KU LEUVEN

DEPARTMENT OF COMPUTER SCIENCE

Approximate inference for Neural Probabilistic Logic Programming

Robin Manhaeve[1], Giuseppe Marra[1], Luc De Raedt[1,2]
[1]KU Leuven, [2]Örebro University
robin.manhaeve@cs.kuleuven.be

Integrating reasoning and perception

- Integrating low-level perception with high-level reasoning is one of the oldest, and yet most active open challenges in AI.
- Low-level perception is typically handled by deep learning.
- High-level reasoning is typically addressed using (probabilistic) logical representations and inference.
- Joining the full flexibility of high-level probabilistic reasoning with the representational power of deep neural networks is still an open problem.

Our approach

- Instead of integrating reasoning capabilities into a complex neural network architecture, we proceed the other way round.
- DeepProbLog is a probabilistic logic programming language incorporating deep learning
  - It contains expressive probabilistic-logical modeling and reasoning
  - It encapsulates general-purpose neural networks
  - It can be trained end-to-end on examples
- Our approach has:
  - The expressiveness and strengths of both worlds
  - A clean separation between the neural and the logic components
  - Clear semantics

DeepProbLog

- ProbLog: Prolog + Probabilities
  - (Probabilistic) facts:

$$0.1 :: \texttt{burglary}. \quad 0.2 :: \texttt{earthquake}. \quad 0.5 :: \texttt{hears\_alarm(mary)}. \quad 0.4 :: \texttt{hears\_alarm(john)}.$$

  - Rules:

$$\texttt{alarm} :- \texttt{earthquake}. \quad \texttt{alarm} :- \texttt{burglary}. \quad \texttt{calls} :- \texttt{alarm}, \texttt{hears\_alarm(X)}.$$

- DeepProbLog: ProbLog + Neural predicates
  - Neural predicate: represents relation between input and output in logic
  - Neural Annotated Disjunction (nAD):

$$nn(m_r, \vec{I}, O, \vec{d}) :: r(\vec{I}, O).$$

  - Evaluates a neural network m_r on input $\vec{I}$
  - It defines a probability distribution over the output domain $\vec{d}$.

Exact inference scales poorly

- Standard example: classify the sum of two MNIST numbers, e.g.: 35 + 28 $= 63$
- Encode the background knowledge of the sum.
  - Define the neural predicate for classifying the digits.

$$\texttt{nn(classifier, [X], Y, [0, ..., 9]) :: classify(X, Y).}$$

  - Define the addition.

```
times10plus(X,Y,Z) :- Z is 10 * Y + X.
images_to_number(I,N) :-
    maplist(classify, I, L),
    foldl(timesplus, L, 0, N).

addition(I1,I2,R) :-
    images_to_number(I1,N1),
    images_to_number(I2,N2),
    R is N1 + N2.
```

- Number of proofs generally grows exponentially with length of number.
- Exact inference quickly becomes intractable as it considers all possible proofs
- On proof should hold all the probability mass
- All other proofs are irrelevant

Approximate Inference

- Approximate inference: search for the best subset of proofs
  - An A* search in SLD tree
  - Heuristics are essential to performance
- Probability of (partial) proofs

$$P(\mathcal{E}) = P(\bigwedge_{f \in \mathcal{E}^f} f \bigwedge_{g \in \mathcal{E}^g} g) = \left(\prod_{f \in \mathcal{E}^f} P(f) \right) P(\bigwedge_{g \in \mathcal{E}^g} g \mid \bigwedge_{f \in \mathcal{E}^f} f)$$

- A* search

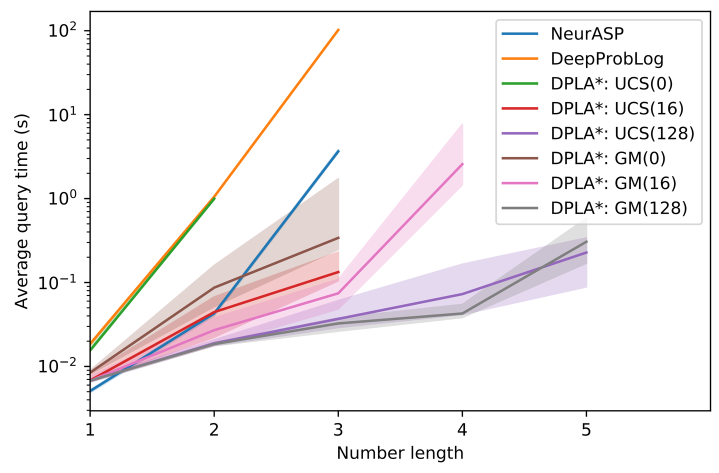
$$\mathrm{cost}(\mathcal{E}) = -\log \left(\prod_{f \in \mathcal{E}^f} P(f) \right) \qquad h(\mathcal{E}) = -\log H(\mathcal{E}) \approx -\log P(\bigwedge_{g \in \mathcal{E}^g} g \mid \bigwedge_{f \in \mathcal{E}^f} f)$$

- Heuristics
  - Uniform cost search
  - One proof should hold all the probability mass: geometric mean heuristic
  - Generalizability between goals that contain sub-symbolic data: learned neural heuristic
- Exploration

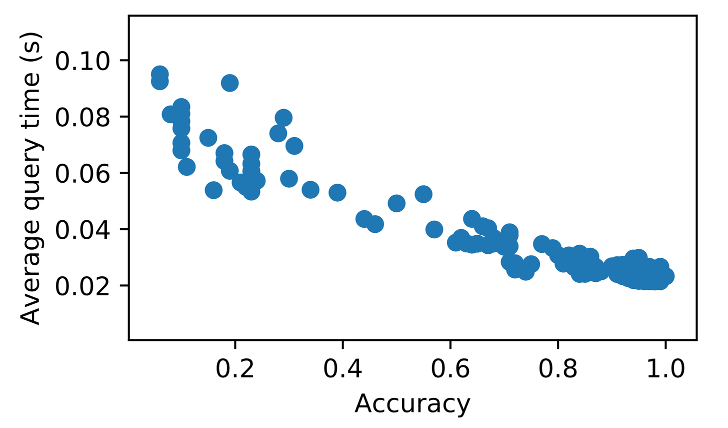
$$\mathrm{cost}(\mathcal{E}) = -\log \left(\prod_{f \in \mathcal{E}^f} P(f) + P_{\mathrm{UCB}}(f) - P(f) P_{\mathrm{UCB}}(f) \right)$$

Experimental evaluation

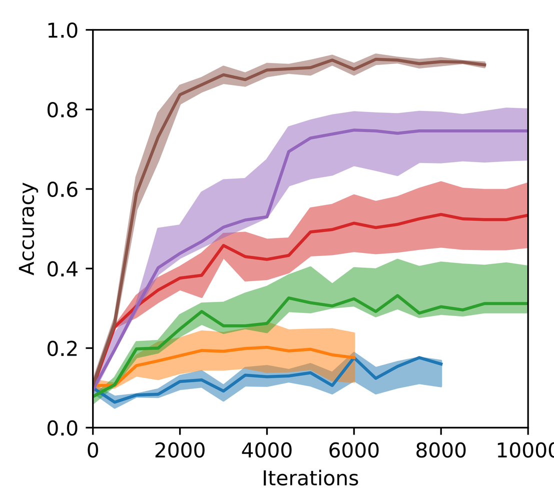
- Experiments:
  - MNIST addition [3, 4] + [7, 5] $= 49$
  - MIL MNIST addition (3 + 4 $= 7$) $\vee$ (7 + 5 $= 7$)
  - HWF
  - CLUTRR



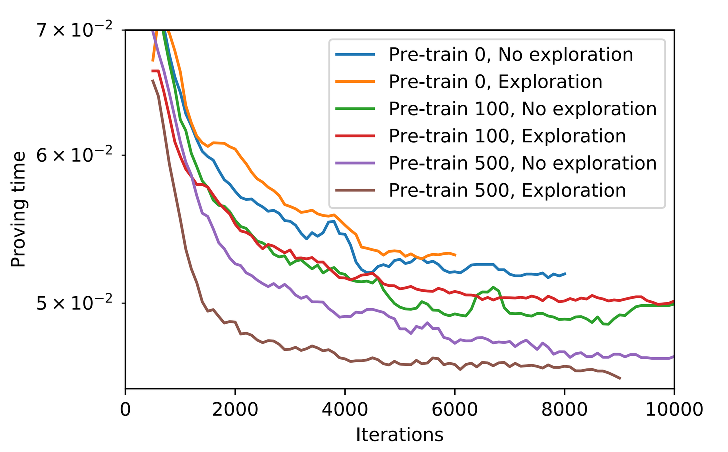
a) The average query time for different methods on the MNIST addition set.



b) The average query time on the MNIST addition task for DPLA* w.r.t. accuracy.



c) Neural predicate accuracy for the MIL Addition task



d) Moving average of the query time for the MIL Addition task

Conclusion

- We introduced **DPLA*** that implements approximate inference for neural-symbolic probabilistic logic programming by using only a subset of all proofs.
- It uses A*-search and heuristics to efficiently search for the best proofs.
- DPLA* is **more scalable** than exact inference methods.
- **Approximate inference** and **learning** can cause issues in convergence and stability.
- **Exploration** or **curriculum learning** can be used to solve these issues.

KU Leuven
Department of Computer Science
Celestijnenlaan 200A box 2402
3001 Leuven, Belgium

DTAI CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD GHENT UNIVERSITY fwo