Unification by Error-Correction

Ekaterina Komendantskaya

INRIA Sophia Antipolis, France

NeSy'08 4th International Workshop on Neural-Symbolic Learning and Reasoning
21 July 2008, Patras, Greece

Outline

Outline

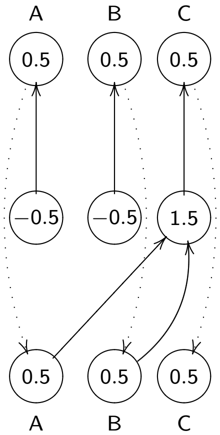
1. Motivation
2. Background Definitions

Outline

Outline

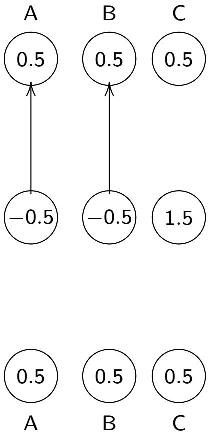
A Simple Example

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



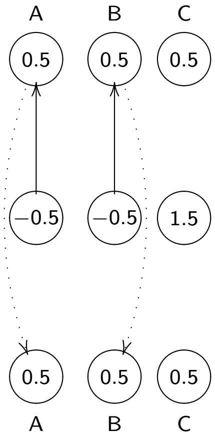
A Simple Example

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



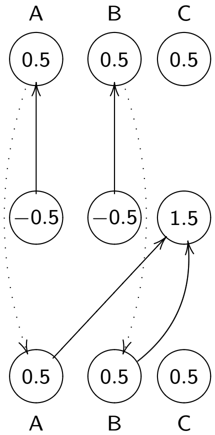
A Simple Example

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



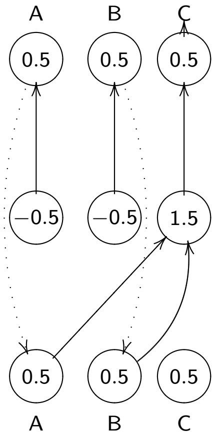
A Simple Example

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



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

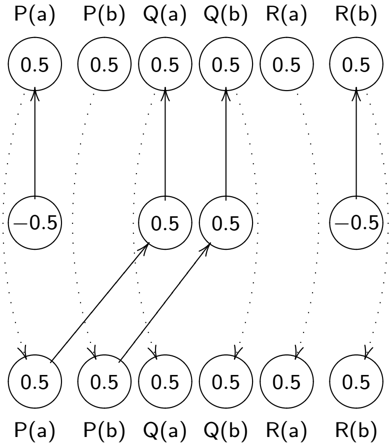
$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

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

Another Example: First-Order Case

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

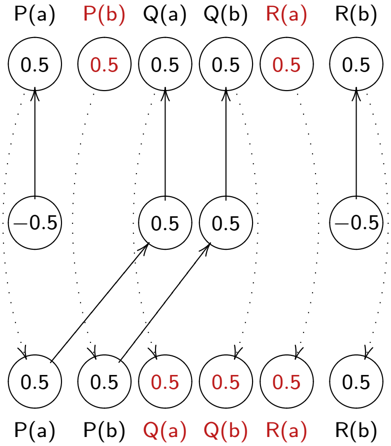
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

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

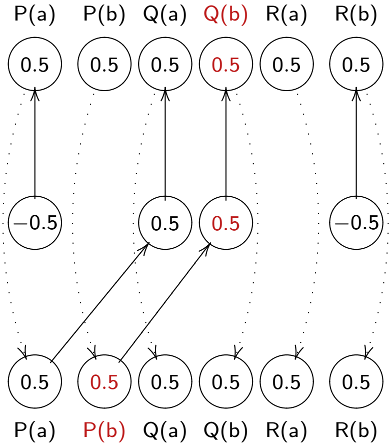
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 = \{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

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

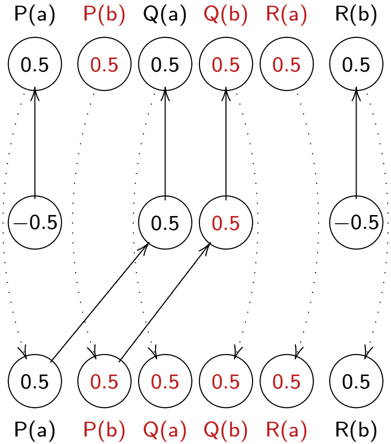
$T_P \uparrow 0 = \{P(a), R(b)\}$
$lfp(T_P) = T_P \uparrow 1 =$
$\{P(a), R(b), Q(a)\}$



Another Example: First-Order Case

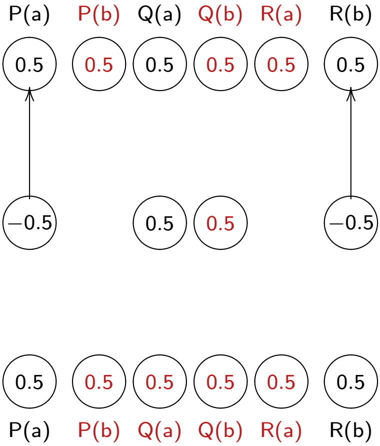
$P(a) \leftarrow$
$Q(x) \leftarrow P(x)$
$R(b) \leftarrow$

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



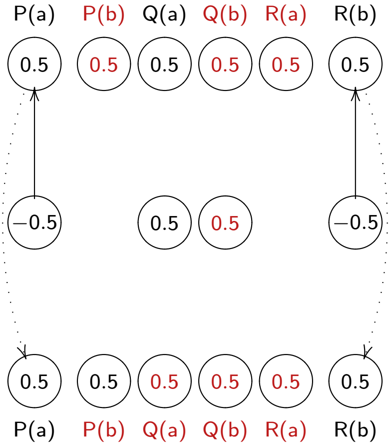
Another Example: First-Order Case

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



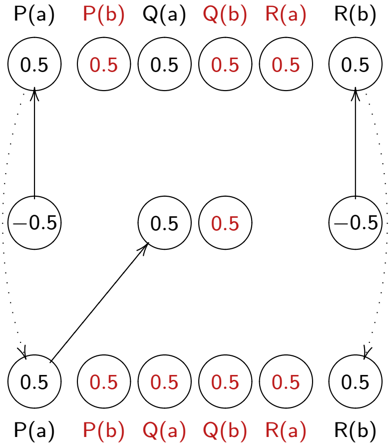
Another Example: First-Order Case

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



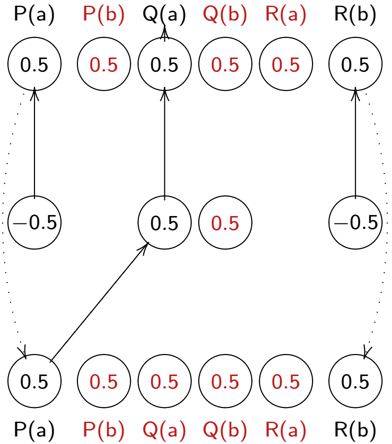
Another Example: First-Order Case

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



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)

What causes the problems?

1. We compute T_P-operator, which forces us to work with Herbrand base and Herbrand model;
2. First-order atoms are not represented in the neural networks directly, instead truth values 0 and 1 are propagated.
3. $2 \Longrightarrow$
   - Only ground atoms are processed; so essentially we are able to work only with propositional case.
   - Require infinitely long layers in the first-order case.
4. Status of learning?

My Wish-List

I wish for

- A Neural Theorem Prover

My Wish-List

I wish for

- A Neural Theorem Prover
- ...implementing a goal oriented search algorithm (as opposed to iterational method of T_P-operator),

My Wish-List

I wish for

- A Neural Theorem Prover
- ...implementing a goal oriented search algorithm (as opposed to iterational method of T_P-operator),
- ...able to compute (non-guarded) substitutions,

My Wish-List

I wish for

- A Neural Theorem Prover
- ...implementing a goal oriented search algorithm (as opposed to iterational method of T_P-operator),
- ...able to compute (non-guarded) substitutions,
- ...at least first-order,

My Wish-List

I wish for

- A Neural Theorem Prover
- ...implementing a goal oriented search algorithm (as opposed to iterational method of T_P-operator),
- ...able to compute (non-guarded) substitutions,
- ...at least first-order,
- ...expandable to higher-order logics.

My Wish-List

I wish for

- A Neural Theorem Prover
- ...implementing a goal oriented search algorithm (as opposed to iterational method of T_P-operator),
- ...able to compute (non-guarded) substitutions,
- ...at least first-order,
- ...expandable to higher-order logics.

I do not wish to deal with truth values and semantic operators of any sorts.

My Wish-List

I wish for

- A Neural Theorem Prover
- ...implementing a goal oriented search algorithm (as opposed to iterational method of T_P-operator),
- ...able to compute (non-guarded) substitutions,
- ...at least first-order,
- ...expandable to higher-order logics.

I do not wish to deal with truth values and semantic operators of any sorts.

Example

I wish to be able to distinguish/prove properties of natural numbers without listing the whole (infinite) set $\{1, 2, 3, 4, \ldots\}$.

Most General Unifier

MGU

Let S be a finite set of atoms. A substitution θ is called a unifier for S if S 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 $S = (P(x), P(0))$, then $\theta = \{x/0\}$ is the mgu.

Disagreement set

Disagreement set

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 term beginning at that symbol position. The set of all such terms is the disagreement set.

Example: For $S = (Q(f(x, y)), Q(f(a, b)))$ we have $D_S = \{x,\ a\}$.

Unification algorithm

1. Put $k = 0$ and $\sigma_0 = \varepsilon$.
2. If $S\sigma_k$ is a singleton, then stop; σ_k is an mgu of S. Otherwise, find the disagreement set D_k of $S\sigma_k$.
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. Put $k = 0$ and $\sigma_0 = \varepsilon$.
2. If $S\sigma_k$ is a singleton, then stop; σ_k is an mgu of S. Otherwise, find the disagreement set D_k of $S\sigma_k$.
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 theorem.

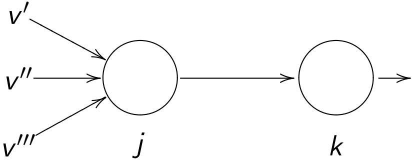
Functions we define and embed:

- **Disagreement set**: $\ominus$;
- **Concatenation**: $\oplus$;
- **Applying the substitution**: $g \odot s$.

Neurons in Connectionist Neural Networks

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t)\right) - \Theta_k$$

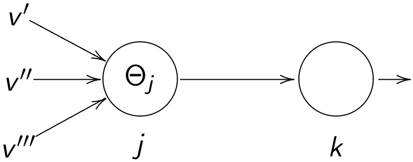
$$v_k(t+\Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Neurons in Connectionist Neural Networks

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t)\right) - \Theta_k$$

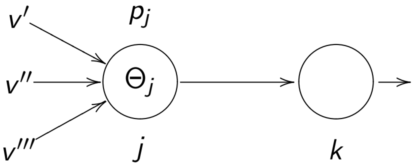
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if} \;\; p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Neurons in Connectionist Neural Networks

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t)\right) - \Theta_k$$

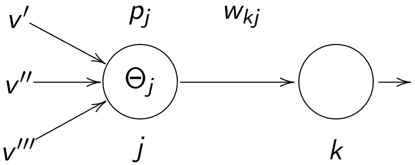
$$v_k(t+\Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } \; p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Neurons in Connectionist Neural Networks

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t)\right) - \Theta_k$$

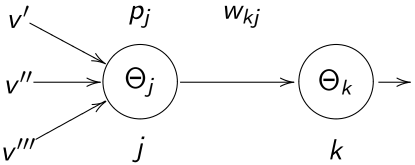
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } \; p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Neurons in Connectionist Neural Networks

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t)\right) - \Theta_k$$

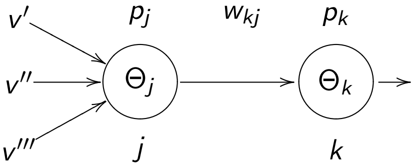
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } \; p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Neurons in Connectionist Neural Networks

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t)\right) - \Theta_k$$

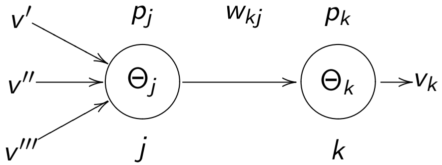
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } \; p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



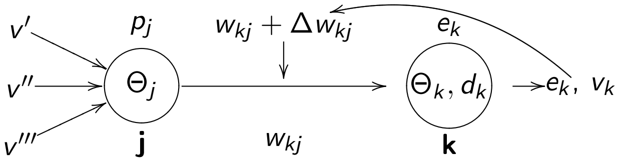
Neurons in Connectionist Neural Networks

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t)\right) - \Theta_k$$

$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$

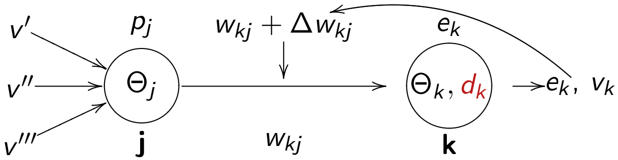


Error-Correction (Supervised) Learning



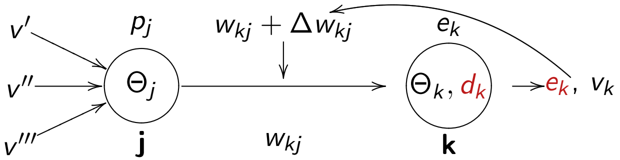
Error-Correction (Supervised) Learning

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



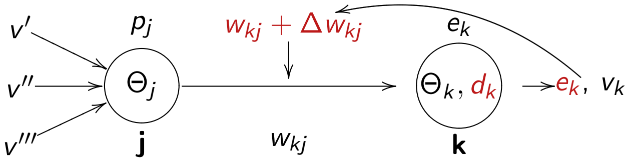
Error-Correction (Supervised) Learning

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

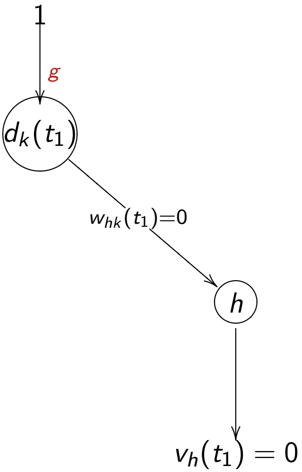


Main Result

Theorem

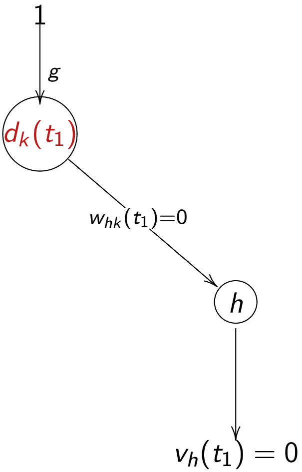
Given two first-order atoms A and B, there exists a two-neuron learning neural network that performs the algorithm of unification for A and B.

Example of Unification in Neural Networks: time $= t_1$.



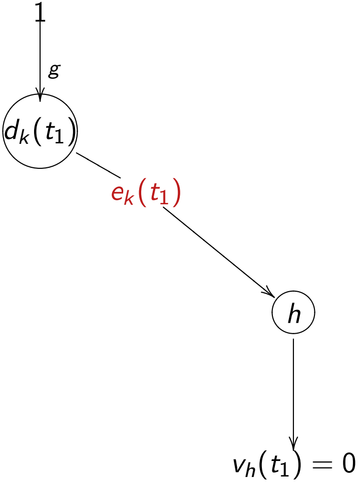
$w_{ik}(t_1) = v_i(t_1) = g$ is some encoding of $P(x)$;
$d_k(t_1) = h$ is some encoding of $P(0)$.

Example of Unification in Neural Networks: time $= t_1$.



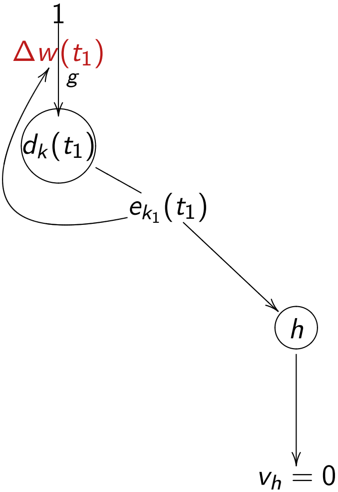
$w_{ik}(t_1) = v_i(t_1) = g$ is some encoding of $P(x)$;
$d_k(t_1) = h$ is some encoding of $P(0)$.

Example of Unification in Neural Networks: time $= t_1$.



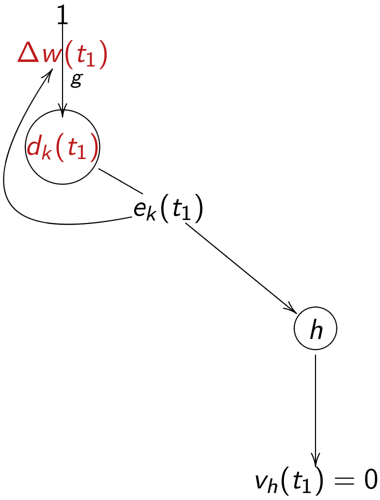
$w_{ki}(t_1) = v_i(t_1) = g$ is the number of $P(x)$;
$d_k(t_1) = h$ is the number of $P(0)$;
Compute $e_k(t_1) = d_k(t_1) \ominus v_k(t_1)$ - the the disagreement set for $\{P(0), P(x)\}$.

Example of Unification in Neural Networks: time $= t_1$.



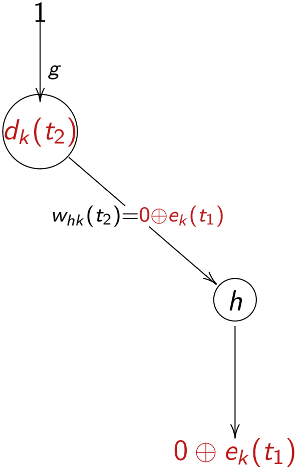
$w_{ki}(t_1) = v_i(t_1) = g$ is the number of $P(x)$;
$d_k(t_1) = h$ is the number of $P(0)$;
Compute $e_k(t_1) = d_k(t_1) \ominus v_k(t_1)$ - the the disagreement set for $\{P(0), P(x)\}$. $\Delta w(t_1) = v_i(t_1)e_k(t_1) = e_k(t_1)$.

Example of Unification in Neural Networks: time $= t_{1-2}$.



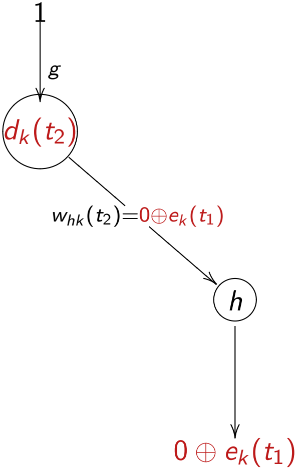
$w_{ki}(t_1) = v_i(t_1) = g$ is the number of $P(x)$;
$d_k(t_1) = h$ is the number of $P(0)$;
Compute $e_k(t_1) = d_k(t_1) \ominus v_k(t_1)$ - the the disagreement set for $\{P(0), P(x)\}$.
$\Delta w(t_1) = v_i(t_1) e_k(t_1) = e_k(t_1)$.
Substitutions are applied:
$w_{ki}(t_2) = w_{ki}(t_1) \odot \Delta w(t_1)$ and $d_k(t_2) = d_k(t_1) \odot \Delta w(t_1)$.

Example of Unification in Neural Networks: time $= t_2$.



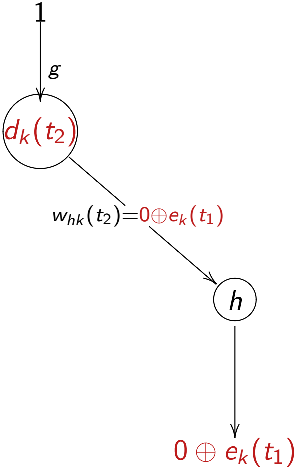
$$w_{hk}(t_2) = w_{hk}(t_1) \oplus e_k(t_1).$$

Example of Unification in Neural Networks: time = t_2.



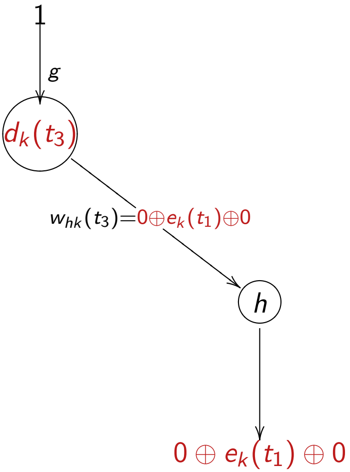
$w_{hk}(t_2) = w_{hk}(t_1) \oplus e_k(t_1)$.
$w_{ik}(t_2) = v_i(t_2) = g$ is the number of $P(0)$;
$d_k(t_2) = g$ is the number of $P(0)$.

Example of Unification in Neural Networks: time $= t_2$.



$w_{hk}(t_2) = w_{hk}(t_1) \oplus e_k(t_1)$.
$w_{ik}(t_2) = v_i(t_2) = g$ is the number of $P(0)$;
$d_k(t_2) = g$ is the number of $P(0)$.
$v_i(t_2) \ominus d_k(t_2) = \emptyset$. This means that we set $e_k(t_2) = 0$.

Example of Unification in Neural Networks: time $= t_3$.



$w_{hk}(t_3) = w_{hk}(t_2) \oplus 0$;
$v_h(t_3) = w_{hk}(t_3)$.
When v_h starts and ends with 0, computation stops.

Conclusions

Properties of these neural networks

- First-order atoms are embedded directly into a neural network.
- Neural networks are finite and give deterministic results, comparing with infinite layers needed to perform substitutions in T_P-neural networks.
- Unification algorithm is performed as an adaptive process, which corrects one piece of data relatively to the other piece of data.

Conclusions

Discussion

- Does the main theorem really define a connectionist neural network?
- Does the network really learn?
- Can we use these networks for massively parallel computations?
- What is the significance of these neural networks?

Future Work

- Practical implementations of SLD neural networks.

Future Work

- Practical implementations of SLD neural networks.
- Theoretical development:
  - SLD neural networks allow higher-order generalisations.
  - ...can therefore be extended to higher-order Horn logics, hereditary Harrop logics...
  - ...can be extended to non-classical logic programs: linear, many-valued, etc...
  - Inductive logic and SLD neural networks.
  - Try proof methods such as sequent calculus and tableaux instead of SLD-resolution...

My Super-Wish-List

I wish...

- ...to use parallelism of NNs in implementations of SLD-resolution, and thus to show that these Neural networks bring computational advantage to proof theory.
  - Undecidability of second-order unification would be a target...

My Super-Wish-List

I wish...

- ...to use parallelism of NNs in implementations of SLD-resolution, and thus to show that these Neural networks bring computational advantage to proof theory.
    * Undecidability of second-order unification would be a target...
- ...to show that learning laws bring advantages to computational logic...

Thank you!