Parallel Rewriting in Neural Networks

Ekaterina Komendantskaya

School of Computer Science, University of St Andrews

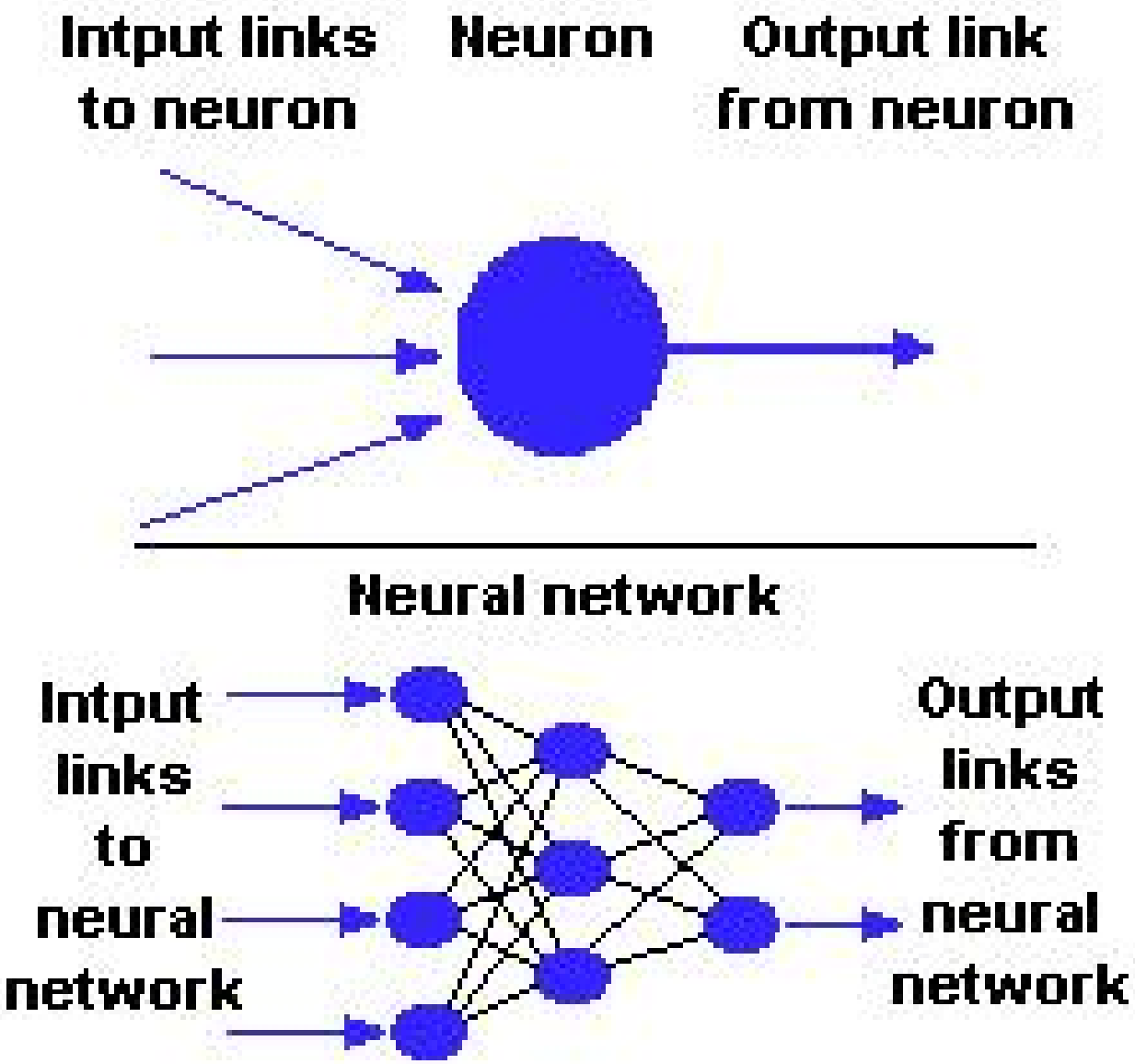
CiE'09, Heidelberg

Outline

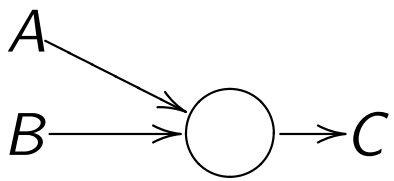
1. CLANN project: http://www.cs.st-andrews.ac.uk/~ek/CLANN/

Outline

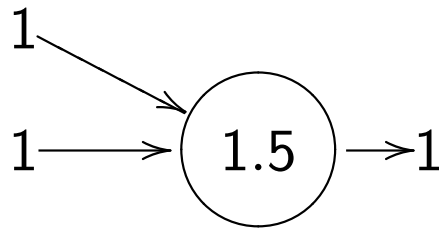
Neural Network: definitions



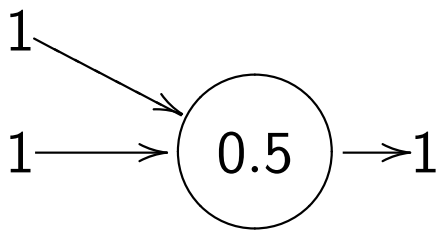
Logic and Neurons: McCullogh and Pitts, 1943.



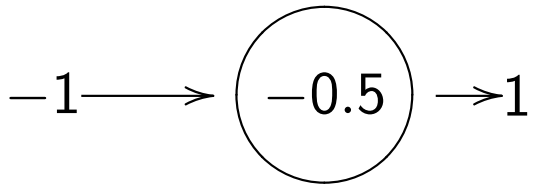
If A and B then C.



$(A = 1)$ and $(B = 1)$.



$(A = 1)$ or $(B = 1)$.



Not $(A = -1)$.

Logic and Neurons - Level of abstraction 1: Automata

(Minsky 1954; Kleene 1956; von Neumann 1958: Neural and digital hardware are equally suitable for symbolic computations.

Logic and Neurons - Level of abstraction 1: Automata

(Minsky 1954; Kleene 1956; von Neumann 1958: Neural and digital hardware are equally suitable for symbolic computations. The picture is due to Alexander & Morton, 1996)

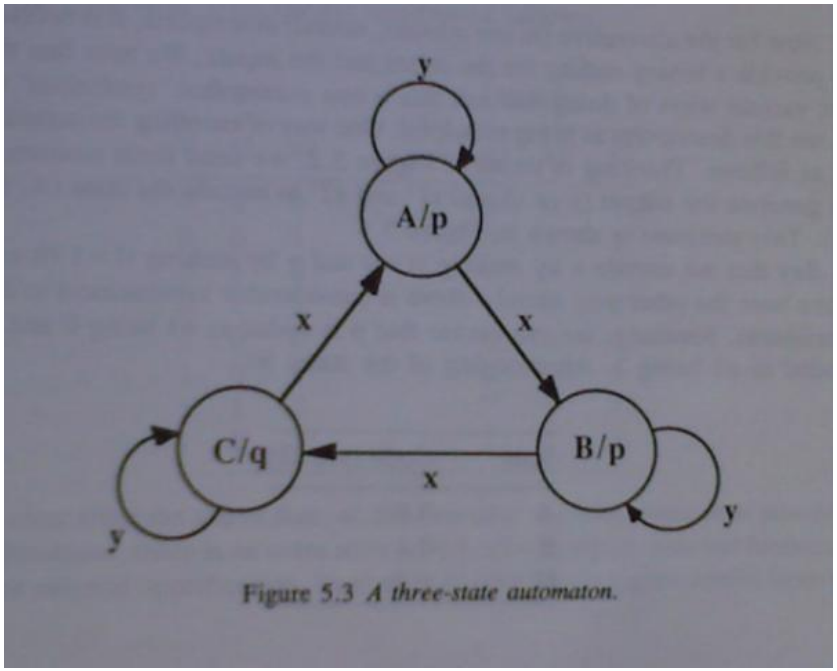


Figure 5.3 *A three-state automaton.*

Logic and Neurons - Level of abstraction 1: Automata

(Minsky 1954; Kleene 1956; von Neumann 1958: Neural and digital hardware are equally suitable for symbolic computations. The picture is due to Alexander & Morton, 1996)

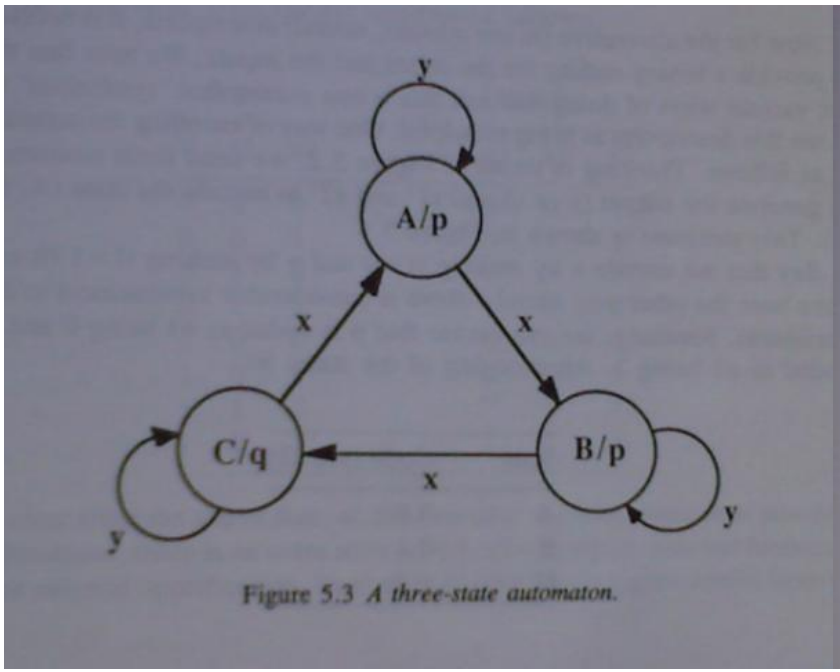


Figure 5.3 *A three-state automaton.*

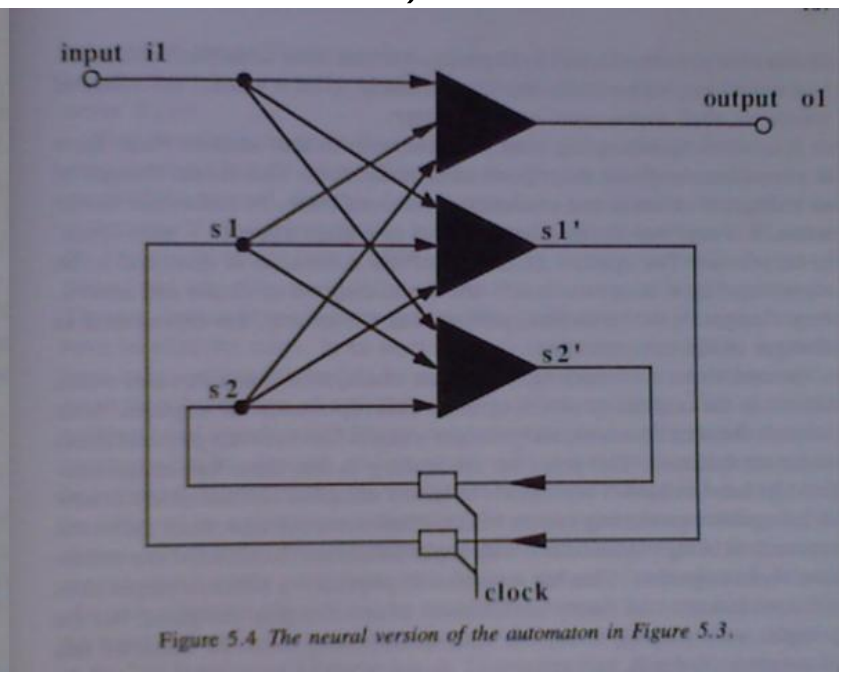


Figure 5.4 *The neural version of the automaton in Figure 5.3.*

Logic and NNs: summary [Siegelmann]

Finite Automata $\longrightarrow$ Binary threshold networks
Turing Machines $\longrightarrow$ Neural networks with rational weights
Probabilistic Turing Machines $\longrightarrow$ NNs with rational weights
9-neuron network will suffice to simulate Universal Turing machine
[Siegelmann and Sontag]

Levels of Abstraction: from 1 to 2

The results we have mentioned form the 1st, theoretical, level of abstraction are general and powerful enough to claim that, given a neural computer, we can transform hardware and software architectures of digital computers to fit the neural hardware. However, in 2009, unlike in 1959, the development of digital and neural hardware do not come hand in hand. As soon as digital computers started to take over, another level of abstraction, much less general, became popular.

Levels of Abstraction: from 1 to 2

The results we have mentioned form the 1st, theoretical, level of abstraction are general and powerful enough to claim that, given a neural computer, we can transform hardware and software architectures of digital computers to fit the neural hardware. However, in 2009, unlike in 1959, the development of digital and neural hardware do not come hand in hand. As soon as digital computers started to take over, another level of abstraction, much less general, became popular.

Given a Neural Network simulator, what kind of practical problems can I solve with it? where can I apply it?

(Parallelism, classification.)

Implementations of Computational Logic in NNs...

A Simple Example

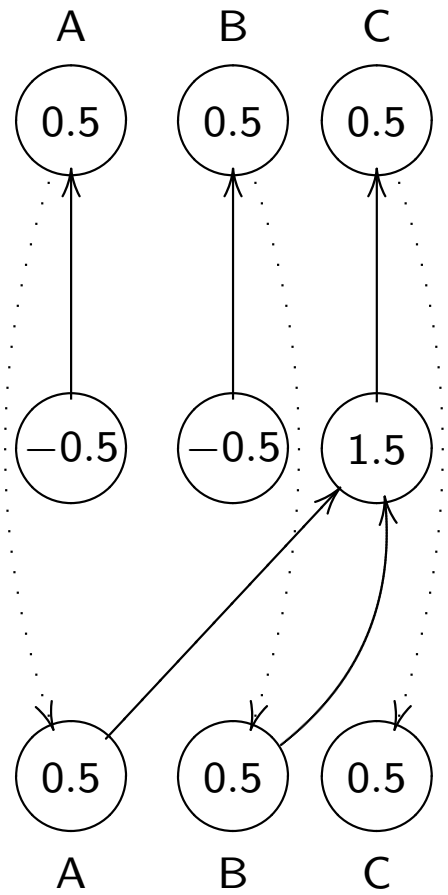
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$

A Simple Example

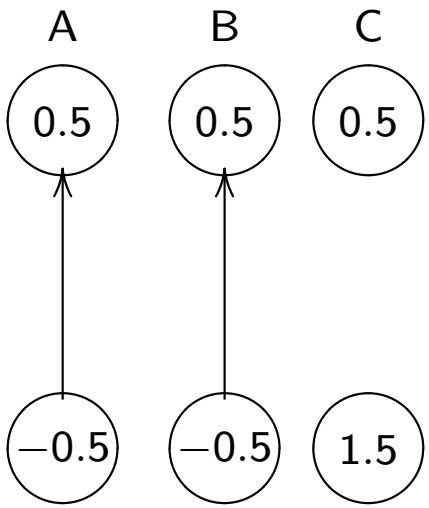
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

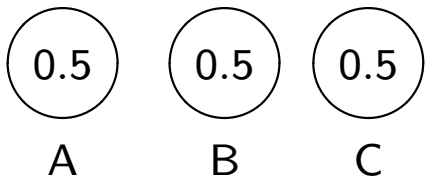
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



A Simple Example

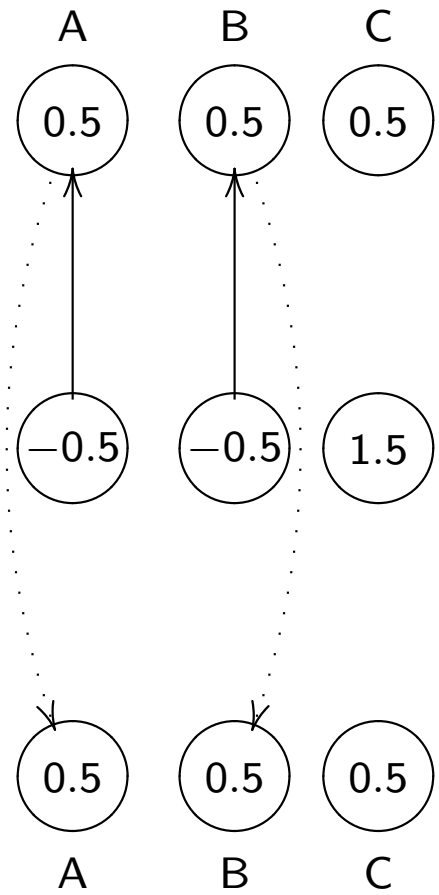
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$





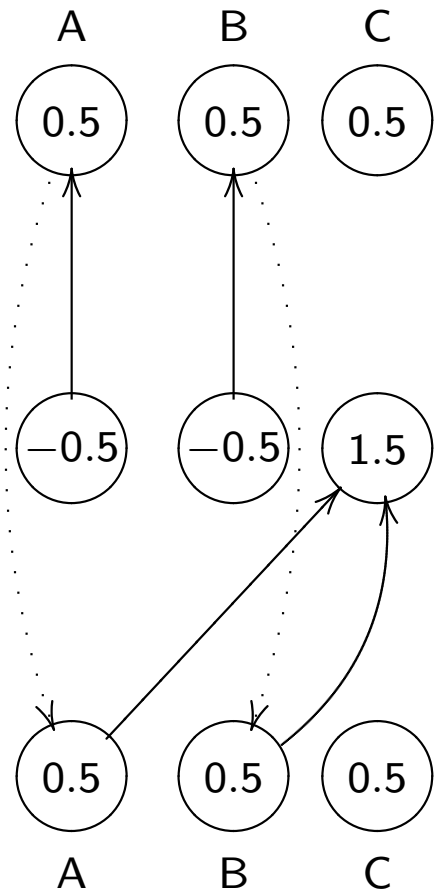
A Simple Example

$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



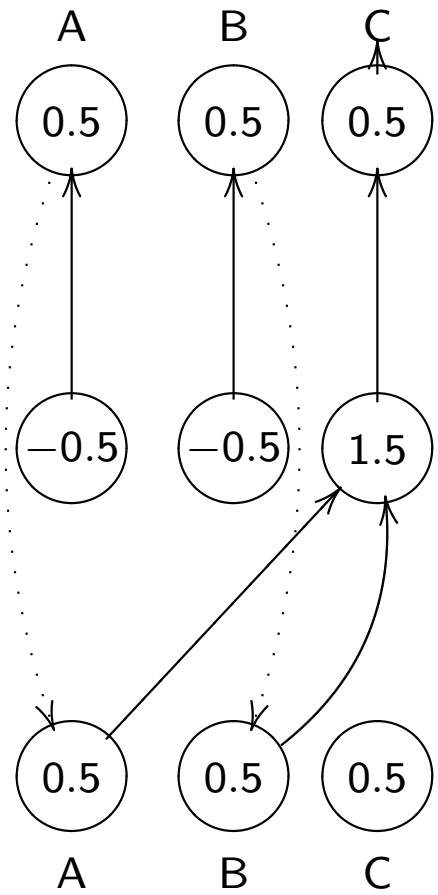
A Simple Example

$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



A Simple Example

$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



Problems with logic reasoning by boolean networks

$P(0) \leftarrow$
$P(s(x)) \leftarrow P(x)$

Problems with logic reasoning by boolean networks

$P(0) \leftarrow$
$P(s(x)) \leftarrow P(x)$

Problems with logic reasoning by boolean networks

$P(0) \leftarrow$
$P(s(x)) \leftarrow P(x)$

Paradox:
(computability,
complexity,
proof theory,
proximity to conventional or biological neural networks is illusory and deceiving...)

Neuro-symbolic architectures of other kinds:

This problem of processing truth values instead of the syntax, causes the same problem in most of the existing Neuro-Symbolic systems, e.g.:

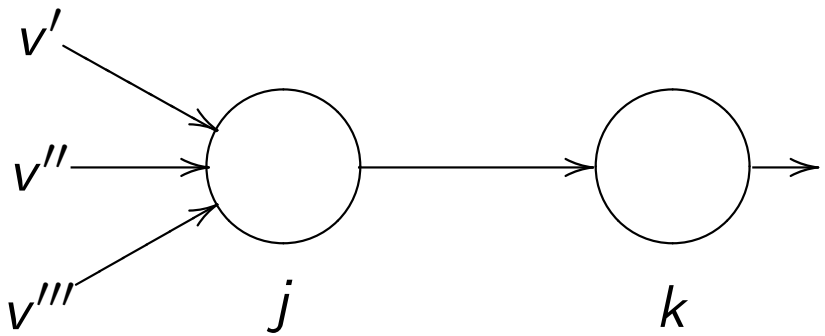
- The "core" method of computing semantic operators for logic programs [Holldobler et al, 1994- 2009]
- Markov Logic and Markov networks [Domingos 2006-2009]
- Inductive and Modal logics in Neural Networks [Broda, Garcez et al. 2002,2008]
- Fuzzy Logic Programming in Fuzzy Networks.

Methods we suggest:

1. Logic terms can be processed directly, without processing their truth values; one can use a one-to-one numerical encoding, if necessary.
2. Although only scalar numbers are allowed to be processed by a single neuron, a layer of neurons processes vectors of symbols, and a network of several layers processes matrices of signals. One can use vectors as representatives of strings; and matrices - as representatives of trees.
3. Parallel algorithms can be easily applied in neural networks.
4. Many techniques of computational logic - such as unification or term-rewriting naturally arise - in non-symbolic forms - in conventional learning algorithms of neurocomputing.
5. Learning Functions one uses in Neuro-Symbolic networks can be arbitrary, not necessarily the conventional functions of neurocomputing.

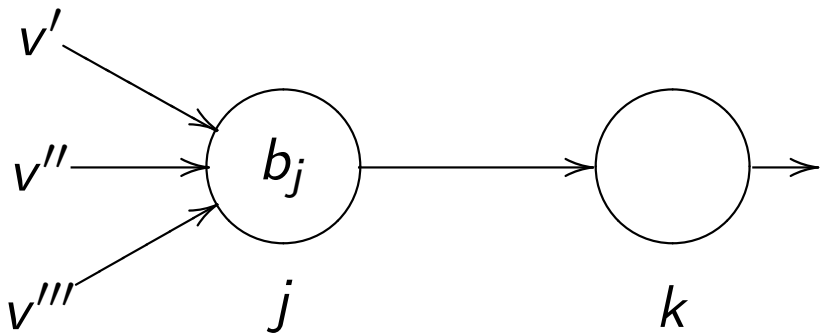
Neurons

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) + b_k \right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



Neurons

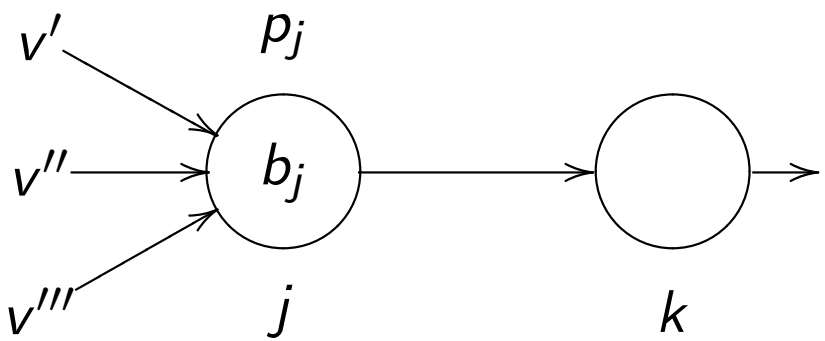
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) + b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



The following parameters can be trained: weights w_{kj}, biases b_k, b_j.

Neurons

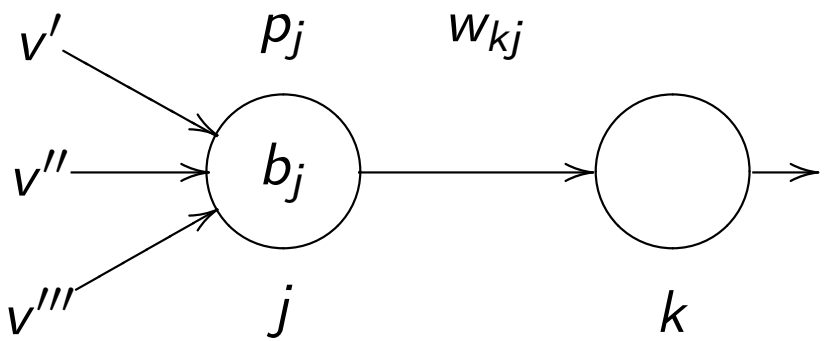
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) + b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



The following parameters can be trained: weights w_{kj}, biases b_k, b_j.

Neurons

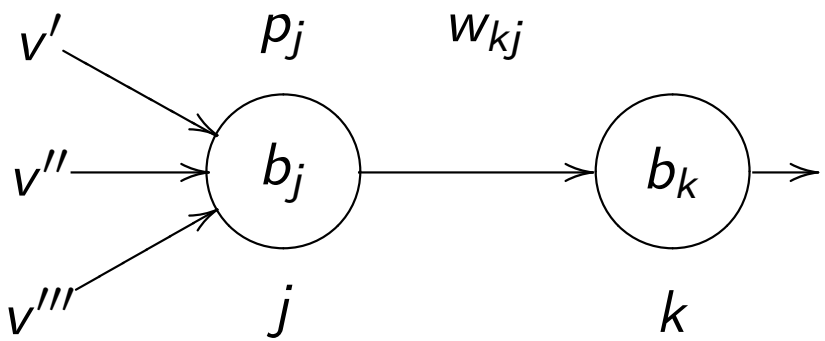
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) + b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



The following parameters can be trained: weights w_{kj}, biases b_k, b_j.

Neurons

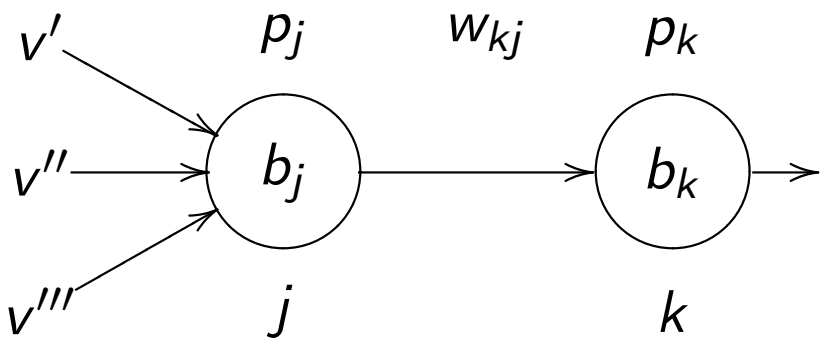
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) + b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



The following parameters can be trained: weights w_{kj}, biases b_k, b_j.

Neurons

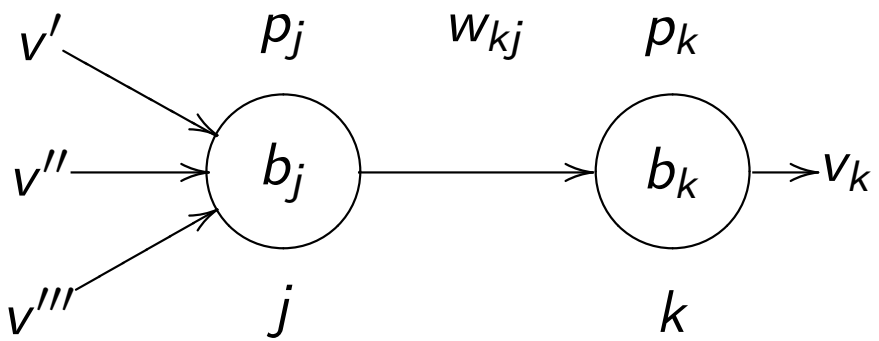
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) + b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



The following parameters can be trained: weights w_{kj}, biases b_k, b_j.

Neurons

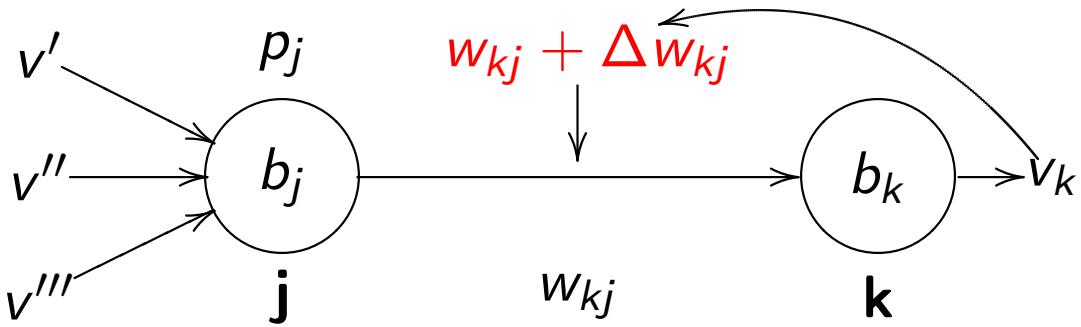
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) + b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



The following parameters can be trained: weights w_{kj}, biases b_k, b_j.

Hebbian (Unsupervised) Learning

Unsupervised learning rule: $\Delta w_{kj}(t) = F(v_k(t), v_j(t))$, where F is some function. Very often, it is $\Delta w_{kj}(t) = \eta v_k(t) v_j(t)$, for some constant η, called **the rate of learning**.



Example 1. Parallel Rewriting

Consider a string [1:2:3:1:2:3:3:1:2:3:1:2] and rewriting rules - ground instantiations of $x \longrightarrow 3x$ for $1, 2, 3$

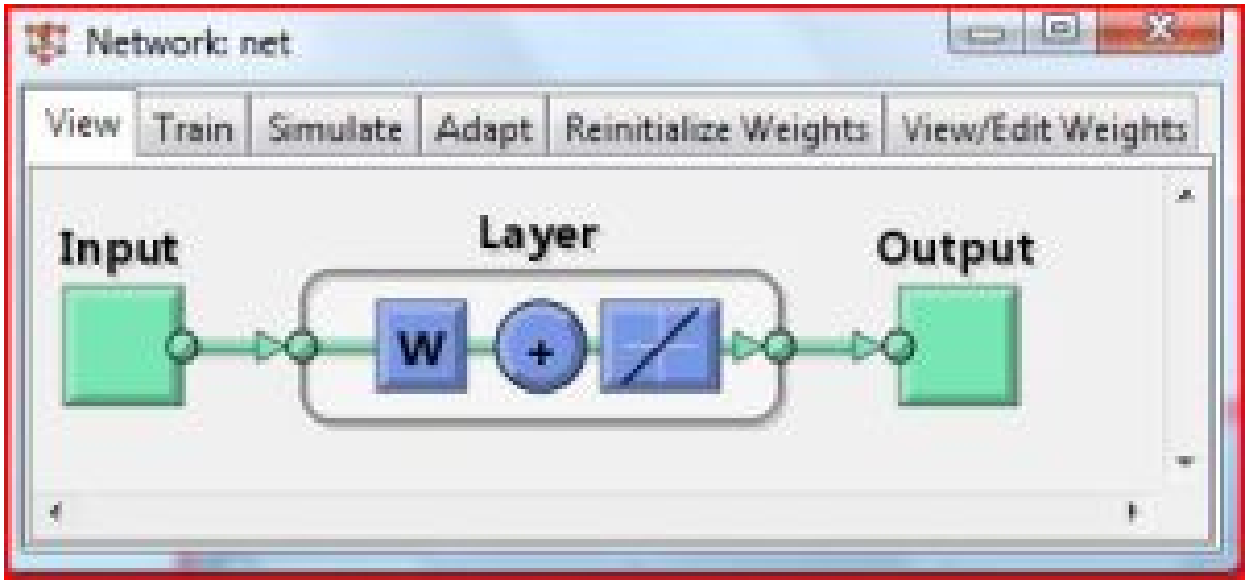
The parallel rewriting step will give us
$[3 : 6 : 9 : 3 : 6 : 9 : 9 : 3 : 6 : 9 : 3 : 6]$.

Example 1. Parallel Rewriting

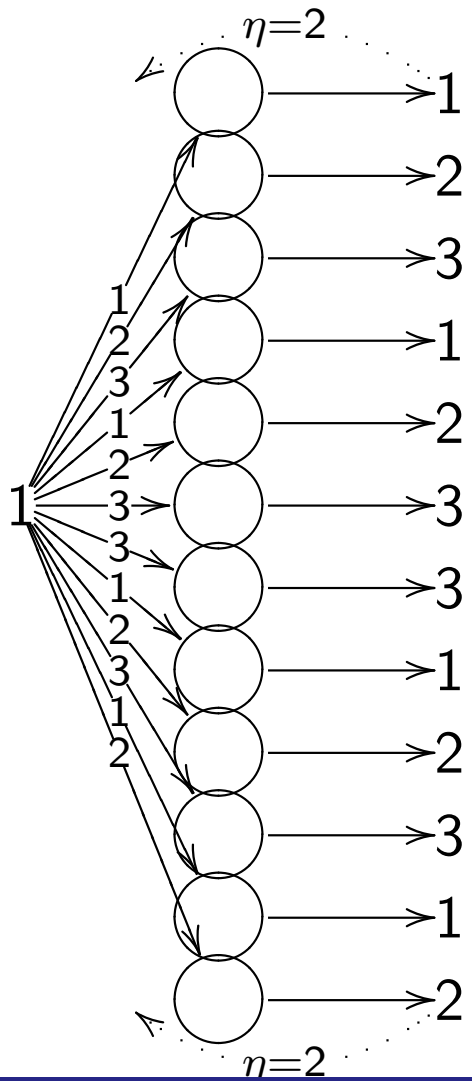
Consider a string [1:2:3:1:2:3:3:1:2:3:1:2] and rewriting rules - ground instantiations of $x \longrightarrow 3x$ for $1, 2, 3$

The parallel rewriting step will give us
$[3:6:9:3:6:9:9:3:6:9:3:6]$.

This can be done in unsupervised learning network `net`: the rate of learning $\eta = 2$; $\Delta\mathbf{w} = \eta\mathbf{yx} = 2\mathbf{w}$; $\mathbf{w}^{\mathtt{new}} = \mathbf{w} + \Delta\mathbf{w} = 3\mathbf{w}$.



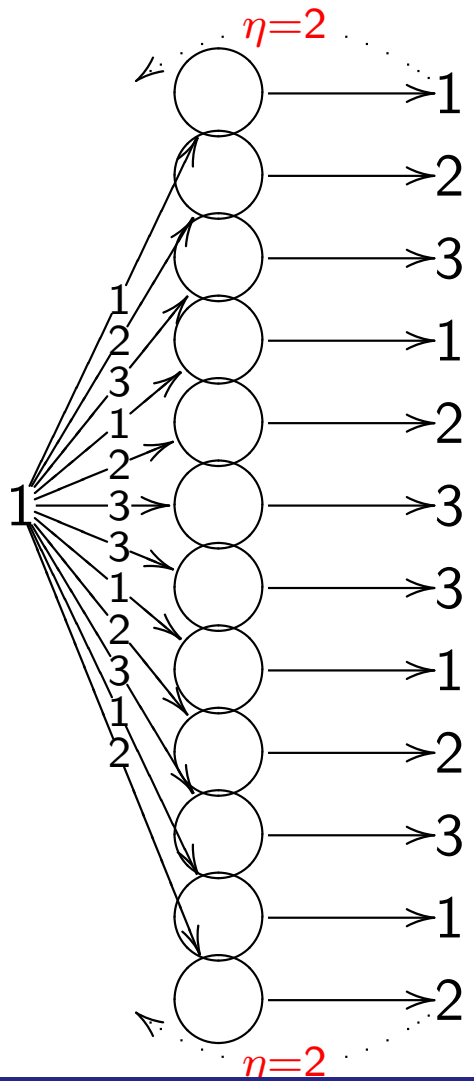
Parallel rewriting on strings - a closer look:



$$w^{\text{new}} = w^{\text{old}} + \Delta w$$

$\Delta w =$
2
4
6
2
4
6
6
2
4
6
2
4

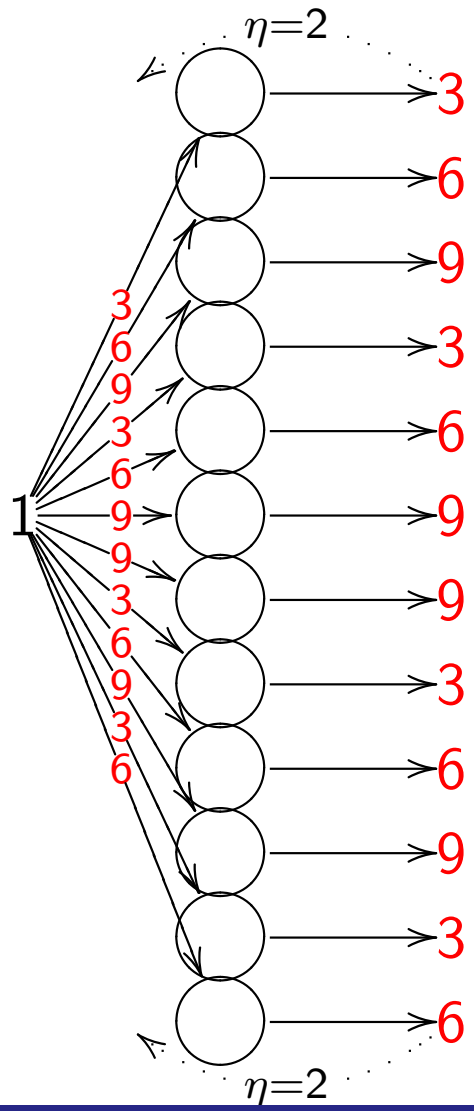
Parallel rewriting on strings - a closer look:



$$w^{\text{new}} = w^{\text{old}} + \Delta w$$

$\Delta w =$
2
4
6
2
4
6
6
2
4
6
2
4

Parallel rewriting on strings - a closer look:



Parallel (Term) Rewriting.

An *abstract rewriting system* (ARS)

is a structure $\mathcal{A} = (A, \{\rightarrow_\alpha | \alpha \in I\})$ consisting of a set A and a set of binary relations $\rightarrow_\alpha$ on A, indexed by a set I.

A *term rewriting system* (TRS)

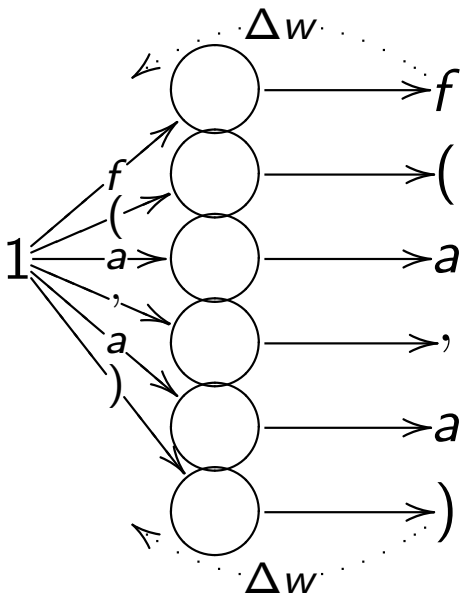
is a pair $\mathcal{R} = (\Sigma, R)$ of a first-order signature Σ and a set of rewriting rules R for Σ, (subject to certain restrictions).

Parallel rewriting step

Let a term t contain some disjoint redexes $s_1, s_2, \ldots, s_n$; that is, suppose we have $t \equiv C[s_1, s_2, \ldots, s_n]$, for some context C. If their contracta are respectively $s'_1, s'_2, \ldots, s'_n$, in n steps the reduct $t' \equiv C[s'_1, s'_2, \ldots, s'_n]$ can be reached. These n steps together are called a *parallel step*.

Parallel Term rewriting

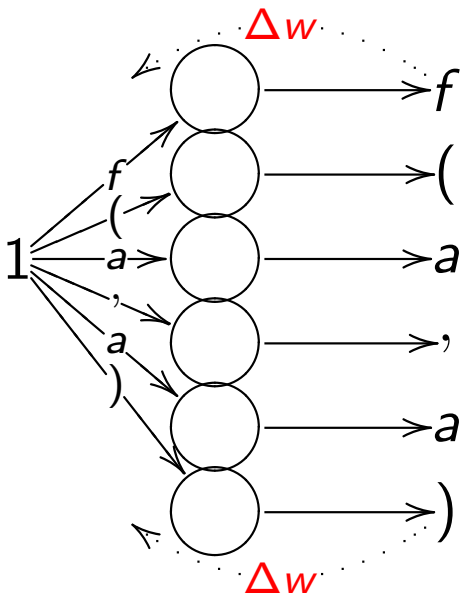
We assume here, that symbols f, a, $($, $)$ and b are encoded as numbers $|f|$, $|a|$, $|(|$, $|)|$ and $|b|$. The rewriting rule is $a \longrightarrow b$.



$$\Delta w = \begin{matrix} 0 \\ 0 \\ |b| - |a| \\ 0 \\ |b| - |a| \\ 0 \end{matrix}$$

Parallel Term rewriting

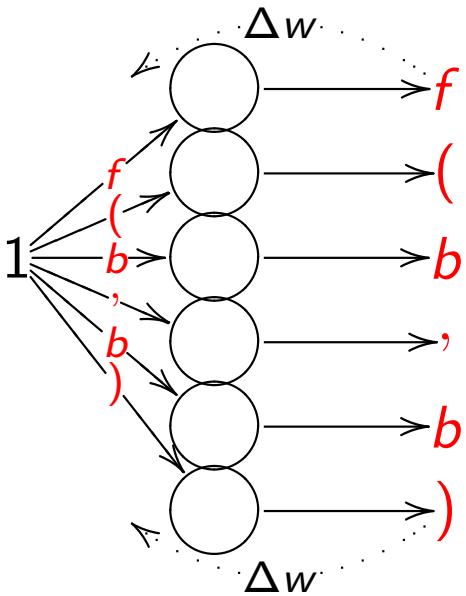
We assume here, that symbols f, a, $($, $)$ and b are encoded as numbers $|f|$, $|a|$, $|(|$, $|)|$ and $|b|$. The rewriting rule is $a \longrightarrow b$.



$$\Delta w = \begin{matrix} 0 \\ 0 \\ |b| - |a| \\ 0 \\ |b| - |a| \\ 0 \end{matrix}$$

$w^{\text{new}} = w^{\text{old}} + \Delta w$ that is,
$[|f|; |(|; |a|; |)|] + [0; 0; |b| - |a|; 0] = [|f|; |(|; |b|; |)|]$.

Parallel Term rewriting



Complications

- The learning rule and change vectors may or may not be describable by a continuous function (or a function that is conventionally used in Neurocomputing). In the first example, we used the conventional learning function ($\Delta\mathbf{w} = \eta\mathbf{yx} = 2\mathbf{w}$)... But not in the second.
- Terms may grow at each substitution: e.g., the term $f(b)$ may well be transformed into $f(g(a))$, which will require more neurons.

This is why, the learning functions we use need to be more clever than just arithmetic operations...

Solutions

- We widen the range of available learning functions, expressing some of them algorithmically. E.g, the learning function for the 2nd example would need to be formulated (roughly) as follows: `for a rewriting rule` $a \longrightarrow b$, `find` $|a|$ `in the vector` w, `and form the vector` $\Delta w = [0; 0; |b| - |a|; 0; |b| - |a|; 0]$.
- We define an algorithm for adding neurons to the layer, which roughly follows the idea of "growing neural gas" [Fritzke,94]. Having a set of rewriting rules, one extends the layer of neurons after each parallel rewriting step to allow terms to grow as rewriting proceeds.
- We extend this to several rewriting rules.

This set of new functions is formalised in MATLAB neural network simulator, and the library is ready to use. The networks perform parallel rewriting for an arbitrary term-rewriting system.

Conclusions

1. The change in style: The networks work directly with terms, and not truth values.
2. Now the burden to do symbolic computations is taken by learning functions, and not by architectures of the networks; - this gives control & flexibility.
3. Because we exploit the conventional rewriting abilities of neural networks, for certain number of cases purely neural approach will suffice. For more general cases of term-rewriting, all we need to do is to switch a new learning function, which may be rather "symbolic" (=algorithmic).
4. This kind of networks can be further implemented in hybrid (neuro-symbolic) systems.
5. Generally, there is always a trade-off between the amount of symbolism we allow in neuro-symbolic networks and the level of generality of logical problems they can solve.

Literature: Level of Abstraction 1.


I. Alexander and H. Morton.
Neurons and Symbols.
Chapman&Hall, 1993.


S.C. Kleene.
Neural Nets and Automata.
Automata Studies, pp. 3 – 43, 1956, Princeton University Press.


W.S. McCulloch and W. Pitts.
A logical calculus of the ideas immanent in nervous activity.
Bulletin of Math. Bio., Vol. 5, pp. 115-133, 1943.


M. Minsky.
Finite and Infinite Machines.
Prentice-Hall, 1969.


H. Siegelmann.
Neural Networks and Analog Computation: Beyond the Turing Limit.
Birkhauser, 1999.


J. Von Neumann.
The Computer and The Brain.
1958, Yale University Press.

Literature. Level of Abstration 2


M. Richardson and P. Domingos.
Markov Logic Networks.
Machine Learning. To appear.


A. Garcez, K.B. Broda and D.M. Gabbay.
Neural-Symbolic Learning Systems: Foundations and Applications,
Springer-Verlag, 2002.


A. Garcez, L.C. Lamb and D.M. Gabbay.
Neural-Symbolic Cognitive Reasoning,
Cognitive Technologies, Springer-Verlag, 2008.


B. Hammer and P. Hitzler.
Perspectives on Neural-Symbolic Integration.
Studies in Computational Intelligence, Springer Verlag, 2007.


S. Hölldobler, Y. Kalinke and H.P. Storr.
Approximating the Semantics of Logic Programs by Recurrent Neural Networks.
Applied Intelligence, 11, 1999, pp. 45–58.


P. Smolensky and G. Legendre.
The Harmonic Mind.
MIT Press, 2006.