Neurons OR Symbols: Why Does OR Remain Exclusive?

Ekaterina Komendantskaya

School of Computer Science, University of St Andrews

STP'09, St Andrews

Outline

1. Neural networks
   - Definitions

Outline

1. Neural networks
   - Definitions
2. Level of Abstraction 1
   - Boolean networks and automata

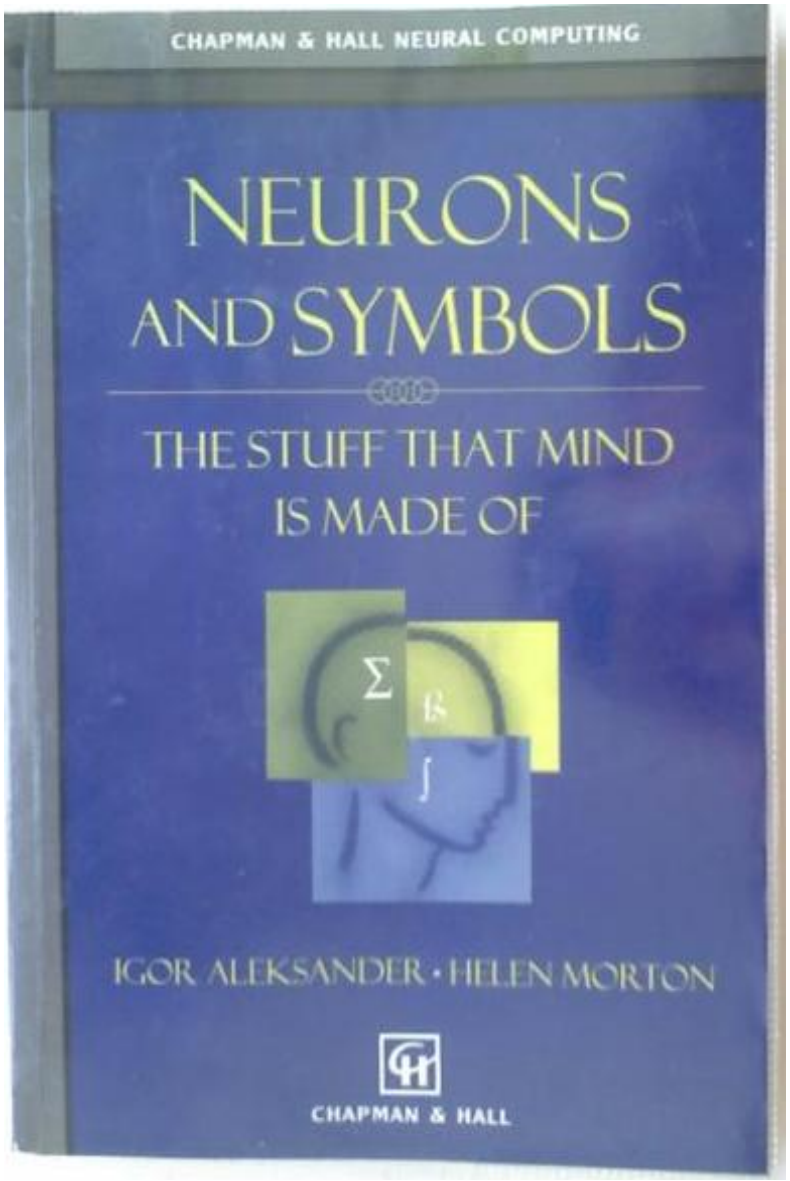
Outline

1. Neural networks
   - Definitions
2. Level of Abstraction 1
   - Boolean networks and automata
3. Level of Abstraction 2.
   - Main stream work
   - What IS wrong?

Outline

1. Neural networks
   - Definitions
2. Level of Abstraction 1
   - Boolean networks and automata
3. Level of Abstraction 2.
   - Main stream work
   - What IS wrong?
4. How to fix it?

Neurons AND Symbols



The title of this talk is inspired by the book **"Neurons and Symbols"**, [Alexander and Morton, 1993]. It claims that there is no, and should not be, divide between symbolic and neuro-computing.

Programmed computing involves:

- devising an algorithm to solve a given problem;
- using software to encode it;
- using the hardware to implement the software

Note!

This has always had close theoretical relation to Mathematical Logic (Church, λ-calculus, Kleene, etc.), the theory of computable functions, Turing machines, von Neumann machines.

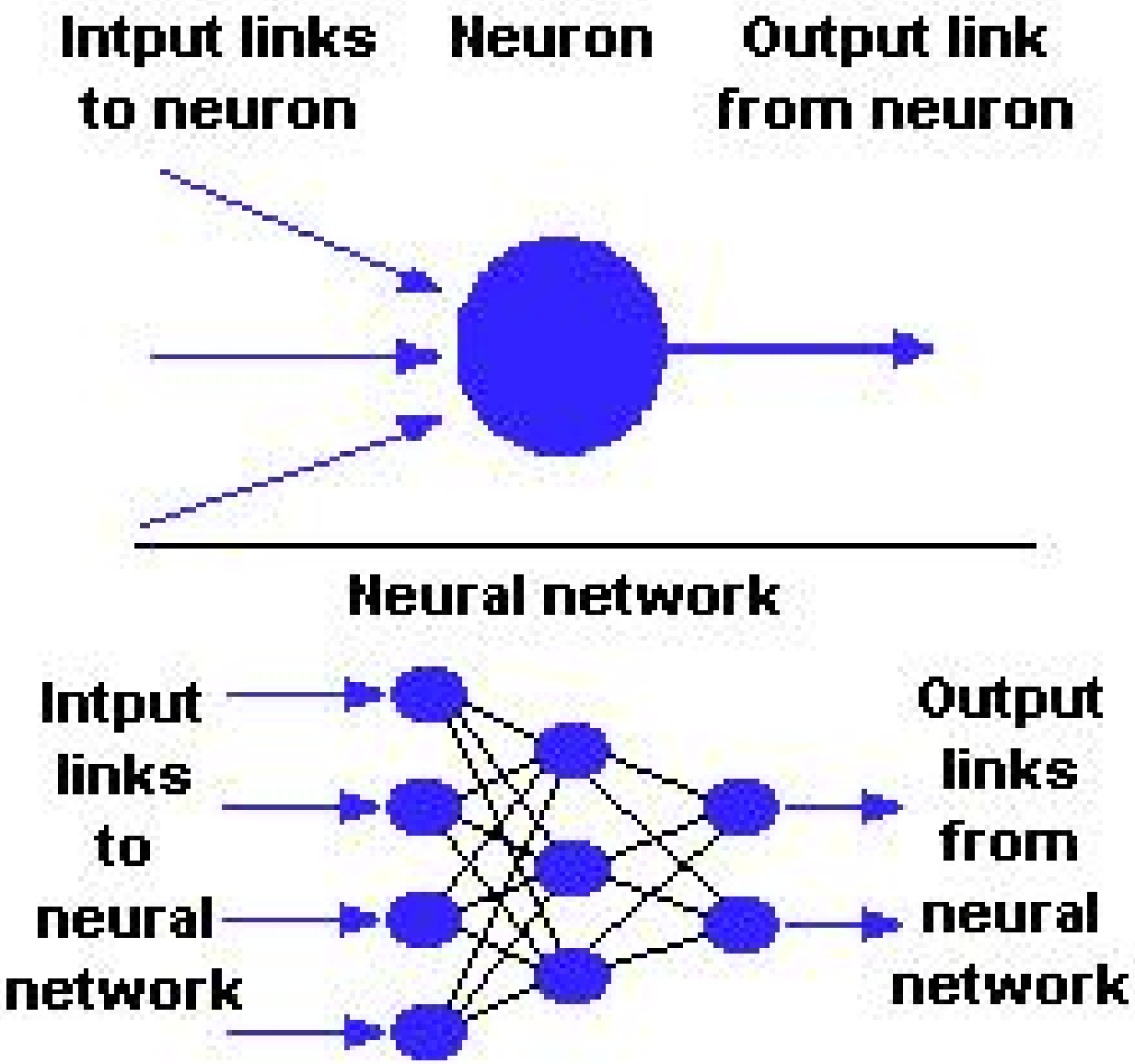
Neurocomputing

- Takes inspiration from (biological) neural networks, rather than from logic.
- Does not require a ready algorithm, but is capable to "learn" the algorithm from examples.
- Artificial Neural networks are parallel, distributed, adaptive processing systems that develop information processing capabilities in response to exposure to an information environment.

Note

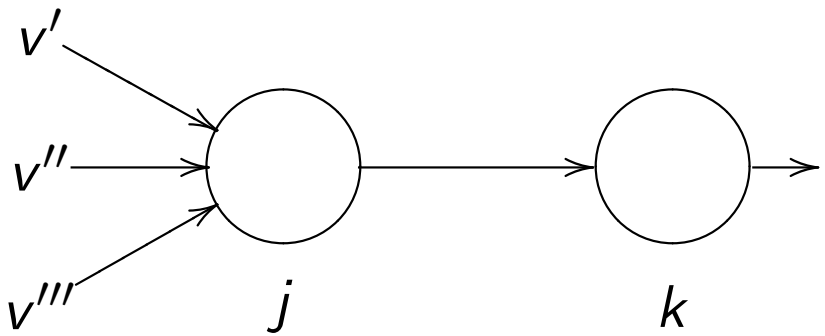
The major advantages: parallelism, learning, ability to adapt.

Neural Network: definitions



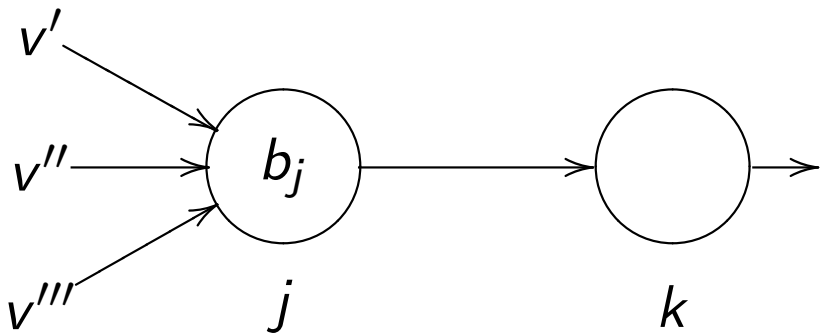
Neurons

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



Neurons

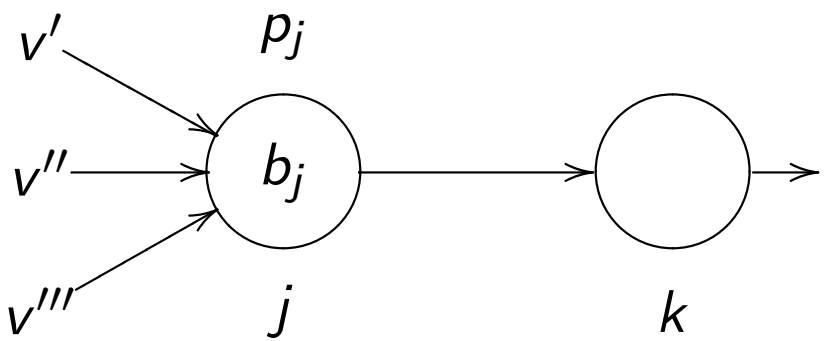
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

Neurons

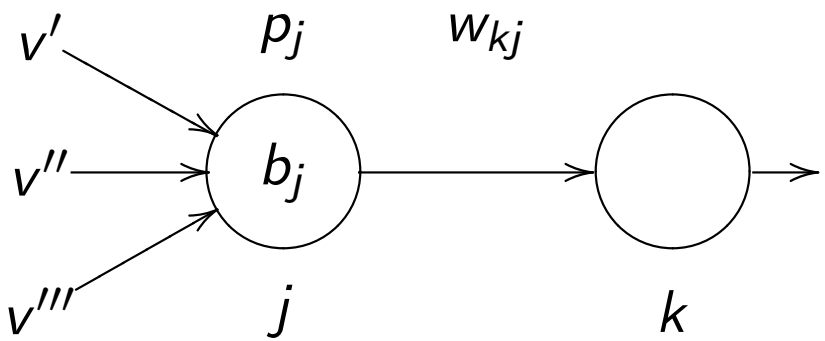
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

Neurons

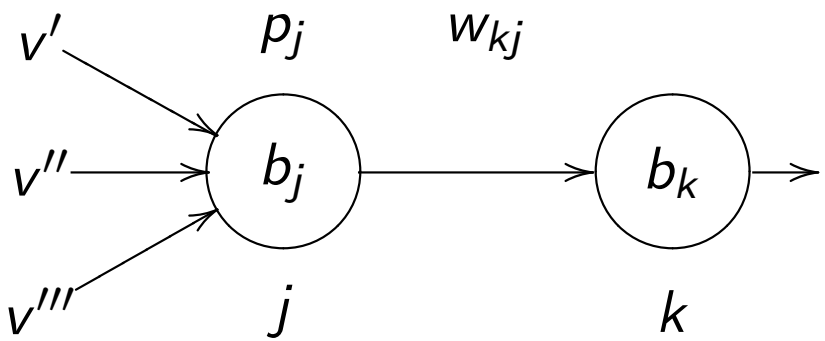
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

Neurons

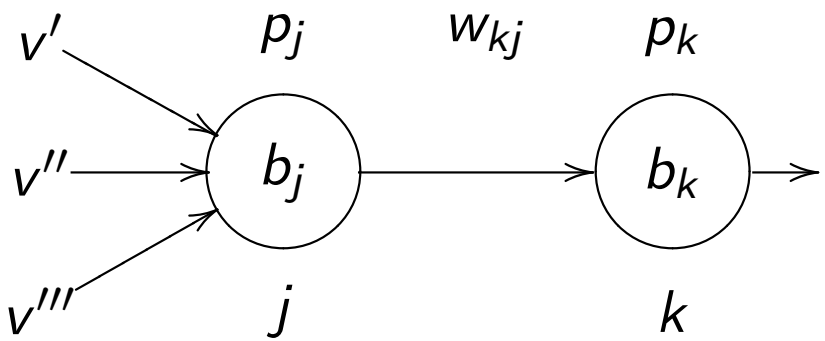
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

Neurons

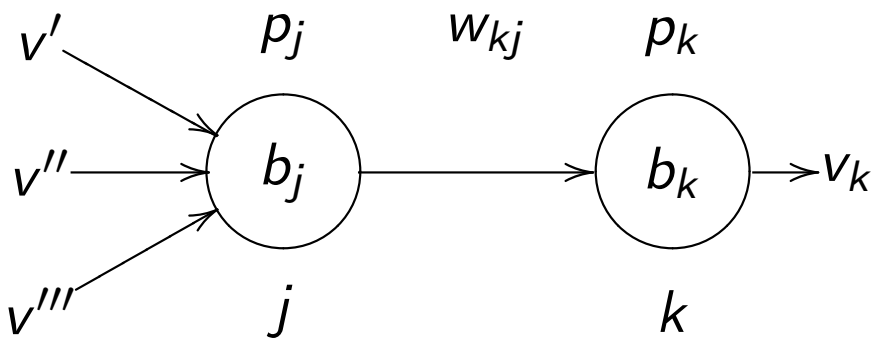
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

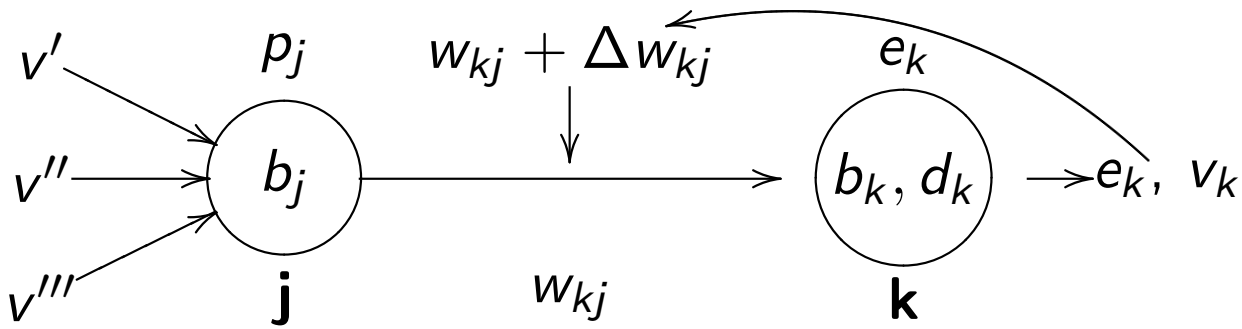
Neurons

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



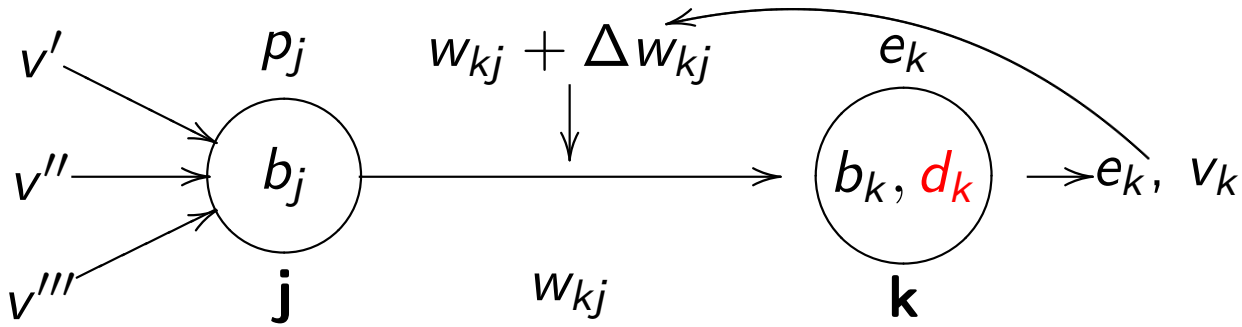
The following parameters can be trained: weights w_{kj}, biases b_k, b_j.

Error-Correction (Supervised) Learning



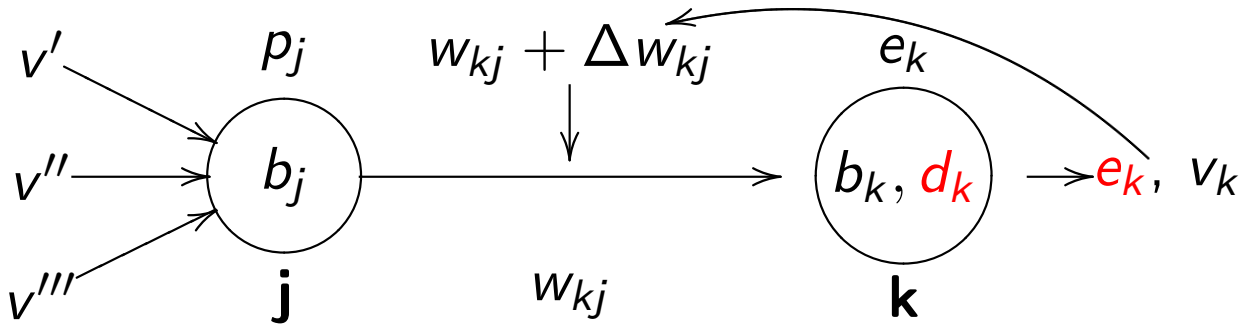
Error-Correction (Supervised) Learning

We embed a new parameter, **desired response** d_k into neurons;



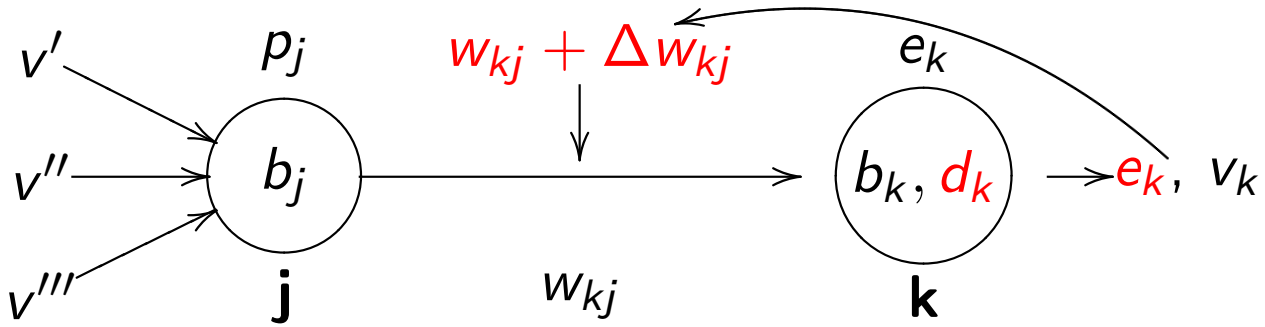
Error-Correction (Supervised) Learning

We embed a new parameter, **desired response** d_k into neurons;
Error-signal: $e_k(t) = d_k(t) - v_k(t)$;



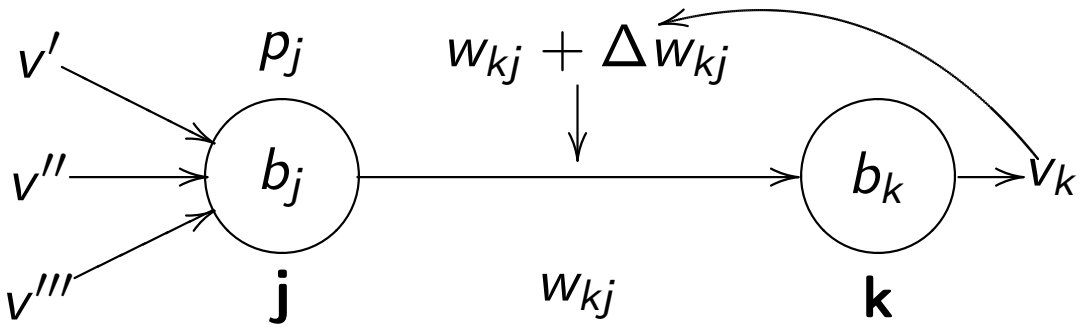
Error-Correction (Supervised) Learning

We embed a new parameter, **desired response** d_k into neurons;
Error-signal: $e_k(t) = d_k(t) - v_k(t)$;
Error-correction learning rule: $\Delta w_{kj}(t) = F(e_k(t), v_j(t))$, very often, $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.



Hebbian (Unupervised) 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**.

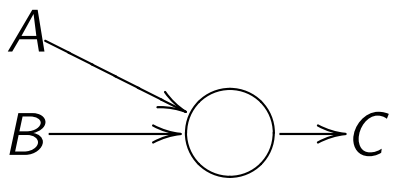


Logic and networks

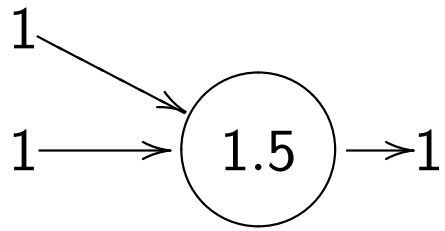
Among early results relating logic and neural networks were:

- Boolean networks - networks receiving and emitting boolean values and performing boolean functions. There exist networks that can learn how to perform Boolean functions from examples.
- XOR problem and perceptron.

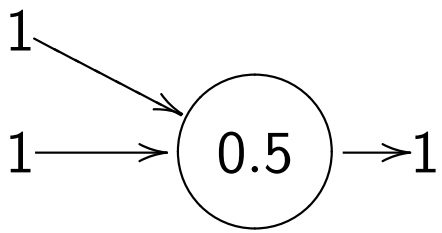
Example: Logical connectives: McCullogh and Pitts, 1943.



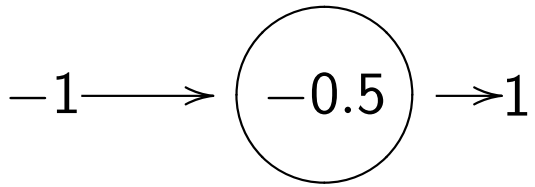
If A and B then C.



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



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



Not $(A = -1)$.

Level of abstraction 1: NN - Automata

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

Level of abstraction 1: NN - 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)

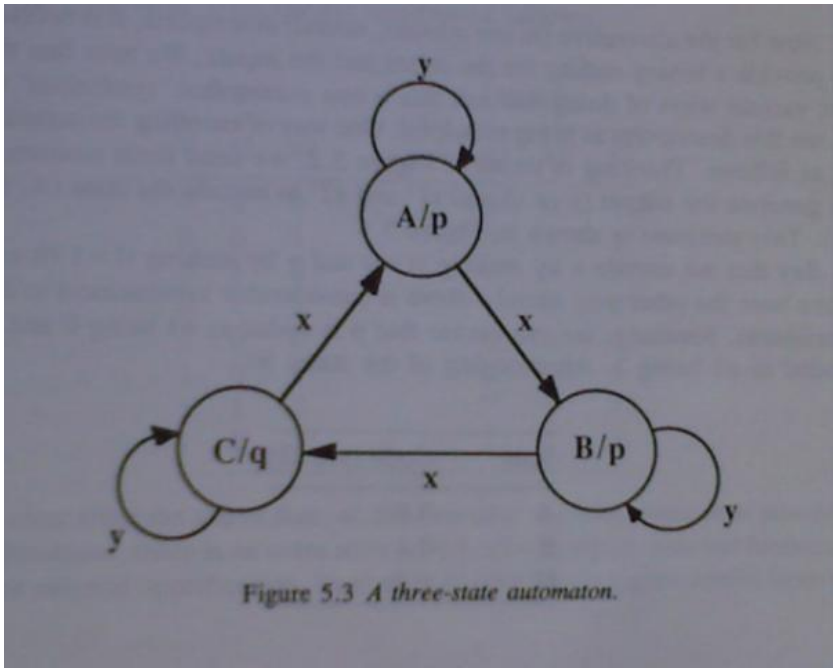


Figure 5.3 *A three-state automaton.*

Level of abstraction 1: NN - 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)

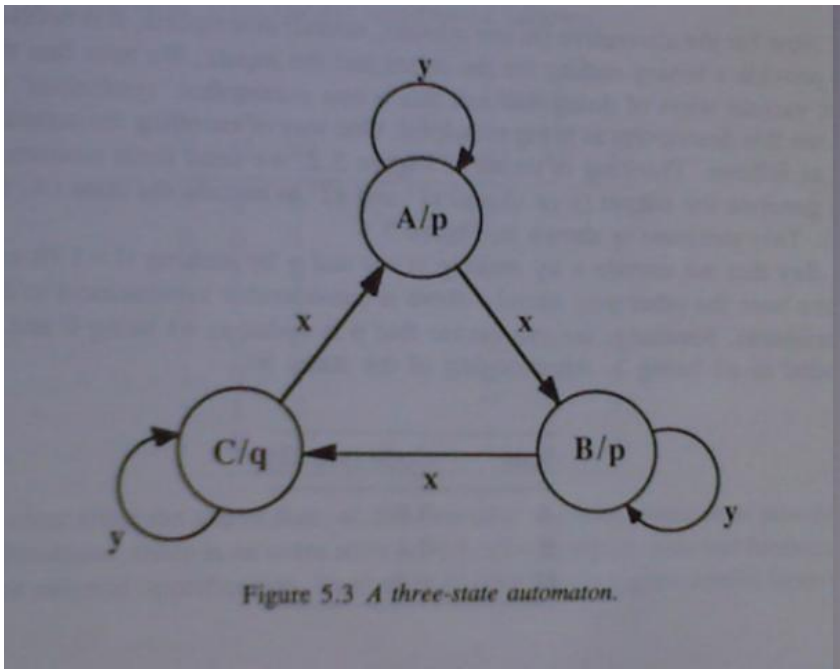


Figure 5.3 *A three-state automaton.*

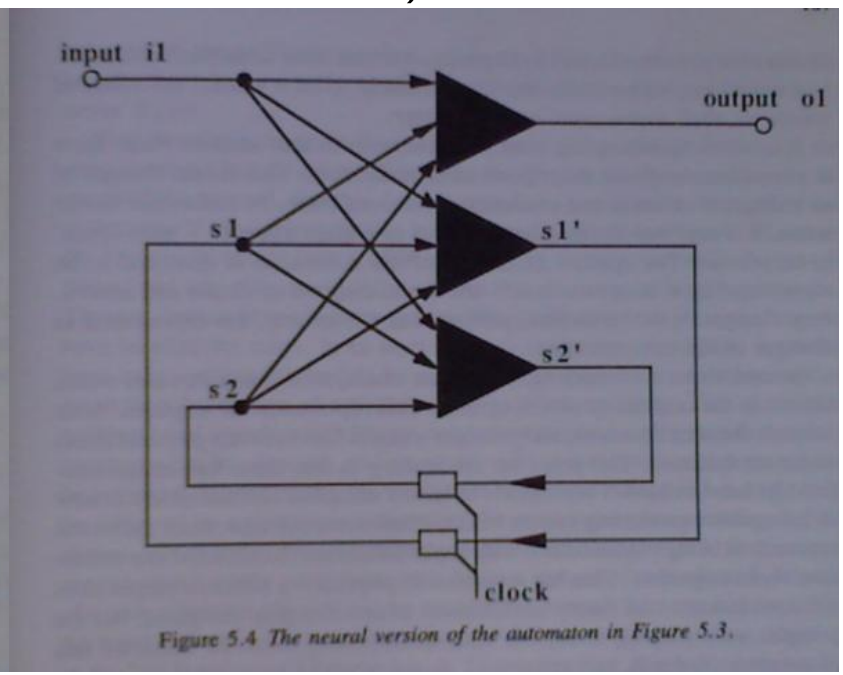


Figure 5.4 *The neural version of the automaton in Figure 5.3.*

Logic and NNs: summary [Siegelmann]

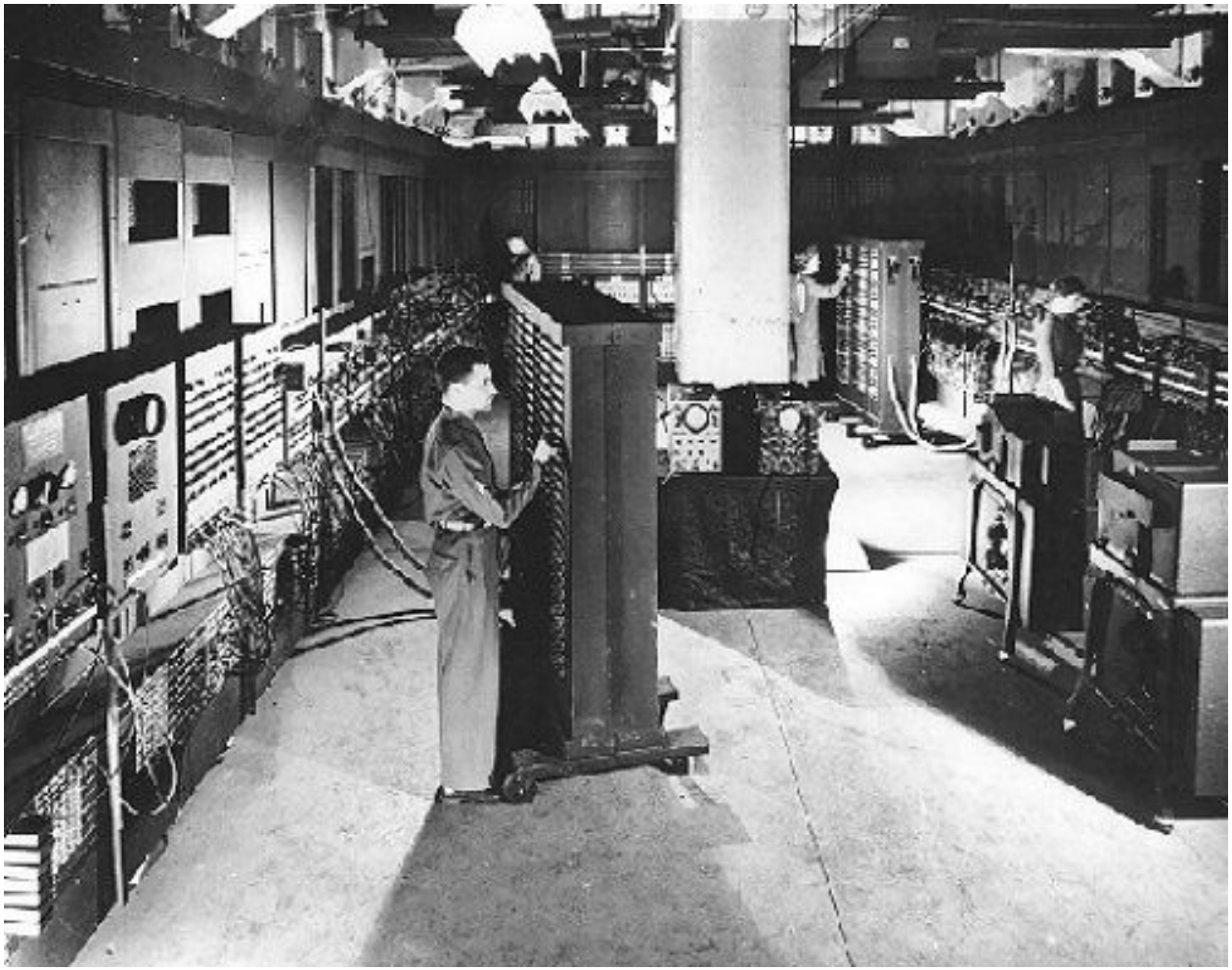
Finite Automata $\longrightarrow$ Binary threshold networks
Turing Machines $\longrightarrow$ Neural networks with rational weights
Probabilistic Turing Machines $\longrightarrow$ NNs with rational weights
Super-turing computations $\longrightarrow$ NNs with real weights.

Early Digital and Neuro computers:

In 1946, the first useful electronic digital computer (ENIAC) is created: it was a happy start for the programmed computing.

Early Digital and Neuro computers:

In 1946, the first useful electronic digital computer (ENIAC) is created: it was a happy start for the programmed computing.



First Engineering insights:

Mark 1 and Mark 2 Perceptrons (1948 - 1958)

Levels of Abstraction: from 1 to 2

The results we have mentioned form the 1st, theoretical, level of abstraction. They are general and powerfull 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. They are general and powerfull 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...

Level of Abstration 2. Neuro-Symbolic Networks and Logic Programs [Holldobler & al.]

Logic Programs

- $A \leftarrow B_1, \ldots, B_n$

Level of Abstration 2. Neuro-Symbolic Networks and Logic Programs [Holldobler & al.]

Logic Programs

- $A \leftarrow B_1, \ldots, B_n$
- $T_P(I) = \{A \in B_P : \ A \leftarrow B_1, \ldots, B_n$ is a ground instance of a clause in P and $\{B_1, \ldots, B_n\} \subseteq I\}$

Level of Abstration 2. Neuro-Symbolic Networks and Logic Programs [Holldobler & al.]

Logic Programs

- $A \leftarrow B_1, \ldots, B_n$
- $T_P(I) = \{A \in B_P : \ A \leftarrow B_1, \ldots, B_n$ is a ground instance of a clause in P and $\{B_1, \ldots, B_n\} \subseteq I\}$
- $\mathtt{lfp}(T_P \uparrow \omega) =$ the least Herbrand model of P.

Theorem

For each propositional program P, there exists a 3-layer feedforward neural network that computes T_P.

- No learning or adaptation;
- Require infinitely long layers in the first-order case.

A Simple Example

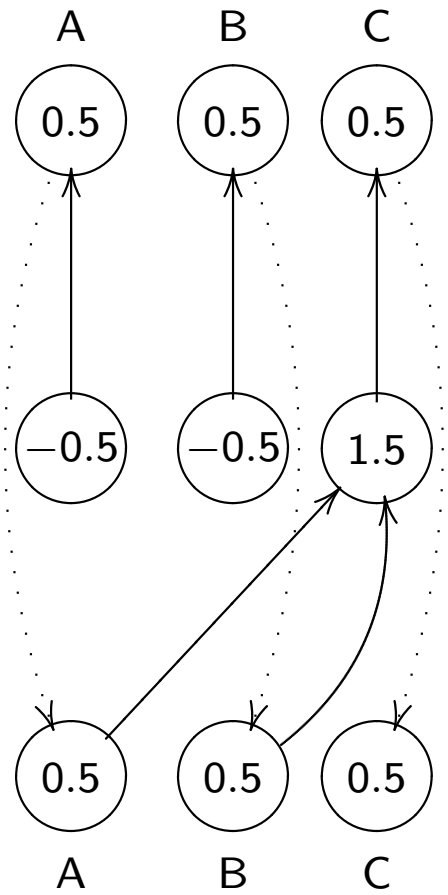
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

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

A Simple Example

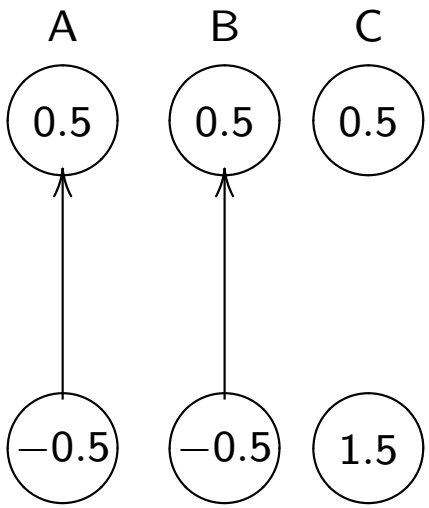
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

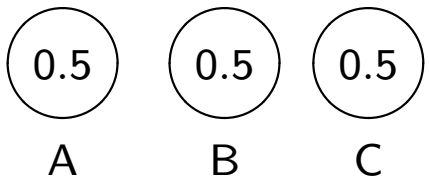
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



A Simple Example

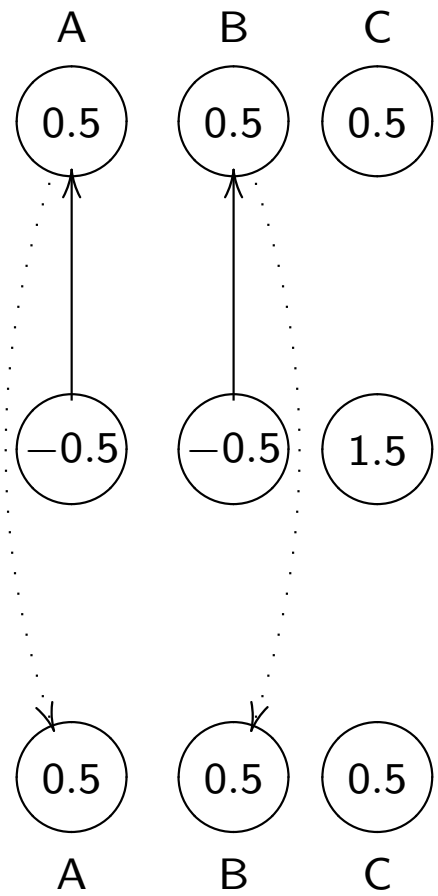
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$





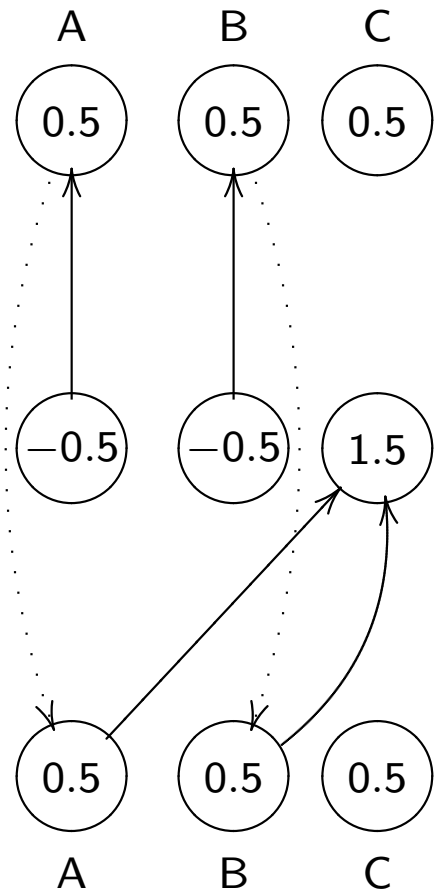
A Simple Example

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



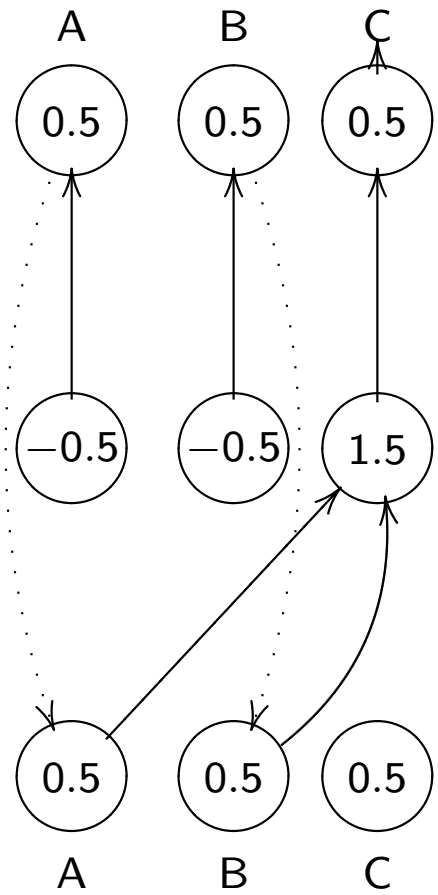
A Simple Example

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



A Simple Example

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



Another Example: First-Order Case

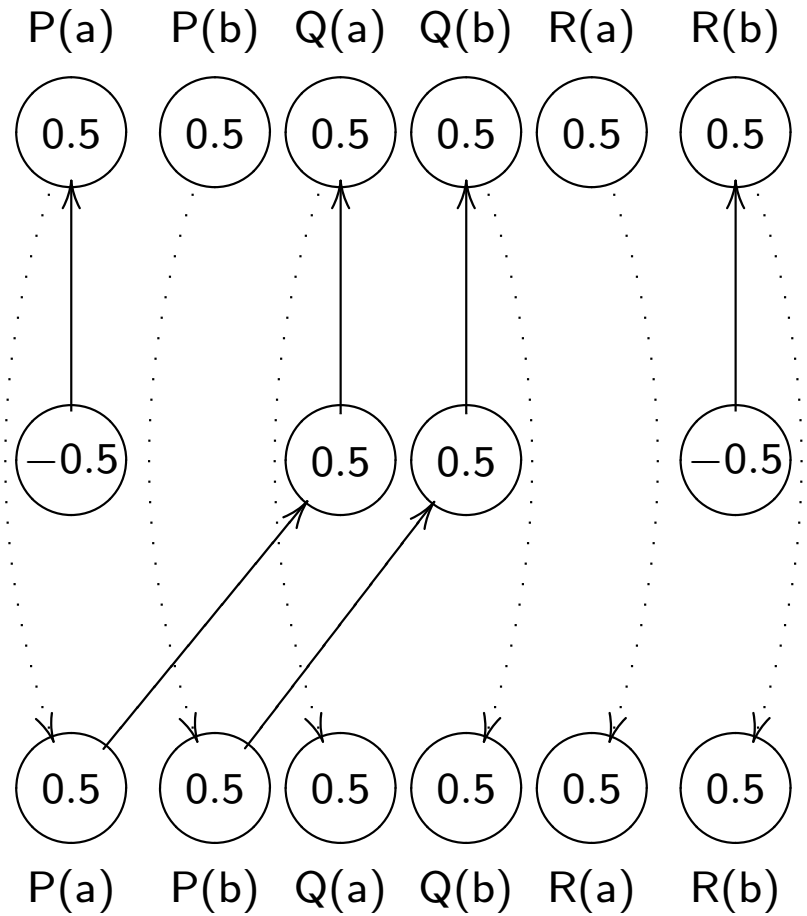
$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$

Another Example: First-Order Case

$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

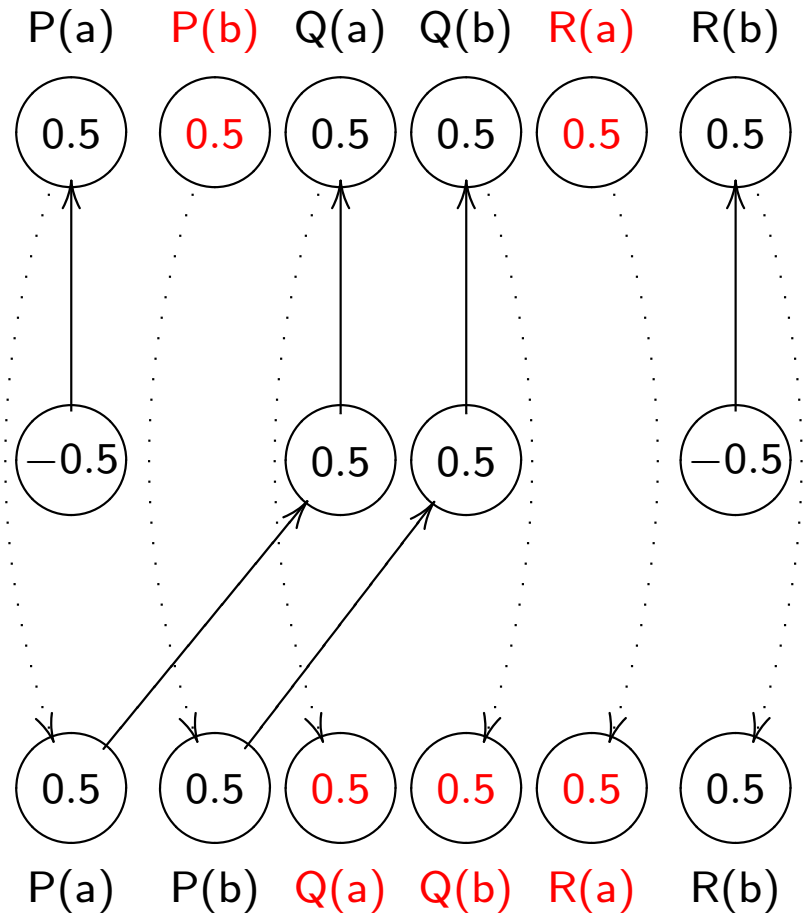
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

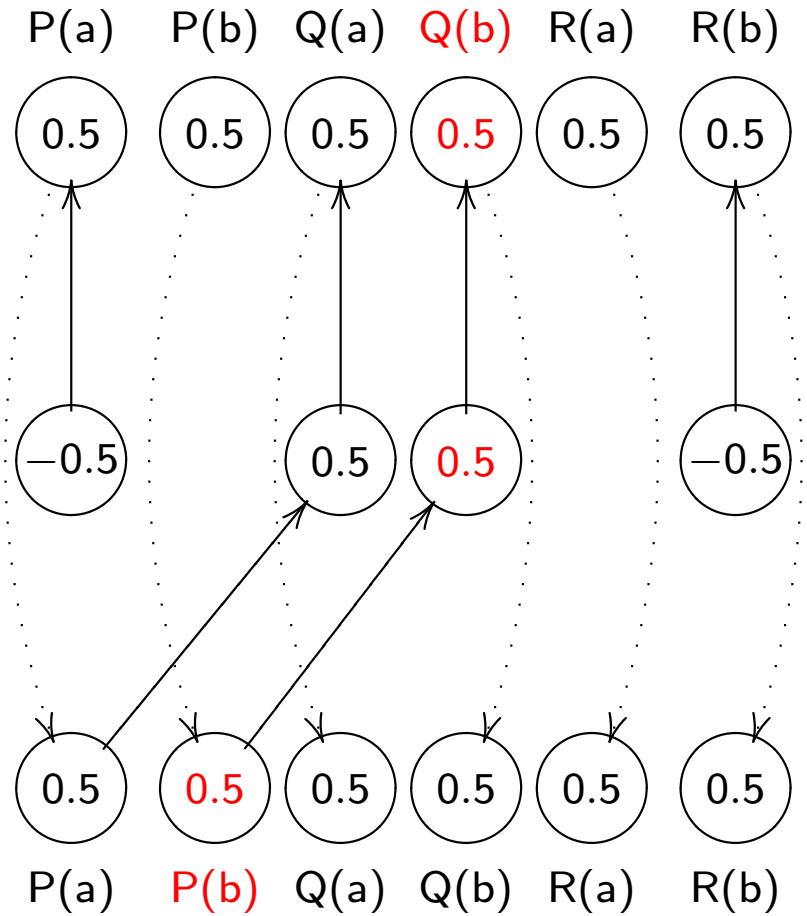
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

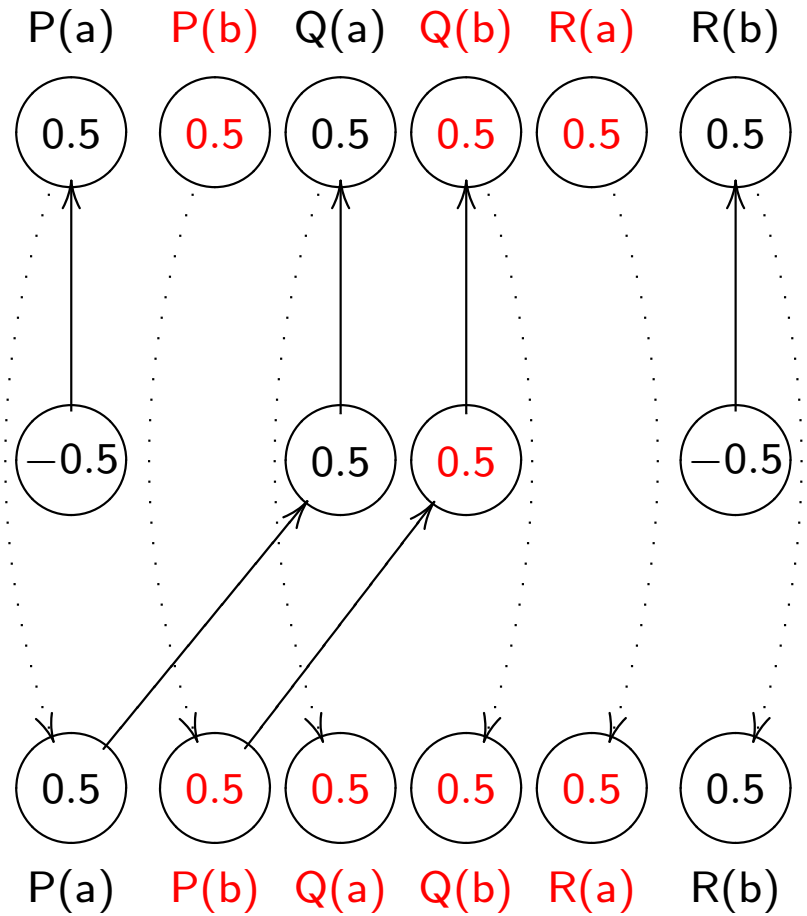
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

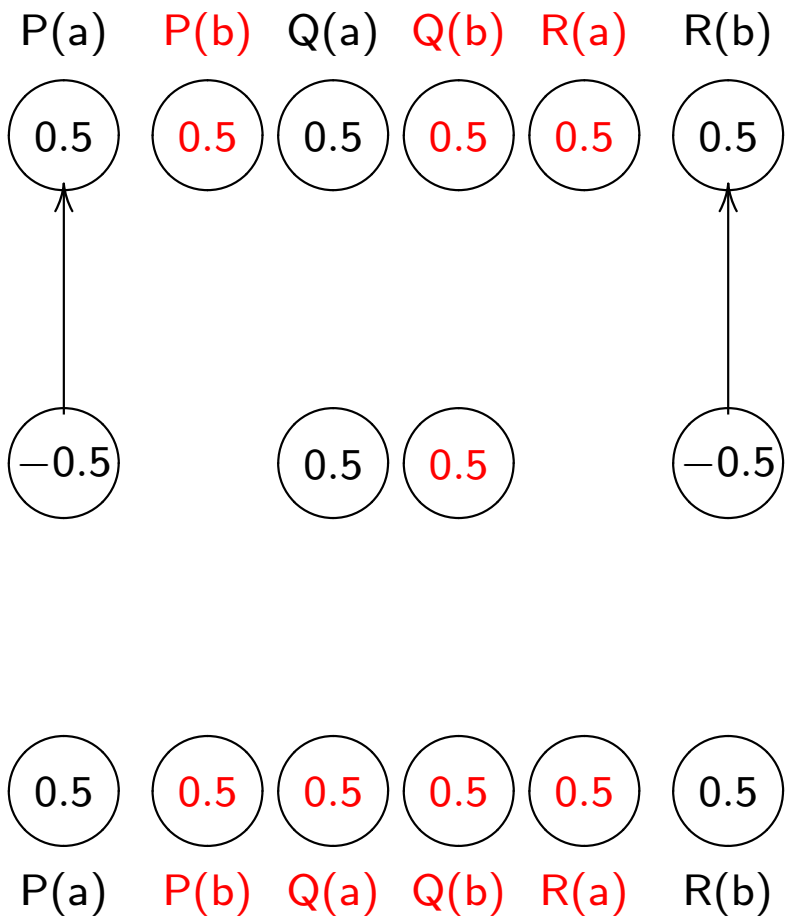
$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



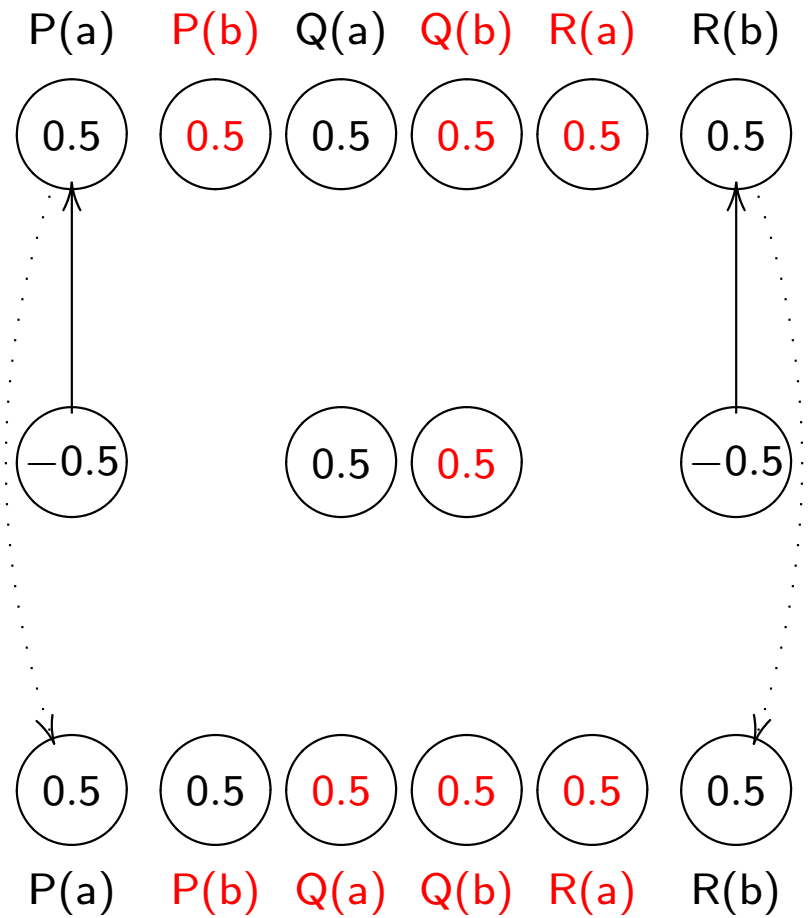
Another Example: First-Order Case

$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



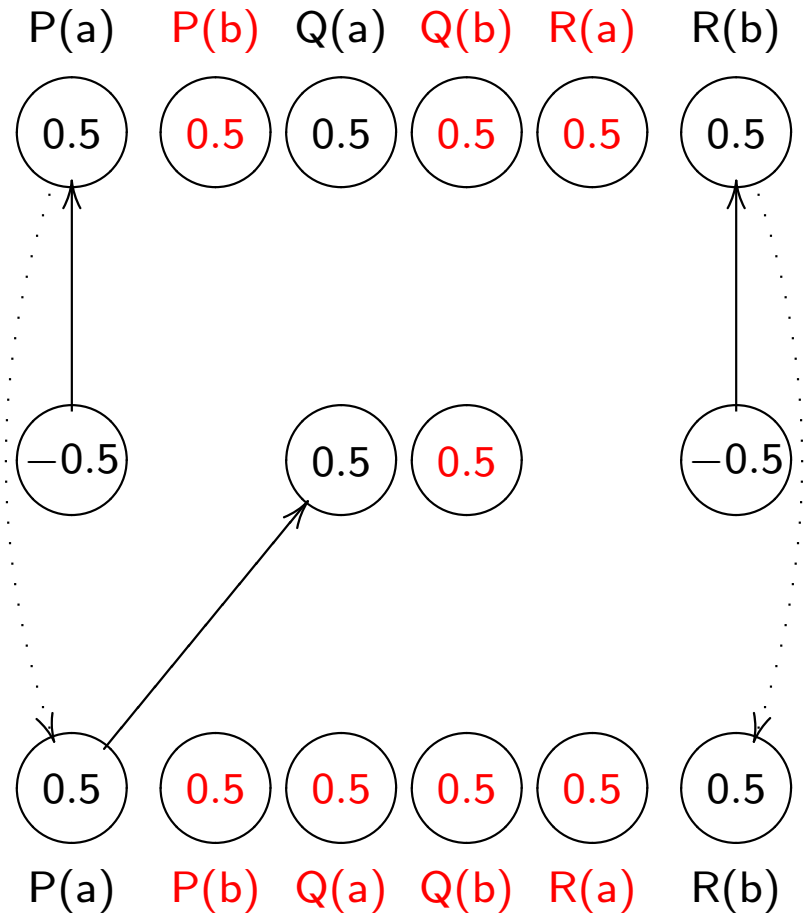
Another Example: First-Order Case

$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



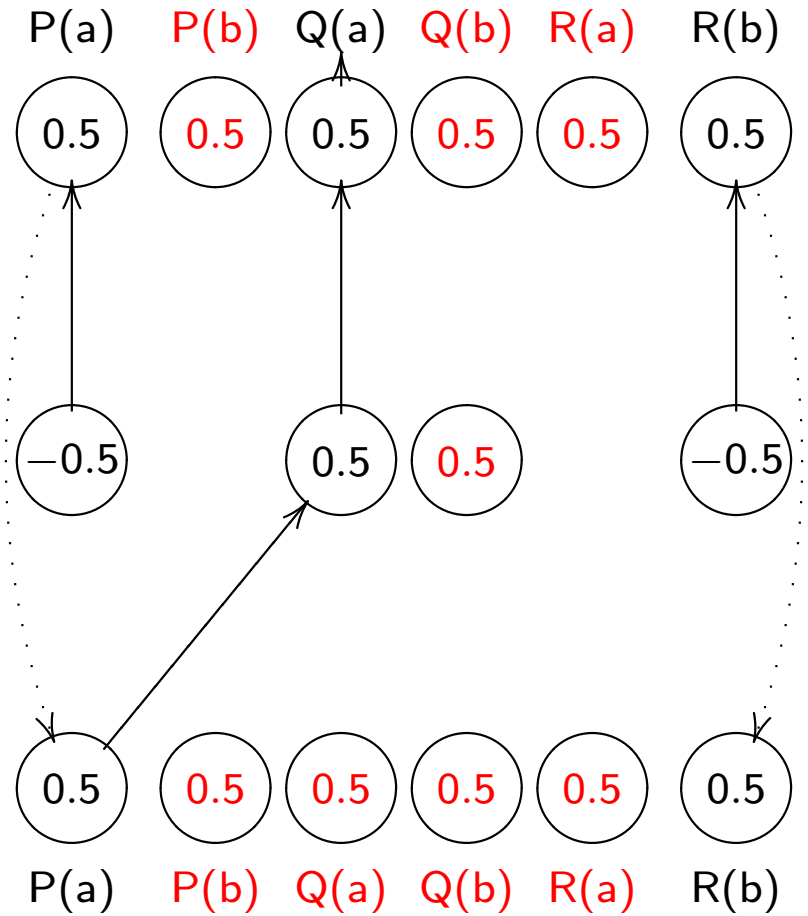
Another Example: First-Order Case

$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Example 3

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

$T_P \uparrow 0 = \{P(0)\}$
$lfp(T_P) = T_P \uparrow \omega =$
$\{0, s(0), s(s(0)),$
$s(s(s(0))), \ldots\}$

Example 3

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

$T_P \uparrow 0 = \{P(0)\}$
$lfp(T_P) = T_P \uparrow \omega =$
$\{0, s(0), s(s(0)),$
$s(s(s(0))), \ldots\}$

Example 3

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

$T_P \uparrow 0 = \{P(0)\}$
$lfp(T_P) = T_P \uparrow \omega =$
$\{0, s(0), s(s(0)),$
$s(s(s(0))), \ldots\}$
Paradox:
(computability,
complexity,
proof theory)

Three characteristic properties of T_P-neural networks.

1. The number of neurons in the input and output layers is the number of atoms in the Herbrand base B_P.
2. Signals are binary, and this provides the computations of truth value functions $\wedge$ and $\leftarrow$.
3. First-order atoms are not presented in the neural network directly, and only truth values 1 and 0 are propagated.

Three characteristic properties of T_P-neural networks.

1. The number of neurons in the input and output layers is the number of atoms in the Herbrand base B_P.
2. Signals are binary, and this provides the computations of truth value functions $\wedge$ and $\leftarrow$.
3. First-order atoms are not presented in the neural network directly, and only truth values 1 and 0 are propagated.

Three main implications. The networks can't:

1. ... deal with recursive programs, that is, programs that can have infinitely many ground instances.
2. ... deal with non-ground reasoning, which is very common in computational logic.
3. ... cover proof-theoretic aspect, only model-theoretic one...

Neuro-symbolic architectures of other kinds:

This problem of processing ground instances of terms instead of terms, using model theory instead of proof theory causes the same problem in most (if not all), the existing Neuro-Symbolic systems, e.g.:

- 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.

As a result, basic algorithms and techniques of computational logic, such as term rewriting or first-order unification have not received any implementation in Neuro-Symbolic networks.

Get methodology right

1 Do we need to implement computational logic in NNs? Why?

2. Do we need the implementation to mimic what happens in the brain or we aim for efficiency irrespective of cognitive plausibility?

3. What is learning: is it a change of parameters of a given system (technical view) or it is the process of acquiring new knowledge?

Get methodology right

1 Do we need to implement computational logic in NNs? Why?
Yes, because we wish to use logic in hybrid neuro-symbolic systems of the future; and because parallelism can bring some speed-up.
2. Do we need the implementation to mimic what happens in the brain or we aim for efficiency irrespective of cognitive plausibility?

3. What is learning: is it a change of parameters of a given system (technical view) or it is the process of acquiring new knowledge?

Get methodology right

1 Do we need to implement computational logic in NNs? Why?
Yes, because we wish to use logic in hybrid neuro-symbolic systems of the future; and because parallelism can bring some speed-up.
2. Do we need the implementation to mimic what happens in the brain or we aim for efficiency irrespective of cognitive plausibility?
We aim for efficiency, and resource-consciousness.
3. What is learning: is it a change of parameters of a given system (technical view) or it is the process of acquiring new knowledge?

Get methodology right

1 Do we need to implement computational logic in NNs? Why?
Yes, because we wish to use logic in hybrid neuro-symbolic systems of the future; and because parallelism can bring some speed-up.
2. Do we need the implementation to mimic what happens in the brain or we aim for efficiency irrespective of cognitive plausibility?
We aim for efficiency, and resource-consciousness.
3. What is learning: is it a change of parameters of a given system (technical view) or it is the process of acquiring new knowledge?
Learning is taken technically - as a process of changing of the essential parameters of a system. Stated like this, deduction and learning are not contradicting terms.

Main assumptions:

1. Symbols can be processed at the same level of abstraction as numbers; 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 learning algorithms of neurocomputing.
5. Learning Functions one uses in Neuro-Symbolic networks can be arbitrary, not necessarily the conventional (arithmetic and statistical) functions of neurocomputing.

Example 1. Parallel (Term) Rewriting

Consider a string [1:2:3:1:2:3:3:1:2:3:1:2] and ground instantiations of the rewriting rule $x \longrightarrow 3x$

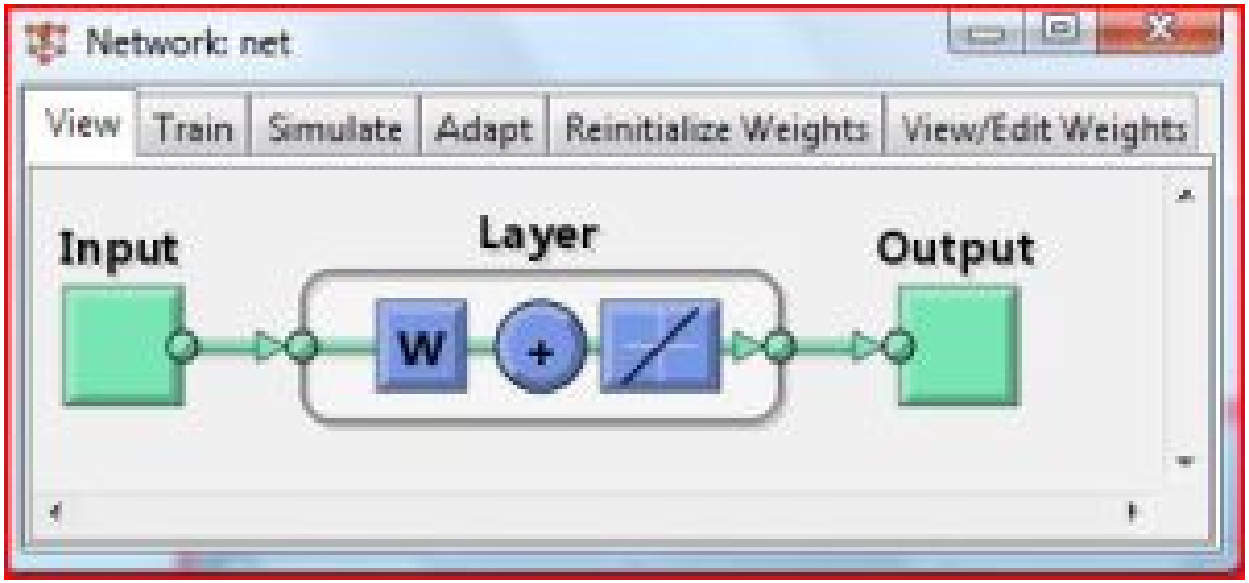
The parallel rewriting step will give us
$[3:6:9:3:6:9:9:3:6:9:3:6]$.

Example 1. Parallel (Term) Rewriting

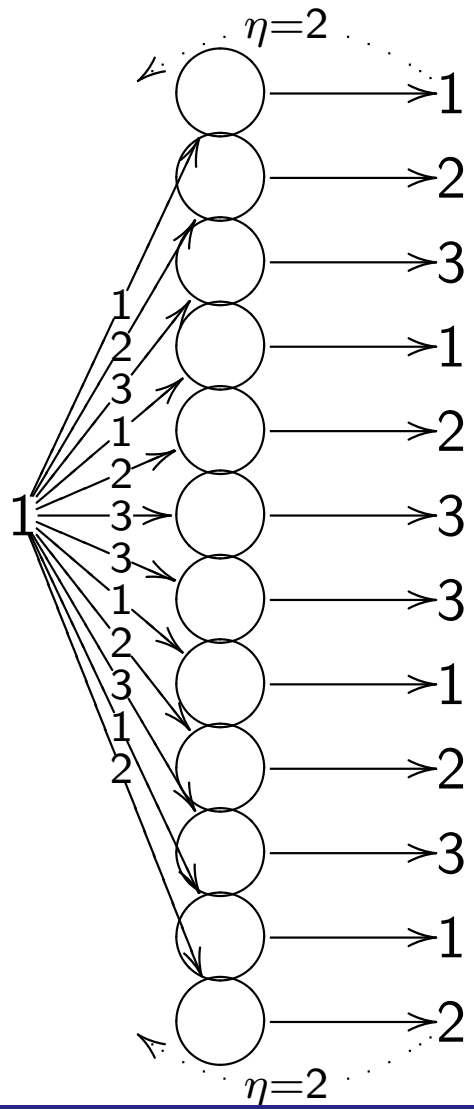
Consider a string [1:2:3:1:2:3:3:1:2:3:1:2] and ground instantiations of the rewriting rule $x \longrightarrow 3x$

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

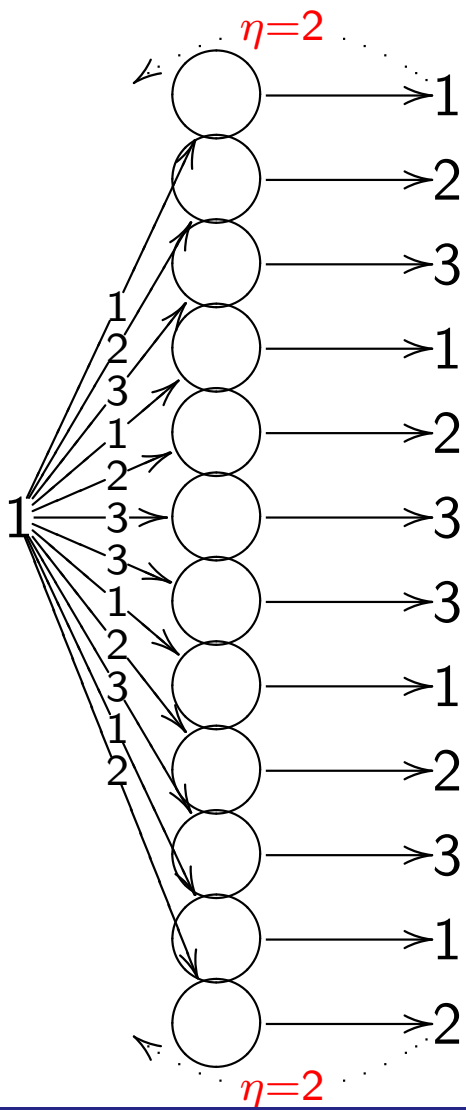
This can be done in unsupervised learning network `net`: the rate of learning $\eta = 2$; $\Delta\mathbf{w} = \eta\mathbf{yx} = 2\mathbf{w}$; $\mathbf{w}^{\mathtt{new}} = \mathbf{w} + \Delta\mathbf{w} = 3\mathbf{w}$.



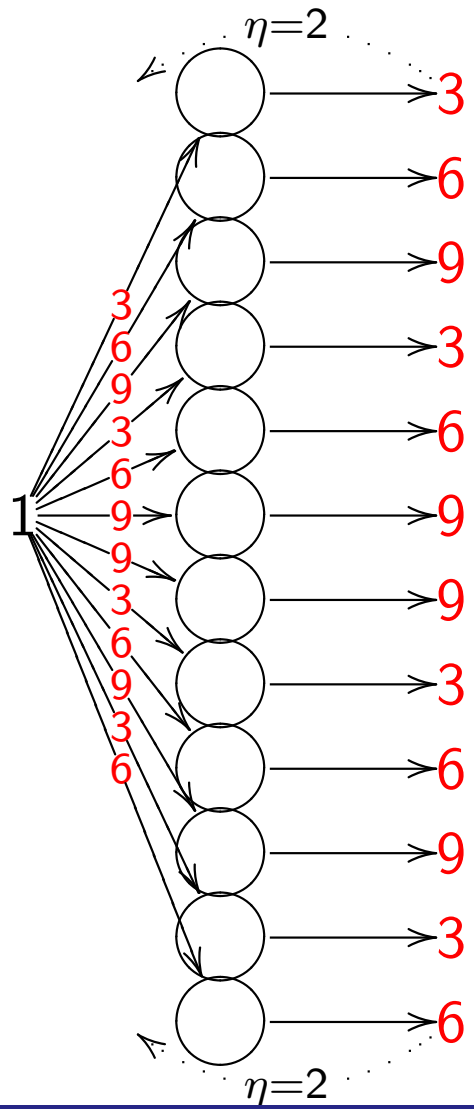
Parallel rewriting - a closer look:



Parallel rewriting - a closer look:



Parallel rewriting - a closer look:



Example 2. Unification

Consider two atoms $P(x)$ and $P(a)$.

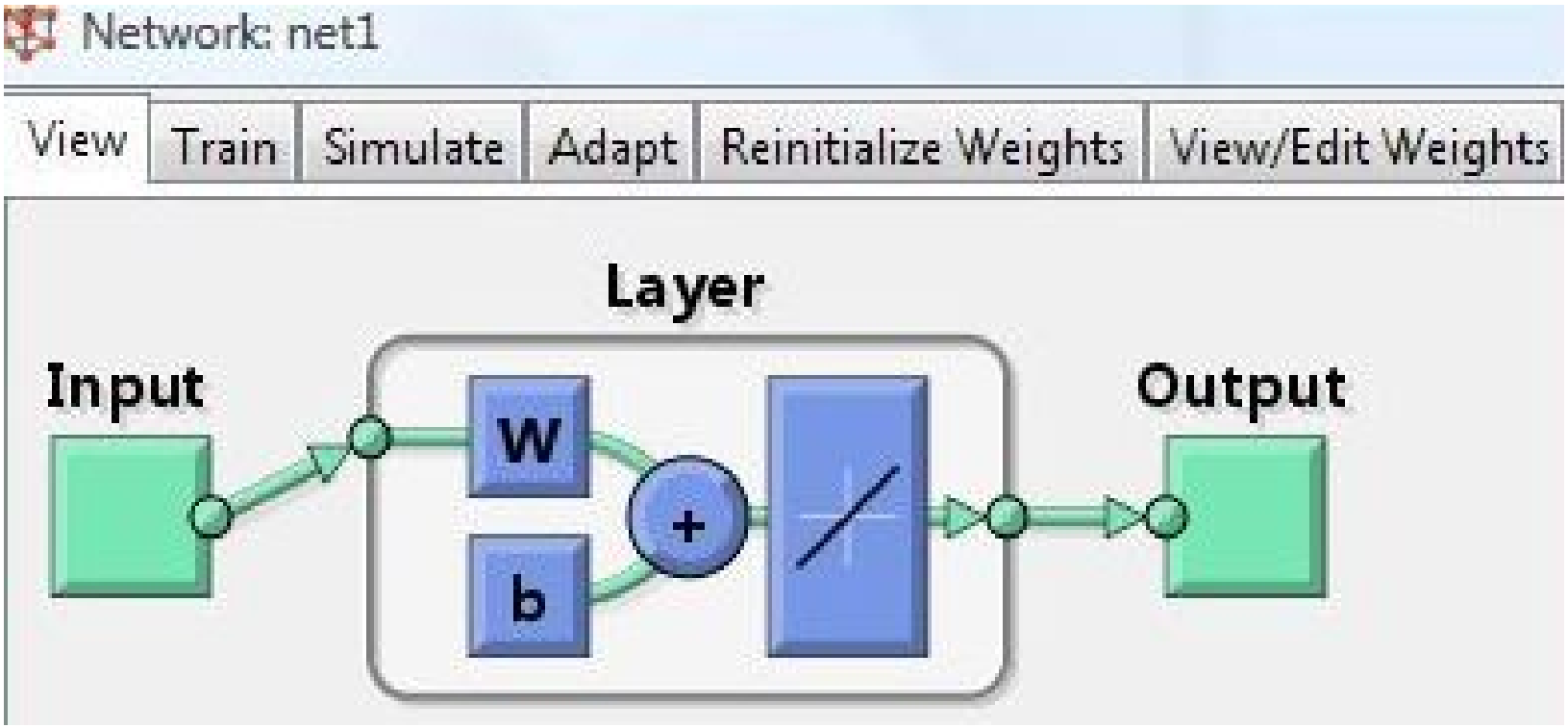
The most general unifier for them will be x/a, which would bring both to the form $P(a)$.

Example 2. Unification

Consider two atoms $P(x)$ and $P(a)$.

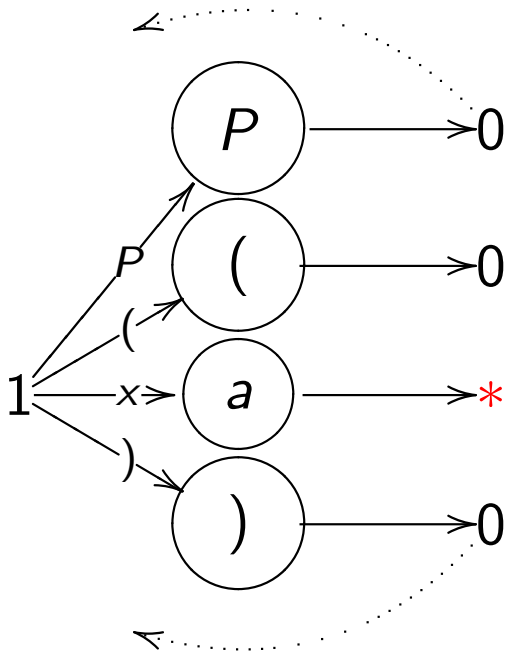
The most general unifier for them will be x/a, which would bring both to the form $P(a)$.

This can be done in error-correction learning network `net1`:



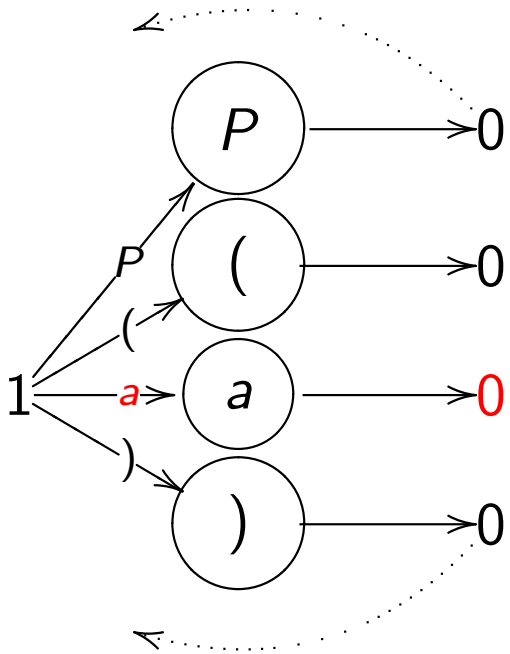
Unification - a closer look:

Target vector is $0; 0; 0; 0$.



Unification - a closer look:

Target vector is $0; 0; 0; 0$.



Conclusion

The simple examples we have shown are all tested using conventional neural networks with corresponding learning functions. However, if we take algorithms of term-rewriting and unification in their full generality, there are many details that need to be taken care of. For example, occurrence check, growth of terms in the process of unification and rewriting, etc.
This is why, the learning functions we use need to be more clever than just arithmetic operations...
The definitions of such functions and the full MATLAB development of the networks suitable for arbitrary complex parallel term-rewriting and unification can be found on the webpage of the project http://www.cs.st-andrews.ac.uk/ ek/CLANN/.

Last Sentence

Neurons OR Symbols: Why does OR remain exclusive?

The Neuro-Symbolic networks (of Level 2) are not "symbolic" enough, if at all. There should be made steps to allow certain symbolic and neural features mix, otherwise the resulting hybrid systems will not be able to compete with the state-of-the-art techniques of computational logic.

Lift the restrictions imposed by architectures from the Level 1, change the attitude to learning functions.

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.