Neural-Symbolic Integration

A selfcontained introduction

Sebastian Bader Pascal Hitzler

ICCL, Technische Universität Dresden, Germany

AIFB, Universität Karlsruhe, Germany

Outline of the Course

- Introduction and Motivation
- The Core Method for Propositional Logic
- Applications of the Propositional Core Method
- The Core Method for First-Order Logic
- More on First-Order & other Perspectives

Part

The Core-Method for Propositional Logic

Outline: The Core-Method for Propositional Logic

Propositional Logic Programs

The Core Method for Propositional Logic

CILLP and some Derivatives

Conclusions

Outline: The Core-Method for Propositional Logic

Propositional Logic Programs

The Core Method for Propositional Logic

CILLP and some Derivatives

Conclusions

Propositional Logic Programs – An Example

$A \leftarrow \neg B.$	*% A is true, if B is false.*
$B \leftarrow A \wedge \neg B.$	*% B is true, if A is true and B is false.*
$B \leftarrow B.$	*% B is true, if B is true.*

Propositional Logic Programs – The Syntax

Definition (Propositional Variables & Connectives)

$$A, B, C, D, \ldots \qquad \wedge = \text{“and”} \qquad \leftarrow = \text{“if-then”} \qquad \neg = \text{“not”}$$

Definition (Clause)

$$\underbrace{H}_{\text{head}} \leftarrow \underbrace{L_1 \wedge L_2 \wedge \ldots \wedge L_n.}_{\text{body with } L_i \text{ either } X \text{ or } \neg X}$$

Definition (Propositional Logic Program)

A propositional logic program is a finite set of clauses.

Propositional Logic Programs – The Semantics

Definition (Herbrand Base $B_{\mathcal{L}}$)

The Herbrand base is the set of all variables occuring in P.

Example ($B_{\mathcal{L}}$ for the running example)

$$B_{\mathcal{L}} = \{A, B\}$$

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

Definition (Interpretation)

An interpretation is a subset of the Herbrand base.

Example (Interpretations for the running example)

$$I_1 = \emptyset \qquad I_2 = \{A\} \qquad I_3 = \{B\} \qquad I_4 = \{A, B\}$$

Propositional Logic Programs – The Semantics Ctd.

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

Example (For $I_2 = \{A\}$)

$(A)^{I_2} = \mathit{true}$	$(\neg A)^{I_2} = \mathit{false}$
$(B)^{I_2} = \mathit{false}$	$(\neg B)^{I_2} = \mathit{true}$

Propositional Logic Programs – The Semantics Ctd.

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$

Example (For $I_2 = \{A\}$)

$$(A)^{I_2} = \mathit{true} \qquad (\neg A)^{I_2} = \mathit{false}$$

$$(B)^{I_2} = \mathit{false} \qquad (\neg B)^{I_2} = \mathit{true}$$

$$(A \leftarrow \neg B)^{I_2} = \mathit{true} \qquad (B \leftarrow B)^{I_2} = \mathit{true}$$

$$(A \wedge \neg B)^{I_2} = \mathit{true} \qquad (B \leftarrow A \wedge \neg B)^{I_2} = \mathit{false}$$

Propositional Logic Programs – The Semantics Ctd.

Definition (Model)

An interpretation M satisfying every clause of a program P is called a model of P (in symbols $M \models P$).

Example (Models of the running example)

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

$$\emptyset \not\models P$$
$$\{A\} \not\models P$$
$$\{B\} \models P$$
$$\{A, B\} \models P$$

The Immediate Consequence Operator T_P

Definition (T_P)

$T_P(I) = \{A \mid \text{there is a clause } A \leftarrow \mathit{body} \text{ in } P \text{ and } I \models \mathit{body}\}$

- The T_P-operator propagates truth along the clauses.

The Immediate Consequence Operator T_P

Definition (T_P)

$$T_P(I) = \{A \mid \text{there is a clause } A \leftarrow \mathit{body} \text{ in } P \text{ and } I \models \mathit{body}\}$$

- The T_P-operator propagates truth along the clauses.

Example (T_P for our running example)

$$\begin{aligned} A &\leftarrow \neg B. \\ B &\leftarrow A \wedge \neg B. \\ B &\leftarrow B. \end{aligned}$$

$$\begin{aligned} \{\} &\mapsto \{A\} \\ \{A\} &\mapsto \{A, B\} \\ \{B\} &\mapsto \{B\} \\ \{A, B\} &\mapsto \{B\} \end{aligned}$$

- For definite programs, T_P converges to the least model.

Outline: The Core-Method for Propositional Logic

Constructing the Core-Network

1. For each element of $B_{\mathcal{L}}$, add an input unit and an output unit with threshold 0.5.
2. For each clause $H \leftarrow L_1 \dots L_n$ do the following:
   2.1 Add a hidden unit c and a connection to H' ($w = 1.0$).
   2.2 Connect every L_i and c with
   $$w = \begin{cases} +1.0 & \text{if } L_i \text{ is positive,} \\ -1.0 & \text{if } L_i \text{ is negated.} \end{cases}$$
   2.3 Set the threshold of c to "number of pos. L_i"-0.5.

Example

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

Constructing the Core-Network

1. For each element of $B_{\mathcal{L}}$, add an input unit and an output unit with threshold 0.5.
2. For each clause $H \leftarrow L_1 \ldots L_n$ do the following:
   2.1 Add a hidden unit c and a connection to H' ($w = 1.0$).
   2.2 Connect every L_i and c with
   $$w = \begin{cases} +1.0 & \text{if } L_i \text{ is positive,} \\ -1.0 & \text{if } L_i \text{ is negated.} \end{cases}$$
   2.3 Set the threshold of c to "number of pos. L_i"-0.5.

Example

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

Constructing the Core-Network

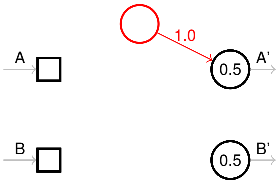
1. For each element of $B_{\mathcal{L}}$, add an input unit and an output unit with threshold 0.5.
2. For each clause $H \leftarrow L_1 \ldots L_n$ do the following:
   2.1 Add a hidden unit c and a connection to H' ($w = 1.0$).
   2.2 Connect every L_i and c with
   $$w = \begin{cases} +1.0 & \text{if } L_i \text{ is positive,} \\ -1.0 & \text{if } L_i \text{ is negated.} \end{cases}$$
   2.3 Set the threshold of c to "number of pos. L_i"-0.5.

Example

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$

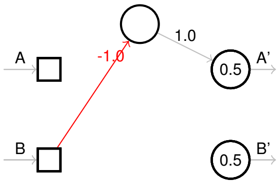


Constructing the Core-Network

1. For each element of $B_{\mathcal{L}}$, add an input unit and an output unit with threshold 0.5.
2. For each clause $H \leftarrow L_1 \ldots L_n$ do the following:
   2.1 Add a hidden unit c and a connection to H' ($w = 1.0$).
   2.2 Connect every L_i and c with
   $$w = \begin{cases} +1.0 & \text{if } L_i \text{ is positive,} \\ -1.0 & \text{if } L_i \text{ is negated.} \end{cases}$$
   2.3 Set the threshold of c to "number of pos. L_i"-0.5.

Example

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

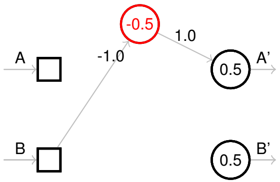


Constructing the Core-Network

1. For each element of $B_{\mathcal{L}}$, add an input unit and an output unit with threshold 0.5.
2. For each clause $H \leftarrow L_1 \ldots L_n$ do the following:
   2.1 Add a hidden unit c and a connection to H' ($w = 1.0$).
   2.2 Connect every L_i and c with
   $$w = \begin{cases} +1.0 & \text{if } L_i \text{ is positive,} \\ -1.0 & \text{if } L_i \text{ is negated.} \end{cases}$$
   2.3 Set the threshold of c to "number of pos. L_i"-0.5.

Example

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

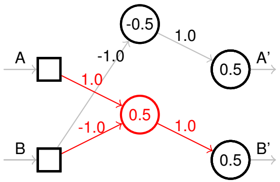


Constructing the Core-Network

1. For each element of $B_{\mathcal{L}}$, add an input unit and an output unit with threshold 0.5.
2. For each clause $H \leftarrow L_1 \ldots L_n$ do the following:
   2.1 Add a hidden unit c and a connection to H' ($w = 1.0$).
   2.2 Connect every L_i and c with
   $$w = \begin{cases} +1.0 & \text{if } L_i \text{ is positive,} \\ -1.0 & \text{if } L_i \text{ is negated.} \end{cases}$$
   2.3 Set the threshold of c to "number of pos. L_i"-0.5.

Example

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

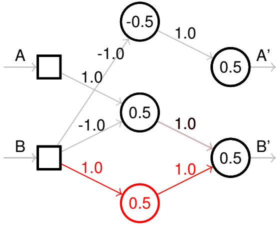


Constructing the Core-Network

1. For each element of $B_{\mathcal{L}}$, add an input unit and an output unit with threshold 0.5.
2. For each clause $H \leftarrow L_1 \dots L_n$ do the following:
   2.1 Add a hidden unit c and a connection to H' ($w = 1.0$).
   2.2 Connect every L_i and c with
   $$w = \begin{cases} +1.0 & \text{if } L_i \text{ is positive,} \\ -1.0 & \text{if } L_i \text{ is negated.} \end{cases}$$
   2.3 Set the threshold of c to "number of pos. L_i"-0.5.

Example

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

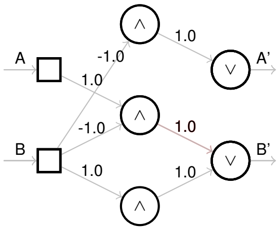


Constructing the Core-Network

1. For each element of $B_{\mathcal{L}}$, add an input unit and an output unit with threshold 0.5.
2. For each clause $H \leftarrow L_1 \ldots L_n$ do the following:
   2.1 Add a hidden unit c and a connection to H' ($w = 1.0$).
   2.2 Connect every L_i and c with
   $$w = \begin{cases} +1.0 & \text{if } L_i \text{ is positive,} \\ -1.0 & \text{if } L_i \text{ is negated.} \end{cases}$$
   2.3 Set the threshold of c to "number of pos. L_i"-0.5.

Example

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

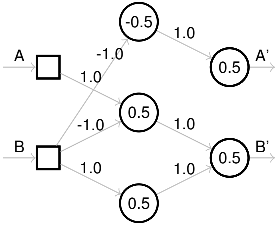


One Application of T_P

$A \leftarrow \neg B.$

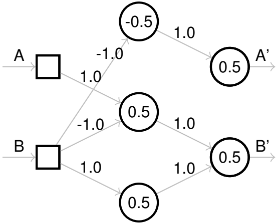
$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$

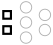


One Application of T_P

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$



$$\{\} \mapsto \{A\}$$

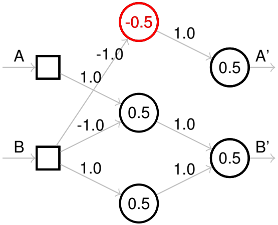


One Application of T_P

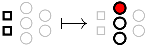
$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$



$\{\} \mapsto \{A\}$

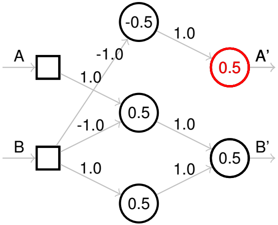


One Application of T_P

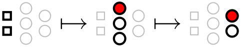
$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$

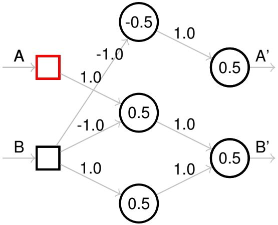


$\{\} \mapsto \{A\}$

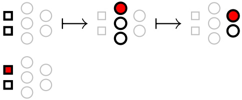


One Application of T_P

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

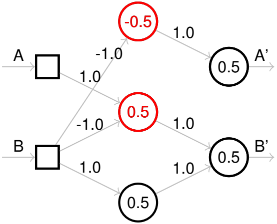


$$\{\} \mapsto \{A\}$$
$$\{A\} \mapsto \{A, B\}$$

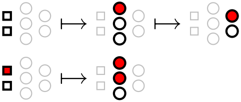


One Application of T_P

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$



$$\{\} \mapsto \{A\}$$
$$\{A\} \mapsto \{A, B\}$$

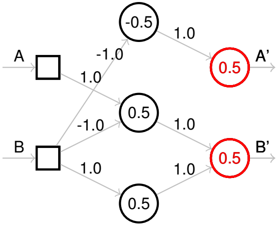


One Application of T_P

$A \leftarrow \neg B.$

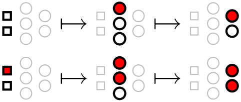
$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$



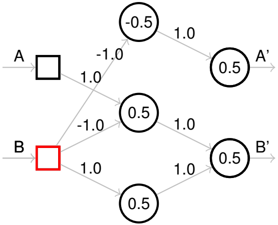
$\{\} \mapsto \{A\}$

$\{A\} \mapsto \{A, B\}$

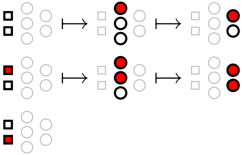


One Application of T_P

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

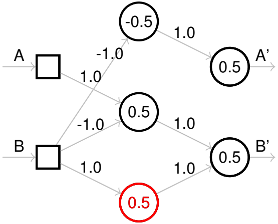


$$\{\} \mapsto \{A\}$$
$$\{A\} \mapsto \{A, B\}$$
$$\{B\} \mapsto \{B\}$$

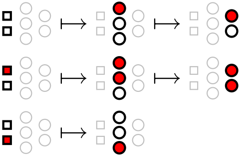


One Application of T_P

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$



$$\{\} \mapsto \{A\}$$
$$\{A\} \mapsto \{A, B\}$$
$$\{B\} \mapsto \{B\}$$

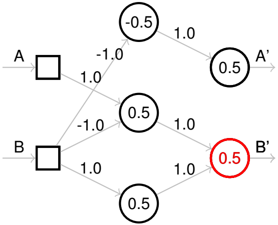


One Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

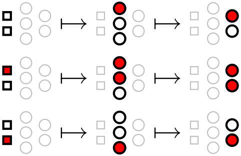
$B \leftarrow B.$



$\{\} \mapsto \{A\}$

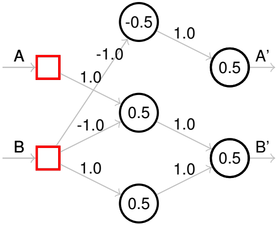
$\{A\} \mapsto \{A, B\}$

$\{B\} \mapsto \{B\}$

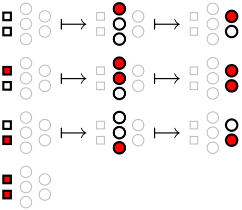


One Application of T_P

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

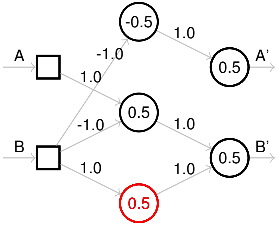


$$\{\} \mapsto \{A\}$$
$$\{A\} \mapsto \{A, B\}$$
$$\{B\} \mapsto \{B\}$$
$$\{A, B\} \mapsto \{B\}$$

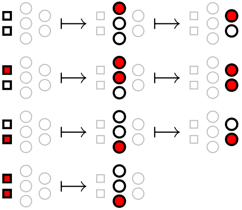


One Application of T_P

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow B.$$

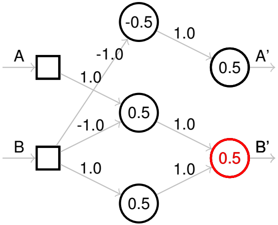


$$\{\} \mapsto \{A\}$$
$$\{A\} \mapsto \{A, B\}$$
$$\{B\} \mapsto \{B\}$$
$$\{A, B\} \mapsto \{B\}$$

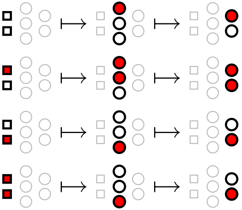


One Application of T_P

$$A \leftarrow \neg B.$$
$$B \leftarrow A \wedge \neg B.$$
$$\color{red}{B} \leftarrow B.$$



$$\{\} \mapsto \{A\}$$
$$\{A\} \mapsto \{A, B\}$$
$$\{B\} \mapsto \{B\}$$
$$\{A, B\} \mapsto \{B\}$$

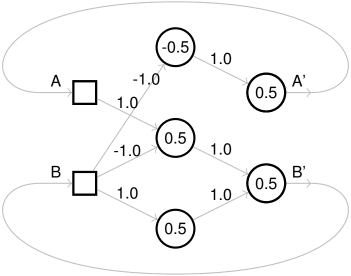


Repetitive Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$

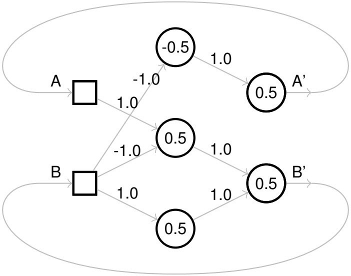


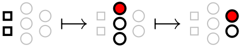
Repetitive Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$



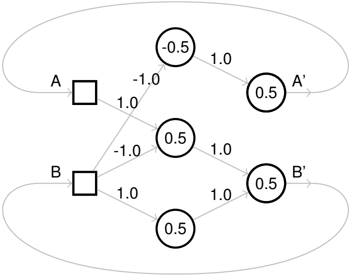


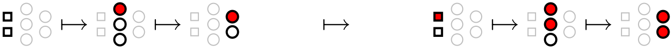
Repetitive Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$



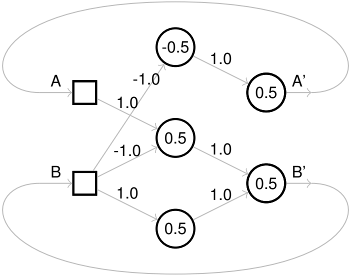


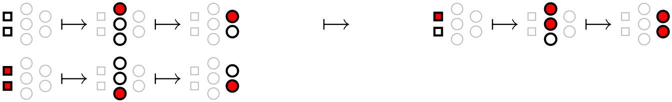
Repetitive Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$



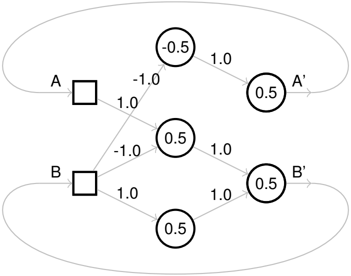


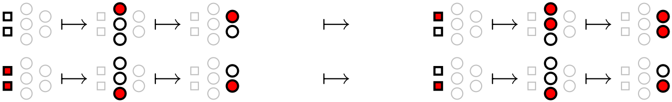
Repetitive Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$



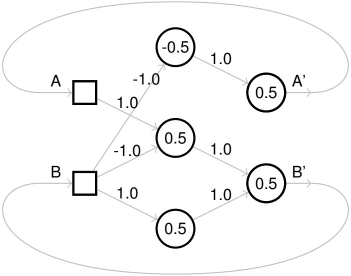


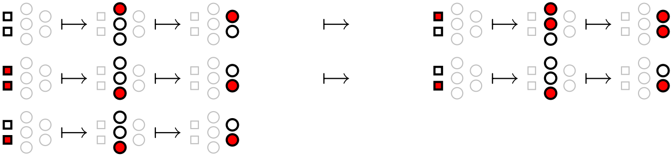
Repetitive Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$



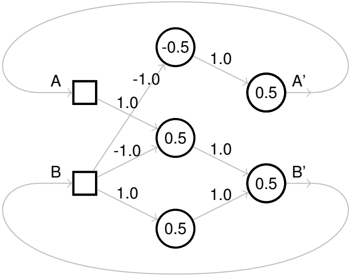


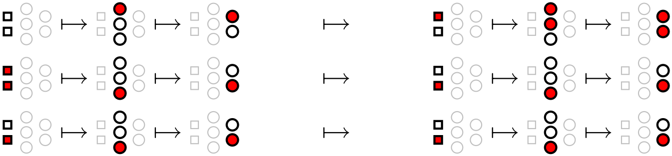
Repetitive Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$



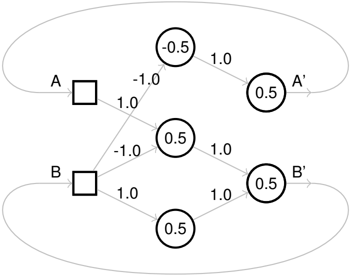


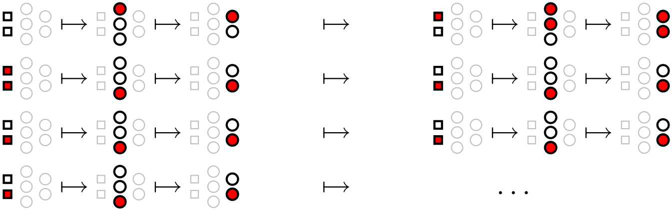
Repetitive Application of T_P

$A \leftarrow \neg B.$

$B \leftarrow A \wedge \neg B.$

$B \leftarrow B.$





Main Results (Hölldobler & Kalinke, 1994)

- 2-layer networks cannot compute T_P.
- For each program P there exists a 3-layer kernel computing T_P.

Space and Time Complexity

Let n be the number of clauses, m be the number of propositional variables:

- $2m + n$ units, $2mn$ connections in the kernel.
- $T_P(I)$ is computed in 2 steps.
- The parallel model to compute T_P is optimal.
- The recurrent network settles down in at most $3n$ steps.

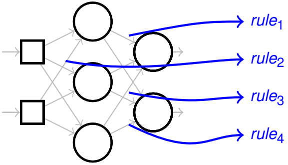
Extraction Methods

- Single units do not necessarily correspond to single rules.
- In general: It is NP-complete to find the minimal logical description for a trained network (Golea, 1996).
- There is not allways a single minimal program (Lehmann, Bader & Hitzler, 2005).

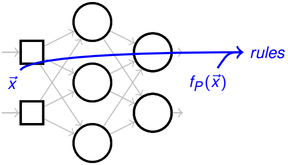
Extraction Methods

- Single units do not necessarily correspond to single rules.
- In general: It is NP-complete to find the minimal logical description for a trained network (Golea, 1996).
- There is not allways a single minimal program (Lehmann, Bader & Hitzler, 2005).

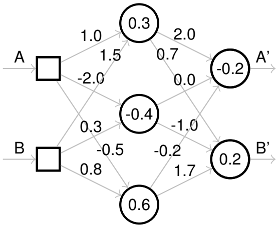
Decompositional



Pedagogical



Extraction – A Pedagogical Approach



A	**B**	c_1	c_2	c_3	**A'**	**B'**
0	**0**	0.0 / 0.0	0.0 / 1.0	0.0 / 0.0	0.0 / **1**	-1.0 / **0**
0	**1**	1.5 / 1.0	0.3 / 1.0	0.8 / 1.0	1.8 / **1**	0.7 / **1**
1	**0**	1.0 / 1.0	-2.0 / 0.0	-0.5 / 0.0	2.0 / **1**	0.7 / **1**
1	**1**	2.5 / 1.0	-1.7 / 0.0	0.3 / 0.0	2.0 / **1**	0.7 / **1**

Extraction – A Pedagogical Approach

A	B	A'	B'
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

$$A \leftarrow \neg A \wedge \neg B.$$
$$A \leftarrow \neg A \wedge B.$$
$$A \leftarrow A \wedge \neg B.$$
$$A \leftarrow A \wedge B.$$
$$B \leftarrow \neg A \wedge B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow A \wedge B.$$

Extraction – A Pedagogical Approach

$$A \leftarrow \neg A \wedge \neg B.$$
$$A \leftarrow \neg A \wedge B.$$
$$A \leftarrow A \wedge \neg B.$$
$$A \leftarrow A \wedge B.$$
$$B \leftarrow \neg A \wedge B.$$
$$B \leftarrow A \wedge \neg B.$$
$$B \leftarrow A \wedge B.$$

$$A.$$
$$B \leftarrow \neg A \wedge B.$$
$$B \leftarrow A.$$

Extraction – A Pedagogical Approach

- ☺ Sound, i.e. every extracted rule is a rule implemented by the network.
- ☺ Complete, i.e. every rule implemented by the network will be extracted.

- ☹ Bad time-complexity, due to the exponential blow-up.
- ☹ Does not create the smallest program automatically.

Main Results (Hölldobler & Kalinke, 1994)

- 2-layer networks cannot compute T_P.
- For each program P there exists a 3-layer kernel computing T_P.
- For each 3-layer kernel K there exists a program P, such that K computes T_P.
- Let n be the number of clauses, m be the number of propositional variables
  - $2m + n$ units, $2mn$ connections in the kernel.
  - $T_P(I)$ is computed in 2 steps.
  - The parallel model to compute T_P is optimal.
  - The recurrent network settles down in at most $3n$ steps.

Outline: The Core-Method for Propositional Logic

Propositional Logic Programs

The Core Method for Propositional Logic

CILLP and some Derivatives

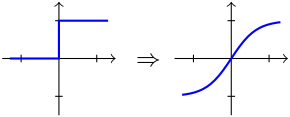
Conclusions

The CILLP-System

? Can the learning capabilities of ANNs be combined with the Core Method (Garcez & Zaverucha, 1999)?

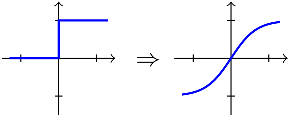
The CILLP-System

- **?** Can the learning capabilities of ANNs be combined with the Core Method (Garcez & Zaverucha, 1999)?
- ► Using sigmoidal / tanh functions, we obtain a standard 3-layer feed-forward neural network.



The CILLP-System

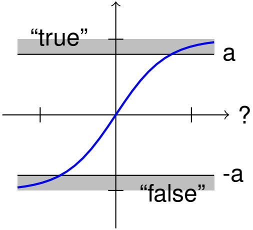
- ? Can the learning capabilities of ANNs be combined with the Core Method (Garcez & Zaverucha, 1999)?
- ▶ Using sigmoidal / tanh functions, we obtain a standard 3-layer feed-forward neural network.



- ▶ This network is trainable using back-propagation.

CILLP - The Construction

- Define ranges for ”true” and “false”:



- Compute *a*, the weights and thresholds such that the sigmoidal kernel computes T_P (Garcez & Zaverucha, 1999).

CILLP - Extracting a Learned Program

- The pedagogical approach would work, but ...
- Garcez, Broda & Gabbay (2001) proposed a suitable method, which ...
  - ☺ is sound.
  - ☺ is computational feasible due to clever restriction of the search space.
  - ☹ is not necessarily complete.
  - ☹ does not necessarily create the small programs.

CILLP - The MONK's Problems

- Robots are described by 6 properties,
  e.g. *head-shape* $\in$ {*round, square, octagon*}, ...
- Classification task: *"Recognice robots with (body-shape = head-shape) or (jacket-color = red)"*
- Network architecture:
  - 17 input units: one for each attribute.
  - 3 hidden layer units.
  - 1 output unit: indicating answer "yes" or "no".
- 100% performance of the network and extracted rules.
- Pruning: from 131072 possible inputs for some hidden unit, only 18724 were queried.

CILLP - Conclusions

- Successfully used for ...
  - classification tasks like the MONK's problem.
  - DNA sequence analysis (Promoter Recognition, Splice Junction Determination).
  - Power system fault diagnosis.
- Extensions of the CILLP-System:
  - Metalevel priorities between rules (Garcez, Broda & Gabbay , 2000).
  - Intuitionistic logic (Garcez, Lamb & Gabbay, 2003).
  - Modal logic (Garcez, Lamb, Broda & Gabbay, 2004).

Outline: The Core-Method for Propositional Logic

Propositional Logic Programs

The Core Method for Propositional Logic

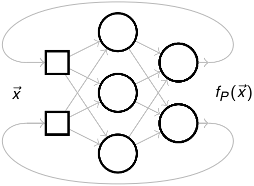
CILLP and some Derivatives

Conclusions

The Core Method

- Relate logic programs and connectionist systems
- Embed interpretations into (vectors of) real numbers.
- Hence, obtain an embedded version of the T_P-operator.
- Construct a network computing one application of f_P.
- Add recurrent connections from output to input layer.

$$
\begin{tikzcd}
I_{\mathcal{L}} \arrow[r, "T_P"] \arrow[d, "\iota"] & I_{\mathcal{L}} \\
\mathbb{R}^m \arrow[r, "f_P"] & \mathbb{R}^m \arrow[u, "\iota^{-1}"']
\end{tikzcd}
$$

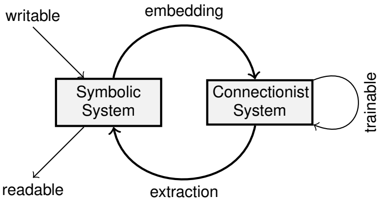


Major Problems in Neural-Symbolic Integration

- How can symbolic knowledge be *represented* within connectionist systems? (What is ι?)
- How can symbolic knowledge be *extracted* from connectionist systems? (What is ι^{-1}?)
- How can symbolic knowledge be *learned* using connectionist systems?
- How can connectionist *learning be guided* by symbolic background knowledge?

Conclusions

We have a complete system implementing the NeSy-Cycle for propositional logic programs.



Main Results

- 3-layer feedforward networks can compute T_P.
- Using sigmoidal units, the network is trainable using Back-Propagation.
- Logic programs can be extracted.
- Successfully applied to real world problems.

Outline of the Course

- Introduction and Motivation
- The Core Method for Propositional Logic
- Applications of the Propositional Core Method
- The Core Method for First-Order Logic
- More on First-Order & other Perspectives