Neural-Symbolic Integration
A self-contained 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.
- $B \leftarrow A \land \neg B$. % $B$ 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)

\[ A, B, C, D, \ldots \land = \text{“and”} \quad \leftarrow = \text{“if-then”} \quad \neg = \text{“not”} \]

Definition (Clause)

\[ \begin{aligned}
& \text{\{Head\}} \quad \leftarrow \\
& \text{\{Body\} with } L_i \text{ either } X \text{ or } \neg X
\end{aligned} \]

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 occurring in \( P \).

**Example (\( B_\mathcal{L} \) for the running example)**
\[
B_\mathcal{L} = \{A, B\}
\]

**Definition (Interpretation)**
An interpretation is a subset of the Herbrand base.

**Example (Interpretations for the running example)**
\[
\begin{align*}
l_1 &= \emptyset \\
l_2 &= \{A\} \\
l_3 &= \{B\} \\
l_4 &= \{A, B\}
\end{align*}
\]
Propositional Logic Programs – The Semantics Ctd.

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

\[
\begin{align*}
(A)^{I_2} &= \text{true} & (\neg A)^{I_2} &= \text{false} \\
(B)^{I_2} &= \text{false} & (\neg B)^{I_2} &= \text{true}
\end{align*}
\]

\[
\begin{align*}
A &\leftarrow \neg B.
B &\leftarrow A \land \neg B.
B &\leftarrow B.
\end{align*}
\]
Example (For $I_2 = \{A\}$)

$$(A)^{I_2} = true$$

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

$$(B)^{I_2} = false$$

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

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

$$(B \leftarrow B)^{I_2} = true$$

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

$$(B \leftarrow A \land \neg B)^{I_2} = false$$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$
**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)**

\[\begin{align*}
A & \leftarrow \neg B. \\
B & \leftarrow A \land \neg B. \\
B & \leftarrow B.
\end{align*}\]

\[\begin{align*}
\emptyset & \not\models P \\
\{A\} & \not\models P \\
\{B\} & \models P \\
\{A, B\} & \models P
\end{align*}\]
The Immediate Consequence Operator $T_P$

**Definition ($T_P$)**

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

- The $T_P$-operator propagates truth along the clauses.
The Immediate Consequence Operator $T_P$

**Definition ($T_P$)**

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

- The $T_P$-operator propagates truth along the clauses.

**Example ($T_P$ for our running example)**

\[
\begin{align*}
A & \leftarrow \neg B. \\
B & \leftarrow A \land \neg B. \\
B & \leftarrow B.
\end{align*}
\]

\[
\begin{align*}
\emptyset & \mapsto \{A\} \\
\{A\} & \mapsto \{A, B\} \\
\{B\} & \mapsto \{B\} \\
\{A, B\} & \mapsto \{B\}
\end{align*}
\]

- For definite programs, $T_P$ converges to the least model.
Outline: The Core-Method for Propositional Logic

Propositional Logic Programs

The Core Method for Propositional Logic
- Embedding a Propositional Program into a Network
- Reasoning using the Core-Network
- Extracting Propositional Programs from Core-Networks

CILLP and some Derivatives

Conclusions
Constructing the Core-Network

1. For each element of $B_L$, 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

\begin{align*}
A & \leftarrow \neg B. \\
B & \leftarrow A \land \neg B. \\
B & \leftarrow B.
\end{align*}
Constructing the Core-Network

1. For each element of $B_L$, 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

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$
Constructing the Core-Network

1. For each element of $B_L$, 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

\[
\begin{align*}
A & \leftarrow \neg B. \\
B & \leftarrow A \land \neg B. \\
B & \leftarrow B.
\end{align*}
\]
Constructing the Core-Network

1. For each element of $B_L$, 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

$A \leftarrow \neg B$.  
$B \leftarrow A \land \neg B$.  
$B \leftarrow B$.  

![Diagram of the Core-Network](image_url)
Constructing the Core-Network

1. For each element of $B_L$, 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

$A \leftarrow \neg B.$
$B \leftarrow A \land \neg B.$
$B \leftarrow B.$

![Network Diagram]

Neural-Symbolic Integration (Sebastian Bader, Pascal Hitzler)
Constructing the Core-Network

1. For each element of $B_L$, 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

\[
\begin{align*}
A & \leftarrow \neg B. \\
B & \leftarrow A \land \neg B. \\
B & \leftarrow B.
\end{align*}
\]
Constructing the Core-Network

1. For each element of $B_L$, 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

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$
Constructing the Core-Network

1. For each element of $B_L$, 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

\[
\begin{align*}
A & \leftarrow \neg B. \\
B & \leftarrow A \land \neg B. \\
B & \leftarrow B.
\end{align*}
\]
One Application of $T_P$

\[ A \leftarrow \neg B. \]
\[ B \leftarrow A \land \neg B. \]
\[ B \leftarrow B. \]
One Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$

\[
\{\} \rightarrow \{A\}
\]

\[
\begin{array}{c}
A \\
\otimes \\
\end{array}
\]

\[
\begin{array}{c}
B \\
\otimes \\
\end{array}
\]

\[
\begin{array}{c}
A' \\
0.5 \\
\end{array}
\]

\[
\begin{array}{c}
B' \\
0.5 \\
\end{array}
\]

\[
\begin{array}{c}
0.5 \\
\text{A'} \\
\end{array}
\]

\[
\begin{array}{c}
0.5 \\
\text{B'} \\
\end{array}
\]
One Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$

\[
\{\} \rightarrow \{A\}
\]
One Application of $T_P$

$$A \leftarrow \neg B.$$  
$$B \leftarrow A \land \neg B.$$  
$$B \leftarrow B.$$  

$$\{\} \mapsto \{A\}$$
One Application of $T_P$

$$A \leftarrow \neg B.$$  
$$B \leftarrow A \land \neg B.$$  
$$B \leftarrow B.$$  

\[
\{\} \mapsto \{A\} \\
\{A\} \mapsto \{A, B\} \\
\} \mapsto \{\} \\
\} \mapsto \{A, B\}
\]
One Application of $T_P$

\[ A \leftarrow \neg B. \]
\[ B \leftarrow A \land \neg B. \]
\[ B \leftarrow B. \]

\[ \{\} \rightarrow \{A\} \]
\[ \{A\} \rightarrow \{A, B\} \]
One Application of $T_P$

\[
A \leftarrow \neg B.
\]

\[
B \leftarrow A \land \neg B.
\]

\[
B \leftarrow B.
\]

\[
\{\} \mapsto \{A\}
\]

\[
\{A\} \mapsto \{A, B\}
\]
One Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$

$\emptyset \mapsto \{A\}$

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

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

Neural-Symbolic Integration (Sebastian Bader, Pascal Hitzler)
One Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$

\[
\begin{align*}
\{\} & \mapsto \{A\} \\
\{A\} & \mapsto \{A, B\} \\
\{B\} & \mapsto \{B\}
\end{align*}
\]
One Application of $T_P$

$$A \leftarrow \neg B.$$  
$$B \leftarrow A \land \neg B.$$  
$$B \leftarrow B.$$  

$$\{\} \mapsto \{A\}$$  
$$\{A\} \mapsto \{A, B\}$$  
$$\{B\} \mapsto \{B\}$$
One Application of $T_P$

\[ A \leftarrow \neg B. \]
\[ B \leftarrow A \land \neg B. \]
\[ B \leftarrow B. \]

\[
\begin{align*}
\{\} & \mapsto \{A\} \\
\{A\} & \mapsto \{A, B\} \\
\{B\} & \mapsto \{B\} \\
\{A, B\} & \mapsto \{B\}
\end{align*}
\]
One Application of $T_P$

\[ A \leftarrow \neg B. \]
\[ B \leftarrow A \land \neg B. \]
\[ B \leftarrow B. \]

\[
\begin{align*}
\{\} & \mapsto \{A\} \\
\{A\} & \mapsto \{A, B\} \\
\{B\} & \mapsto \{B\} \\
\{A, B\} & \mapsto \{B\}
\end{align*}
\]
One Application of $T_P$

\[
\begin{align*}
A & \leftarrow \neg B. \\
B & \leftarrow A \land \neg B. \\
B & \leftarrow B.
\end{align*}
\]

\[
\begin{align*}
\{\} & \mapsto \{A\} \\
\{A\} & \mapsto \{A, B\} \\
\{B\} & \mapsto \{B\} \\
\{A, B\} & \mapsto \{B\}
\end{align*}
\]
Repetitive Application of $T_P$

\[ A \leftarrow \neg B. \]
\[ B \leftarrow A \land \neg B. \]
\[ B \leftarrow B. \]
Repetitive Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$
Repetitive Application of $T_P$

\[ A \leftarrow \neg B. \]
\[ B \leftarrow A \land \neg B. \]
\[ B \leftarrow B. \]
Repetitive Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$
Repetitive Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$
Repetitive Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$
Repetitive Application of $T_P$

$A \leftarrow \neg B.$

$B \leftarrow A \land \neg B.$

$B \leftarrow B.$
Repetitive Application of $T_P$

$A \leftarrow \neg B$.  
$B \leftarrow A \land \neg B$.  
$B \leftarrow B$.  

![Diagram showing the repetitive application of $T_P$]
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 always 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 always a single minimal program (Lehmann, Bader & Hitzler, 2005).
Extraction – A Pedagogical Approach

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$A'$</th>
<th>$B'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.0 / 0.0</td>
<td>0.0 / 1.0</td>
<td>0.0 / 0.0</td>
<td>0.0 / 1</td>
<td>-1.0 / 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1.5 / 1.0</td>
<td>0.3 / 1.0</td>
<td>0.8 / 1.0</td>
<td>1.8 / 1</td>
<td>0.7 / 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1.0 / 1.0</td>
<td>-2.0 / 0.0</td>
<td>-0.5 / 0.0</td>
<td>2.0 / 1</td>
<td>0.7 / 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2.5 / 1.0</td>
<td>-1.7 / 0.0</td>
<td>0.3 / 0.0</td>
<td>2.0 / 1</td>
<td>0.7 / 1</td>
</tr>
</tbody>
</table>
Extraction – A Pedagogical Approach

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A'</th>
<th>B'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
A \leftarrow \neg A \land \neg B. \\
A \leftarrow \neg A \land B. \\
A \leftarrow A \land \neg B. \\
A \leftarrow A \land B. \\
B \leftarrow \neg A \land B. \\
B \leftarrow A \land \neg B. \\
B \leftarrow A \land B.
\]
Extraction – A Pedagogical Approach

\[ A \leftarrow \lnot A \land \lnot B. \]
\[ A \leftarrow \lnot A \land B. \]
\[ A \leftarrow A \land \lnot B. \]
\[ A \leftarrow A \land B. \]
\[ B \leftarrow \lnot A \land B. \]
\[ B \leftarrow \lnot A \land B. \]
\[ B \leftarrow A \land \lnot B. \]
\[ B \leftarrow A \land B. \]

\[ A. \]
\[ B \leftarrow \lnot A \land B. \]
\[ B \leftarrow A. \]
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”:

- “true”
- “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 ∈ \{round, square, octagon\}, ...
- Classification task: “Recognize 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.
Major Problems in Neural-Symbolic Integration

- How can symbolic knowledge be represented within connectionist systems? (What is $\iota$?)
- How can symbolic knowledge be extracted from connectionist systems? (What is $\iota^{-1}$?)
- 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