Neural-Symbolic Integration Strategies

Neural-Symbolic Integration

Unification Strategies

Hybrid Systems

Neuronal Modeling

Connectionist Logic Systems

Hybrid by Translation

Hybrid by Function

Neural-Symbolic Learning Systems
CILP: Connectionist Inductive Logic Programming System

Objective: To benefit from the integration of Artificial Neural Networks and Symbolic Rules.

C ← F, ~G; F ← A ← B,C,~D; A ← E,F; B ←

Exploiting Background Knowledge
Explanation Capability

Efficient Learning
Massively Parallel Computation
1. Adding Background Knowledge (BK)
2. Computing BK in Parallel
3. Adding Training with Examples
4. Extracting Knowledge
5. Closing the Cycle
Theory Refinement

Contents

• Inserting Background Knowledge
• Performing Inductive Learning with Examples
• Adding Classical Negation
• Adding Metalevel Priorities
• Experimental Results
Inserting Background Knowledge

Example: \( P = \{ A \leftarrow B, C, \neg D; \ A \leftarrow E, F; \ B \leftarrow \} \)

**General clause:** \( A_0 \leftarrow A_1 \ldots A_m, \neg A_{m+1} \ldots \neg A_n \)
Theorem: For each general logic program $P$, there exists a feedforward neural network $N$ with exactly one hidden layer and semi-linear neurons such that $N$ computes $T_P$.

Corollary (analogous to [Holldobler and Kalinke 94]): Let $P$ be an acceptable general program. There exists a recurrent neural network $N_r$ with semi-linear neurons such that, starting from an arbitrary initial input, $N_r$ converges to the unique stable model of $P$. 
Computing with Background Knowledge

Example: \( P = \{ \ A \leftarrow \ B, C, \neg D; \ A \leftarrow E, F; \ B \leftarrow \} \).

Recurrently connected, \( N \) converges to stable state:
\{A = \text{false}, B = \text{true}, C = \text{false}, D = \text{false}, E = \text{false}, F = \text{false}\}

\[
\begin{align*}
W_r &= 1 \\
-(1+A_{\min})W/2 \\
(1+A_{\min})W \\
(1+A_{\min})W/2 \\
0
\end{align*}
\]
CILP translation algorithm

Produces neural network N given logic program P
N can be trained with backpropagation subsequently

Given P with clauses of the form: A if L1, ..., Lk
Let p be the number of positive literals in L1, ..., Lk
Let m be the number of clauses in P with A in the head
Amin denotes the minimum activation for a neuron to be true
Amax denotes the maximum activation for a neuron to be false

\[ \Theta_h = \frac{(1+A_{\text{min}})(k-1)W}{2} \]  (threshold of hidden neuron)

\[ \Theta_A = \frac{(1+A_{\text{min}})(1-m)W}{2} \]  (threshold of output neuron)

\[ W > 2 \frac{\ln(1+A_{\text{min}}) - \ln(1-A_{\text{min}})}{\max(k,m)(A_{\text{min}}-1) + A_{\text{min}} + 1} \]

\[ A_{\text{min}} > \frac{\max(k,m)-1}{\max(k,m)+1} \]

\[ A_{\text{max}} = -A_{\text{min}} \text{ (for simplicity)} \]
Performing Inductive Learning with Background Knowledge

• Neural Networks may be trained with examples to approximate the operator $T_P$ associated with a Logic Program $P$.

• A differentiable activation function, e.g. the bipolar semi-linear function $h(x) = (2 / (1 + e^{-x})) - 1$, allows efficient learning with Backpropagation.
Performing Inductive Learning with Background Knowledge

• We add extra input, output and hidden neurons, depending on the application
• We fully-connect the network
• We use Backpropagation
Adding classical negation

General Program:
Cross ← ~ Train

School bus crosses rail line in the absence of proof of approaching train

Extended Program:
Cross ← ¬ Train

School bus crosses rail line if there is proof of no approaching train

Extended clause: \( L_0 \leftarrow L_1 \ldots L_m, \sim L_{m+1} \ldots \sim L_n \)
The Extended CILP System

\[ r_1: A \leftarrow B, \neg C; \]

\[ r_2: \neg C \leftarrow B, \sim \neg E; \]

\[ r_3: B \leftarrow \sim D \]
Adding Classical Negation

**Theorem**: For each extended logic program $P$, there exists a feedforward neural network $N$ with exactly one hidden layer and semi-linear neurons such that $N$ computes $T_P$.

**Corollary**: Let $P$ be a consistent acceptable extended program. There exists a recurrent neural network $N_r$ with semi-linear neurons such that, starting from an arbitrary initial input, $N_r$ converges to the unique answer set of $P$. 

Computing with Classical Negation

Example: $P = \{ B \leftarrow \sim C; \\
A \leftarrow B, \sim \neg D; \\
\neg B \leftarrow A \}$. 

Recurrently connected, $N$ converges to stable state: 
{$A = \text{true}, B = \text{true}, \neg B = \text{true}, \\
C = \text{false}, \neg D = \text{false}$}
Adding Metalevel Priorities

\[ r_1 > r_2 = \text{“} x \text{ is preferred over } \neg x \text{”}, \text{i.e. when } r_1 \text{ fires it should block the output of } r_2 \]
Learning Metalevel Priorities

\[ P = \{ r_1: \text{guilty} \leftarrow \text{fingertips}, \]
\[ r_2: \neg \text{guilty} \leftarrow \text{alibi}, \]
\[ r_3: \text{guilty} \leftarrow \text{super-grass} \} \]

\[ r_1 > r_2 > r_3 \]

Training examples include:

\[ [(-1, 1, *), (-1, 1)] \]
\[ [(1, *, *), (1, -1)] \]

where * means “don’t care”
Learning Metalevel Priorities

\[ W_{\text{guilty}, r1} = 3.97, \]
\[ W_{\text{guilty}, r2} = -1.93, \quad W_{\neg \text{guilty}, r1} = -1.93 \]
\[ W_{\text{guilty}, r3} = 1.94, \quad W_{\neg \text{guilty}, r2} = 1.94 \]
\[ W_{\neg \text{guilty}, r3} = 0.00 \]

\[ \theta_{\text{guilty}} = -1.93, \quad \theta_{\neg \text{guilty}} = 1.93 \]

r1 > r2 > r3 ?
Setting Linearly Ordered Theories

\[ W_{\text{guilty},r_3} = W \]
\[ W_{\text{guilty},r_2} = -W + \delta \]
\[ W_{\text{guilty},r_1} = W_{\text{guilty},r_3} - W_{\text{guilty},r_2} + \delta = 2W \]

If \( W = 2 \), \( \delta = 0.01 \):
\[ W_{\text{guilty},r_3} = 2 \]
\[ W_{\text{guilty},r_2} = -1.99 \]
\[ W_{\text{guilty},r_1} = 4 \]

\[-3 < \theta_{\text{guilty}} < 1\]
\[-1 < \theta_{\neg \text{guilty}} < 3\]

\[ r_1 > r_2 > r_3 \]
Partially Ordered Theories

Each of \( r_2 \), \( r_3 \) and \( r_4 \) should block the conclusion of \( r_1 \)
Problematic Case

layEggs(platypus) monotreme(platypus)
hasFur(platypus) hasBill(platypus)

r1: mammal(x) ← monotreme(x)

r2: mammal(x) ← hasFur(x)

r3: ¬ mammal(x) ← layEggs(x)  \( r1 > r3 \)

r4: ¬ mammal(x) ← hasBill(x)  \( r2 > r4 \)

Cannot have \( r1 > r3 \), \( r2 > r4 \) without also having \( r1 > r4 \) and \( r2 > r3 \)
CILP Experimental Results

• Test Set Performance (how well it generalises)

• Test Set Performance over small/increasing training sets (how important BK is)

• Training Set Performance (how fast it trains)
CILP Experimental Results

- Promoter Recognition:
  - A short DNA sequence that preceeds the beginning of genes.

Background Knowledge

<table>
<thead>
<tr>
<th>Promoter ← Contact, Conformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact ← Minus10, Minus35</td>
</tr>
<tr>
<td>Minus 10 ← @ -14 ‘tataat’</td>
</tr>
<tr>
<td>Minus 10 ← @ -13 ‘tataat’</td>
</tr>
<tr>
<td>Minus 35 ← @ -37 ‘cttgac’</td>
</tr>
<tr>
<td>Minus 35 ← @ -36 ‘ttgaca’</td>
</tr>
<tr>
<td>Conformation ← @ -45 ‘aa’, @ -41 ‘a’</td>
</tr>
<tr>
<td>Conformation ← @ -45 ‘a’, @ -41 ‘a’, @ -28 ‘tt’, @ -23 ‘t’, @ -21 ‘aa’, @ -17 ‘t’, @ -15 ‘t’, @ -4 ‘t’</td>
</tr>
<tr>
<td>Conformation ← @ -49 ‘a’, @ -44 ‘t’, @ -27 ‘t’, @ -22 ‘a’, @ -18 ‘t’, @ -16 ‘tg’, @ -1 ‘a’</td>
</tr>
<tr>
<td>Conformation ← @ -47 ‘caa’, @ -43 ‘tt’, @ -40 ‘ac’, @ -22 ‘g’, @ -18 ‘t’, @ -16 ‘c’, @ -8 ‘gcgcc’, @ -2 ‘cc’</td>
</tr>
</tbody>
</table>
An Example Bioinformatics Rule

Minus5 ← @-1'gc', @5't'

Diagram:

```
  Minus5
     /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   \   /   ```
Promoter Recognition

53 examples of promoters
53 examples of non-promoters

Initial Topology of the Network

- Minus35
- Minus10
- Conform.
- Contact
- Promoter

-50 DNA +7 Minus35 Minus10 Conform. Contact
Test Set Performance (promoter recognition)

- Comparison with systems that learn from examples only (i.e. no BK)
Test Set Performance (promoter recognition)

- Comparison with systems that learn from examples and background knowledge

<table>
<thead>
<tr>
<th>System</th>
<th>Test Set Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBCNN</td>
<td>98.1</td>
</tr>
<tr>
<td>C-IL2P</td>
<td>97.2</td>
</tr>
<tr>
<td>KBANN</td>
<td>92.5</td>
</tr>
<tr>
<td>Labyrinth</td>
<td>86.8</td>
</tr>
<tr>
<td>FOCL</td>
<td>85.8</td>
</tr>
<tr>
<td>Either</td>
<td>79.0</td>
</tr>
</tbody>
</table>
Test set performance on small/increasing training sets

- Promoter recognition: comparison with Backprop and KBANN
Training Set Performance (promoter recognition)

• Comparison with Backprop and KBANN
CILP Experimental Results

- Splice Junction Determination

% Points on a DNA sequence at which the cell removes superfluous DNA during the process of protein creation.

![DNA and mRNA Diagram]

Background Knowledge

<table>
<thead>
<tr>
<th>EI</th>
<th>@ -3 ‘aagtaagt’, ~EI-Stop</th>
<th>EI</th>
<th>@ -3 ‘caggtaagt’, ~EI-Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>EI-Stop</td>
<td>@ -3 ‘taa’</td>
<td>EI-Stop</td>
<td>@ -4 ‘taa’</td>
</tr>
<tr>
<td>EI-Stop</td>
<td>@ -3 ‘tag’</td>
<td>EI-Stop</td>
<td>@ -4 ‘tag’</td>
</tr>
<tr>
<td>EI-Stop</td>
<td>@ -3 ‘tga’</td>
<td>EI-Stop</td>
<td>@ -4 ‘tga’</td>
</tr>
<tr>
<td>IE</td>
<td>@ -3 ‘tagg’, Piramidal, ~IE-Stop</td>
<td>IE</td>
<td>@ -3 ‘cagg’, Piramidal, ~IE-Stop</td>
</tr>
<tr>
<td>Piramidal</td>
<td>@ -15 ‘tttttttttt’</td>
<td>Piramidal</td>
<td>@ -15 ‘cccececece’</td>
</tr>
<tr>
<td>IE-Stop</td>
<td>@ 1 ‘taa’</td>
<td>IE-Stop</td>
<td>@ 2 ‘taa’</td>
</tr>
<tr>
<td>IE-Stop</td>
<td>@ 1 ‘tag’</td>
<td>IE-Stop</td>
<td>@ 2 ‘tag’</td>
</tr>
<tr>
<td>IE-Stop</td>
<td>@ 1 ‘tga’</td>
<td>IE-Stop</td>
<td>@ 2 ‘tga’</td>
</tr>
</tbody>
</table>
Splice-Junction Determination

3190 examples:
- 25% examples of E/I boundaries
- 25% examples of I/E boundaries
- 50% of non-examples

Initial Topology of the Network
Test Set Performance
(Splice-Junction Determination)

- Comparison with systems that learn from examples only (i.e. no BK)
Test Set Performance (Splice-Junction Determination)

- Comparison with systems that learn from examples and background knowledge
Test set performance on small/increasing training sets

- Splice-junction determination: comparison with Backprop and KBANN
Training Set Performance
(Splice-junction determination)

• Comparison with Backprop and KBANN

![Graph showing the comparison of different models. The x-axis represents Training Epochs ranging from 0 to 30, and the y-axis represents RMS Error Rate ranging from 0 to 0.6. The graph includes lines for Backprop, KBANN, and C-IL2P models.]
Experimental Results summary

- CILP's test set performance is comparable with Backpropagation and KBANN

- CILP's test set performance in the presence of few training examples is better than Backprop and comparable with KBANN

- CILP's training set performance is superior than Backprop and KBANN
Sources of CILP strength

• CILP uses Backpropagation

• CILP uses Background Knowledge

• CILP's translation of BK into N is compact and correct:
  – Single-hidden layer network
  – Provably sound translation algorithm
The combination of theory and data learning provides more effective machine learning systems.

Single hidden layer neural networks can be used to represent and learn extended logic programs.

Preference relations can be encoded into neural networks in order to adjudicate conflicts between rules.