Computational Logic and Neural Networks: a personal perspective

Ekaterina Komendantskaya

School of Computing, University of Dundee

9 February 2011, Aberdeen

Outline

Outline

Outline

Outline

Outline

Outline

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, gold medal for excellency).
In 2004-2007 - PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)
2007- 2008 - My first postdoc was with Yves Bertot in INRIA, France - on guardedness of corecursive functions in Coq.
2008 - 2011 - EPSRC research fellowship CLANN first in St Andrews, later transferred to the School of Computing in Dundee.
My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning)
- Higher-order Interactive Theorem Provers
- Neuro-Symbolic networks
- Categorical Semantics of Computations

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, gold medal for excellency).
In 2004-2007 - PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)
2007- 2008 - My first postdoc was with Yves Bertot in INRIA, France - on guardedness of corecursive functions in Coq.
2008 - 2011 - EPSRC research fellowship CLANN first in St Andrews, later transferred to the School of Computing in Dundee.
My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning) (PhD thesis)
- Higher-order Interactive Theorem Provers
- Neuro-Symbolic networks
- Categorical Semantics of Computations

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, gold medal for excellency).
In 2004-2007 - PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)
2007- 2008 - My first postdoc was with Yves Bertot in INRIA, France - on guardedness of corecursive functions in Coq.
2008 - 2011 - EPSRC research fellowship CLANN first in St Andrews, later transferred to the School of Computing in Dundee.
My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning) (PhD thesis)
- Higher-order Interactive Theorem Provers (Postdoc in INRIA)
- Neuro-Symbolic networks
- Categorical Semantics of Computations

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, gold medal for excellency).
In 2004-2007 - PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)
2007- 2008 - My first postdoc was with Yves Bertot in INRIA, France - on guardedness of corecursive functions in Coq.
2008 - 2011 - EPSRC research fellowship CLANN first in St Andrews, later transferred to the School of Computing in Dundee.
My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning) (PhD thesis)
- Higher-order Interactive Theorem Provers (Postdoc in INRIA)
- Neuro-Symbolic networks (PhD Thesis, current EPSRC fellowship)
- Categorical Semantics of Computations

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, gold medal for excellency).
In 2004-2007 - PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)
2007- 2008 - My first postdoc was with Yves Bertot in INRIA, France - on guardedness of corecursive functions in Coq.
2008 - 2011 - EPSRC research fellowship CLANN first in St Andrews, later transferred to the School of Computing in Dundee.
My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning) (PhD thesis)
- Higher-order Interactive Theorem Provers (Postdoc in INRIA)
- Neuro-Symbolic networks (PhD Thesis, current EPSRC fellowship)
- Categorical Semantics of Computations (in parallel to the above)

School of Computing, University of Dundee

- Assistive and healthcare technologies;
- Computational systems (Computer Vision; Theory of Argumentation);
- Interactive systems design;
- Space technology centre.

Computational Logic in Neural Networks

Symbolic Logic as Deductive System

- Deduction in logic calculi;
- Logic programming;
- Higher-order proof assistants...

Sound symbolic methods we can trust

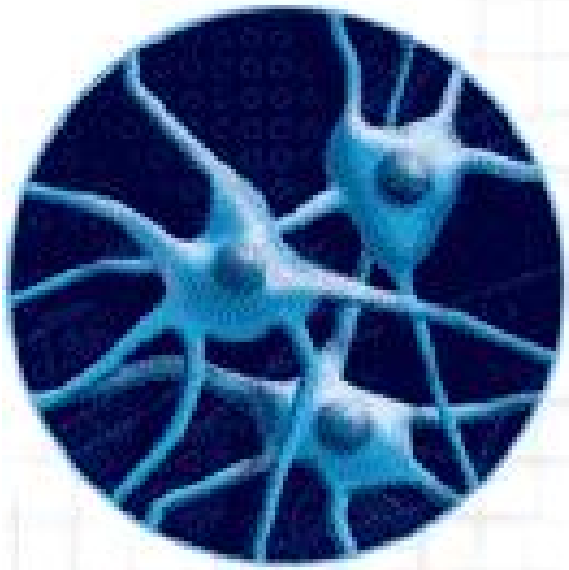
Computational Logic in Neural Networks

Symbolic Logic as Deductive System

- Deduction in logic calculi;
- Logic programming;
- Higher-order proof assistants...

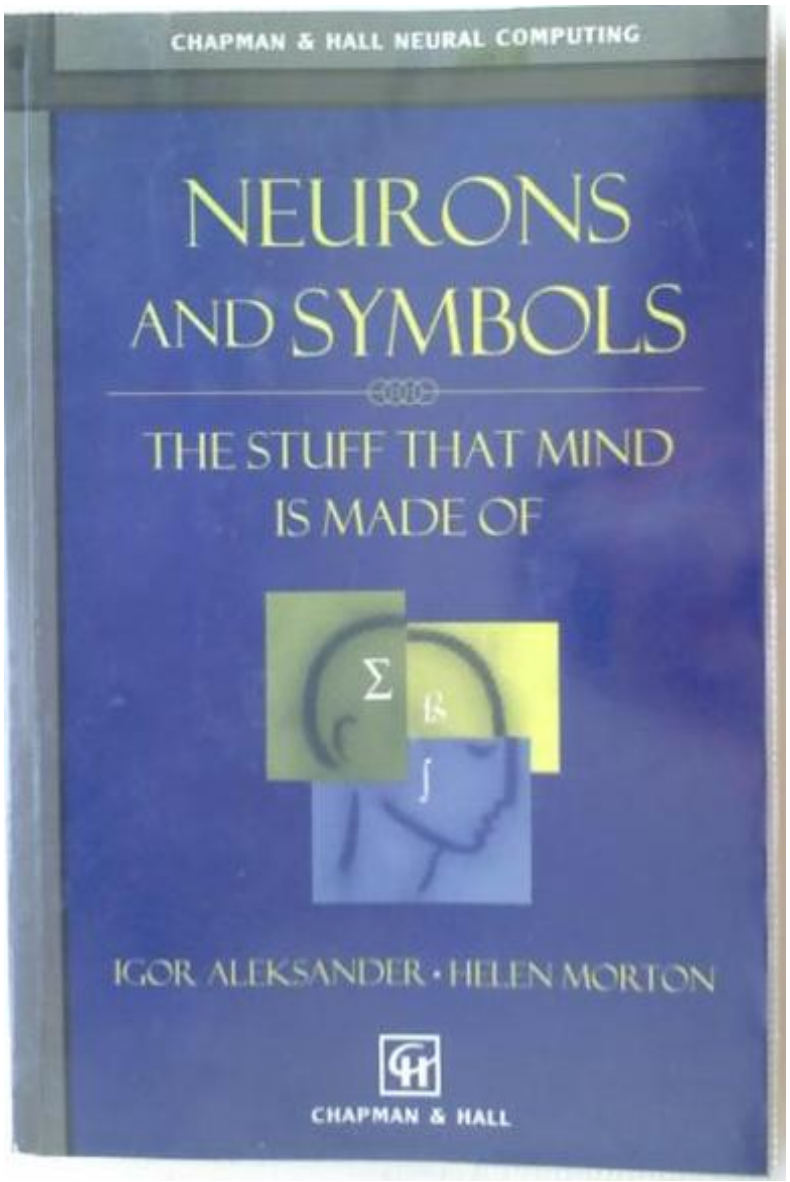
Sound symbolic methods we can trust

Neural Networks



- spontaneous behavior;
- learning and adaptation;
- parallel computing.

Neurons AND Symbols: a long story



Many attempts have been made to merge, integrate, unite, etc.etc... the two.

On the picture: **"Neurons and Symbols"**, [Alexander and Morton, 1993] claims that there is no, and should not be, divide between symbolic and neural computing.

Symbolic and subsymbolic levels

Divide! [Smolensky, 2006]

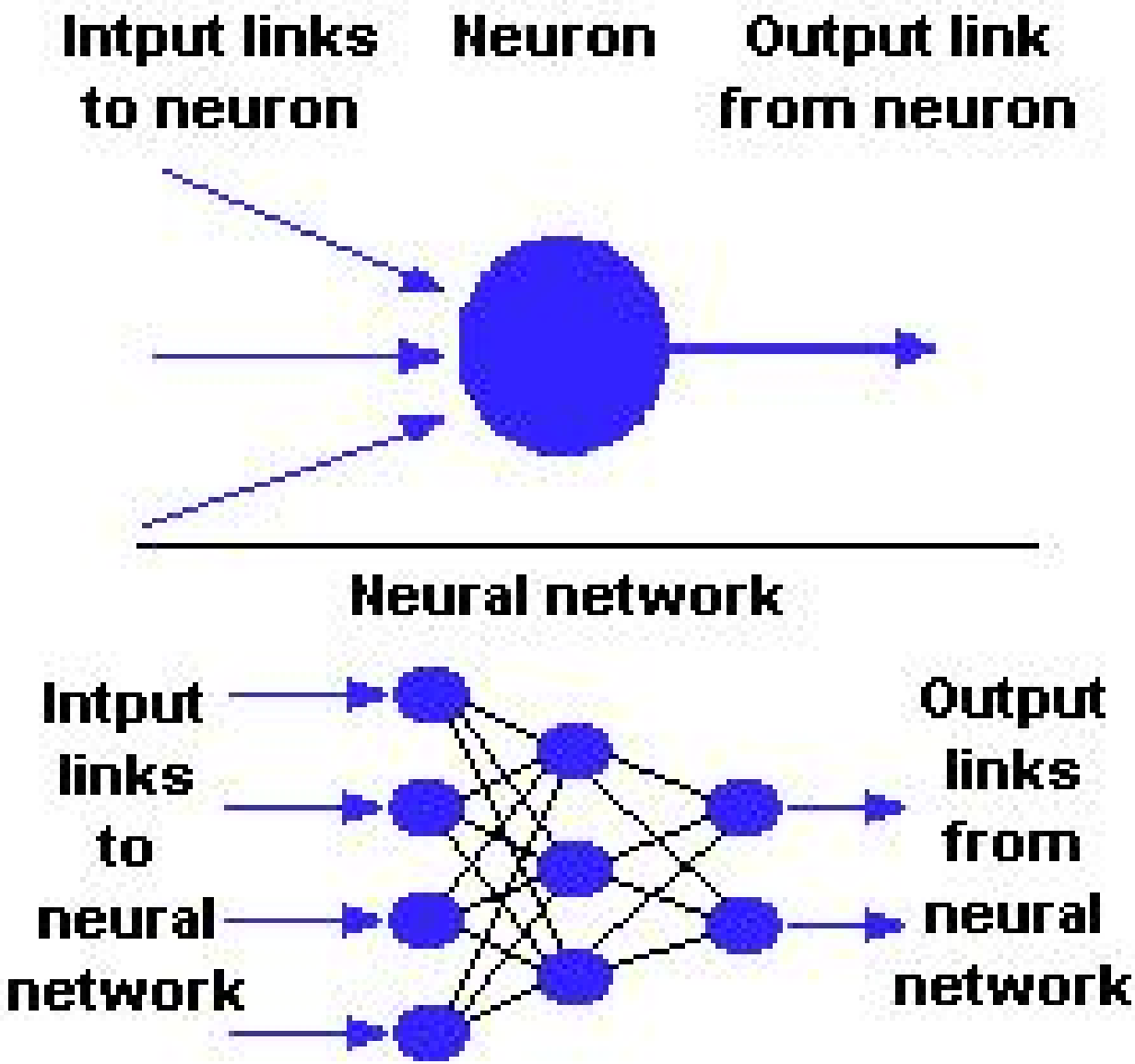
Connectionism often divides the two cognitive functions of neural network: on a subsymbolic level, we have primitive, massive and parallel neurons, on a symbolic level, their massive and chaotic actions form a symbolic, structured, reasoning. Analogy: atoms in physics - have one sort of properties on micro-level, the objects we deal with on macro-level are made of atoms, but obey different laws of physics.

Not exactly proven by experiments with the real brain neurons...

No divide! [Alexander, 1993]

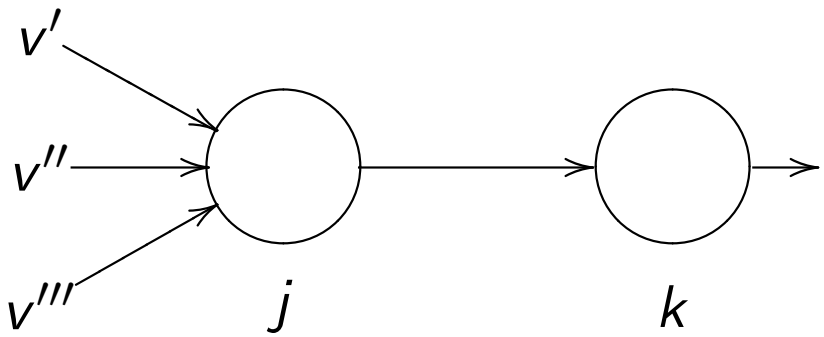
The rival approach claims that the divide is artificial: both styles of computing can be seen as variants of one computational technique - the use of state machines. Thus Neurocomputing is totally compatible with symbolic (or Turing) style.

Neural Network: definitions



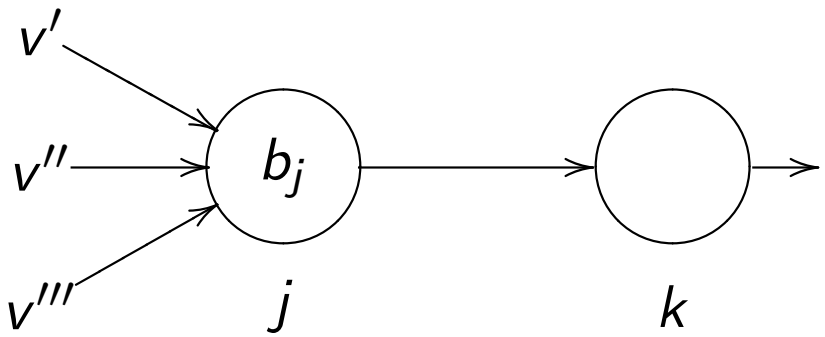
Neurons

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



Neurons

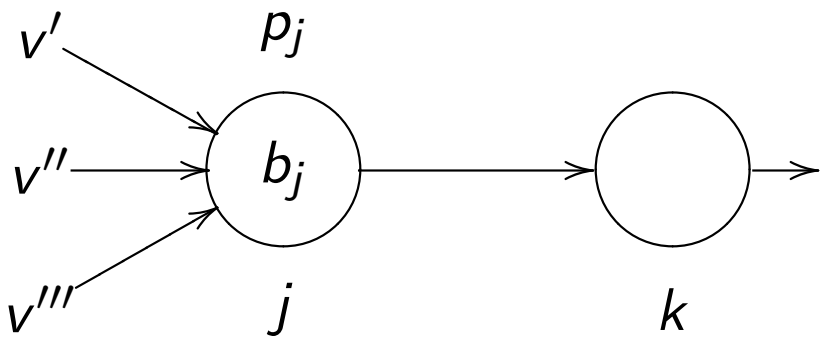
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

Neurons

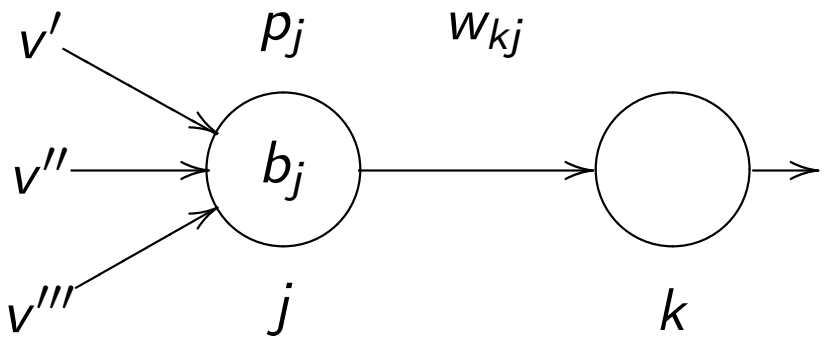
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

Neurons

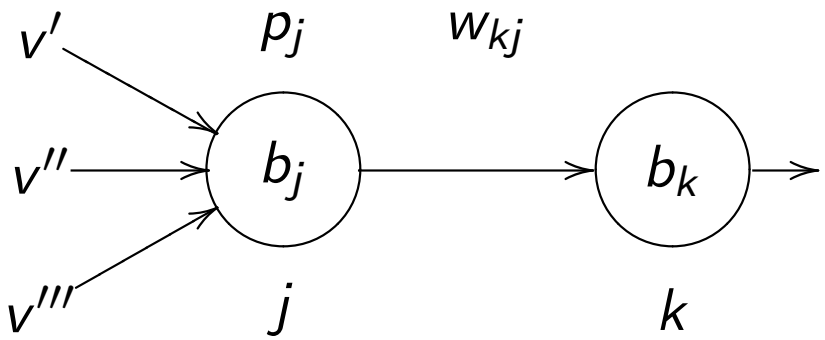
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

Neurons

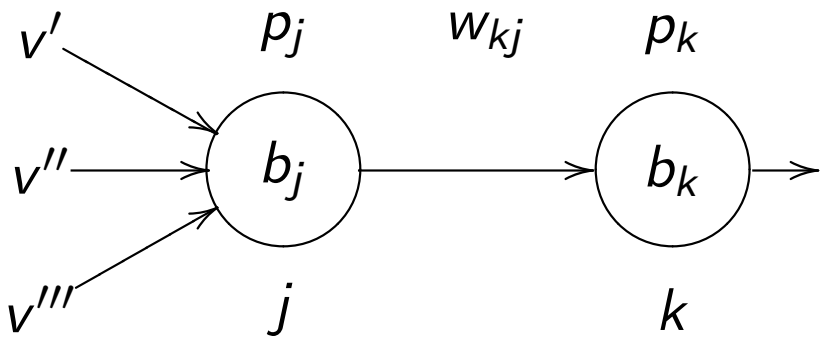
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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

Neurons

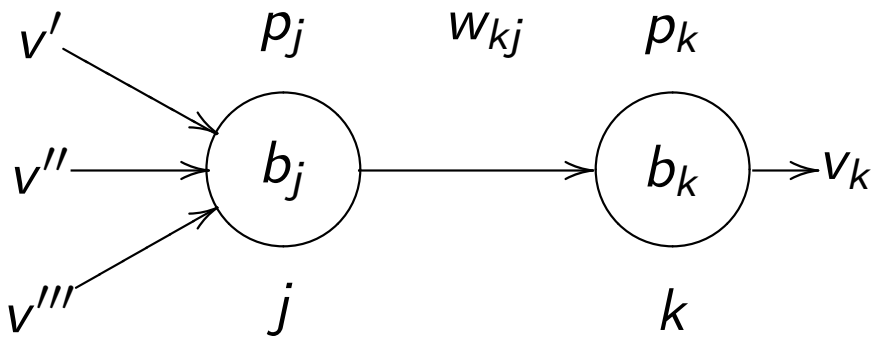
$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) - b_k\right)$$
$$v_k(t + \Delta t) = \psi(p_k(t))$$



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.

First-order syntax

We fix the *alphabet* **A** to consist of

- constant symbols a, b, c, possibly with finite subscripts;
- variables $x, y, z, x_1 \ldots$;
- function symbols $f, g, h, f_1 \ldots$, with arities;
- predicate symbols $P, Q, R, P_1, P_2 \ldots$, with arities;

Term:

$$t = a \mid x \mid f(t)$$

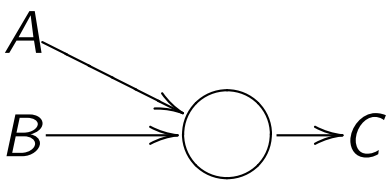
Atomic Formula: $At = P(t_1, \ldots, t_n)$, where P is a predicate symbol of arity n and t_i is a term.

Logic and networks

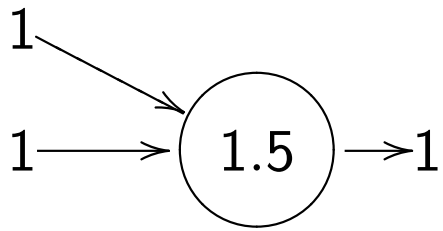
Among early results relating logic and neural networks were:

- 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.

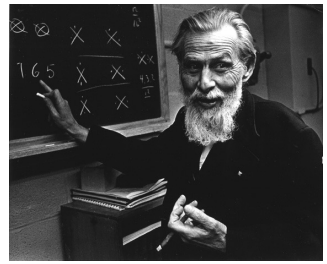
Boolean Networks of McCullogh and Pitts, 1943.

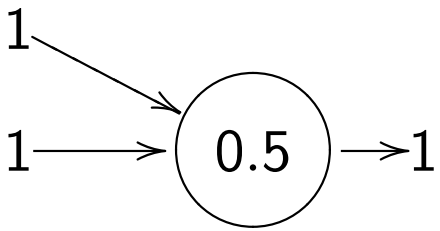


If A and B then C.

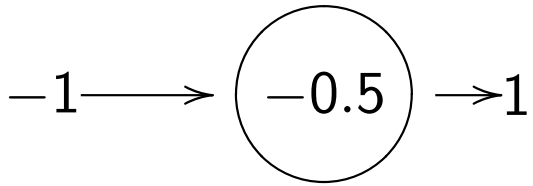


$(A = 1)$ and $(B = 1)$.





$(A = 1)$ 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)

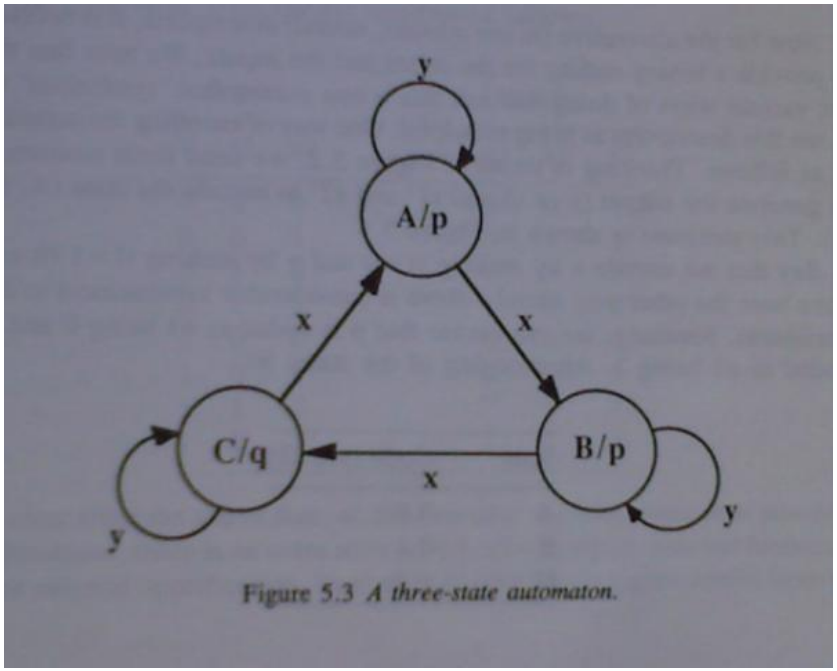


Figure 5.3 *A three-state automaton.*

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)

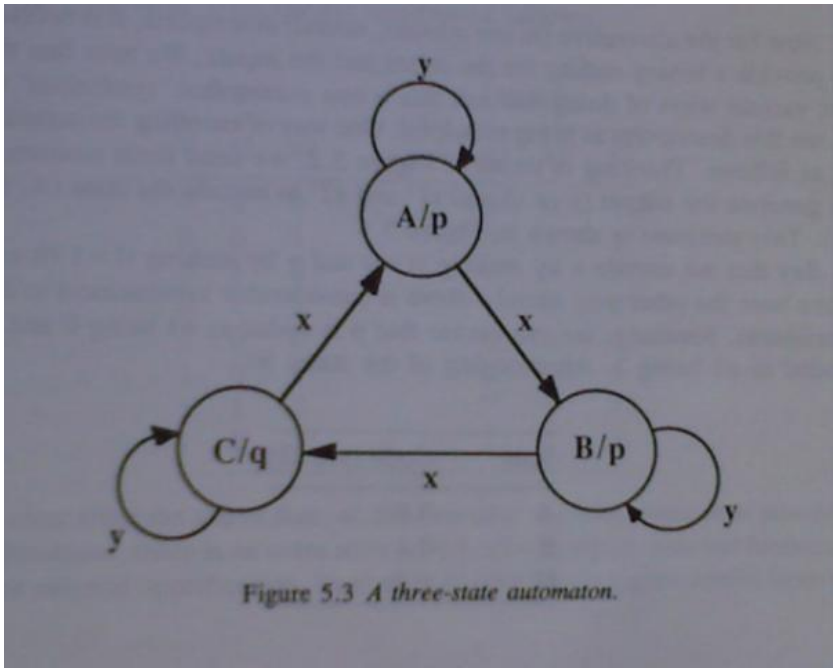


Figure 5.3 *A three-state automaton.*

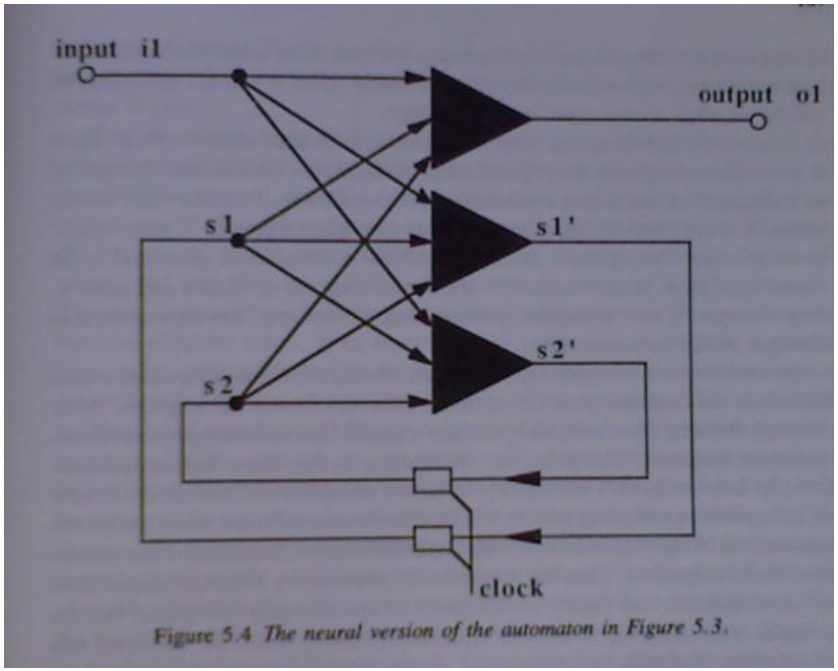


Figure 5.4 *The neural version of the automaton in Figure 5.3.*

Logic and NNs: summary [Siegelmann]

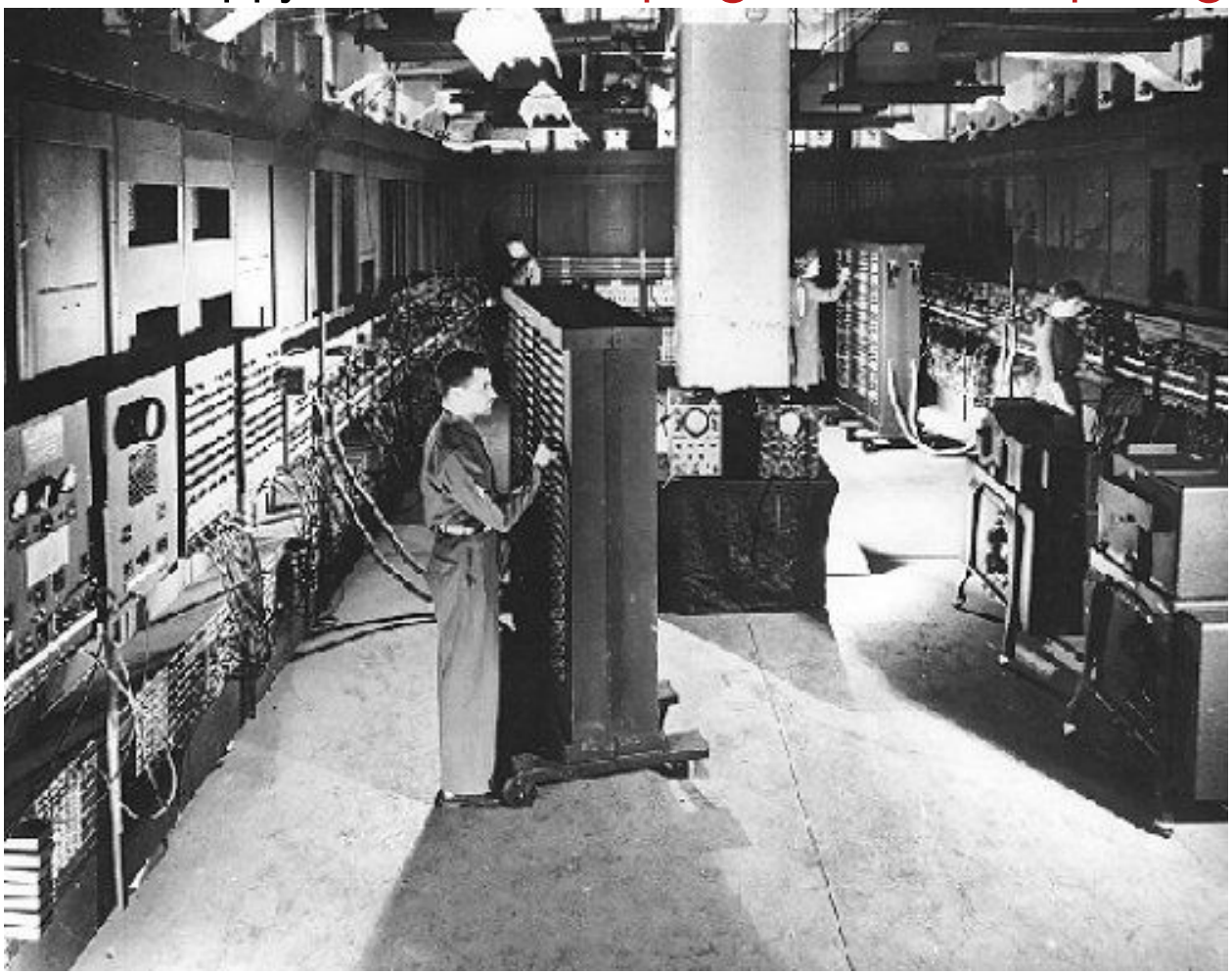
Finite Automata $\longrightarrow$ Binary threshold networks
Turing Machines $\longrightarrow$ Neural networks with rational weights
Probabilistic Turing Machines $\longrightarrow$ NNs with rational weights
Super-turing computations $\longrightarrow$ NNs with real weights.

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)

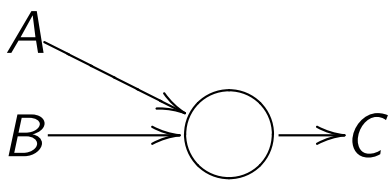
Levels of Abstraction: from 1 to 2

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

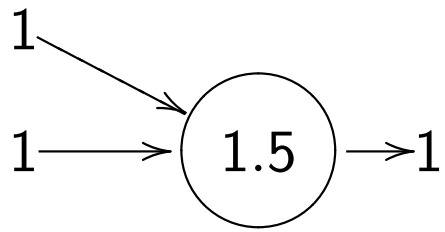
Levels of Abstraction: from 1 to 2

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

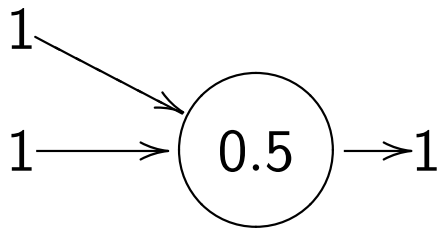
Level of abstraction-2. Capitalising on Boolean networks:



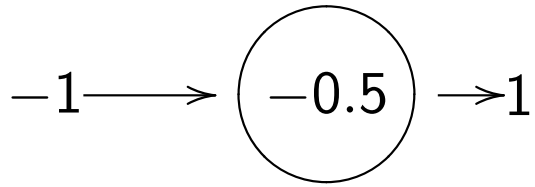
If A and B then C.



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



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



Not $(A = -1)$.

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

Logic Programs

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

Level of Abstraction 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 Abstraction 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

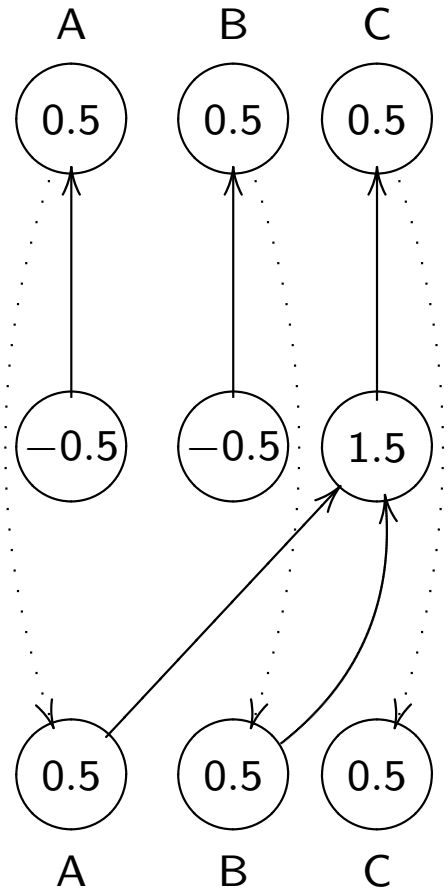
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

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

A Simple Example

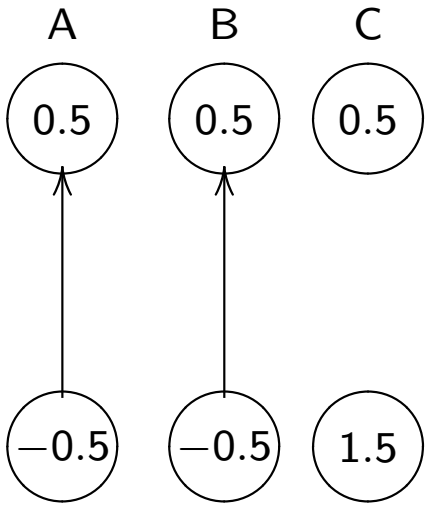
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

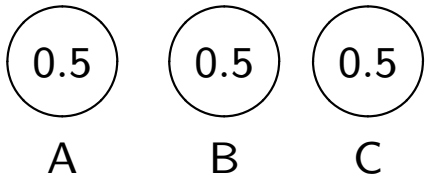
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



A Simple Example

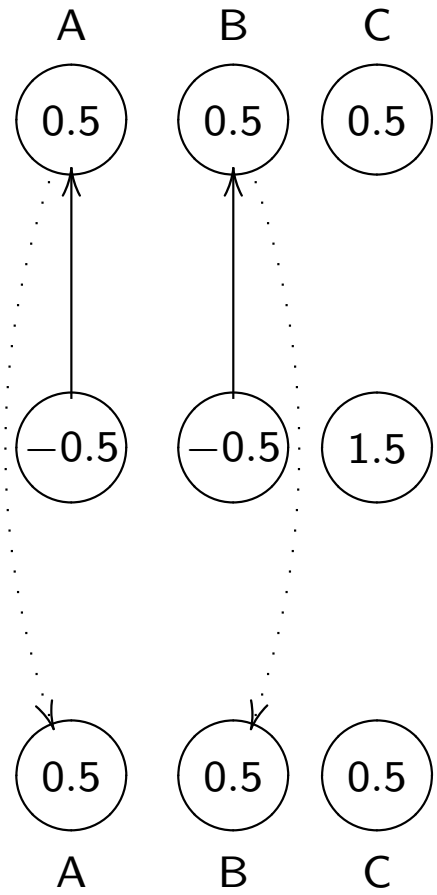
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$





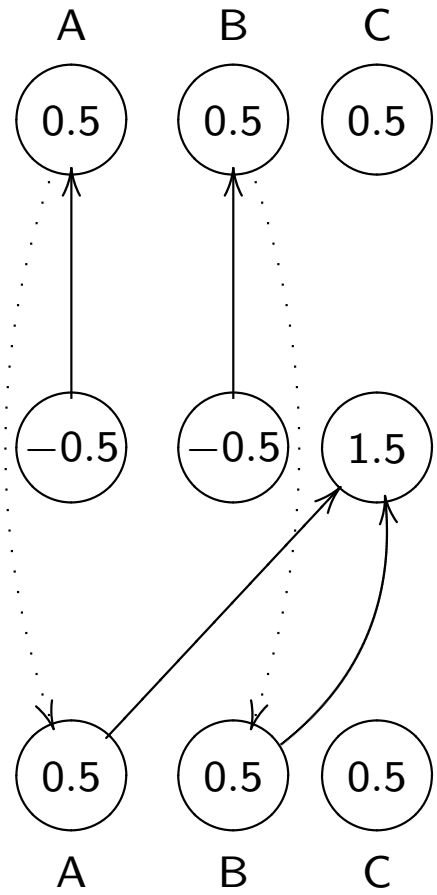
A Simple Example

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



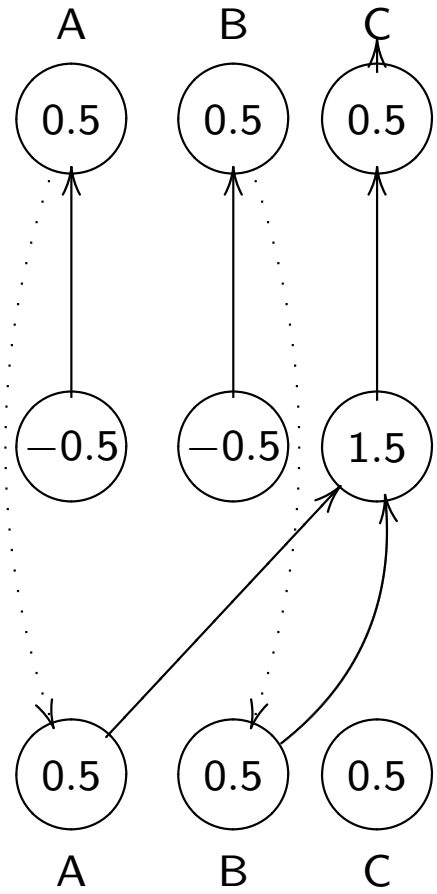
A Simple Example

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



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

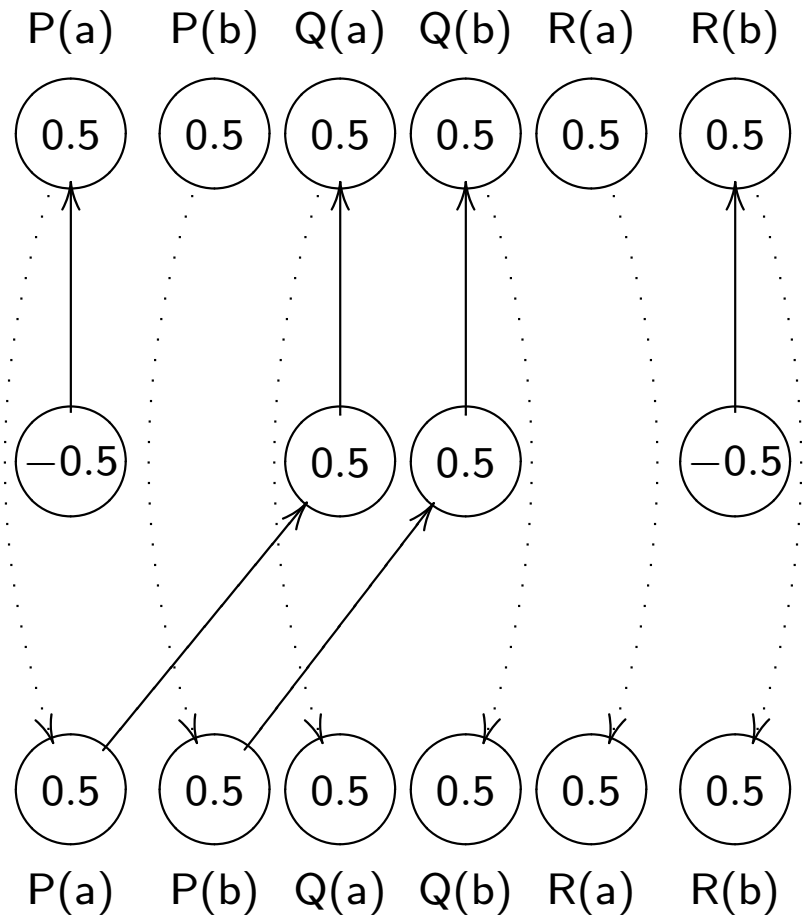
$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

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

Another Example: First-Order Case

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

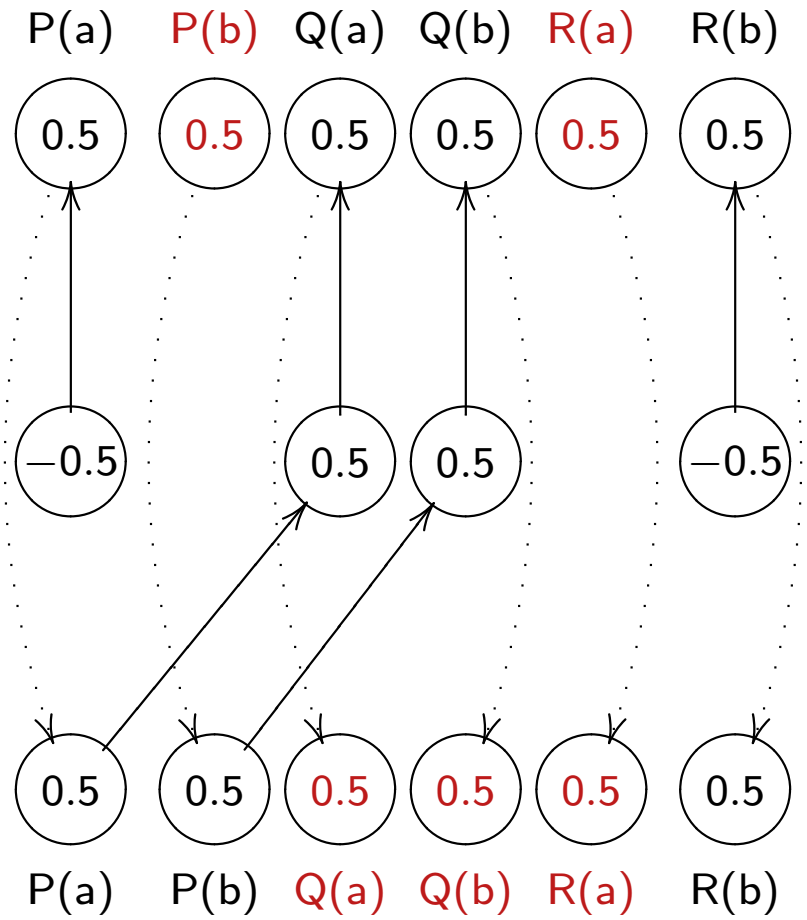
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

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

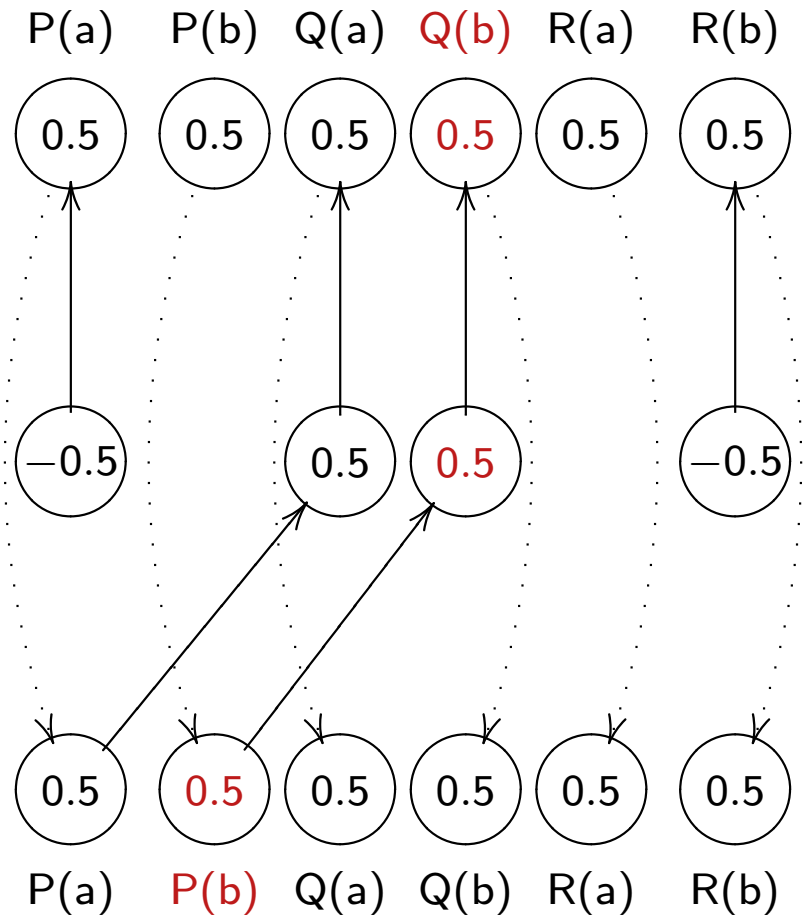
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

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

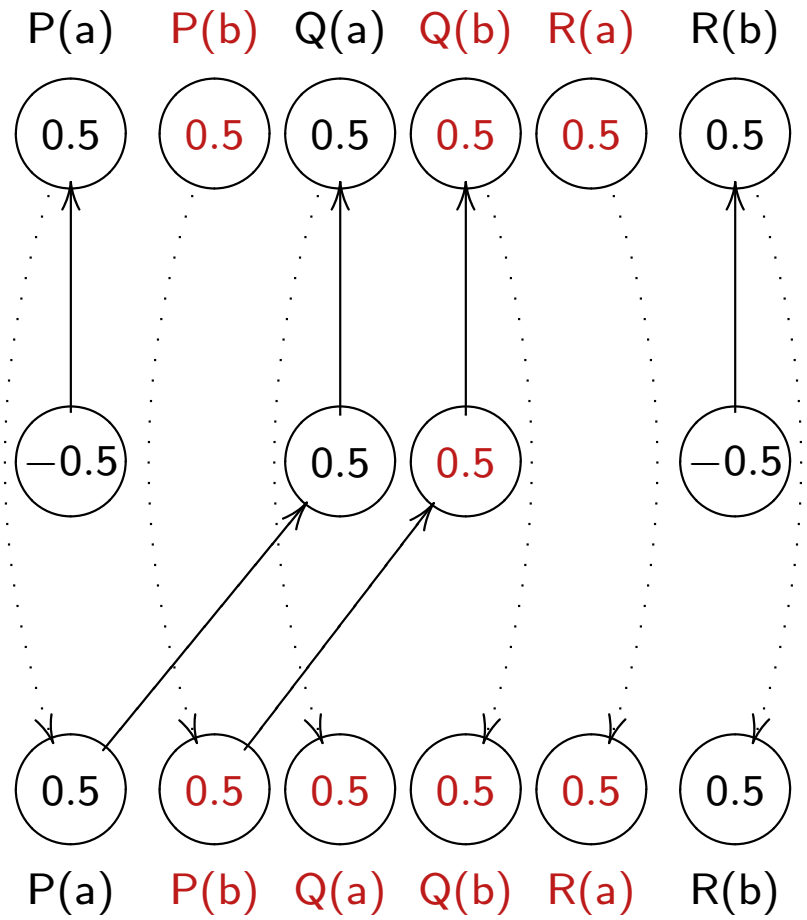
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

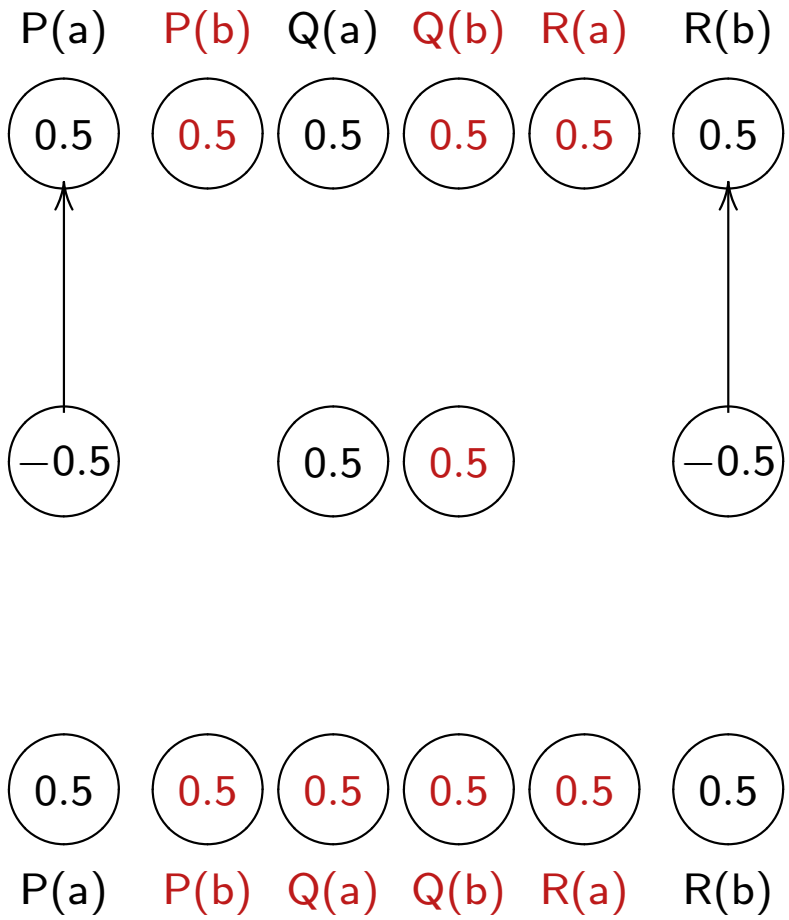
$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

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



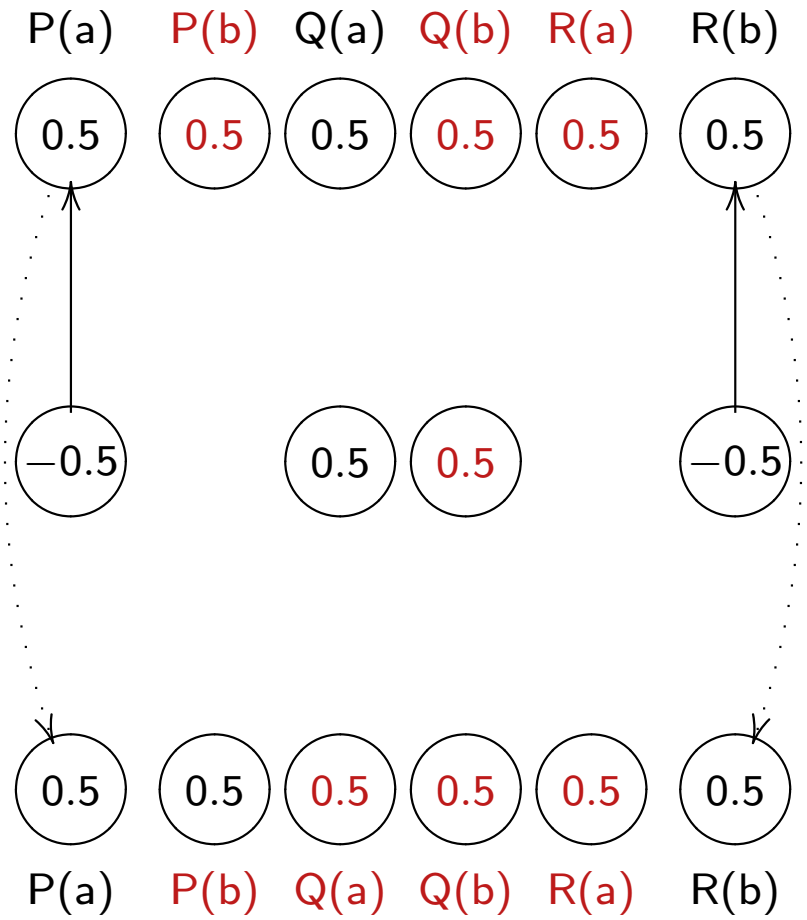
Another Example: First-Order Case

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



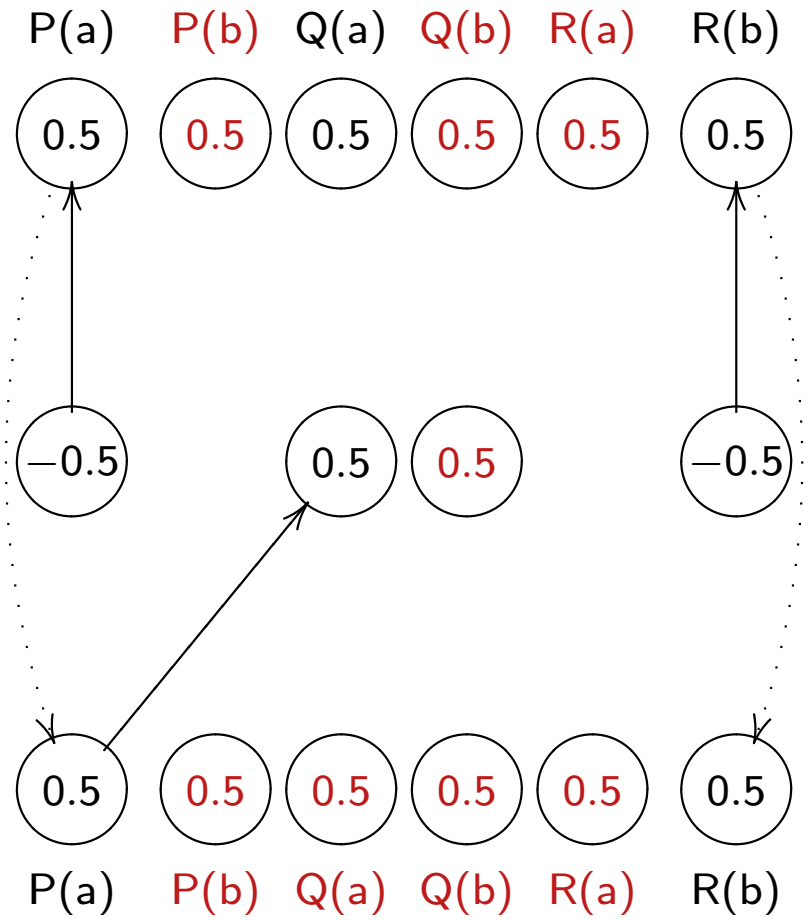
Another Example: First-Order Case

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



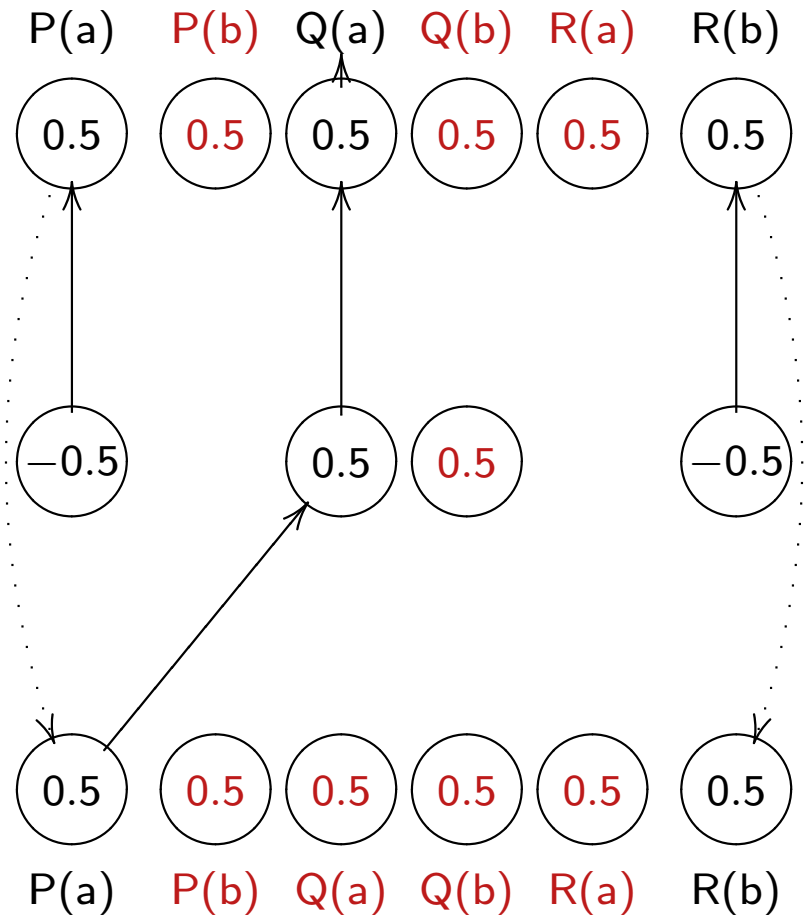
Another Example: First-Order Case

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



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 makes computation of truth value functions $\wedge$ and $\leftarrow$ possible.
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 makes computation of truth value functions $\wedge$ and $\leftarrow$ possible.
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 based on the same methodology:

The approach of McCulloch and Pitts to processing truth values has dominated the area, and many modern neural network architectures consciously or unconsciously follow and develop this old method.

- Core Method: massively parallel way to compute minimal models of logic programs. [Holldobler et al, 1999 - 2009]
- Markov Logic and Markov networks: statistical AI and Machine learning implemented in NN. [Domingos et al., 2006-2009]
- Inductive Reasoning in Neural Networks [Broda, Garcez et al. 2002,2008]
- Fuzzy Logic Programming in Fuzzy Networks [Zadeh at al].

Inevitable mismatch?

From my personal experience:

- It is hard to find a neuro-symbolic system that is both natural from the point of view of neurocomputing, and is useful for logical reasoning.
- Depending on the background of researchers (Logic or Neurocomputing) - integrated systems tend to sacrifice one set of features or another.

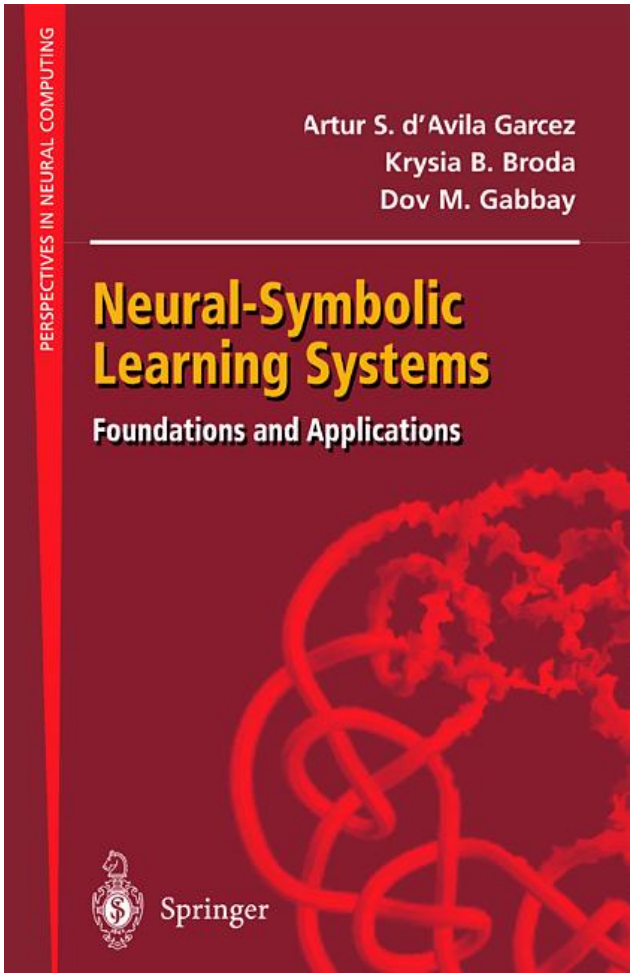
Inevitable mismatch?

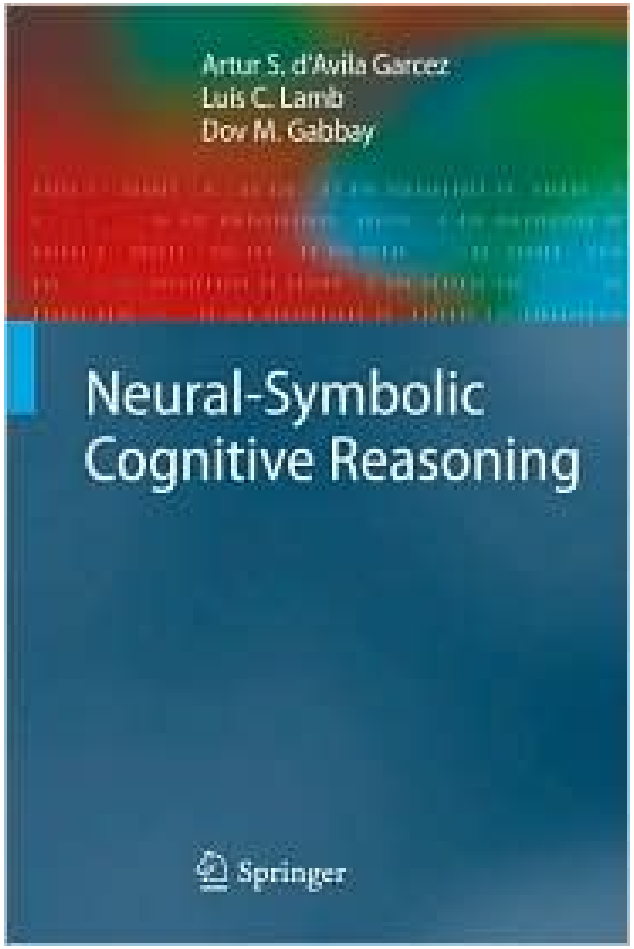
From my personal experience:

- It is hard to find a neuro-symbolic system that is both natural from the point of view of neurocomputing, and is useful for logical reasoning.
- Depending on the background of researchers (Logic or Neurocomputing) - integrated systems tend to sacrifice one set of features or another.

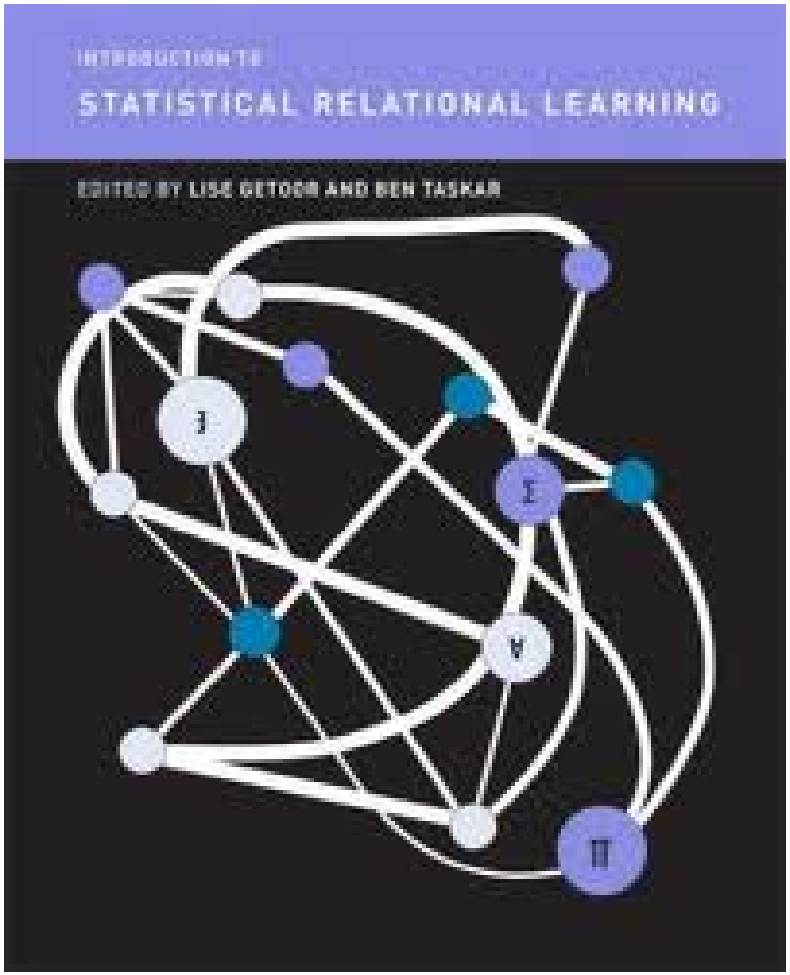
In fact, there are two different subjects that merge neural networks and symbolic logic from the two different ends:

Neuro-Symbolic integration





Statistical Relational Learning (featuring a disjoint set of authors)



Difference in approach: statistical relational learning

Primary goal is the learning task, e.g.

- detect a face on an image (pixels are the main "data")
- classify email messages into spam/not-spam (URLs are the main data)

Difference in approach: statistical relational learning

Primary goal is the learning task, e.g.

- detect a face on an image (pixels are the main "data")
- classify email messages into spam/not-spam (URLs are the main data)
- or analyse hyperlinked webpages?

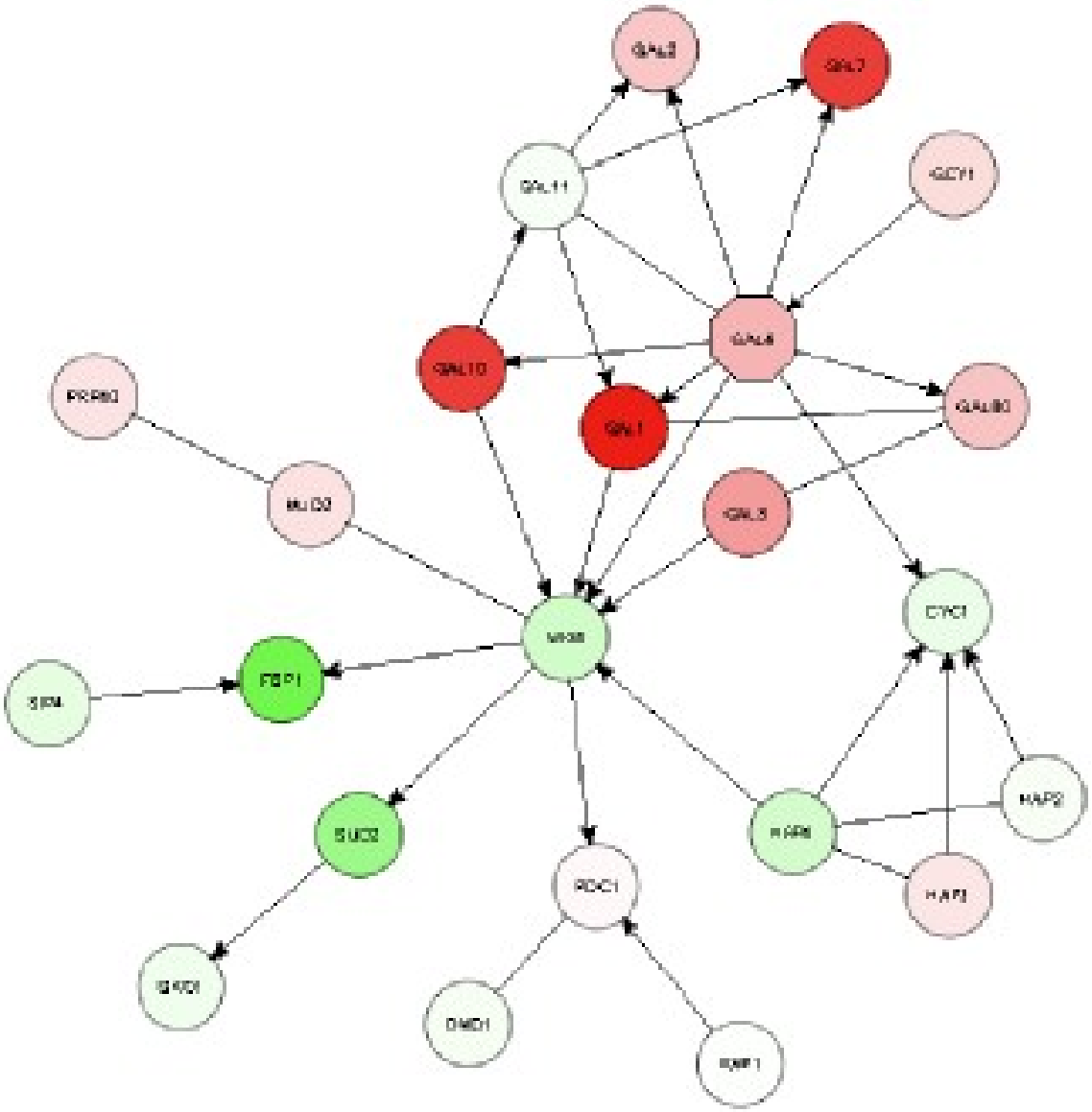
Main method: vector representation.

Need for a "relational" data, such as:

Is the face smiling?
Is it a man, a woman, a boy, a girl?
Is it a friends' message or a "work-related" message?

Main method: collective classification, joint probabilities. e.g. relational Markov networks over relational data set.

Example of Markov nets



Markov Networks have been (most impressively) applied by [Domingos et al.]

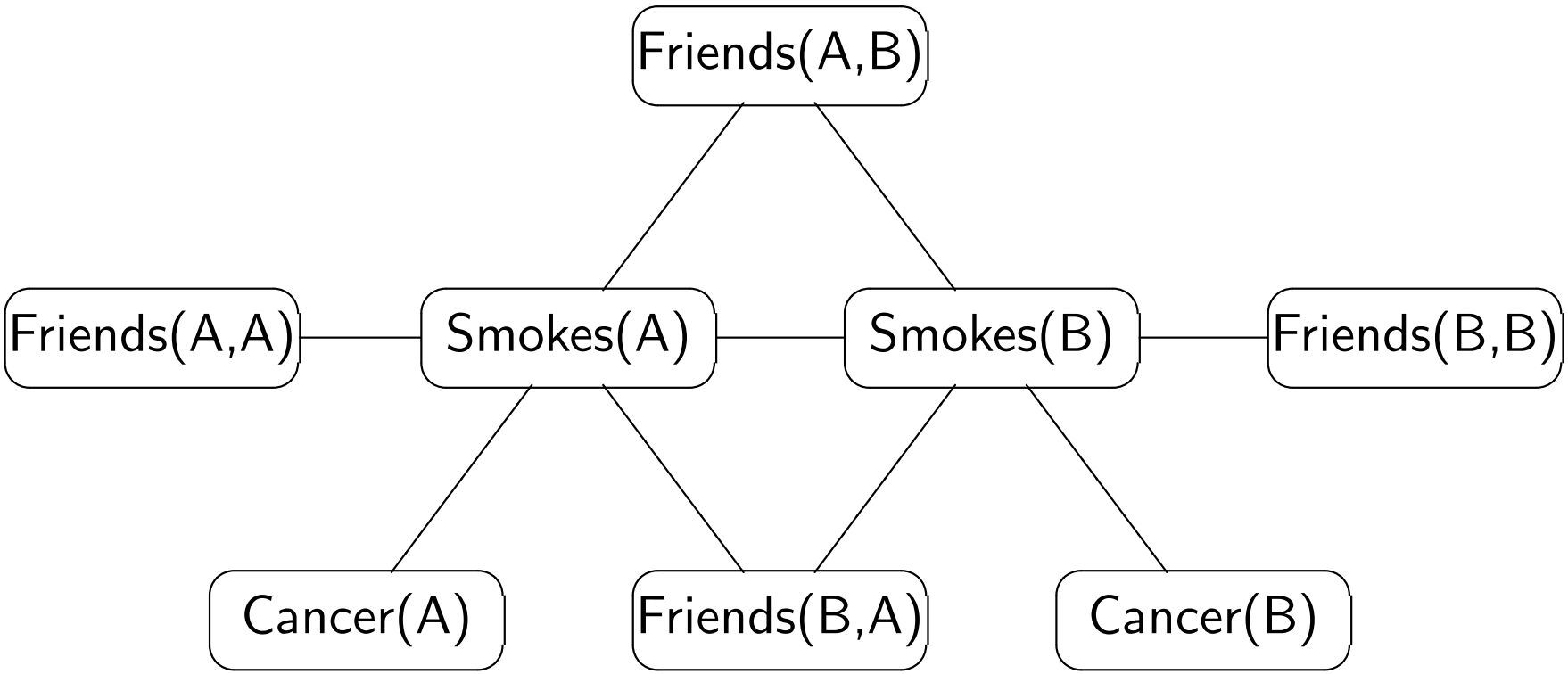
Markov networks have been successfully applied in a variety of areas.

A system based on them recently won a competition on information extraction for biology. They have been successfully applied to problems in information extraction and integration, natural language processing, robot mapping, social networks, computational biology, and others, and are the basis of the open-source Alchemy system. Applications to Web mining, activity recognition, natural language processing, computational biology, robot mapping and navigation, game playing and others are under way.

P. Domingos and D. Lowd. Markov Logic: An Interface Layer for Artificial Intelligence. San Rafael, CA: Morgan and Claypool, 2009.

Example of Markov nets - Any relation to logic programs?

Friend(x,z) ← Friend(x,y), Friend(y,z);
Smokes(x) ← Friend(x,none);
Cancer(x) ← Smokes(x);
Smokes(x) ← Friend(x,y), Smokes(y)



Any solutions?

In my thesis, I proposed to modify N-S nets and run proof search in them

Any solutions?

In my thesis, I proposed to modify N-S nets and run proof search in them

- For goal-oriented inference - one needs to implement first-order unification in Neural networks

Any solutions?

In my thesis, I proposed to modify N-S nets and run proof search in them

- For goal-oriented inference - one needs to implement first-order unification in Neural networks
- For "search" - need of backtracking.

Unifier

Definition

Let S be a finite set of atoms. A substitution θ is called a *unifier* for S if $S\theta$ is a singleton. A unifier θ for S is called a *most general unifier (mgu)* for S if, for each unifier σ of S, there exists a substitution γ such that $\sigma = \theta\gamma$.

Example: If $S = (Q(f(x_1, x_2)), Q(f(a_1, a_2)))$, then $\theta = \{x_1/a_1;\ x_2/a_2\}$ is the mgu.

Disagreement set

Definition

Let S be a finite set of atoms. To find the disagreement set D_S of S locate the leftmost symbol position at which not all atoms in S have the same symbol and extract from each atom in S the subexpression (term) beginning at that symbol position. The set of all such expressions is the *disagreement set* of S.

Example: For $S = (Q(f(x_1, x_2)), Q(f(a_1, a_2)))$ we have $D_S = \{x_1,\ a_1\}$.

Unification algorithm

1. Let $k = 0$ and $\theta_0 = \varepsilon$.
2. If $S\theta_k$ is a singleton, then stop; θ_k is an mgu of S. Otherwise, find the disagreement set D_k of $S\theta_k$.
3. If there exist a variable v and a term t in D_k such that v does not occur in t, then put $\theta_{k+1} = \theta_k\{v/t\}$, increment k and go to 2. Otherwise, stop; S is not unifiable.

Unification algorithm

1. Let $k = 0$ and $\theta_0 = \varepsilon$.
2. If $S\theta_k$ is a singleton, then stop; θ_k is an mgu of S. Otherwise, find the disagreement set D_k of $S\theta_k$.
3. If there exist a variable v and a term t in D_k such that v does not occur in t, then put $\theta_{k+1} = \theta_k\{v/t\}$, increment k and go to 2. Otherwise, stop; S is not unifiable.

Example.
Take $S = (Q(f(x_1, x_2)), Q(f(a_1, a_2)))$ as before.

- $S\varepsilon$ is not a singleton. So, compute $D_1 = \{x_1,\ a_1\}$ and $\theta_1 = \{x_1/a_1\}$.

Unification algorithm

1. Let $k = 0$ and $\theta_0 = \varepsilon$.
2. If $S\theta_k$ is a singleton, then stop; θ_k is an mgu of S. Otherwise, find the disagreement set D_k of $S\theta_k$.
3. If there exist a variable v and a term t in D_k such that v does not occur in t, then put $\theta_{k+1} = \theta_k\{v/t\}$, increment k and go to 2. Otherwise, stop; S is not unifiable.

Example.
Take $S = (Q(f(x_1, x_2)), Q(f(a_1, a_2)))$ as before.

- $S\varepsilon$ is not a singleton. So, compute $D_1 = \{x_1,\ a_1\}$ and $\theta_1 = \{x_1/a_1\}$.
- Apply $S\theta_1 = (Q(f(a_1, x_2)), Q(f(a_1, a_2)))$, it's not a singleton. So, compute $D_2 = \{x_2,\ a_2\}$ and $\theta_2 = \{x_2/a_2\}$.

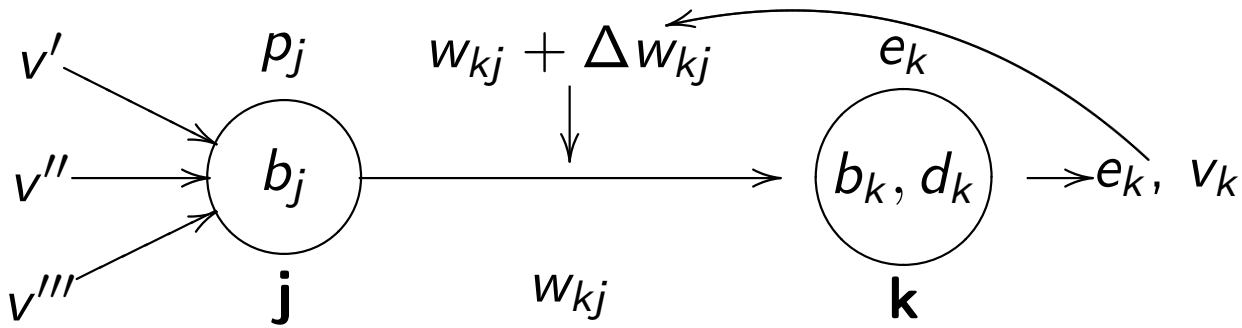
Unification algorithm

1. Let $k = 0$ and $\theta_0 = \varepsilon$.
2. If $S\theta_k$ is a singleton, then stop; θ_k is an mgu of S. Otherwise, find the disagreement set D_k of $S\theta_k$.
3. If there exist a variable v and a term t in D_k such that v does not occur in t, then put $\theta_{k+1} = \theta_k\{v/t\}$, increment k and go to 2. Otherwise, stop; S is not unifiable.

Example.
Take $S = (Q(f(x_1, x_2)), Q(f(a_1, a_2)))$ as before.

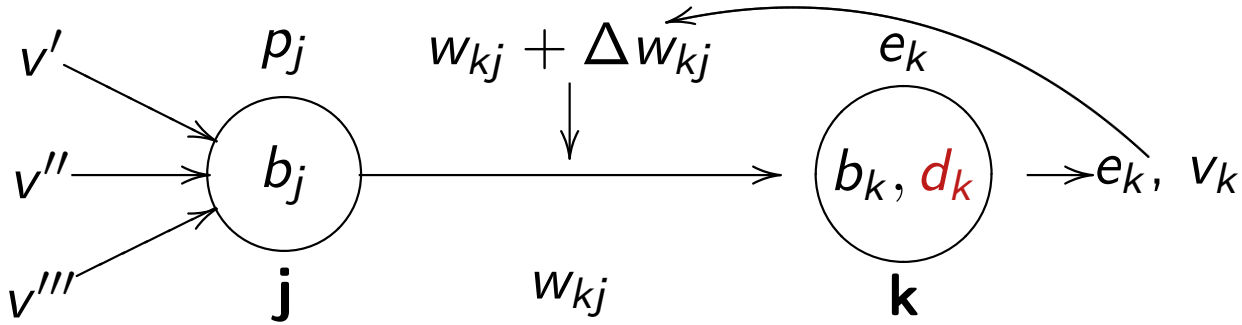
- $S\varepsilon$ is not a singleton. So, compute $D_1 = \{x_1,\ a_1\}$ and $\theta_1 = \{x_1/a_1\}$.
- Apply $S\theta_1 = (Q(f(a_1, x_2)), Q(f(a_1, a_2)))$, it's not a singleton. So, compute $D_2 = \{x_2,\ a_2\}$ and $\theta_2 = \{x_2/a_2\}$.
- Apply $S\theta_1\theta_2 = (Q(f(a_1, a_2))$ - it's a singleton. Stop.

Error-Correction (Supervised) Learning



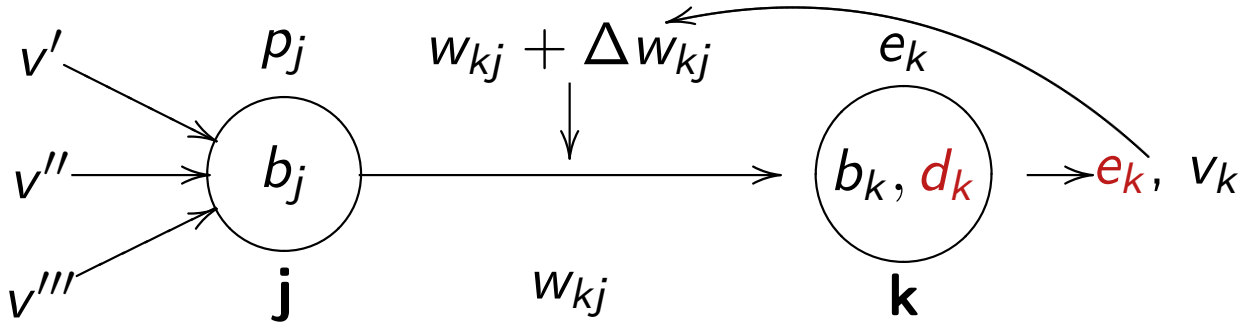
Error-Correction (Supervised) Learning

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



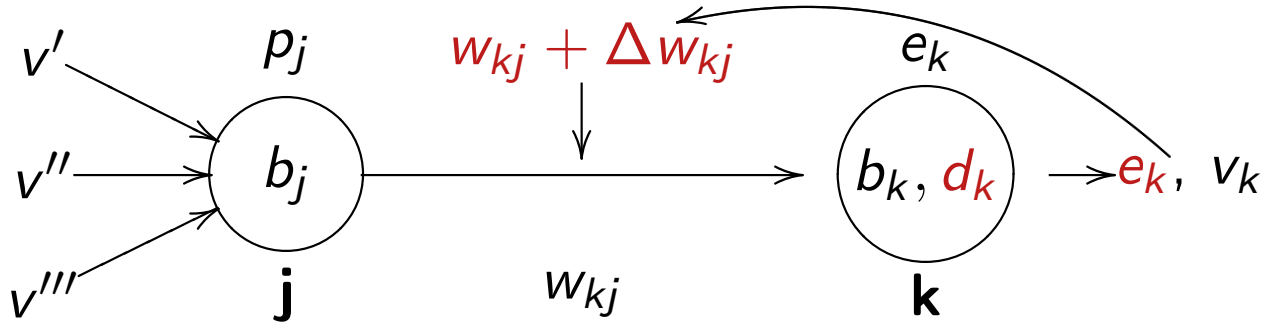
Error-Correction (Supervised) Learning

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, $\Delta w_{kj}(t) = \eta e_k(t) v_j(t)$.

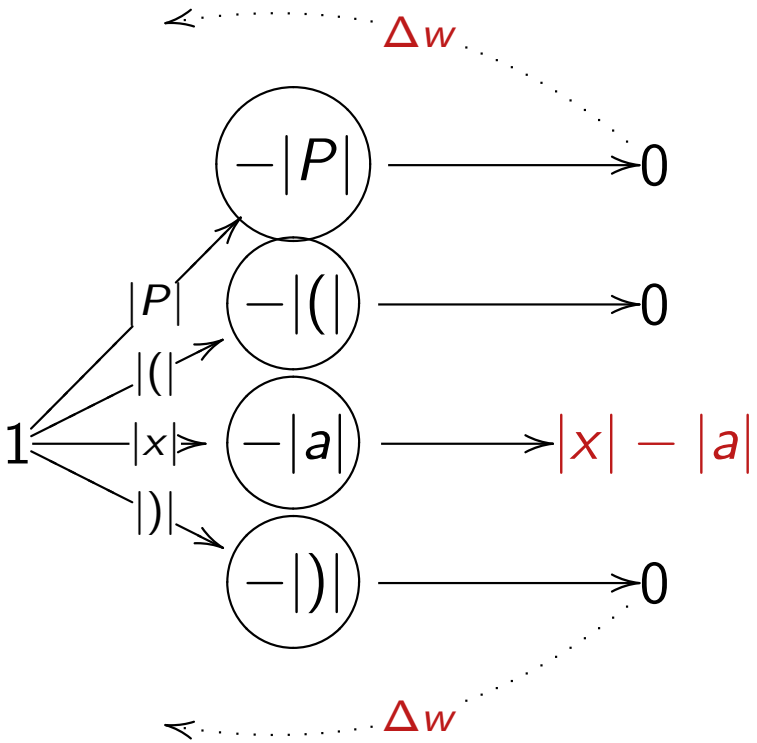


Unification by Error-Correction: Example 1

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

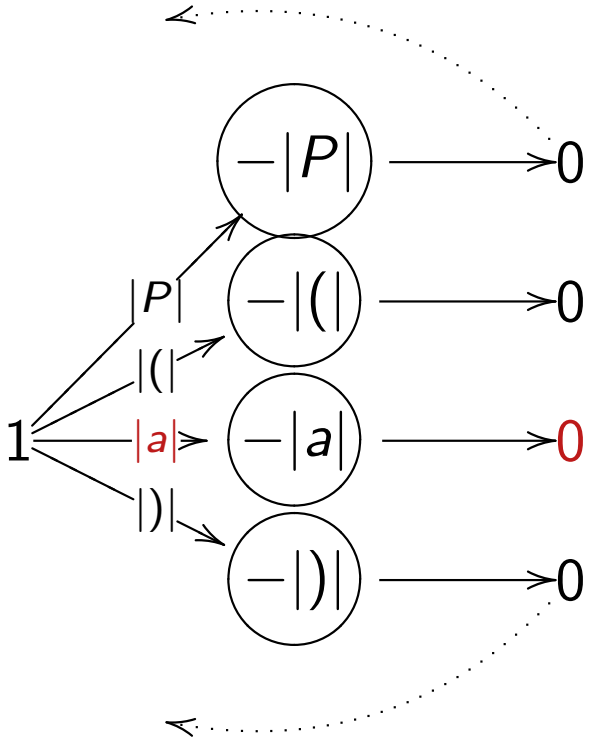
encoded numerically using function $|\ |$.

Target (*desired response*) vector is $0; 0; 0; 0$.

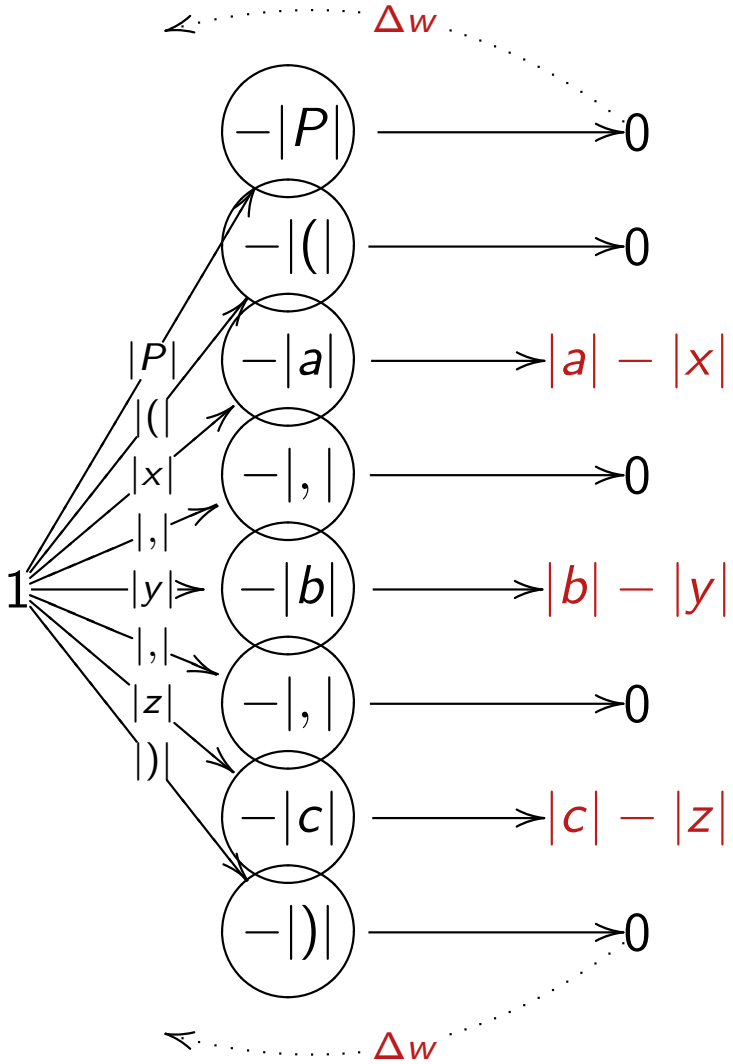


Example 1 - ctnd

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



Capacity for Parallelisation: Example 2.



Main Theorems (Journal of Algorithms in Cognition, Informatics and Logic)

A Theorem for Logicians

For some restricted cases of unification - such as term matching - error-correction in neural networks can perform parallel unification. For the general case of unification, one redefines the learning function. This new learning function is defined and implemented in MATLAB neural Network Toolbox.

A Theorem for Neurocomputing

Error-correction implemented in neural networks of certain kinds does unification (term-matching) known in Logic. Namely, the neural networks that do this are one layer networks; and, with the vector of weights w, the vector of biases b, and the vector of targets set to 0, the given layer performs parallel term-matching for w and b.

Logically sound. And natural???

- Example shown above can be implemented in conventional neural nets, and conventional error-correction learning.
- However, for the general case of unification, allowing growth of terms or their shortening, a lot of control is given to new learning functions that adjust network size, do occurrence check, and generally take care of all the subtleties of logical reasoning.

Example

Constant c can be substituted for variable x, but not the other way around...

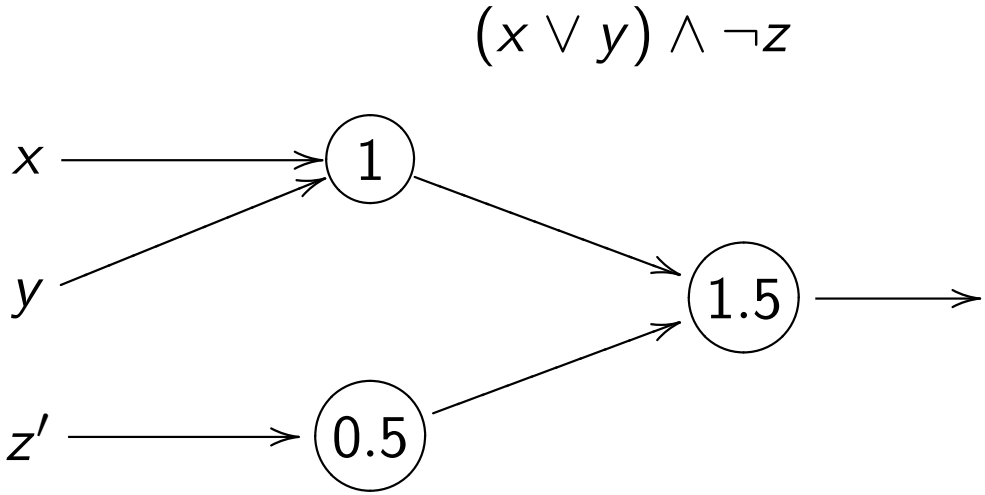
Similar results: ICNC'2009

Certain cases of Term-rewriting naturally occur in Unsupervised (Hebbian) learning.

Search for both relevant and natural applications...

- Type-checking for neural networks.
- Higher-order unification in Hopfield networks.

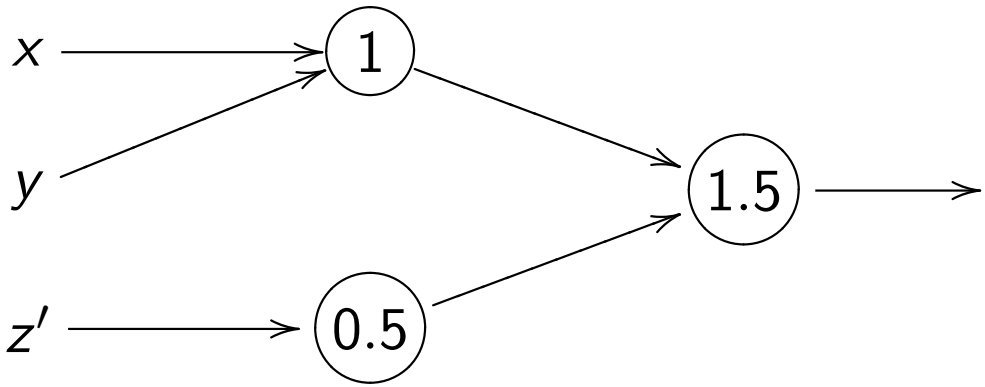
Boolean networks: How do we know that they are correct?



Such network would not be able to distinguish "logical" data (values 0 and 1) from any other type of data, and would output the same result both for sound inputs like $x := 1, y := 1, z := 0$, and for non-logical values such as $x := 100.555, y := 200.3333\ldots, z := 0$. Imagine a user monitors the outputs of a big network, and sees outputs 1, standing for "true", whereas in reality the network is receiving some uncontrolled or excessive data.

Boolean networks: How do we know that they are correct?

$$(x \vee y) \wedge \neg z$$



Such network would not be able to distinguish "logical" data (values 0 and 1) from any other type of data, and would output the same result both for sound inputs like $x := 1, y := 1, z := 0$, and for non-logical values such as $x := 100.555, y := 200.3333\ldots, z := 0$. Imagine a user monitors the outputs of a big network, and sees outputs 1, standing for "true", whereas in reality the network is receiving some uncontrolled or excessive data.

The network gives correct answers on the condition that the input is well-typed.

Relational Reasoning and Learning.

In [Garcez et al, 2009], were built networks that can learn relations. E.g., given examples $Q(b, c) \rightarrow P(a, b)$ and $Q(d, e) \rightarrow P(c, d)$, they can infer a more general relation $Q(y, z) \rightarrow P(x, y)$.

Example

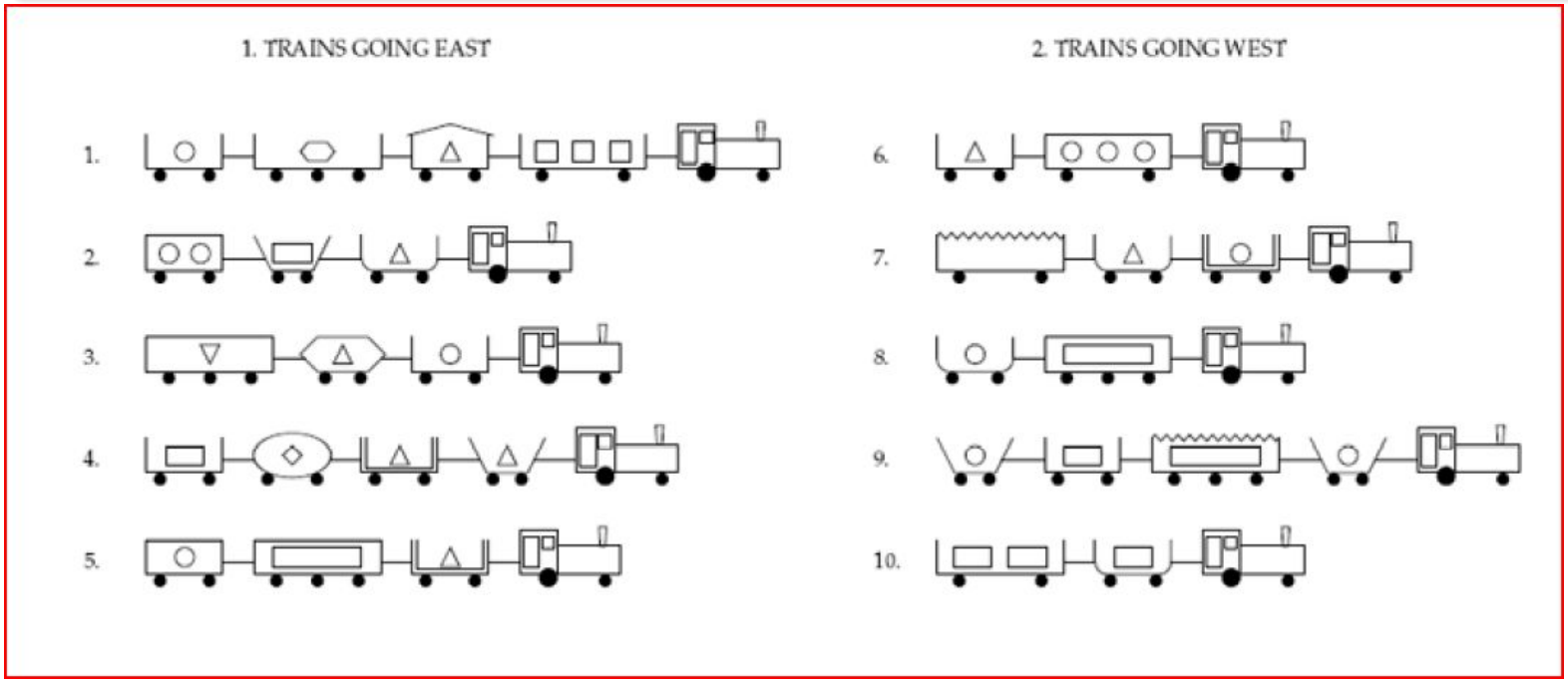
Learning a relation "grandparent" by examining families. Classification of trains according to certain characteristics.

Relational Reasoning and Learning.

In [Garcez et al, 2009], were built networks that can learn relations. E.g., given examples $Q(b, c) \rightarrow P(a, b)$ and $Q(d, e) \rightarrow P(c, d)$, they can infer a more general relation $Q(y, z) \rightarrow P(x, y)$.

Example

Learning a relation "grandparent" by examining families. Classification of trains according to certain characteristics.



Problems with this method

Such relational learning can be applied as long as input data is well-typed.

Well-typed, in our previous examples, means that only related people, and not any other objects, are given to the network that learns relation "grandparent". And there are only trains of particular, known in advance, configuration, that are considered by the network that classifies trains.

This means that users have to make the preliminary classification and filtering of data before it is given to such networks; and NNs would not be able to warn the users if the data are ill-typed :-(.

Problems with this method

Such relational learning can be applied as long as input data is well-typed.

Well-typed, in our previous examples, means that only related people, and not any other objects, are given to the network that learns relation "grandparent". And there are only trains of particular, known in advance, configuration, that are considered by the network that classifies trains.

This means that users have to make the preliminary classification and filtering of data before it is given to such networks; and NNs would not be able to warn the users if the data are ill-typed :-(.

Generally, as it turns out, typing is important for correct reasoning.

E.g., one can generalise from "This dog has four legs, and hence it can run" to the statement "Everything that has four legs can run". However, we know that there are some objects, such as chairs, that have four legs but do not move. Hence we (often unconsciously) use typing in such cases, e.g., apply the generalisation only to all animals.

Solutions: K.K., K. Broda, A.Garcez [CiE'2010, ICANN 2010]

Solution

As an alternative to the manual pre-processing of data, we propose neural networks that can do the same automatically. We use *neural networks called type recognisers*; and implement such networks to ensure the correctness of neural computations; both for classical cases (McCulloch & Pitts) and for the relational reasoning and learning.

Solutions: K.K., K. Broda, A.Garcez [CiE'2010, ICANN 2010]

Solution

As an alternative to the manual pre-processing of data, we propose neural networks that can do the same automatically. We use *neural networks called type recognisers*; and implement such networks to ensure the correctness of neural computations; both for classical cases (McCulloch & Pitts) and for the relational reasoning and learning.

The solution involves techniques like pattern-matching, inductive type definitions, etc. that are used in functional programming, type theory, and interactive theorem provers!

The main result

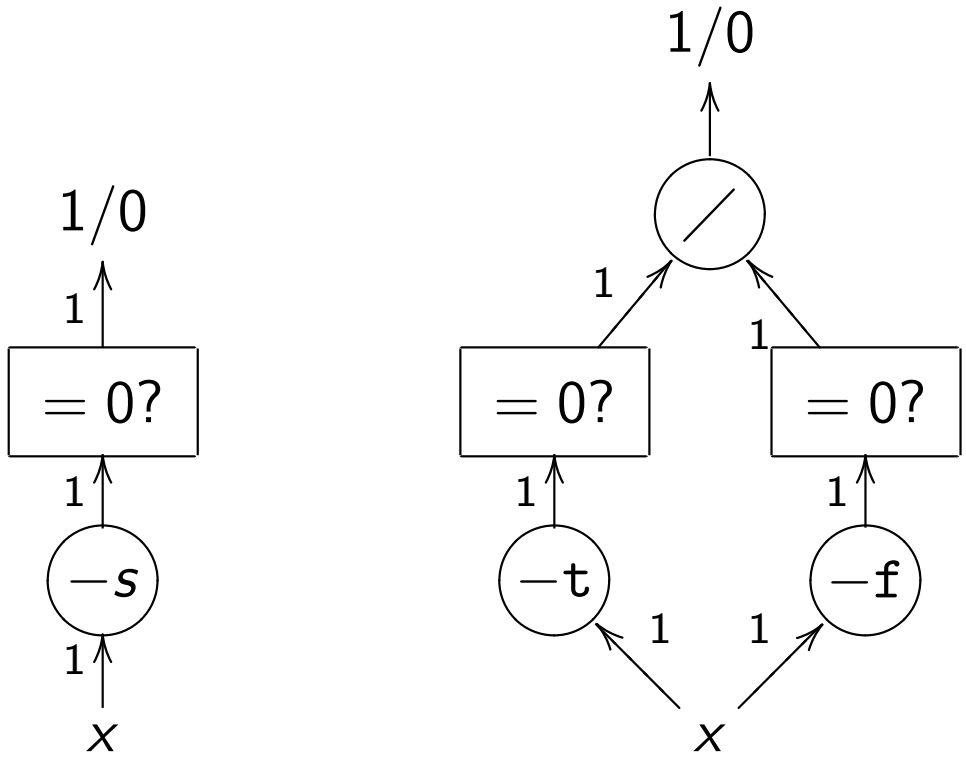
A method of using Types for ensuring the correctness of Neural or Neuro-Symbolic computations.

Theorem

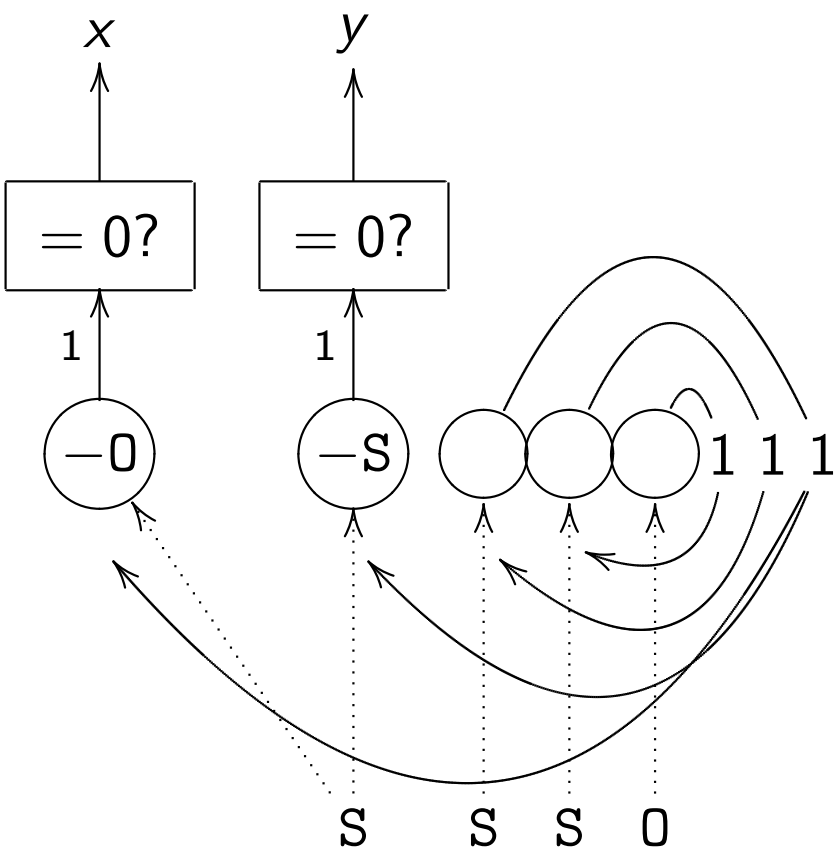
For any a type A, given an expression E presented in a form of a numerical vector, we can construct a neural network that recognises whether E is of type A.

Such networks are called Type recognisers, and for each given type A, the network that recognises A is called an A-recogniser. This construction covers simple types, such as Bool, as well as more complex inductive types, such as natural numbers, lists; or even dependent inductive types, such as lists of natural numbers.

Example: Inductive recogniser for bool



Example: Inductive recogniser for `nat`



x	y	
1	0	- success
0	1	- working
1	1	- impossible
0	0	- failure

Conclusions

- Neuro-symbolic integration (or, else, statistical relational learning) is an exciting field;
- It is both new and has a long history, - depending at which level of abstraction you come to it;
- It is still waiting for its Eureka moment, when Proof-theoretical and Neurocomputing methods will merge both elegantly and with good practical applications...

E.g., there are two research project currently running in the UK (AI4FM: Newcastle, Edinburgh, Heriot-Watt, Southampton) and Netherlands (Radboud University Nijmegen) that apply neural nets to learning proof strategies.

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 Abstraction 2


M. Richardson and P. Domingos.
Markov Logic Networks.
Machine Learning, 62, 107-136, 2006.


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.


Introduction to Statistical Relational Learning.
MIT Press, 2007.


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