How to make logics neurosymbolic

Luc De Raedt
Giuseppe Marra & Vincent Derkinderen & Sebastijan Dumancic & Robin Manhaeve & Thomas Winters & Angelika Kimmig

© Luc De Raedt and colleagues

Learning and Reasoning both needed

- System 1 - thinking fast - can do things like 2+2 = ? and recognise objects in image
- System 2 - thinking slow - can reason about solving complex problems - planning a complex task
- alternative terms — data-driven vs knowledge-driven, symbolic vs subsymbolic, solvers and learners, neuro-symbolic...

- A lot of work on integrating learning and reasoning, neural symbolic computation to integrate logic / symbols reasoning with neural networks

  see also arguments by Marcus, Darwiche, Levesque, Tenenbaum, Geffner, Bengio, Le Cun, Kautz, ...

Real-life problems involve two important aspects.

Thinking fast
MAIN PARADIGM in AI
Focus on Learning

Who can go first?
A. The red car
B. The blue van
C. The white car

https://www.theorie-blokken.be/nl/gratis-proefexamen

NEURAL
Thinking slow = reasoning

TWO MAIN PARADIGMS in AI

PROBABILITY

LOGIC

Their integration has been well studied in Probabilistic (Logic) Programming and Statistical Relational AI (StarAI).

Neurosymbolic = Neuro + Logic

LOGIC

NEURAL

WARNING!
TALK MAY NOT COVER ALL of NESY

Neurosymbolic = Neuro + Logic + Probability

PROBABILITY

LOGIC

NEURAL

see Manhaeve et al. NeSy Book

interpret PROBABILITY broadly (including fuzzy)

Key Message 1

The NeuroSymbolic alphabet-soup

check our survey on arxiv — Marra, Dumancic, Manhaeve & De Raedt, 23
StarAI and NeSy share similar problems and thus similar solutions apply

See also [De Raedt et al., IJCAI 20; Marra et al, arxiv]

Key Message 2

Provide recipe for

Neural : Symbolic

“an interface layer (<> pipeline) between neural & symbolic components”

Key Message 3

Part 1: NeSy AI - a little Survey

Part 2: The Recipe

Part 3: DeepStochLog and DeepProbLog

check our survey on arxiv — Marra, Dumancic, Manhaeve & De Raedt, 23
Logic Programs
as in the programming language Prolog

Propositional logic program

0.1 \cdot \text{burglary}.
0.05 \cdot \text{earthquake}.

\text{alarm} \leftarrow \text{earthquake}.
\text{alarm} \leftarrow \text{burglary}.

\text{calls} \leftarrow \text{alarm}.
0.7:: \text{calls} \leftarrow \text{alarm}.
0.6:: \text{calls} \leftarrow \text{alarm}.

Logic as constraints
as in SAT solvers

Propositional logic

\text{IFF}
\text{calls} \leftarrow \text{hears} \land \text{hears} \land \text{alarm}.
\text{AND}
\text{calls} \leftarrow \text{hears} \land \text{hears} \land \text{alarm}.

\text{OR}
\text{alarm} \leftarrow \text{earthquake} \lor \text{burglary}.

A proof-theoretic view

Two types of Neural Symbolic Systems

Logic as a kind of neural program

-directed StarAI approach and logic programs

undirected StarAI approach and (soft) constraints

Also, many NeSy systems are doing knowledge based model construction KBMC where logic is used as a template

Just like in StarAI
Logic as a neural program

directed StarAI approach and logic programs

- KBANN (Towell and Shavlik AJJ 94)
- Turn a (propositional) Prolog program into a neural network and learn

\[ A \rightarrow B, Z \]
\[ B \rightarrow C, D \]
\[ B \rightarrow E, F, G \]
\[ Z \rightarrow Y, \neg X \]
\[ Y \rightarrow S, T \]

REWRITE

Directed StarAI approach and logic programs

Logic as a neural program

Logic as a neural program

Directed StarAI approach and logic programs

ADD LINKS — ALSO SPURIOUS ONES

and then learn

(Details of activation & loss functions not mentioned)

Lifted Relational Neural Networks

directed StarAI approach and logic programs

- Directed (fuzzy) NeSy
- similar in spirit to the Bayesian Logic Programs and Probabilistic Relational Models
- Of course, other kind of (fuzzy) operations for AND, OR and Aggregation (cf. later)

Neural Theorem Prover

directed StarAI approach and logic programs

the logic is encoded in the network

how to reason logically?

[Sourek, Kuzelka, et al JAIR]

[Rocktäschel Riedel, NeurIPS 17; Minervini et al.]
Two types of Neural Symbolic Systems

Logic as a kind of neural program  
- directed StarAI approach and logic programs

Logic as the regularizer (reminiscent of Markov Logic Networks)
- undirected StarAI approach and (soft) constraints

Also, many NeSy systems are doing knowledge based model construction KBMC where logic is used as a template

Logic as constraints

undirected StarAI approach and (soft) constraints

multi-class classification

This constraint should be satisfied

\[(\neg x_1 \land \neg x_2 \land x_3) \lor (\neg x_1 \land x_2 \land \neg x_3) \lor (x_1 \land \neg x_2 \land \neg x_3)\]

figures and example from Xu et al., ICML 2018

Semantic Loss:

- Use logic as constraints (very much like "propositional MLNs")
- Semantic loss

\[SLoss(T) \propto - \log \sum_{X=T} \prod_{x \in X} p_i \prod_{\neg x \in X} (1 - p_i)\]
- Used as regulariser

\[Loss = \text{TraditionalLoss} + w \cdot SLoss\]

- Use weighted model counting, close to StarAI

Probability that constraint is satisfied

\[(1 - x_1)(1 - x_2)x_3 + (1 - x_1)x_2(1 - x_3) + x_1(1 - x_2)(1 - x_3)\]
**Logic Tensor Networks**

undirected StarAI approach and (soft) constraints

\[ P(x, y) \rightarrow A(y), \text{ with } G(x) = v \text{ and } G(y) = u \]

Serafi & Garcez

---

**Semantic Based Regularization**

undirected StarAI approach and (soft) constraints

\[
F = \forall x \exists y (P(x \rightarrow \neg A(y)) \\
F' = \forall x \exists y (R(x, y) \rightarrow (A(x) \land A(y)) \lor (\neg A(x) \land \neg A(y))) \\
C = (d_1, d_2) \\
P_x(d_i) = 1 \\
R(d_1, d_2) = 1
\]

Output Layer

Quantifier Layer

Proportional Layer

the logic is encoded in the network

how to reason logically?

Diligenti et al. AIJ

---

**Two types of Neural Symbolic Systems**

Logic as a kind of **neural program**

directed StarAI approach and logic programs

Logic as the **regularizer** (reminiscent of Markov Logic Networks)

undirected StarAI approach and (soft) constraints

Consequence:

the logic is encoded in the network

the ability to logically reason is lost

logic is not a special case

Part 2: The Recipe
A recipe for NeSy

**STEP 1**

1. Take your favorite symbolic (logic / rule based) representation

**NeSy Data Point**
- `calls(image32, signal42, mary)`
- `image32 = ✢, signal42 = ☐`

**NeSy Model**
- Logic Rules:
  - `alarm(B,E) IF burglary(B) OR earthquake(E).`
  - `calls(B,E,X) IF alarm(B,E) AND hears_alarm(X).`
- Logic Facts:
  - `hears_alarm(mary).`
  - `hears_alarm(john).`

**Neural Net Modules**
- `neural-net(image_perception(B))`
- `neural-net(signal_analysis(E))`

(legal on DeepProbLog) © Luc De Raedt

**STEP 2**

2. Interpret neural networks as neural predicates

**NeSy Data Point**
- `calls(image32, signal42, mary)`
- `image32 = ✢, signal42 = ☐`

**NeSy Model**
- Logic Rules:
  - `alarm(B,E) IF burglary(B) OR earthquake(E).`
  - `calls(B,E,X) IF alarm(B,E) AND hears_alarm(X).`
- Logic Facts:
  - `hears_alarm(mary).`
  - `hears_alarm(john).`

**Neural Predicates**
- `burglary(B) IF neural-net(image_perception(E))`
- `earthquake(E) IF neural-net(signal_analysis(E))`

**Neural Net Modules**
- `image_perception = ✢, signal_analysis = ☐`

(legal on DeepProbLog) © Luc De Raedt

3. Turn the 0/1 or True/False into Probabilistic or Fuzzy Interpretation

**STEP 3**

4. Construct logical proof / explanation for example

**NeSy Network**
- `P(calls(image32, signal42, mary)) = 0.246`
- `AND` node
- `OR` node

(legal on DeepProbLog) © Luc De Raedt
A recipe for NeSy

4. Construct logical proof / explanation for example

5. Add the neural networks to the corresponding predicates (*reparametrise*)

6. Replace OR and AND by $\oplus$ and $\otimes$

7. Differentiate

---

A recipe for NeSy

Where do the numbers come from?

From logic formulae to circuits

\[ \ell'(A \land B) \rightarrow C \quad \ell(Q) \]

\[ \ell_F(A \land B) \rightarrow C \quad \ell(Q) \]

What is the algebraic structure? = Parametric circuit

What operators?

What labeling functions?

The query Q determines the structure (potentially after knowledge compilation)
A recipe for NeSy

Where do the numbers come from?

**Boolean**

\[ \ell_f((A \land B) \rightarrow C) \]

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p\land q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

The query Q determines the structure (potentially after knowledge compilation)

**What operators?**

- \( \ell_f \)
- \( \ell(A) \)
- \( \ell(B) \)
- \( \ell(C) \)

**What labeling functions?**

- \( \ell(f) \)
- \( \ell(A) \)
- \( \ell(B) \)
- \( \ell(C) \)

**Fuzzy**

- t-norm extends conjunction to \([0,1]\) interval
- Three fundamental t-norms:
  - \( t_L(x, y) = \max(0, x + y - 1) \)
  - \( t_G(x, y) = \min(x, y) \)
  - \( t_P(x, y) = x \cdot y \)

Other operators derived from the t-norm

<table>
<thead>
<tr>
<th>Product</th>
<th>Łukasiewicz</th>
<th>Gödel</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \cdot y )</td>
<td>( x \cdot y )</td>
<td>( \min(x, y) )</td>
</tr>
<tr>
<td>( x + y )</td>
<td>( x + y )</td>
<td>( \max(x, y) )</td>
</tr>
<tr>
<td>( x \land y )</td>
<td>( 1 - x )</td>
<td>( 1 - x )</td>
</tr>
<tr>
<td>( x \lor y )</td>
<td>( y \land x )</td>
<td>( \min(1, 1 - x + y) )</td>
</tr>
</tbody>
</table>

**Probability**

- \( P(A \lor B) = P(A) + P(B) - P(A \land B) \)

Knowledge Compilation (computationally expensive)

Probabilistic structure is explicit in compiled formula.

**Why Compile**

- but a measure of vagueness not of uncertainty

Many problems
See [Van Krieken et al AIJ]
**Logic as soft constraints**

**Markov Logic**

**Propositional logic**

Model / Possible World

10: calls(mary) <- hears_alarm(mary) ∧ alarm

20: calls(john) <- hears_alarm(john) ∧ alarm

30: alarm <- earthquake v burglary

**Propositional logic**

Model / Possible World

{burglary, hears_alarm(john), alarm, calls(john)}

The higher the weight, the harder or more logical the constraint

<table>
<thead>
<tr>
<th>Weight</th>
<th>Formula</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>calls(mary) &lt;- hears_alarm(mary) ∧ alarm</td>
<td>$e^{10}$</td>
</tr>
<tr>
<td>20</td>
<td>calls(john) &lt;- hears_alarm(john) ∧ alarm</td>
<td>$e^{20}$</td>
</tr>
<tr>
<td>30</td>
<td>alarm &lt;- earthquake v burglary</td>
<td>$e^{30}$</td>
</tr>
</tbody>
</table>

Using weighted model counting (WMC)

Weights/probabilities are on the formulae (softer constraints)

The higher the weight, the harder or more logical the constraint

Probability of world = $e^{10} \times e^{20} \times e^{30}$

**Logic as soft constraints**

**Probabilistic Soft Logic** [Bach & Getoor]

**Propositional logic**

Model / Possible World

10: calls(mary) <- hears_alarm(mary) ∧ alarm

20: calls(john) <- hears_alarm(john) ∧ alarm

30: alarm <- earthquake v burglary

Atoms are no longer true or false in worlds.

But true or false to a certain degree

Lukasiewicz T-norm:

For 0 and 1 we get boolean logic

Residuum evaluates to 1 when rule is satisfied

A $\lor$ B = min(1, A + B)

A $\land$ B = min(1, A + B - 1)

A $\rightarrow$ B = max(0, 1 - A) (residuum) evaluates to 1 when rule is satisfied

when B $\leq$ A

**Probability**

Rule evaluates to when calls(john) = 0.3 ≥ 0.5

A $\land$ B = min(1, A + B - 1)

A $\rightarrow$ B = max(0, 1 - A) (residuum)

When calls(john) = 0.3

Fuzzy Logic

w = $e^{[−20 x (1−0.8)]}$

See Van Krieken et al AIJ 22

**From StarAI to NeSy**

**Part 3: DeepStochLog and DeepProbLog**
Two types of probabilistic models / programs

• Based on a random graph model
  • Bayesian Nets and ProbLog -> DeepProbLog [AIJ 21]

• Based on a random walk model
  • Probabilistic grammars and Stochastic Logic Programs [Muggleton] -> DeepStochLog [AAAI 22]

Our method/recipe:
Take an existing probabilistic logic and inject neural predicates that act as an interface

DeepProbLog

DeepProbLog = Probability + Logic + Neural Network

DeepProbLog = ProbLog + Neural Network

Related work in NeSy

| Logic is made less expressive | Full expressivity is retained |
| Logic is pushed into the neural network | Maintain both logic and neural network |
| Fuzzy logic | Probabilistic logic programming |
| Language semantics unclear | Clear semantics |

Holds also for DeepStochLog

DeepStochLog = SLPs + Neural Network

46

PART 3 A

From Prolog to ProbLog

Propositional logic program

Two proofs (by refutation)

burglary.
hears_alarm(mary).
earthquake.
hears_alarm(john).
alarm :- earthquake.
alarm :- burglary.
calls(mary) :- alarm, hears_alarm(mary).
calls(john) :- alarm, hears_alarm(john).

A proof-theoretic view
Probabilistic Logic Programs
as in the probabilistic programming language ProbLog

Propositional logic program

Key Idea (Sato & Poole)
the distribution semantics:
unify the basic concepts in logic
and probability:
random variable ~ propositional
variable
an interface between logic and
probability

0.1 :: burglary.
0.3 :: hears_alarm(mary).
0.05 :: earthquake.
0.6 :: hears_alarm(john).

alarm :- earthquake.
alarm :- burglary.
calls(mary) :- alarm, hears_alarm(mary).
calls(john) :- alarm, hears_alarm(john).

Propositional logic program

Disjoint sum problem

P(alarm) = P(burg OR earth)
= P(burg) + P(earth) - P(burg AND earth)
=/= P(burg) + P(earth)

Probability of one proof :
\( \prod_{\text{fact in Proof}} P_f \)

Probabilistic Logic Program
Semantics

[Vennekens et al, ICLP 04]

probabilistic causal laws

earthquake.
0.05 :: burglary.
0.6 :: alarm :- earthquake.
0.8 :: alarm :- burglary.

alarm

no alarm

earthquake

0.1

0.05

0.95

0.6

0.4

0.05 0.93

no burglary

alarm

0.8

0.2

no alarm

P(alarm) = 0.6 \times 0.05 \times 0.8 + 0.6 \times 0.05 \times 0.2 + 0.6 \times 0.95 + 0.4 \times 0.05 \times 0.8
Probabilistic Logic Program Semantics

Bayesian Network

0.1 :: burglary.
0.05 :: earthquake.

alarm :- earthquake.
alarm :- burglary.

0.7::calls(mary) :- alarm.
0.6::calls(john) :- alarm.

Bayesian net encoded as Probabilistic Logic Program
PLPs correspond to directed graphical models

ProbLog has both (directed) probabilistic graphic models,
the programming language Prolog (and probabilistic databases) as special case

Flexible and Compact Relational Model for Predicting Grades

“Program” Abstraction:
• S, C: logical variable representing students, courses
• the set of individuals of a type is called a population
• Int(S), Grade(S, C), D(C) are parametrized random variables

Grounding:
• for every student s, there is a random variable Int(s)
• for every course c, there is a random variable D(c)
for every s, c pair there is a random variable Grade(s,c)
all instances share the same structure and parameters

ProbLog by example: Grading

Shows relational structure

grounded model: replace variables by constants
Works for any number of students / classes (for 1000 students and 100 classes, you get 101100 random variables); still only few parameters

With SRL / PP
build and learn compact models,
from one set of individuals -> other sets;
reason also about exchangeability,
build even more complex models,
incorporate background knowledge

ProbLog applications
Dynamic networks

*Travian:* A massively multiplayer real-time strategy game
- Can we build a model of this world?
- Can we use it for playing better?

**Activity analysis and tracking**
- Track people or objects over time? Even if temporarily hidden?
- Recognize activities?
- Infer object properties?

**Learning relational affordances**
- But focus on learning probabilistic aspects, neural networks still pipelined

**Biology**
- Causes: Mutations
- All related to similar phenotype
- Effects: Differentially expressed genes
- 27,000 cause effect pairs
- Interaction network:
  - 3063 nodes
  - Genes
  - Proteins
  - 16794 edges
  - Molecular interactions
  - Uncertain
- Goal: connect causes to effects through common subnetwork
  - Find mechanism
  - Techniques:
    - DTProbLog
    - Approximate inference

**Techniques:**
- [De Maeyer et al., Molecular Biosystems 13, NAR 15] [Gross et al., Communications Biology, 19]

**Dynamic networks**

**Travian:** A massively multiplayer real-time strategy game
- Can we build a model of this world?
- Can we use it for playing better?
From ProbLog to DeepProbLog

Neural predicate

- Neural networks have uncertainty in their predictions
- A normalized output can be interpreted as a probability distribution
- Neural predicate models the output as probabilistic facts
- No changes needed in the probabilistic host language

Key Idea DeepProbLog
unify the basic concepts in logic and neural networks:
neural predicate ~ neural net
an interface between logic and neural nets

The output of the neural network is probabilistic facts in DeepProbLog

Example:

nn(mnist_net, [X], Y, [0 ... 9]) :: digit(X,Y).

Instantiated into a (neural) Annotated Disjunction:

0.04::digit(5,0) ; 0.35::digit(7,1) ; ... ;
0.53::digit(7,7) ; ... ; 0.014::digit(7,9).
DeepProbLog exemplified: MNIST addition

Task: Classify pairs of MNIST digits with their sum

| 3 | 5 | 8 |
| 0 | 4 | 4 |
| 7 | 2 | 11 |

Benefit of DeepProbLog:

- Encode addition in logic
- Separate addition from digit classification

```
nn(mnist_net, [X], Y, [0 ... 9]) :: digit(X,Y).
addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.
```

Examples:
```
addition(3,5,8), addition(0,4,4), addition(7,2,11), ...
```

MNIST Addition

- Pairs of MNIST images, labeled with sum
- Baseline: CNN
  - Classifies concatenation of both images into classes 0 ... 18
- DeepProblog:
  - CNN that classifies images into 0 ... 9
  - Two lines of DeepProblog code
- Result:
  - Fewer iterations necessary
  - Higher accuracy achieved

Example

Learn to classify the sum of pairs of MNIST digits

Individual digits are not labeled!

E.g. (3, 5, 8)

Could be done by a CNN: classify the concatenation of both images into 19 classes

However: \[3 \times 5 \neq 0 \times 4 \]
Inference & Learning

ProbLog Inference

Answering a query in a ProbLog program happens in four steps
1. Grounding the program w.r.t. the query
2. Rewrite the ground logic program into a propositional logic formula
3. Compile the formula into an arithmetic circuit
4. Evaluate the arithmetic circuit

Query

0.1 :: burglary.
0.5 :: hears_alarm(mary).
0.2 :: earthquake.
0.4 :: hears_alarm(john).
alarm :- earthquake.
alarm :- burglary.
calls(X) :- alarm, hears_alarm(X).

Multi-digit MNIST addition with MNIST

number([H | T] , Acc , Result) :-
digit(H, Nr), Acc2 is Nr + 10*Acc,
number(T, Acc2, Result).
number(X,Y) :- number(X,0,Y).

multiaddition(X,Y,Z) :-
number([H | T] , Acc , Result) :-
digit(H, Nr), Acc2 is Nr + 10*Acc,
number(T, Acc2, Result).
number(X,Y) :- number(X,0,Y).

Noisy Addition

addition(a,b,1)

fraction of noise

Baseline
94.46 87.85 82.49 52.67 8.79 5.87
DeepProbLog
97.20 95.78 94.50 92.90 46.42 0.88
DeepProbLog w/ explicit noise
96.64 95.96 95.58 94.12 73.22 2.92
Learned fraction of noise
0.00 0.21 0.41 0.61 0.80 0.98

Table 3: The accuracy on the test set for T4.
ProbLog Inference

Answering a query in a ProbLog program happens in four steps

1. **Grounding the program w.r.t. the query** (only relevant part!)
2. Rewrite the ground logic program into a propositional logic formula
3. Compile the formula into an arithmetic circuit
4. Evaluate the arithmetic circuit

 ProbLog Inference

Answering a query in a ProbLog program happens in four steps

1. **Grounding the program w.r.t. the query**
2. Rewrite the ground logic program into a propositional logic formula
3. Compile the formula into an arithmetic circuit
4. Evaluate the arithmetic circuit

### Example

0.1 :: burglary.
0.5 :: hears_alarm(mary).
0.2 :: earthquake.
0.4 :: hears_alarm(john).

alarm :- earthquake.
alarm :- burglary.
calls(mary) :- alarm, hears_alarm(mary).
calls(john) :- alarm, hears_alarm(john).

0.1 :: burglary.
0.5 :: hears_alarm(mary).
0.2 :: earthquake.
0.4 :: hears_alarm(john).

alarm :- earthquake.
alarm :- burglary.
calls(mary) :- alarm, hears_alarm(mary).
calls(john) :- alarm, hears_alarm(john).

### Useful Semirings

<table>
<thead>
<tr>
<th>Semiring</th>
<th>task</th>
<th>A</th>
<th>e⁺</th>
<th>e⁻</th>
<th>0</th>
<th>α(0)</th>
<th>α(-1)</th>
<th>ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>[true, false]</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>ref</td>
</tr>
<tr>
<td>#SAT</td>
<td>N</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>WTA</td>
<td>R₂⁰</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>PROB</td>
<td>R₂⁰</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>SENS</td>
<td>K</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>GRAD</td>
<td>R₂⁰</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>MPP</td>
<td>K</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>MPATH</td>
<td>N⁺</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>WPATH</td>
<td>N⁺</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>LWEIGHT</td>
<td>K</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>WHY</td>
<td>N⁺</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
</tbody>
</table>

Table 1: Examples of commutative semirings and labeling functions. The **WHY** and **LWEIGHT** 

### From Kimming, Vanden Broeck and De Raedt, 2016

<table>
<thead>
<tr>
<th>Semiring</th>
<th>task</th>
<th>A</th>
<th>e⁺</th>
<th>e⁻</th>
<th>0</th>
<th>α(0)</th>
<th>α(-1)</th>
<th>ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>[true, false]</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>ref</td>
</tr>
<tr>
<td>#SAT</td>
<td>N</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>WTA</td>
<td>R₂⁰</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>PROB</td>
<td>R₂⁰</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>SENS</td>
<td>K</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>GRAD</td>
<td>R₂⁰</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>MPP</td>
<td>K</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>MPATH</td>
<td>N⁺</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>WPATH</td>
<td>N⁺</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>LWEIGHT</td>
<td>K</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
<tr>
<td>WHY</td>
<td>N⁺</td>
<td>0</td>
<td>1</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ref</td>
</tr>
</tbody>
</table>

Table 1: Examples of commutative semirings and labeling functions. The **WHY** and **LWEIGHT** semirings apply to positive literals only. Referenee list: B (Racah et al., 2009), BT (Ibarra and Theodorakopoulou, 2010), C (Ehren, 2002), G (Goodman, 1999), K (Green et al., 2007), N (Kimmig et al., 2011), S (Larossa et al., 2010), M (Meseguer et al., 2006). More examples can be found in these references.
Gradient Semiring

nn(mnist_net, [X], Y, [0 ... 9]) :-
digit(X,Y).

addition(X,Y,Z) :-
digit(X,N1),
digit(Y,N2),
Z is N1+N2.

The ACs are differentiable and there is an interface with the neural nets

(Pretty elegant in ProbLog we use the “gradient” semi-ring)

Example DeepProbLog

---

Program Induction/Sketching

In Neural Symbolic methods

- Rule Induction — work with templates
  \( P(X) \rightarrow R(X,Y), Q(Y) \)
- and have the “predicate” variables / slots RQ, R determined by the NN
- Simpler form, fill just a few slots / holes

Approach similar to ‘Programming with a Differentiable Forth Interpreter’ [1]

- Partially defined Forth program with slots / holes
- Slots are filled by neural network (encoder / decoder)

- Fully differentiable interpreter: NNs are trained with input / output examples

---

DeepSeaProbLog

(discrete and continuous distributions [De Smet UAI 23])

useful for robotics and perception

dim is neural net returning parameters of normal distribution.

length(Obj) ~ normal(dim(Obj, Image)).

large(Obj) :- length(Obj) > 100.

---

determining order digits to determine year

---
DeepSeaProbLog

discrete and continuous distributions [De Smet UAI 23]
generative model with variational autoencoders (see also [Misoni et al NeurIPS 22])

So far from input 8 3 to output 11 so that \( \text{SUM}(8, 3, 11) \) holds

In DeepSeaProbLog, you can query \( \text{SUM}(\_ \_ , X, 5) \)

Soft Unification

- NTP: "grandpa" softly unifies with "grandfather"; as embeddings are close
- DeepProblog: define

  \[
  \text{softunification}(X, Y) : - \text{embed}(X, EX), \text{embed}(Y, EY), \text{rbf}(EX, EY).
  \]

  \[
  \text{softunification}(X, Y) \text{ returns 1 if } X \text{ and } Y \text{ unify}
  \]

  \[
  \text{otherwise returns } \exp(\frac{-|e_X - e_Y|^2}{2\mu^2})
  \]

  \[
  \text{grandPaOf}(X, Y) : \text{softunification}(\text{grandPaOf}(X), R(X, Y)).
  \]

Neural Theorem Prover

So far from input 8 3 to output 11 so that \( \text{SUM}(8, 3, 11) \) holds

Probabilistic Logic Shield for Reinforcement Learning

Assuming noisy sensors

\[
\begin{align*}
&\text{Shield} \\
&\text{Will } \text{car} \text{ stay undamaged?} \\
&P_{\text{safe}}(a, s) = 0.28 \\
&P_{\text{left}}(a) = 0.92 \\
&P_{\text{right}}(a) = 0.8 \\
\end{align*}
\]

Probability of staying safe if following \( \pi \)?

\[
P_{\text{safe}}(\pi) = 0.576
\]

What is a safer policy \( \pi^* \)?

\[
\begin{align*}
&\pi^*(\text{accelerate}) = 0.24 \\
&\pi^*(\text{left}) = 0.46 \\
&\pi^*(\text{right}) = 0.28
\end{align*}
\]
DeepStochLog: Neural Definite Clause Grammars

DeepStochLog

- Based on a different semantics
  - probabilistic graphical models vs grammars
  - random graphs vs random walks
- Underlying StarAI representation is Stochastic Logic Programs (Muggleton, Cussens)
  - close to Probabilistic Definite Clause Grammars, aka probabilistic unification based grammar formalism
  - again the idea of neural predicates
- Scales better, is faster than DeepProbLog

PART 3 B

DeepStochLog: Neural Definite Clause Grammars

CFG: Context-Free Grammar

Useful for:
- Is sequence an element of the specified language?
- What is the "part of speech"-tag of a terminal
- Generate all elements of language
**PCFG: Probabilistic Context-Free Grammar**

0.5 :: E -> N
0.5 :: E -> E, P, N
1.0 :: P -> [“*”]
0.1 :: N -> [“0”]
0.1 :: N -> [“1”]
0.1 :: N -> [“2”]
0.1 :: N -> [“9”]

Useful for:
- What is the most likely parse for this sequence of terminals? (useful for ambiguous grammars)
- What is the probability of generating this string?

**DCG: Definite Clause Grammar**

e(N) -> n(N),
e(N) -> e(N1), p, n(N2),
{N is N1 + N2}.

Useful for:
- Modelling more complex languages (e.g. context-sensitive)
- Adding constraints between non-terminals thanks to Prolog power (e.g. through unification)
- Extra inputs & outputs aside from terminal sequence (through unification of input variables)

**SDCG: Stochastic Definite Clause Grammar**

0.5 :: e(N) -> n(N),
0.5 :: e(N) -> e(N1), p, n(N2),
{N is N1 + N2}.
1.0 :: p -> [“*”].
0.1 :: n(0) -> [“0”].
0.1 :: n(1) -> [“1”].
0.1 :: n(9) -> [“9”].

Useful for:
- Same benefits as PCFGs give to CFG (e.g. most likely parse)
- But: loss of probability mass possible due to failing derivations

**NDCG: Neural Definite Clause Grammar (= DeepStochLog)**

0.5 :: e(N) -> n(N),
0.5 :: e(N) -> e(N1), p, n(N2),
{N is N1 + N2}.
1.0 :: p -> [“*”].

Useful for:
- Subsymbolic processing: e.g. tensors as terminals
- Learning rule probabilities using neural networks

Probability of this parse = 0.5*0.5*0.5*0.1*1*0.1*1*0.1
= 0.000125

**DeepStochLog**

Probability of this parse = 0.5*0.5*0.5*0.1*1*0.1*1*0.1
= 0.000125
DeepStochLog
Inference

And/Or tree + semiring for different inference types

<table>
<thead>
<tr>
<th>Probability of goal</th>
<th>Most likely derivation</th>
</tr>
</thead>
</table>

Deriving probability of goal for given terminals in NDCG

Proof derivations $d(e(1), [+])$

then turn it into and/or tree

Inference optimisation

- Inference is optimized using
  - SLG resolution: Prolog tables the returned proof tree(s), and thus creates forest → Allows for reusing probability calculation results from intermediate nodes

- Batched network calls: Evaluate all the required neural network queries first → Very natural for neural networks to evaluate multiple instances at once using batching & less overhead in logic & neural network communication

<table>
<thead>
<tr>
<th>Lengths</th>
<th>Answers</th>
<th>No Tabling</th>
<th>Tabling</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>0.067</td>
<td>0.060</td>
</tr>
<tr>
<td>3</td>
<td>95</td>
<td>0.061</td>
<td>0.096</td>
</tr>
<tr>
<td>5</td>
<td>186</td>
<td>5.78</td>
<td>0.95</td>
</tr>
<tr>
<td>7</td>
<td>10356</td>
<td>30.42</td>
<td>10.55</td>
</tr>
<tr>
<td>9</td>
<td>68298</td>
<td>1.041</td>
<td>1.12</td>
</tr>
<tr>
<td>11</td>
<td>418032</td>
<td>104.23</td>
<td>199.60</td>
</tr>
</tbody>
</table>
Mathematical expression outcome

T1: Summing MNIST numbers with pre-specified # digits

\[3 + 4 = 13\]

T2: Expressions with images representing operator or single digit number.

\[\frac{7}{2} + 4 \times 3 = 19\]

Classic grammars, but with MNIST images as terminals


\[\text{\texttt{(((())())}}}\]

T4: inputs are strings \(\texttt{ab|bc|c}\) (or permutations of \([a,b,c]\), and \((k+l+m) \mod 3=0\). Predict 1 if \(k=l=m\), otherwise 0.

\[\text{\texttt{1100222}} = 1\]

\[\text{\texttt{1100222}} = 0\]

Citation networks

T5: Given scientific paper set with only few labels & citation network, find all labels

Challenges

- For NeSy,
- scaling up
- which models and which knowledge to use
- large scale life applications
- peculiarities of neural nets & fuzzy logic
- dynamics / continuous
- This is an excellent area for starting researchers / PhDs
Neurosymbolic = Neuro + Logic + Probability

Key Message 1
StarAI and NeSy share similar problems and thus similar solutions apply
See also [De Raedt et al., IJCAI 20; Marra et al, arxiv]

Key Message 2
Provide recipe for NeSy
Kautz
Neural : Symbolic
“an interface layer (<> pipeline) between neural & symbolic components”

THANKS