

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

1.c.ii Infinite Families of Binary Networks $\{N_n\}$:

- *alternative input protocol:*
  one N_n for each input length $n \geq 0$

- *recognition of a language* $L \subseteq \{0,1\}^\star$:
  an input $\mathbf{x} \in \{0,1\}^n$ is presented to the network N_n, a single output neuron out is read after N_n converges in time $t^\star$:
  $$y_{out}^{(t^\star)} = 1 \text{ iff } \mathbf{x} \in L$$

- the *size* $S(n)$ of $\{N_n\}$ is defined as a function of n

- *polynomial-size* families of binary recurrent networks ($S(n) = O(n^c)$) recognize exactly the languages in the complexity class PSPACE/poly (`Lepley,Miller,1983`)

  `Orponen,1996`:

  – symmetric weights: PSPACE/poly

  – polynomial symmetric weights: P/poly

  $\longrightarrow$ polynomial-size infinite families of binary symmetric networks with polynomial integer weights

  $$\equiv$$

  polynomial-time finite analog asymmetric networks with arbitrary real weights

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

   i. Binary State

   ii. Analog State

   (c) Recurrent Architecture

   i. Finite Size

   A. Asymmetric Weights

   B. Symmetric Weights

   ii. Infinite Families of Networks

   2. **Probabilistic Computation** $\longleftarrow$

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

2 Probabilistic Perceptron Networks:

- a deterministic discrete-time perceptron network is augmented with *additional random binary input units* $i \in \Pi$ with fixed real probabilities $0 \leq p_i \leq 1$:

$$\forall\, t \geq 0 \quad P\{y_i^{(t)} = 1\} = p_i \quad \text{for } i \in \Pi$$

$$\left(\longrightarrow \forall\, t \geq 0 \quad P\{y_i^{(t)} = 0\} = 1 - p_i \quad \text{for } i \in \Pi \right)$$

- the *reference model* that is polynomially (in parameters) related to neural networks with other stochastic behavior, e.g. unreliable in computing states and connecting units (`von Neumann,1956; Siegelmann,1999`); Boltzmann machines (`Parberry,Schnitger,1989`), etc.

- a language $L \subseteq \{0,1\}^n$ is *ε-recognized* $(0 < \varepsilon < \frac{1}{2})$ if for any input $\mathbf{x} \in \{0,1\}^n$ the probability that the network outputs 1 satisfies:

$$P\{y_{out} = 1\} \begin{cases} \geq 1 - \varepsilon & \text{if } \mathbf{x} \in L \\ \leq \varepsilon & \text{if } \mathbf{x} \notin L \end{cases}$$

  this symmetry in the probability of errors ε in accepting and rejecting an input can always be achieved by adding random input units (`Hajnal et al.,1993`)

A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

      i. Binary State

      ii. Analog State

   (c) Recurrent Architecture

      i. Finite Size

         A. Asymmetric Weights

         B. Symmetric Weights

      ii. Infinite Families of Networks

   2. **Probabilistic Computation**

   (a) **Feedforward Architecture** ⟵

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

2.a Probabilistic Binary Feedforward Networks:

- *increasing the reliability:* any language $L \subseteq \{0,1\}^n$ that is ε-recognized $(0 < \varepsilon < 1/2)$ can be λ-recognized for any $0 < \lambda < \varepsilon$ if one extra layer is granted:

$$N_\varepsilon(s,d) \longmapsto N_\lambda\left(2s \cdot \left\lceil \frac{\ln \lambda}{\ln 4\varepsilon(1-\varepsilon)} \right\rceil + 1, d+1\right)$$

- *deterministic simulation:* $1/4 < \varepsilon < 1/2$

$$N_\varepsilon(s,d) \longmapsto N_{\mathsf{det}}\left(\left\lceil \frac{8\varepsilon \ln 2}{(1-2\varepsilon)^2} + 1 \right\rceil ns + 1, d+1\right)$$

(Parberry,Schnitger,1989)

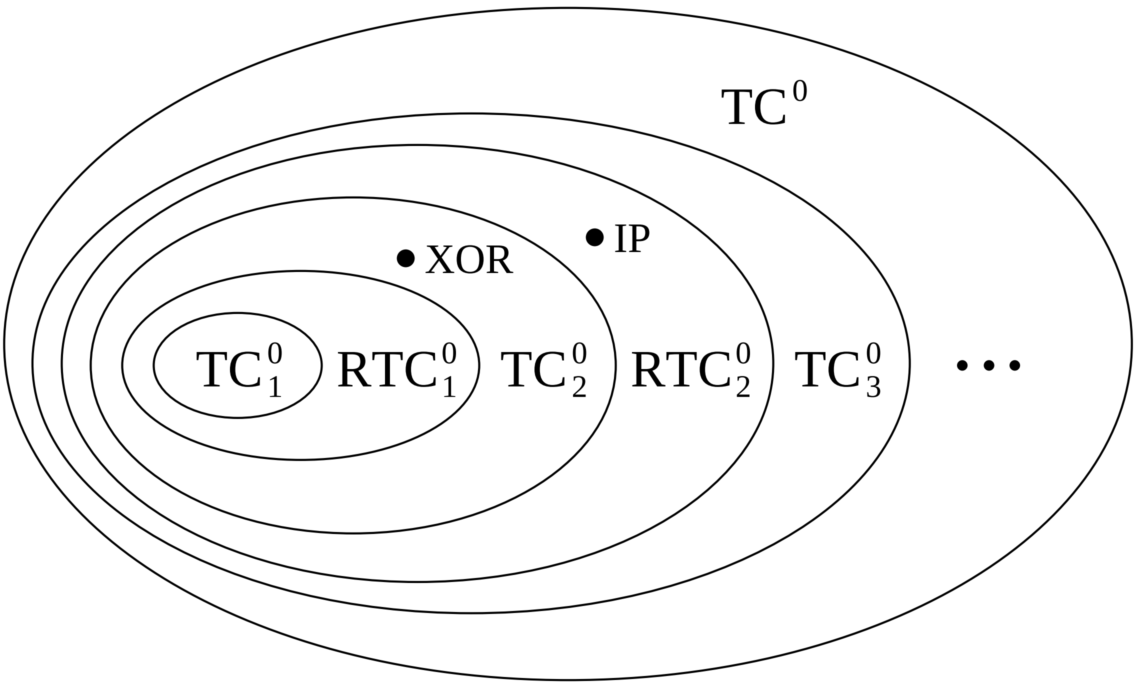
- $\mathbf{RTC^0_d}$ $(d \geq 1)$ is the class of all languages $\varepsilon(n)$-recognized by the families of *polynomial-size* and *polynomial-weight* probabilistic threshold circuits of depth d with the probabilities of errors $\varepsilon(n) = \frac{1}{2} - \frac{1}{n^{O(1)}}$

- `Hajnal et al.,1993:`

$$\mathbf{IP} \in \mathbf{RTC^0_2} \quad (IP \notin TC^0_2)$$

$$\forall\, \mathbf{d} \geq \mathbf{1}\ \ \mathbf{RTC^0_d} \subseteq \mathbf{TC^0_{d+1}} \quad \text{(non-uniformly)}$$

$\longrightarrow$ *at most one layer can be saved by introducing stochasticity in threshold circuits*



A Taxonomy of Neural Network Models

1. **Perceptron**

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

   i. Binary State

   ii. Analog State

   (c) Recurrent Architecture

   i. Finite Size

   A. Asymmetric Weights

   B. Symmetric Weights

   ii. Infinite Families of Networks

   2. **Probabilistic Computation**

   (a) Feedforward Architecture

   (b) **Recurrent Architecture** $\longleftarrow$

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

2.b Probabilistic Analog Recurrent Networks

with the saturated-linear activation function (Siegelmann,1999)

weights	deterministic networks		probabilistic networks	
	unbounded time	polynomial time	unbounded time	polynomial time
integer	regular	regular	regular	regular
rational	recursive	P	recursive	BPP
real	arbitrary	P/poly	arbitrary	P/poly

1. *integer weights:* the *binary-state* probabilistic networks ε-recognize the *regular languages*

2. *rational parameters:* analog probabilistic networks can in *polynomial time* ε-recognize exactly the languages from the complexity class **BPP** (polynomial-time bounded-error probabilistic Turing machines)

   or nonuniform **Pref-BPP/log** for rational weights and arbitrary real probabilities

3. *arbitrary real weights:* *polynomial-time* computations correspond to the complexity class **P/poly**

$\longrightarrow$ *stochasticity plays a similar role in neural networks as in conventional Turing computations*

A Taxonomy of Neural Network Models

1. **Perceptron**

    Discrete Time

    Deterministic Computation

    (a) Single Perceptron

    (b) Feedforward Architecture

    i. Binary State

    ii. Analog State

    (c) Recurrent Architecture

    i. Finite Size

    A. Asymmetric Weights

    B. Symmetric Weights

    ii. Infinite Families of Networks

    2. Probabilistic Computation

    (a) Feedforward Architecture

    (b) Recurrent Architecture

    3. **Continuous Time** $\longleftarrow$

4. RBF Unit

5. Winner-Take-All Unit

6. Spiking Neuron

3 Continuous-Time Perceptron Networks:

- the dynamics of the analog state $\mathbf{y}(t) \in [0,1]^s$ is defined for every *real time* instant $t > 0$ as the solution of a system of s *differential equations*:

  $$\frac{dy_j}{dt}(t) = -y_j(t) + \sigma\left(\sum_{i=0}^{s} w_{ji} y_i(t)\right) \qquad j = 1, \ldots, s$$

  with the *boundary conditions* given by $\mathbf{y}(0)$
  e.g. σ is the saturated-linear activation function

- *symmetric networks* $(w_{ji} = w_{ij})$: Liapunov function

  $$E(\mathbf{y}) = -\sum_{j=1}^{s} y_j \left(w_{j0} + \frac{1}{2}\sum_{i=1}^{s} w_{ji} y_i\right) + \sum_{j=1}^{s} \int_0^{y_j} \sigma^{-1}(y)dy$$

  $\longrightarrow$ *converge* from any initial state $\mathbf{y}(0)$ to some *stable state* satisfying $dy_j/dt = 0$ for all $j = 1, \ldots, s$ (`Cohen,Grossberg,1983`)

  which may take an *exponential time* in terms of s (`Šíma,Orponen,2003`)

- simulation of *finite binary-state discrete-time networks* by *asymmetric* (`Orponen,1997`) and *symmetric* (`Šíma, Orponen,2003`) continuous-time networks:

  $$N_{\text{discrete}}(s, T(n)) \longmapsto N_{\text{continuous}}(O(s), O(T(n)))$$

  $\longrightarrow$ *polynomial-size families of continuous-time (symmetric) networks recognize at least PSPACE/poly*

A Taxonomy of Neural Network Models

1. Perceptron

    Discrete Time

    Deterministic Computation

    (a) Single Perceptron

    (b) Feedforward Architecture

    i. Binary State

    ii. Analog State

    (c) Recurrent Architecture

    i. Finite Size

    A. Asymmetric Weights

    B. Symmetric Weights

    ii. Infinite Families of Networks

    2. Probabilistic Computation

    (a) Feedforward Architecture

    (b) Recurrent Architecture

    3. Continuous Time

4. **RBF Unit** $\longleftarrow$

5. Winner-Take-All Unit

6. Spiking Neuron

4 RBF Networks:

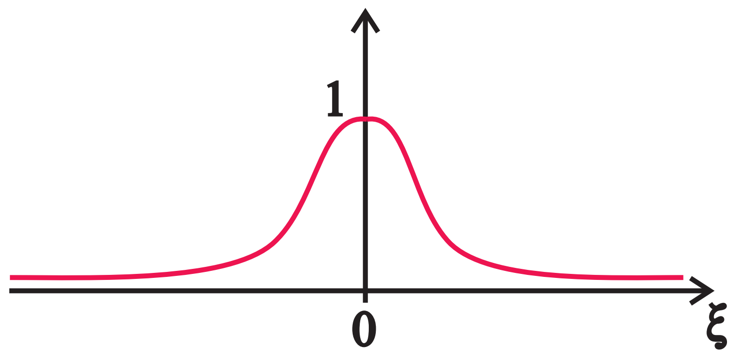
the units compute *Radial Basis Functions:*

$$\text{"excitation"} \quad \xi_j^{(t)} = \frac{\left\| \mathbf{x}_j^{(t)} - \mathbf{w}_j \right\|}{w_{j0}} > 0 \quad \text{of unit } j \in V$$

where $\mathbf{x}_j^{(t)}$ is the *input* from the incident units, the "weight" vector $\mathbf{w}_j$ represent a *center*, and a "bias" $w_{j0} > 0$ determines the *width*

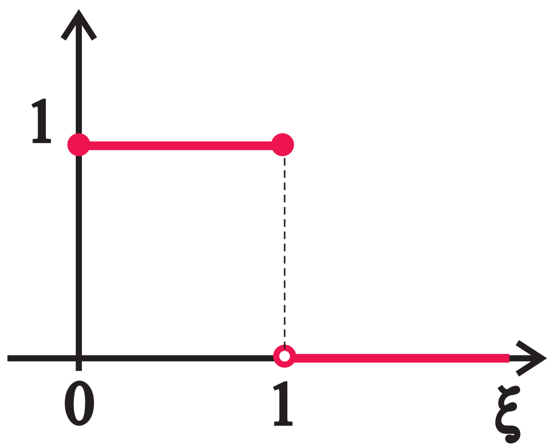
$$\longrightarrow \quad \text{output} \quad y_j^{(t+1)} = \sigma\left(\xi_j^{(t)}\right)$$

e.g. the *Gaussian* activation function $\sigma(\xi) = \mathbf{e}^{-\xi^2}$



or the binary activation function

$$\sigma(\xi) = \begin{cases} 1 & \text{if } 0 \leq \xi \leq 1 \\ 0 & \text{if } \xi > 1 \end{cases}$$



the computational power of RBF networks:

- binary RBF units with the Euclidean norm compute exactly the Boolean linear threshold functions (Friedrichs,Schmitt,2005), i.e.

  binary RBF unit $\equiv$ perceptron

- digital computations using analog RBF unit: two different analog states of RBF units represent the binary values 0 and 1

- an analog RBF unit with the maximum norm can implement the universal Boolean *NAND gate* over multiple literals (input variables or their negations)

  $\longrightarrow$ a *deterministic finite automaton* with q states can be simulated by a recurrent network with $O(\sqrt{q \log q})$ RBF units in a robust way (Šorel,Šíma,2000)

- the *Turing universality* of finite RBF networks is still an open problem

A Taxonomy of Neural Network Models

1. Perceptron

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

   i. Binary State

   ii. Analog State

   (c) Recurrent Architecture

   i. Finite Size

   A. Asymmetric Weights

   B. Symmetric Weights

   ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. **Winner-Take-All Unit** $\longleftarrow$

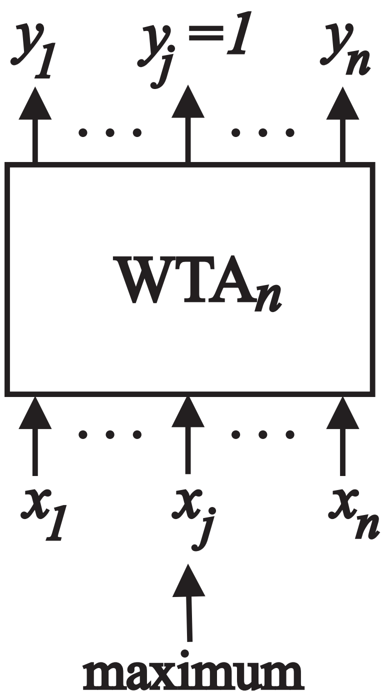
6. Spiking Neuron

5 Winner-Take-All (WTA) Networks:

- competitive principle (e.g. Kohonen networks)
- efficient analog VLSI implementations
- the units compute $k\text{-}WTA_n : \Re^n \longrightarrow \{0,1\}^n$ defined as

$$y_j = 1 \quad \text{iff} \quad |\{i\,;\, x_i > x_j,\, 1 \leq i \leq n\}| \leq k - 1$$

e.g. a WTA_n gate ($k = 1$) indicates which of the n inputs has *maximal* value



- a WTA_n device ($k = 1$, $n \geq 3$) cannot be implemented by any perceptron network having fewer than sufficient $\binom{n}{2} + n$ threshold gates (Maass,2000)

- any Boolean function from TC_0^2 can be computed by a single k-WTA_r gate applied to $r = O(n^c)$ (for some constant c) weighted sums of n input variables with polynomial natural weights (Maass,2000)

- P_{LR}^k (recall $P_{LR}^k(x_1, \ldots, x_k, x'_1, \ldots, x'_k) = 1$ iff $\exists 1 \leq i < j \leq k : x_i = x'_j = 1$) can be computed by a two-layered network consisting of only 2 WTA units (with weighted inputs) and 1 threshold gate, whose total wire length reduces to $\mathbf{O(k)}$ as compared to $O(k \log k)$ perceptrons (Legenstein,Maass,2001)

$\longrightarrow$ *the winner-take-all gates are more efficient than the perceptrons*

A Taxonomy of Neural Network Models

1. Perceptron

   Discrete Time

   Deterministic Computation

   (a) Single Perceptron

   (b) Feedforward Architecture

   i. Binary State

   ii. Analog State

   (c) Recurrent Architecture

   i. Finite Size

   A. Asymmetric Weights

   B. Symmetric Weights

   ii. Infinite Families of Networks

   2. Probabilistic Computation

   (a) Feedforward Architecture

   (b) Recurrent Architecture

   3. Continuous Time

4. RBF Unit

5. Winner-Take-All Unit

6. **Spiking Neuron** $\longleftarrow$

6 Networks of Spiking Neurons:

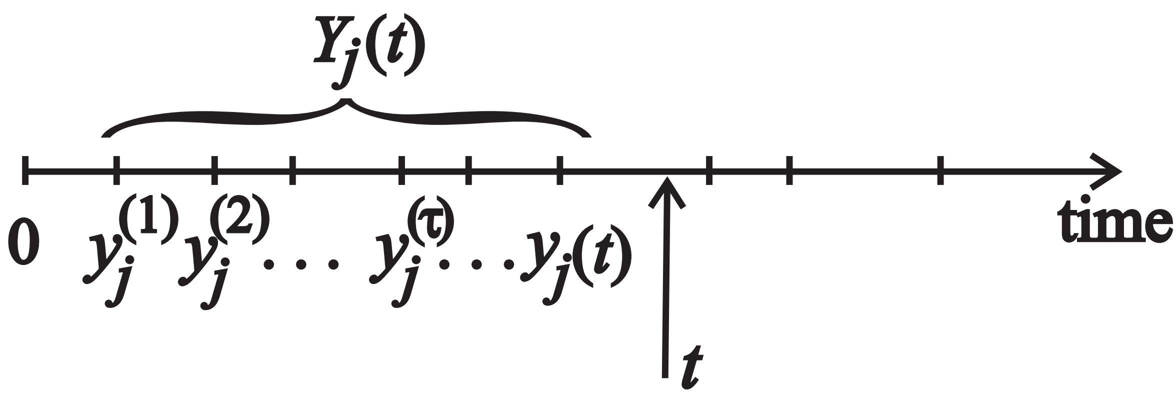
(Artificial Pulsed Neural Networks, Spiking Networks)

- biologically plausible: the states are encoded as *temporal* differences between neuron *spikes (firing times)*, e.g. an input binary string is presented bit after bit by firing or nonfiring within given time intervals

- $0 \leq y_j^{(1)} < y_j^{(2)} < \cdots < y_j^{(\tau)} < \ldots$ a *sequence of firing times* of spiking neuron $j \in V$

  $Y_j(t) = \{y_j^{(\tau)} < t;\, \tau \geq 1\}$ the *set of spikes* of unit j preceding a continuous time instant $t \geq 0$

  $y_j(t) = \max Y_j(t)$ the *last firing time* (for $Y_j(t) \neq \emptyset$)



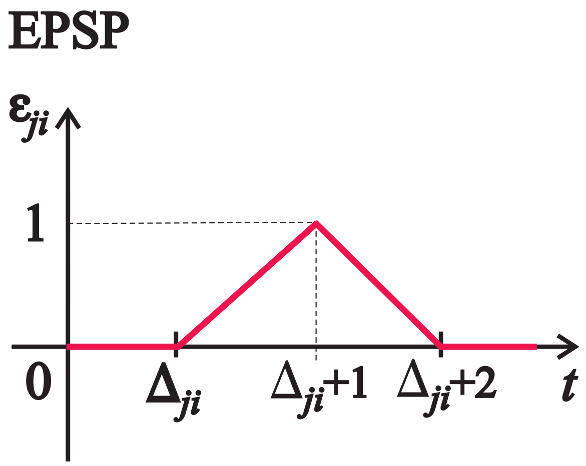
- the *next spike* of neuron j:

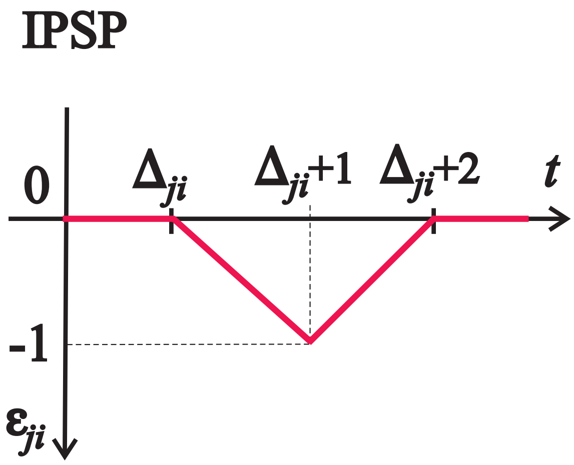
$$y_j^{(\tau)} = \inf \left\{ t > y_j^{(\tau-1)} \,;\, \xi_j(t) \geq 0 \right\}$$

  where excitation

$$\xi_j(t) = w_{j0}(t - y_j(t)) + \sum_{i=1}^{s} \sum_{y \in Y_i(t)} w_{ji} \cdot \varepsilon_{ji}(t - y)$$

- *response function* $\varepsilon_{ji} : \Re_0^+ \longrightarrow \Re$ of unit j to presynaptic spikes from i in time $t \geq 0$ models either *excitatory* (EPSP) $\varepsilon_{ji} \geq 0$ or *inhibitory* (IPSP) $\varepsilon_{ji} \leq 0$ postsynaptic potential, e.g. with a *delay* Δ_{ji}:

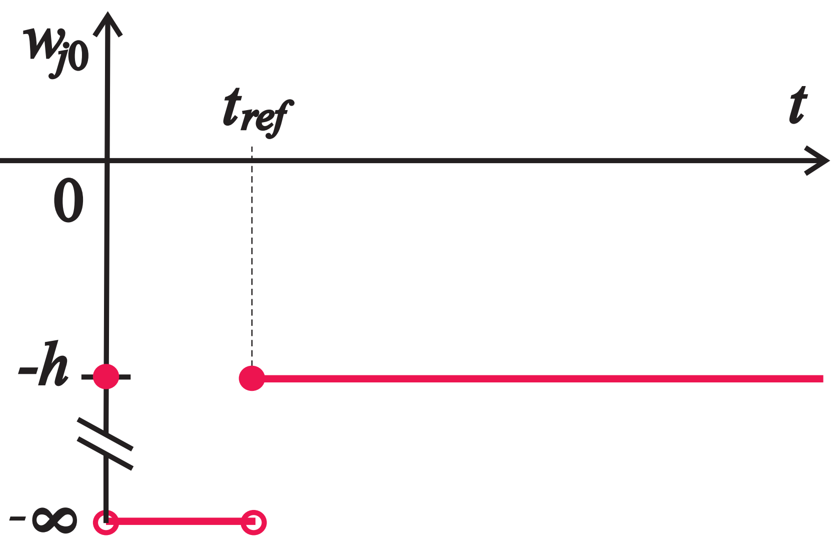




- $w_{ji} \geq 0$ for all $j, i \in V$ while the *bias function* $w_{j0} : \Re_0^+ \longrightarrow \Re_0^- \cup \{-\infty\}$, e.g.

$$w_{j0}(t) = \begin{cases} -\infty & \text{for } 0 < t < t_{ref} \\ -h < 0 & \text{for } t \geq t_{ref} \end{cases}$$

where t_{ref} is a *refractory period*



The Computational Power of Spiking Nets:

Lower Bounds: (Maass,1996)

- any *binary feedforward perceptron network* of size s and depth d can be simulated by a neural network of $O(s)$ spiking neurons within time $O(d)$
- any *deterministic finite automaton* with q states can be simulated by a spiking network with $O(\sqrt{q})$ neurons
- any *Turing machine* can be simulated by a finite spiking network with *rational weights* while any *I/O mapping* can be implemented using *arbitrary real weights*

Upper Bounds: (Maass,1994)

finite spiking networks with any piecewise linear response- and bias-functions

$$\equiv$$

finite discrete-time *analog perceptron networks* with any piecewise linear activations functions (e.g. the saturated-linear and Heaviside functions)

$$\equiv$$

Random Access Machines with $O(1)$ registers working with unbounded arbitrary real numbers

(linear-time pairwise simulations, valid also for rational parameters)

Piecewise Constant Response Functions:

(`Maass,Ruf,1999`)

- easier to implement in hardware

- spiking networks with *piecewise constant response functions* and piecewise linear bias functions with *rational* parameters $\equiv$ *finite automata*

- for *arbitrary real parameters* these networks simulate any *Turing machine* but, in general, *not* within polynomial number of spikes

$\longrightarrow$ *the computational power of spiking networks depends on the shape of the postsynaptic potentials*

Liquid State Machine (LSM) & Online Computation:

inspired by experimental neuroscience and robotics (e.g. walking for 2-legged robots in a terrain):

- online computation: input pieces arrive all the time, not in one batch

- real-time computation: a strict deadline when the result is required

- anytime algorithms: can be interrupted and still should be able to provide a partial answer

$\times$ conventional computation theory and algorithm design have focused on offline computation:

TMs compute the static outputs from the inputs which are completely specified at the beginning a machine M performing online computations maps input streams onto output streams

these are encoded as functions $u : \Re \longrightarrow \Re^n$ of (discrete or continuous) time, i.e. $u(t) \in \Re^n$ provides the information at the time point t

M implements a filter (operator) $F : U \longrightarrow \Re^{\Re}$ that maps input functions u from domain U onto output functions $\mathbf{y}$

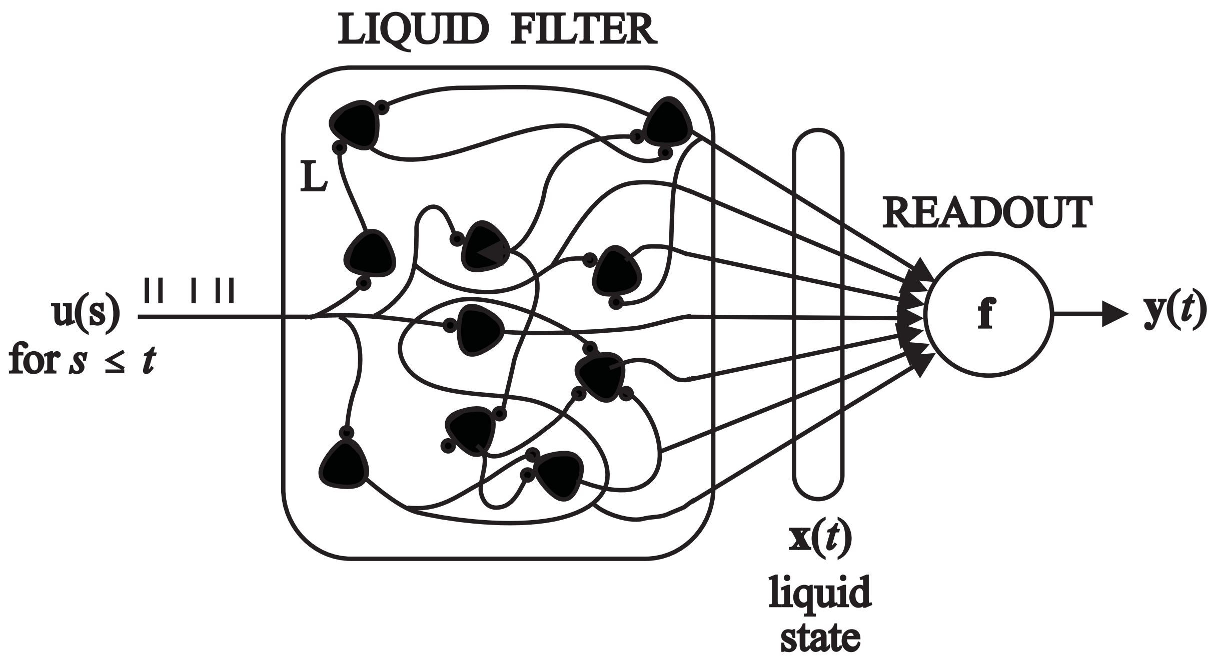
1. time-invariant: the output does not depend on any absolute internal clock of M (input-driven):

   $$(Fu(t+t_0))(t) = (Fu)(t+t_0) \text{ for any } t, t_0 \in \Re$$

   $\longrightarrow$ F is uniquely identified by the values $\mathbf{y}(0) = (Fu)(0)$ (if U is closed under temporal shift) and represents a functional (mapping the input functions $u \in U$ onto real values $(Fu)(0) \in \Re^n$)

2. fading memory: for computing the most significant bits of $(Fu)(0)$ it suffices to know an *approximate* value of input function $u(t)$ for a finite time interval back into the past (i.e. the continuity property of F)

Liquid State Machines can (under reasonable assumptions) approximate time-invariant fading memory filters



1. liquid filter (or circuit) L producing liquid states is implemented by a sufficiently rich fixed bank of basis filters or a general dynamical system, e.g. randomly and sparsely connected spiking neurons
$$\mathbf{x}(t) = (Lu)(t)$$
2. readout function $\mathbf{f}$ which is trained for a specific task, e.g. $\mathbf{f}$ is linear
$$\mathbf{y}(t) = \mathbf{f}(\mathbf{x}(t))$$

digital computations on LSM:

- if L has fading memory then LSM is even unable to implement all FA
- LSM augmented with a feedback from a readout to the liquid circuit is universal for analog computing, e.g. LSM can simulate arbitrary TM

Summary and Open Problems

1. **Unit Type:**

- traditional perceptron networks are best understood
- similar analysis for other unit types still not complete: their taxonomy should be refined for different architectures, parameter domains, probabilistic computation, etc.
- e.g. open problems:
  - Turing universality of finite RBF networks
  - the power of recurrent WTA networks
- RBF networks comparable to perceptron networks
- WTA gates may bring more efficient implementations
- networks of spiking neurons are slightly more efficient than their perceptron counterparts; temporal coding as a new source of efficient computation

2. **Discrete vs. Continuous Time:**

- continuous-time perceptron networks are at least as powerful as the discrete-time models

- the simulation techniques unsatisfying: the continuous-time computation is still basically discretized

- discrete-time mind-set of traditional complexity theory provides no adequate theoretic tools (e.g. complexity measures, reductions, universal computation, etc.) for "naturally" continuous-time computations

- continuous-time neural networks as useful reference models for developing such theoretical tools (`Ben-Hur, Siegelmann,Fishman,2002; Gori,Meer,2002`)

3. **Deterministic vs. Probabilistic Computation:**

- stochasticity represents an additional source of efficient computation in probabilistic perceptron networks (e.g. IP can be computed efficiently using only two-layered probabilistic networks while an efficient deterministic implementation requires 3 layers)

- from the computational power point of view stochasticity plays a similar role in neural networks as in conventional Turing computations

- open problem: e.g. a more efficient implementation of finite automata by binary-state probabilistic neural networks than that by deterministic neural networks

4. **Feedforward vs. Recurrent Architectures:**

- feedforward networks $\equiv$ convergent recurrent networks $\equiv$ symmetric networks

- common interesting functions (e.g. arithmetic operations) can be implemented efficiently with only a small number of layers $\longrightarrow$ the widely spread use of two- or three-layered networks in practical applications

- two layers of perceptrons are not sufficient for an efficient implementation of certain functions

- open problem: is the bounded-depth TC^0 hierarchy infinite? (the separation of three-layer and four-layer networks is unknown)

- the computational power of finite recurrent networks is nicely characterized by the descriptive complexity of the weights, e.g. for rational weights these networks are Turing universal

  $\times$ more realistic models with fixed precision of real parameters or analog noise recognize only regular languages

- practical recurrent networks essentially represent efficient implementations of finite automata

5. **Binary vs. Analog States:**

- analog-state neural networks prove to be at least as powerful and efficient computational devices as their binary-state counterparts
- for feedforward architectures the computational power of binary and analog states is almost equal ($\times$ sometimes the size can be reduced by a logarithmic factor)
- open problem: e.g. the equivalence of sigmoidal and threshold gates in polynomial-size networks for large weights (i.e. $TC_d^0(\sigma) = TC_d^0$ for exponential weights)
- for recurrent architectures infinite amounts of information stored in the analog states drastically increases the computational power of finite networks from finite automata to Turing universality or even more
- analog models of computation may be worth investigating more for their efficiency gains than for their (theoretical) capability for arbitrary-precision real number computation
- open problems: e.g.
  - how efficient implementations of finite automata by analog neural networks can be achieved?
  - how this efficiency depends on the chosen activation function?

6. **Symmetric vs. Asymmetric Weights:**

- for binary-state networks not only do all Hopfield nets converge but all convergent computations can be efficiently implemented using symmetric weights

- for analog networks an external oscillator is needed to boost the power of symmetric networks to that of asymmetric ones

- analog symmetric networks cannot perform arbitrary unbounded computations, i.e. probably less powerful than finite automata

- open problem: convergence conditions for neural networks based on other types of units than perceptrons

Literature on Complexity Theory of Neural Networks

- Anthony, M. (2001). *Discrete Mathematics of Neural Networks: Selected Topics.* Philadelphia, PA: Society for Industrial and Applied Mathematics.

- Parberry, I. (1994). *Circuit Complexity and Neural Networks.* Cambridge, MA: The MIT Press

- Roychowdhury, V. P., Siu, K.-Y., & Orlitsky, A. (Eds.) (1994). *Theoretical Advances in Neural Computation and Learning*. Boston: Kluwer Academic Publishers

- Siegelmann, H. T. (1999). *Neural Networks and Analog Computation: Beyond the Turing Limit*. Boston: Birkhäuser.

- Šíma, J., & Orponen, P. (2003). General-purpose computation with neural networks: A survey of complexity theoretic results. *Neural Computation, 15*(12), 2727–2778. (covers most of this tutorial)

- Siu, K.-Y., Roychowdhury, V. P., & Kailath, T. (1995a). *Discrete Neural Computation: A Theoretical Foundation*. Englewood Cliffs, NJ: Prentice Hall.