

probKanren: A Simple Probabilistic extension for miniKanren

Robert Zinkov¹, William E. Byrd²

¹*Peoples' Friendship University of Russia (RUDN University), 6 Miklukho-Maklaya St, Moscow, 117198, Russian Federation*

²*Hugh Kaul Precision Medicine Institute, University of Alabama at Birmingham, 705 20th Street S., Birmingham, AL 35233, United States of America*

Abstract

Probabilistic programming is sometimes described as a generalisation of logic programming where instead of just returning a set of facts that answer a given query, which return a probability distribution over facts that answer a given query. But many contemporary probabilistic logic programming languages are not implemented as simple extensions of existing logic programming languages but instead involve their own unique implementations. Here we introduce probKanren, a simple extension to miniKanren that transforms

Keywords

Probabilistic Logic Programming, miniKanren, Probabilistic Programming, Sequential Monte Carlo

1. Introduction

Conceptually, logic programming provides a way to model non-determinism. This is accomplished by maintaining a set of answers that satisfy a set of logical constraints. A natural generalisation to this domain is adding a notion of uncertainty to this set of answers by associating with them a probability distribution.

1.1. Illustrated Example

To help explain how to use probKanren we introduce the following example:

```
(run 1000 (q)
  (normal 0 3 q)
  (greatero q 0))
```

This probKanren program draws 1000 samples from a normal distribution truncated below 0.


Woodstock'21: Symposium on the irreproducible science, June 07–11, 2021, Woodstock, NY

✉ zinkov@robots.ox.ac.uk (R. Zinkov); webyrd@uab.edu (W. E. Byrd)

🌐 <https://zinkov.com/> (R. Zinkov); <http://webyrd.net/> (W. E. Byrd)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

1.2. Contributions

2. Related Work

There is a rich history of extending logic programming formalisms to support probabilistic inference. Early systems like PRISM[1] and ProbLog[2] allowed associating discrete distributions with facts. Later work[3] introduces distribution clauses so that some continuous distributions. Other work extended these methods further while focusing on efficient exact inference algorithms like model weight integration[4, 5]. Our work is most similar to [6] except that while they combine their forward reasoning with an importance sampler we use a particle cascade instead which can be more sample efficient.

3. Background

3.1. miniKanren

MINIKANREN[7, 8] is a pure logic programming language embedded in Scheme. The language consists of a set of terms, functions for constructing, a set of goal primitives, and two run functions to answer queries in the language. The goal primitives consist of a fresh form for introducing logic variables, a unification primitive, a conjunction combinator, and a disjunction combinator.

3.2. Probabilistic Programming

Probabilistic Programming Languages[9, 10] are a family of domain specific languages for posing and efficiently solving probabilistic modelling problems. At their core, all have a way to sample from a probability distribution and observe data generated from a probability distribution.

There are many ways to implement inference algorithms for probabilistic programming languages but methods based on likelihood-weighting and sequential monte carlo algorithms are the easiest.

3.3. Sequential Monte Carlo

Sequential Monte Carlo[11] are an efficient online way to sample from probabilistic models especially suited for state-space domains. If we imagine our probabilistic programs as straight-line programs with no control-flow we can imagine numbering every sample function x_1, f_2, \dots, f_n and every observe function g_1, g_2, \dots, g_n then our probability density over our random variables x and observed data y can be defined as:

$$p(x, y) = \prod_i f(x_i \mid x_{0:n-1})g(y_i \mid x_{0:n}) \quad (1)$$

3.4. Sequential Importance Sampling

If we imagine running this program each execution trace can be seen as a sample from the distribution. If we then weight by their likelihood as we run them that collection execution

traces will be an empirical distribution of the given program. We call each of these execution traces particles and the below method is how we obtain them along with their weights w .

$$x_n \sim f(x_n \mid x_{0:n-1}) \quad (2)$$

$$w_n^{(k)} = \frac{g(x_n \mid x_{0:n})f(x_n \mid x_{0:n-1})}{q(x_n \mid x_{0:n-1})} \quad (3)$$

$$W_n^{(k)} = W_{n-1}^{(k)} w_n^{(k)} \quad (4)$$

The weights are then normalised at the end to make a proper probability distribution.

3.5. Sequential Importance Resampling

The problem with the above algorithm is over time for many particles W_k is going to become low and that particle will stop being very informative of the underlying distribution. To mitigate this issue, each time we encounter an observation we resample our particles. Resampling effectively removes particles with low weight and duplicates particles with higher weight by sampling with replacement our existing particles.

The above is called multinomial resampling but there are other methods as well. A survey[12] of resampling methods suggests all of them are helpful to reduce particle degeneracy.

3.6. Particle Cascade

As the SMC resampling step was defined in the previous section Particle Cascades [13] remove this barrier allowing every particle to be resampled asynchronously with the associated weights being relative to a global running average.

4. Proposed Method

We propose to extend MINIKANREN by augmenting each of the search streams with a set of particles. These particles represent the empirical distribution of that stream. Each particle has associated with it a substitution of all the logic and random variables as well as a weight that is proportional to the likelihood of the substitution.

An initial set of particles is created from the probabilistic program when it is first run. As disjunctions like `conde` are encountered, we split evenly the number of particles allocated to each stream. Whenever we encounter a unification primitive, we run a resampling step. This helps to prune low-weight particles and replicate high weight ones.

As an optimisation we may create more particles during resampling based on a globally stored a counter of the effective sample size of all particles across all streams.

We follow [3] and place the following restrictions on our distribution clauses and the random variables they specify.

Firstly, the arguments of distribution clauses must be ground. Secondly, a random variable can not unify with any arithmetic expression

5. Experiments

One of the experiments should be something from the probabilistic logic literature.

Friends who Smoke is a probabilistic logic program which models the social nature of who smokes cigarettes. The model predicts that people who are friends with people who smoke are more likely to smoke.

We could include experiment from Noah Goodman’s dippl work like a semantic parsing example

<http://dippl.org/examples/semanticparsing.html>

We could include program synthesis experiment, where probabilities allow us to specify a soft preference for using certain language primitives in a similar spirit to rKANREN[14] or neural-guided search [15].

6. Conclusions

We made a cool and simple to implement extension to MINIKANREN that let’s us support probabilistic inference on both discrete and continuous distributions.

References

- [1] T. Sato, Y. Kameya, Prism: a language for symbolic-statistical modeling, in: IJCAI, volume 97, Citeseer, 1997, pp. 1330–1339.
- [2] L. De Raedt, A. Kimmig, H. Toivonen, Problog: A probabilistic prolog and its application in link discovery., in: IJCAI, volume 7, Hyderabad, 2007, pp. 2462–2467.
- [3] B. Gutmann, M. Jaeger, L. De Raedt, Extending problog with continuous distributions, in: International Conference on Inductive Logic Programming, Springer, 2010, pp. 76–91.
- [4] M. A. Islam, C. Ramakrishnan, I. Ramakrishnan, Inference in probabilistic logic programs with continuous random variables, Theory and Practice of Logic Programming 12 (2012) 505–523.
- [5] V. Belle, A. Passerini, G. Van den Broeck, Probabilistic inference in hybrid domains by weighted model integration, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.
- [6] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, L. De Raedt, The magic of logical inference in probabilistic programming, Theory and Practice of Logic Programming 11 (2011) 663–680.
- [7] W. E. Byrd, M. Ballantyne, G. Rosenblatt, M. Might, A unified approach to solving seven programming problems (functional pearl), Proceedings of the ACM on Programming Languages 1 (2017) 1–26. URL: <https://doi.org/10.1145/3110252>.
- [8] D. P. Friedman, W. E. Byrd, O. Kiselyov, J. Hemann, The Reasoned Schemer, 2nd ed., MIT Press, 2018.
- [9] F. Wood, J. W. Meent, V. Mansinghka, A new approach to probabilistic programming inference, in: Artificial Intelligence and Statistics, PMLR, 2014, pp. 1024–1032.

- [10] J. van de Meent, B. Paige, H. Yang, F. Wood, An introduction to probabilistic programming, CoRR abs/1809.10756 (2018). URL: <http://arxiv.org/abs/1809.10756>.
- [11] N. Chopin, O. Papaspiliopoulos, et al., An Introduction to Sequential Monte Carlo, volume 4, Springer, 2020.
- [12] R. Douc, O. Cappé, Comparison of resampling schemes for particle filtering, in: ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005., IEEE, 2005, pp. 64–69.
- [13] B. Paige, F. D. Wood, A. Doucet, Y. W. Teh, Asynchronous anytime sequential monte carlo, in: Advances in Neural Information Processing Systems, 2014, pp. 3410–3418. URL: <https://proceedings.neurips.cc/paper/2014/hash/7eb7eabbe9bd03c2fc99881d04da9cbd-Abstract.html>.
- [14] C. Swords, D. Friedman, rkanren: Guided search in minikanren, in: Proceedings of Scheme Workshop, 2013.
- [15] L. Zhang, G. Rosenblatt, E. Fetaya, R. Liao, W. E. Byrd, M. Might, R. Urtasun, R. Zemel, Neural guided constraint logic programming for program synthesis, arXiv preprint arXiv:1809.02840 (2018).