Logic programs with uncertainty and neural computations

Ekaterina Komendantskaya Anthony Seda

Department of Mathematics, University College Cork, Ireland

International Conference Computability in Europe,
30 June – 5 July, 2006

Outline

1. Motivation
   - Neuro-Symbolic Integration
   - Connectionist Neural Networks and Logic Programs

Outline

1. Motivation
   - Neuro-Symbolic Integration
   - Connectionist Neural Networks and Logic Programs
2. Reasoning with Uncertainty
   - Bilattice-based Annotated Logic Programs
   - Introducing Learning into Connectionist Neural Networks
   - Integration of the Two Paradigms

Outline

1. Motivation
   - Neuro-Symbolic Integration
   - Connectionist Neural Networks and Logic Programs
2. Reasoning with Uncertainty
   - Bilattice-based Annotated Logic Programs
   - Introducing Learning into Connectionist Neural Networks
   - Integration of the Two Paradigms
3. Conclusions and Ongoing Work

Motivation

Symbolic Logic as Deductive System

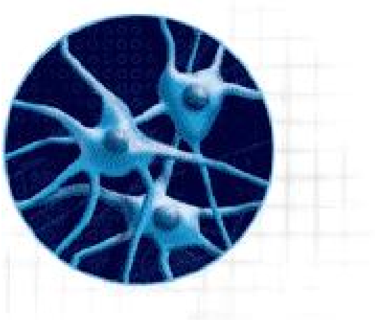
1. Axioms: $(A \supset (B \supset A))$; $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$; $(((\neg B) \supset (\neg A)) \supset (((\neg B) \supset A) \supset B))$; $((\forall x A) \supset S^x_t A)$; $\forall x(A \supset B)) \supset (A \supset \forall x B)))$;
2. Rules: $\dfrac{A \supset B,\ A}{B}$; $\dfrac{A}{\forall x A}$.

Motivation

Symbolic Logic as Deductive System

1. Axioms: $(A \supset (B \supset A))$; $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$; $(((\neg B) \supset (\neg A)) \supset (((\neg B) \supset A) \supset B))$; $((\forall x A) \supset S^x_t A)$; $\forall x(A \supset B)) \supset (A \supset \forall x B)))$;
2. Rules: $\frac{A \supset B,\ A}{B}$; $\frac{A}{\forall x A}$.

Neural Networks



- spontaneous behavior;
- learning and adaptation

Motivation

Logic Programs

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

Motivation

Logic Programs

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

Motivation

Logic Programs

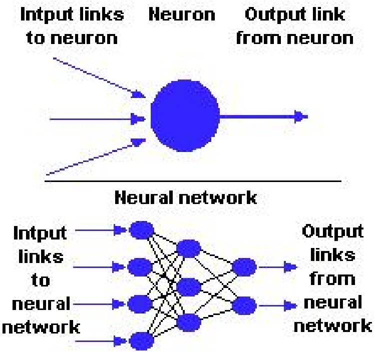
- $A \leftarrow B_1, \ldots, B_n$
- $T_P(I) = \{A \in B_P :$
  $A \leftarrow B_1, \ldots, B_n$
  is a ground instance of a clause in P and
  $\{B_1, \ldots, B_n\} \subseteq I\}$
- $\mathtt{lfp}(T_P \uparrow \omega) =$ the least Herbrand model of P.

Motivation

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
  $\{B_1, \ldots, B_n\} \subseteq I\}$
- $\texttt{lfp}(T_P \uparrow \omega)$ = the least Herbrand model of P.

Artificial Neural Networks



An Important Result, [Kalinke, Hölldobler, Storr]

Theorem

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

- No learning or adaptation

A Simple Example

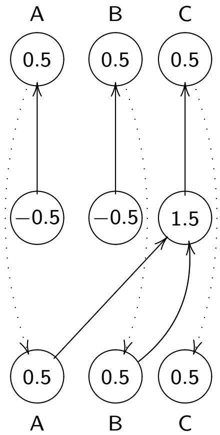
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

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

A Simple Example

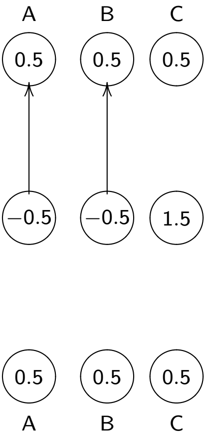
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

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



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

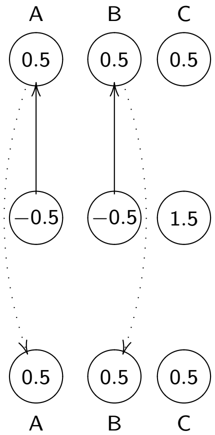
$B \leftarrow$

$A \leftarrow$

$C \leftarrow A, B$

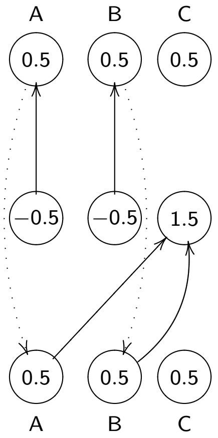
$T_P \uparrow 0 = \{B, A\}$

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



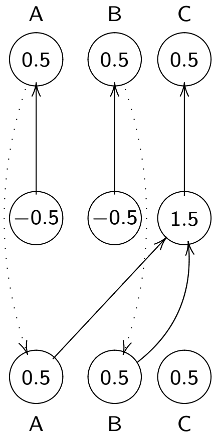
A Simple Example

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



A Simple Example

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



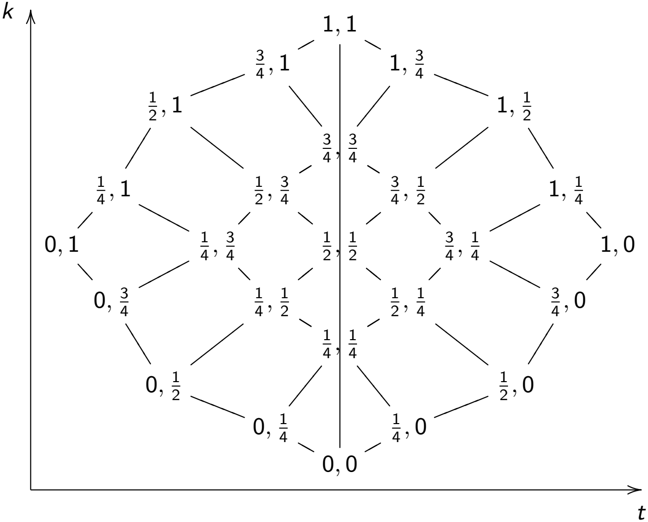
Reasoning with Uncertainty

Reasoning with uncertainty:

- (Automated) Reasoning with incomplete and inconsistent databases
- Requires spontaneous learning
- Possible field for integration of logic and neural networks

Bilattices [Ginsberg]

$\mathbf{B}_{25} = (\mathbf{B}, \leq_k, \leq_t, \neg) = L_1 \times L_2$, with $L_1 = L_2 = (\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}, \leq)$.



Bilattice-based annotated logic programs

$$A \quad \leftarrow L_1 \quad \ldots \quad L_n$$

Bilattice-based annotated logic programs

$$A : (\mu, \nu) \leftarrow L_1 : (\mu_1, \nu_1) \otimes \ \ldots \ \otimes L_n : (\mu_n, \nu_n)$$

Bilattice-based annotated logic programs

$$A : (\mu, \nu) \leftarrow L_1 : (\mu_1, \nu_1) \otimes \ \ldots \ \otimes L_n : (\mu_n, \nu_n)$$

$\mathcal{T}_P(\mathtt{HI}) = A : (\mu, \nu) \in B_P$:

1. $\{L_1 : (\mu'_1, \nu'_1), \ldots, L_n : (\mu'_n, \nu'_n)\} \subseteq \mathtt{HI}$, and one of the following conditions holds for each (μ'_i, ν'_i):
   - $(\mu'_i, \nu'_i) \geq_k (\mu_i, \nu_i)$,

Bilattice-based annotated logic programs

$$A : (\mu, \nu) \leftarrow L_1 : (\mu_1, \nu_1) \otimes \ \ldots \ \otimes L_n : (\mu_n, \nu_n)$$

$\mathcal{T}_P(\mathtt{HI}) = A : (\mu, \nu) \in B_P$:

1. $\{L_1 : (\mu_1', \nu_1'), \ldots, L_n : (\mu_n', \nu_n')\} \subseteq \mathtt{HI}$, and one of the following conditions holds for each (μ_i', ν_i'):
   - $(\mu_i', \nu_i') \geq_k (\mu_i, \nu_i)$,
   - $(\mu_i', \nu_i') \geq_k \otimes_{j \in J_i}(\mu_j, \nu_j)$, where J_i is the finite set of indices such that $L_j = L_i$

Bilattice-based annotated logic programs

$$A : (\mu, \nu) \leftarrow L_1 : (\mu_1, \nu_1) \otimes \ \ldots \ \otimes L_n : (\mu_n, \nu_n)$$

$\mathcal{T}_P(\mathtt{HI}) = A : (\mu, \nu) \in B_P :$

1. $\{L_1 : (\mu'_1, \nu'_1), \ldots, L_n : (\mu'_n, \nu'_n)\} \subseteq \mathtt{HI}$, and one of the following conditions holds for each (μ'_i, ν'_i):
   - $(\mu'_i, \nu'_i) \geq_k (\mu_i, \nu_i)$,
   - $(\mu'_i, \nu'_i) \geq_k \otimes_{j \in J_i}(\mu_j, \nu_j)$, where J_i is the finite set of indices such that $L_j = L_i$
2. or there are annotated strictly ground atoms $A : (\mu^*_1, \nu^*_1), \ldots, A : (\mu^*_k, \nu^*_k) \in \mathtt{HI}$ such that $\langle \mu, \nu \rangle \leq_k \langle \mu^*_1, \nu^*_1 \rangle \oplus \ldots \oplus \langle \mu^*_k, \nu^*_k \rangle$.

A simple example of bilattice-based logic program

$$B : (0,1) \leftarrow,$$
$$B : (1,0) \leftarrow,$$
$$A : (0,0) \leftarrow B : (1,1),$$
$$C : (1,1) \leftarrow A : (1,0) \otimes A : (0,1)$$

A simple example of bilattice-based logic program

$B : (0, 1) \leftarrow,$
$B : (1, 0) \leftarrow,$
$A : (0, 0) \leftarrow B : (1, 1),$
$C : (1, 1) \leftarrow A : (1, 0) \otimes A : (0, 1)$

- $T_P \uparrow 1 = \{B : (0, 1), B : (1, 0)\}$

A simple example of bilattice-based logic program

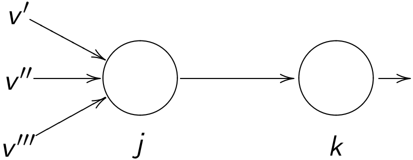
$B : (0,1) \leftarrow,$
$B : (1,0) \leftarrow,$
$A : (0,0) \leftarrow B : (1,1),$
$C : (1,1) \leftarrow A : (1,0) \otimes A : (0,1)$

- $T_P \uparrow 1 = \{B : (0,1), B : (1,0)\}$
- $\mathcal{T}_P \uparrow 3 = \{B : (0,0), B : (0,1), B : (1,0), B : (1,1), A : (0,0), C : (0,0), C(0,1), C(1,0), C : (1,1)\}$

Hebbian learning

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

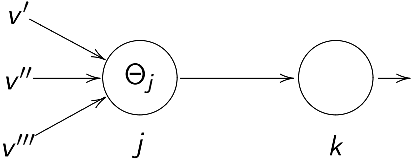
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Hebbian learning

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

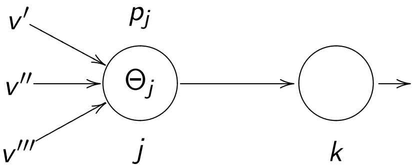
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Hebbian learning

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

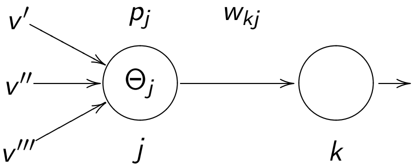
$$v_k(t+\Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } \; p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Hebbian learning

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

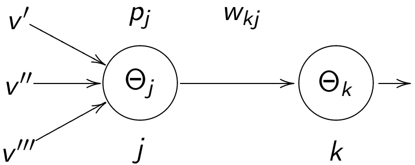
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Hebbian learning

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

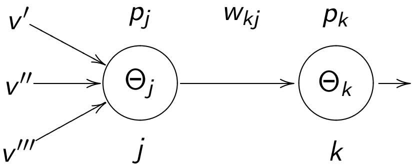
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Hebbian learning

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

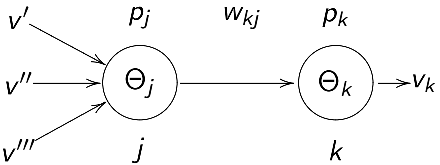
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Hebbian learning

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

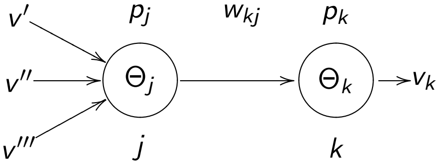
$$v_k(t + \Delta t) = \psi(p_k(t)) = \begin{cases} 1 & \texttt{if } p_k(t) > 0 \\ 0 & \texttt{otherwise.} \end{cases}$$



Hebbian learning

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

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

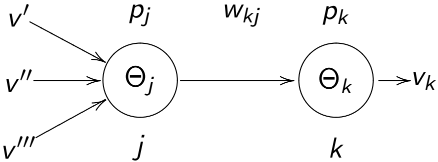


- $\Delta w_{kj}(t) = F(v_j(t), p_k(t))$
- $\Delta w_{kj}(t) = \eta(v_j(t))(p_k(t))$

Hebbian learning

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

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



- $\Delta w_{kj}(t) = F(v_j(t), p_k(t))$
- $\Delta w_{kj}(t) = \eta(v_j(t))(p_k(t))$
- $\phi_1 = \Delta w_{ci}(t) = -(v_i(t))(p_c(t) - 0.5)$
- $\phi_2 = \Delta w_{oc}(t) = (v_c(t))(p_o(t) + 1.5)$

The Main Theorem

Theorem

For each function-free bilattice-based annotated logic program P, there exists a 3-layer feedforward learning artificial neural network which computes $\mathcal{T}_P$.

The Main Theorem

Theorem

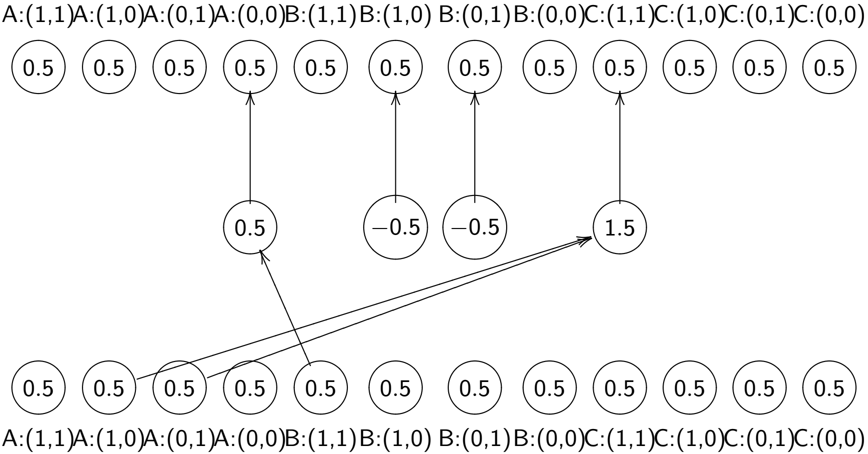
For each function-free bilattice-based annotated logic program P, there exists a 3-layer feedforward learning artificial neural network which computes $\mathcal{T}_P$.

An Approximation Result

For each level of approximation l and for each first-order bilattice-based logic program P with functions in annotations there exists a finite family of finite artificial neural networks which approximates $\texttt{lfp}(\mathcal{T}_P)$.

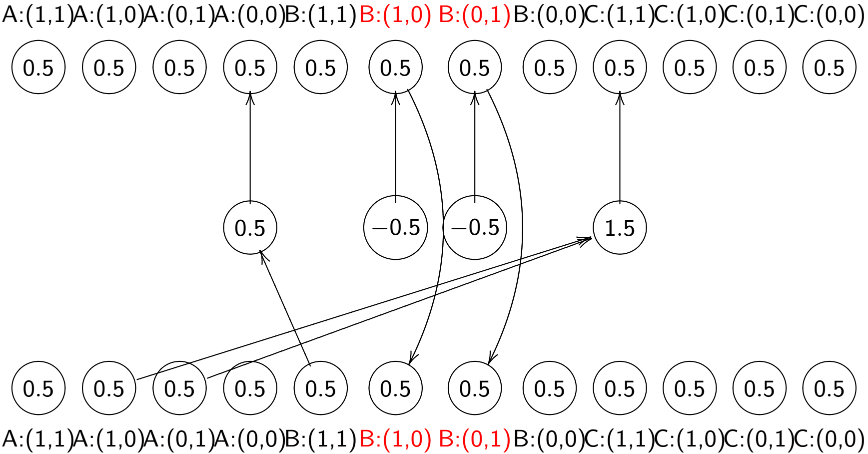
A Learning Neural Network

$B:(0,1) \leftarrow;\quad B:(1,0) \leftarrow;\quad A:(0,0) \leftarrow B:(1,1);$
$C:(1,1) \leftarrow A:(1,0) \otimes A:(0,1)$



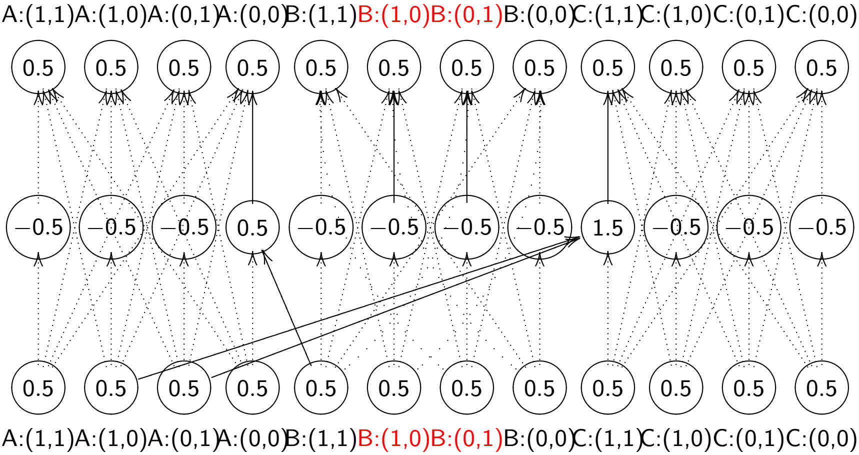
A Learning Neural Network

$B:(0,1) \leftarrow; \quad B:(1,0) \leftarrow; \quad A:(0,0) \leftarrow B:(1,1);$
$C:(1,1) \leftarrow A:(1,0) \otimes A:(0,1)$



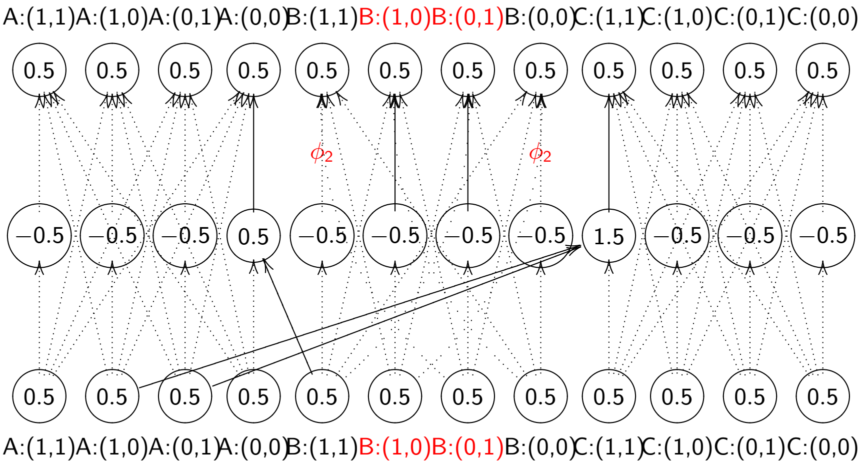
A Learning Neural Network

$$B:(0,1)\leftarrow;\quad B:(1,0)\leftarrow;\quad A:(0,0)\leftarrow B:(1,1);$$
$$C:(1,1)\leftarrow A:(1,0)\otimes A:(0,1)$$



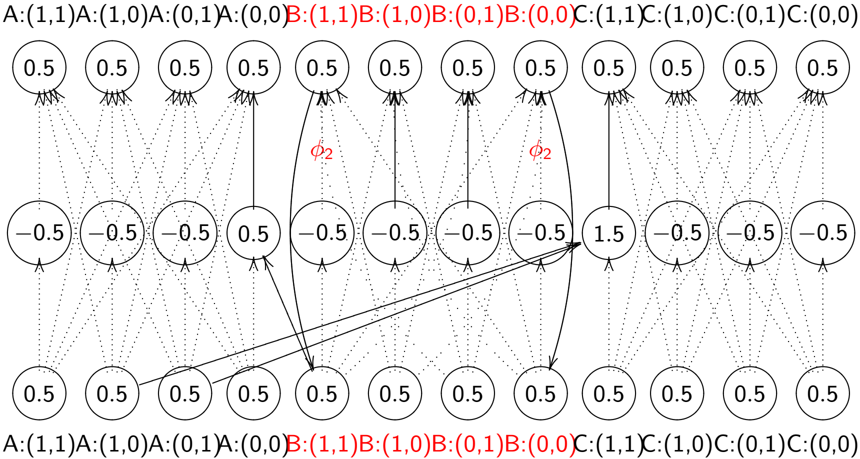
A Learning Neural Network

$B:(0,1) \leftarrow; \quad B:(1,0) \leftarrow; \quad A:(0,0) \leftarrow B:(1,1);$
$C:(1,1) \leftarrow A:(1,0) \otimes A:(0,1)$



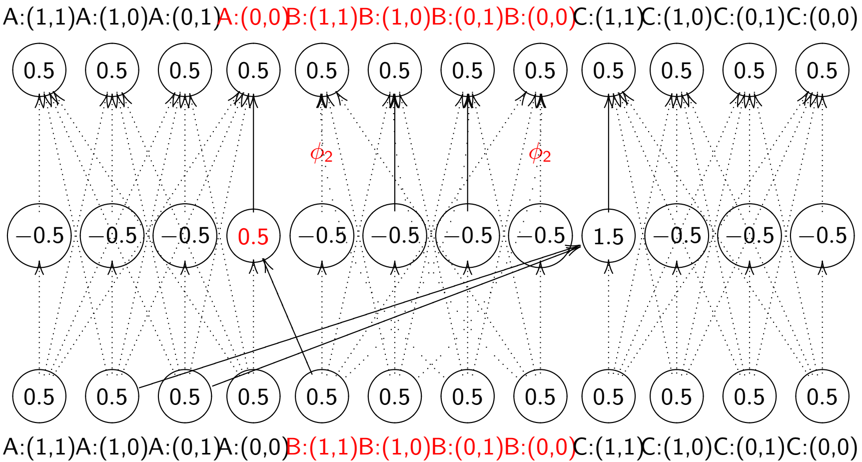
A Learning Neural Network

$B:(0,1)\leftarrow;\quad B:(1,0)\leftarrow;\quad A:(0,0)\leftarrow B:(1,1);$
$C:(1,1)\leftarrow A:(1,0)\otimes A:(0,1)$



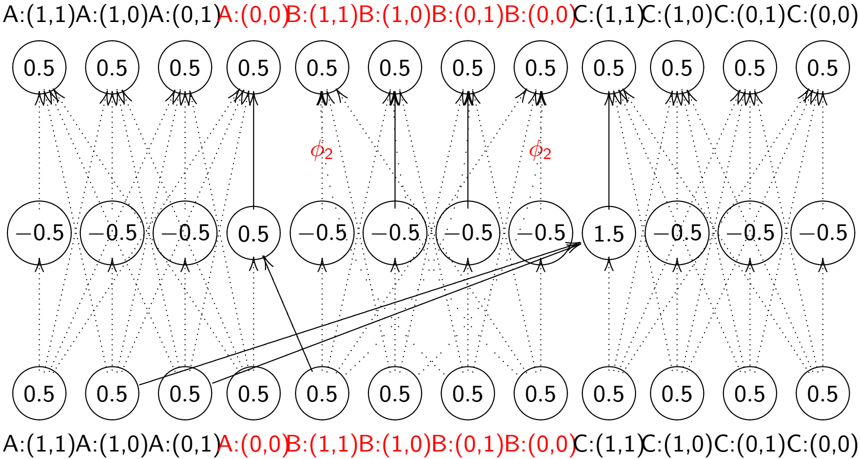
A Learning Neural Network

$$B:(0,1) \leftarrow; \quad B:(1,0) \leftarrow; \quad A:(0,0) \leftarrow B:(1,1);$$
$$C:(1,1) \leftarrow A:(1,0) \otimes A:(0,1)$$



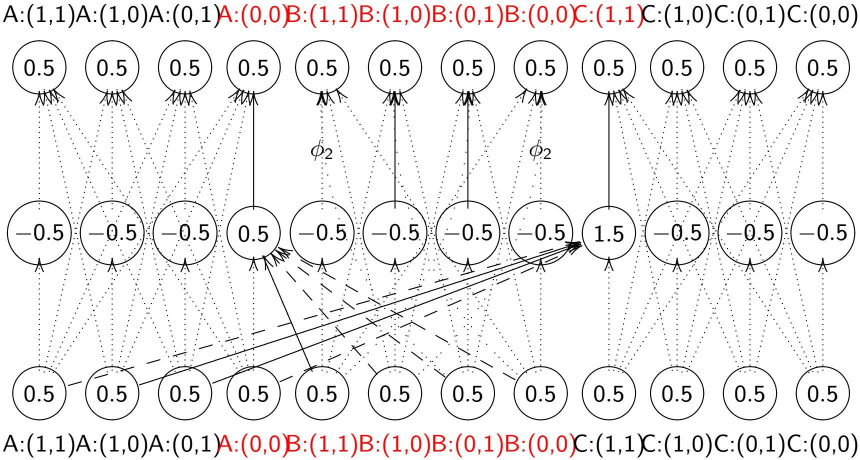
A Learning Neural Network

$B:(0,1)\leftarrow;\quad B:(1,0)\leftarrow;\quad A:(0,0)\leftarrow B:(1,1);$
$C:(1,1)\leftarrow A:(1,0)\otimes A:(0,1)$



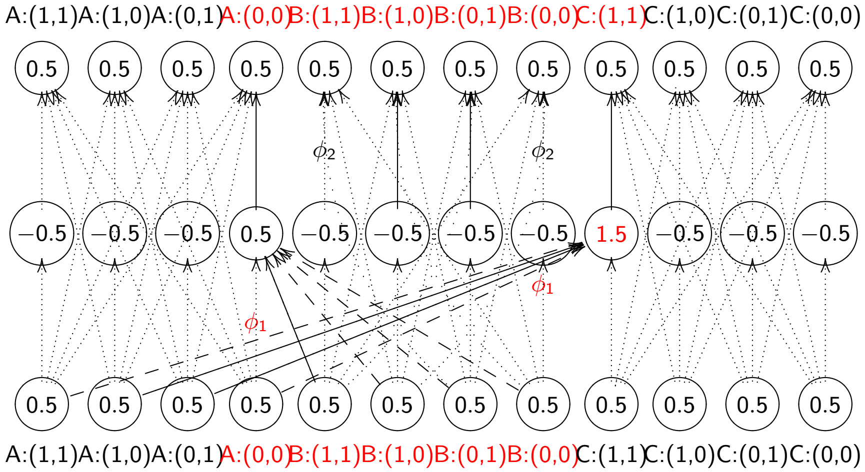
A Learning Neural Network

$$B:(0,1)\leftarrow; \quad B:(1,0)\leftarrow; \quad A:(0,0)\leftarrow B:(1,1);$$
$$C:(1,1)\leftarrow A:(1,0)\otimes A:(0,1)$$



A Learning Neural Network

$B:(0,1)\leftarrow;\quad B:(1,0)\leftarrow;\quad A:(0,0)\leftarrow B:(1,1);$
$C:(1,1)\leftarrow A:(1,0)\otimes A:(0,1)$



Conclusions and Ongoing Work

- Automated reasoning as an opposite to neural computations: SLD-resolution for bilattice-based logic programs.
- Categorical semantics for logic programs and neural networks: search for a uniform picture

Thank you!