IMPLEMENTING A PROBABILISTIC PROGRAMMING LANGUAGE

Maxwell Clarke – maxeonyx@gmail.com – 300559026

COMP440 Assignments 1 & 2 – Wed 16 Jun 2021 – Victoria University of Wellington

See the code on GitHub

ABSTRACT

In this report I describe my implementation of a first-order probabilistic programming language in fulfillment of Assignment 1 and 2 for this COMP440 Directed Individual Study (DIS). These assignments are based on the 2020 UBC course CS532W [1], specifically Homeworks [2] & [3].

The language is an implementation of the first-order probabilistic programming language from the book *An Introduction to Probabilistic Programming* [4]. It is a Lisp-like dynamic language, and implements two evaluation-based inference algorithms, Importance Sampling (Likelihood Weighting), and Single-Site Metropolis-Hastings.

Contents

1. BACKGROUND

1.1. Bayesian Inference

Bayesian inference methods help us approximate the posterior to some prior, given some observations. This is typically used for finding predictive distributions given a model and some data, but can also be used for finding distributions of model parameters given some data, or any other task.

Exact Bayesian inference is sometimes possible, but in most real-world applications becomes intractable. Therefore, we use approximate inference methods, such as:

- Belief propagation (sum-product message passing). This is restricted to finite acyclic models. It passes messages around a graph, and their aggregation converges to the posterior.
- Importance Sampling. This is a Monte Carlo method which samples from the prior and weights the samples according to their probability in the posterior.
- Markov Chain Monte Carlo methods, such as Metropolis-Hastings and Hamiltonian Monte Carlo. These create a sequence of samples by modifying previous samples, and the sequence converges to the posterior.

1.2. Probabilistic Programming Languages

Probabilistic programming languages allow flexible creation of probabilistic models and the ability to perform inference on them. They stand in contrast to ad-hoc inference, and are similar to inference libraries. Some probabilistic languages can be used as embedded languages or libraries from within another language, such as Stan [5]. Examples of probabilistic programming languages are:

- Stan [5].
- PyMC3 [6]
- WebPPL [7]

2. ASSIGNMENTS

For this DIS we set assignments based off the UBC homework assignments. These built off each other and involved implementing a probabilistic programming language to complete a number of tasks.

Assignment 1 involved implementing the parser, interpreter and sampling from the prior.

Assignment 2 completed the language by implementing inference algorithms to compute the posterior.

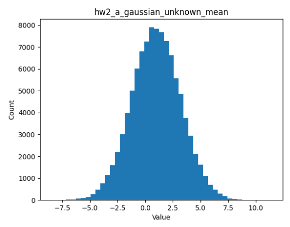

Fig. 1. Prior distribution on μ in Listing 1 (Gaussian Unknown Mean)

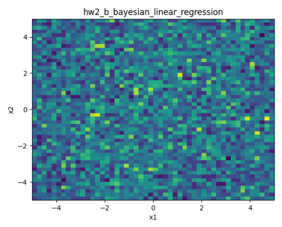

Fig. 2. Prior distribution on **slope** (x_1) and **bias** (x_2) in Listing 2 (Bayesian Linear Regression)

2.1. Assignment 1

Assignment 1 was based off UBC Homework 2 [2]. The task for this assignment was to implement the evaluator for the language, and to sample from the prior of a selection of models.

The UBC homework asked to complete two separate but similar evaluators. The first, which I implemented, was the evaluator for the programming language described in [4]. The second, which I did not implement, was to evaluate the same programs via ancestral sampling after a transformation (provided by the Daphne [8] compiler) to a Probabilistic Graphical Model (PGM). The difference between the two evaluators is only the order of evaluation of the program - in the PGM evaluator, there is an explicit dependency graph, and sampling can be done in any topological order.

The models for this task were:

(a) Gaussian Unknown Mean (Listing 1). Infers the posterior over the mean of a gaussian given some 1 dimensional data.

(b) Bayesian Linear Regression (Listing 2). Infers the slope and bias parameters of a 2D line given (x, y) pairs.

(c) Hidden Markov Model (Listing 3). Infers the hidden states of the hidden markov model given observed emissions.

(d) Bayesian Neural Network (Listing 4). Infers the posterior over the weight parameters given a dataset of (x, y) pairs.

I chose to implement the language using Rust [9] as the host language, because I have prior experience using Rust to implement my own dynamic language called Kal [10].

The implementation of the probabilistic programming language consisted of four major parts:

1. Implementing the parser for the language. This was done using the parser generator library LALRPOP [11]. The grammar is defined in the file grammar.lalrpop.
2. Implementing the interpreter, including basic language features such as variables and functions, and built-in

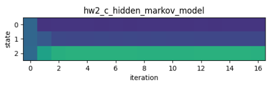

Fig. 3. Prior distribution on all hidden states in Listing 3 (Hidden Markov Model). Each column shows the probability of states 0, 1 and 2 at the given step.

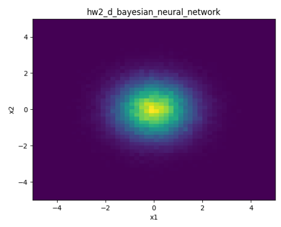

Fig. 4. Prior distribution on weights $W_{0,0}$ (x_1) and $W_{0,1}$ (x_2) from the first layer in Listing 4 (Bayesian Neural Network).

functions. The interpreter implementation is located in interpreter.rs, and the built-in functions are located in functions.rs.

3. Implementing a command line interface for the program and JSON output format.
4. Implementing a python program to visualize the outputs of performing inference on the example programs.

Visualizations of the inference results for each of the task programs are shown in Figures 1 to 4.

2.2. Assignment 2

The second assignment involved implementing inference algorithms to sample from the posterior of the models. The UBC homework task was to implement the following three algorithms, and test them on a selection of programs.

- Importance Sampling (IS) (Likelihood weighting)
- Metropolis-in-Gibbs (MH Gibbs)
- Hamiltonian Monte Carlo (HMC)

The programs to test on were:

(a) Gaussian Unknown Mean (Listing 1). Same as Assignment 1.

(b) Bayesian Linear Regression (Listing 2). Same as Assignment 1.

(c) Gaussian Mixture (Listing 5). Posterior probability (binary distribution) that the first and second data points are in the same cluster.

(d) Sprinkler model (Listing 6). Posterior probability that it is raining in the classic sprinkler model.

(e) Dirac distribution. Dirac distribution is a pdf with density only at one point. I didn't attempt this one.

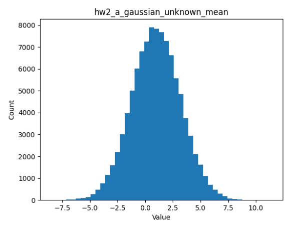

Fig. 5. Posterior distribution of Gaussian Unknown Mean program using importance sampling (IS)

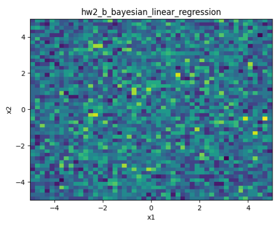

Fig. 6. Posterior distribution of Bayesian Linear Regression program using IS

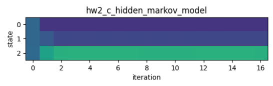

Fig. 7. Posterior distribution of Hidden Markov Model program using IS

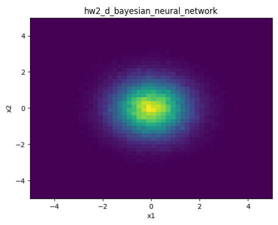

Fig. 8. Posterior distribution of Bayesian Neural Network program using IS

My implementation of IS can be found in likelihood_weighting.rs. MH Gibbs required I had implemented the PGM evaluator in Assignment 1, so I did not implement it. Finally, since we had discussed another inference algorithm during our meetings, single-site Metropolis-Hastings (SSM), I chose to implement this instead of HMC. My implementation of SSMH is located in single_site_metropolis.rs.

Importance sampling works by drawing a sample from the prior, then weighting the sample according to the density at this sample in the posterior (the likelihood of the sample). To do this, a sample is produced by evaluating the program. Then, at each *sample* or *observe* statement, the density of the distribution associated with the *sample* or *observe* statement, at the sampled or observed value, is evaluated and added to a running product. The product at the end of the evaluation is used as the weight for the sample.

Importance sampling is useful for many applications, but performs poorly in applications where there are regions that are very low probability in the prior but very high probability in the posterior. Importance sampling may fail to find these regions because it does not use information about the posterior when sampling. For these applications a Markov chain method is usually more appropriate.

Visualizations of the posterior of the test models using importance sampling are shown in Figures 5 to 10.

Secondly, I implemented single-site Metropolis-Hastings (SSMH). This is an evaluation-based Markov chain method that proposes new samples by changing a single variable (Re-

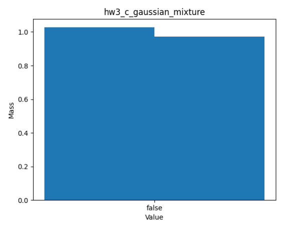

Fig. 9. Posterior distribution of Gaussian Mixture program (Listing 5) using IS

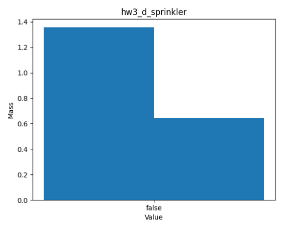

Fig. 10. Posterior distribution of Sprinkler program (Listing 6) using IS

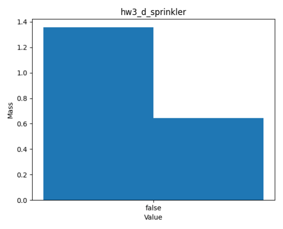

Fig. 15. Posterior distribution of Sprinkler program (Listing 6) using SSMH

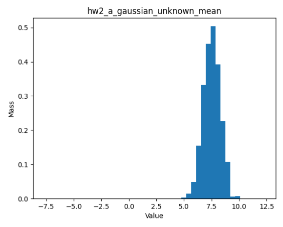

Fig. 11. Posterior distribution of Gaussian Unknown Mean program using SSMH

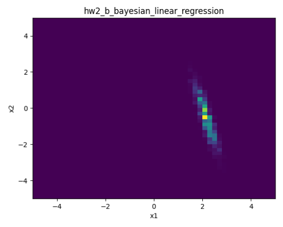

Fig. 12. Posterior distribution of Bayesian Linear Regression program using SSMH

samples a single *sample* statement) between program runs, and computes the acceptance probability by re-evaluating the log weight similarly to importance sampling. This is useful for some programs because a markov chain method can find high-probability regions of the posterior, even if they are low probability in the prior, and draw more samples there.

To implement SSMH, I extended the evaluator to generate a unique identifier for each *sample* and *observe* expresssion in the program. This is neccessary to associate the random variables between program runs, so their values can be reused.

Visualizations of the posterior of the test models using SSMH are shown in Figures 11 to 15.

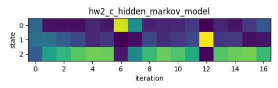

Fig. 13. Posterior distribution of Hidden Markov Model program using SSMH

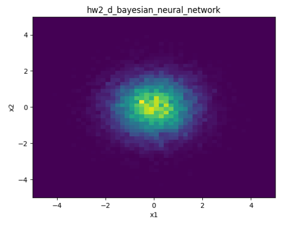

Fig. 14. Posterior distribution of Bayesian Neural Network program using SSMH

3. USAGE GUIDE

The GitHub repository contains the following folders at the root:

- homework-specs. This folder contains backups of the homework assignments in case the UBC course is taken offline.
- ppl-impl. The implementation of this probabilistic programming language.
- stan-programs. Implementations of the demo programs for the essay in Stan.
- webppl-programs. Implementations of the demo programs for the essay in WebPPL.
- thisppl-programs. Implementations of the demo programs for the essay in this PPL implementation.
- webppl*. Files from early in the DIS.

To run the programming language, use either ”cargo run” (if you have installed Rust) or ”thisppl” (if you are using the binary from the GitHub release. The program will provide help.

An example of running the program on Listing 2, is ”./thisppl infer hw2_b_bayesian_linear_regression.ppl likelihood-weighting”. This runs inference using the default number of samples, and writes an output data file to a file called ”./data/hw2_b_bayesian_linear_regression.json”.

4. SUMMARY

I created an implementation of the programming language from [4]. It fulfills the requirements for UBC Homework 2, except for evaluating the programs via ancestral sampling. It fulfills the requirements for UBC Homework 3, except there is no implementation of MH Gibbs, and it does not run the Dirac distribution example. HMC is swapped out for SSMH.

References

[1] F. Wood, “Topics in AI: Probabilistic Programming,” https://www.cs.ubc.ca/~fwood/CS532W-539W/, 2021, accessed: 2021-06-15.

[2] ——, “CS532W Homework 2,” https://www.cs.ubc.ca/~fwood/CS532W-539W/homework/2.html, 2021, accessed: 2021-06-15.

[3] ——, “CS532W Homework 3,” https://www.cs.ubc.ca/~fwood/CS532W-539W/homework/3.html, 2021, accessed: 2021-06-15.

[4] J.-W. van de Meent, B. Paige, H. Yang, and F. Wood, “An Introduction to Probabilistic Programming,” https://arxiv.org/abs/1809.10756, 2018.

[5] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, and A. Riddell, “Stan: a probabilistic programming language.” *Grantee Submission*, vol. 76, no. 1, pp. 1–32, 2017.

[6] F. C. Salvatier J, Wiecki TV, “Probabilistic programming in Python using PyMC3,” https://peerj.com/articles/cs-55/, 2016.

[7] N. D. Goodman and A. Stuhlmüller, “The Design and Implementation of Probabilistic Programming Languages,” http://dippl.org, 2014, accessed: 2021-06-16.

[8] C. Weilbach, B. Beronov, W. Harvey, and F. Wood, “The Daphne probabilistic programming compiler,” https://github.com/plai-group/daphne, 2020, accessed: 2021-06-16.

[9] S. Klabnik and C. Nichols, *The Rust Programming Language*. No Starch Press, 2019.

[10] M. Clarke, “Kal Programming Language,” https://github.com/maxeonyx/kal, 2020, accessed: 2021-06-16.

[11] N. Matsakis, M. Westerlind, and contributors., “LALRPOP Parser Generator,” https://github.com/lalrpop/lalrpop, 2021, accessed: 2021-06-16.

5. APPENDIX

Listing 1. Gaussian Unknown Mean

```
(let [mu (sample (normal 1 (sqrt 5)))
           sigma (sqrt 2)
           lik (normal mu sigma)]
       (observe lik 8)
       (observe lik 9)
    mu)
```

Listing 2. Bayesian Linear Regression

```
(defn observe-data [_ data slope bias]
        (let [xn (first data)
              yn (second data)
              zn (+ (* slope xn) bias)]
          (observe (normal zn 1.0) yn)
          (rest (rest data))))

(let [slope (sample (normal 0.0 10.0))
             bias  (sample (normal 0.0 10.0))
             data (vector 1.0 2.1 2.0 3.9 3.0 5.3
                           4.0 7.7 5.0 10.2 6.0 12.9)]
   (loop 6 data observe-data slope bias)
   (vector slope bias))
```

Listing 3. Hidden Markov Model

```
(defn hmm-step [t states data trans-dists likes]
      (let [z (sample (get trans-dists
                            (last states)))]
        (observe (get likes z)
                  (get data t))
        (append states z)))

(let [data [0.9 0.8 0.7 0.0 -0.025 -5.0 -2.0 -0.1
            0.0 0.13 0.45 6 0.2 0.3 -1 -1]
      trans-dists [(discrete [0.10 0.50 0.40])
                    (discrete [0.20 0.20 0.60])
                    (discrete [0.15 0.15 0.70])]
      likes [(normal -1.0 1.0)
             (normal 1.0 1.0)
             (normal 0.0 1.0)]
      states [(sample (discrete [0.33 0.33 0.34]))]]
  (loop 16 states hmm-step data trans-dists likes))
```

Listing 4. Bayesian Neural Network Learning

```
(let [weight-prior (normal 0 1)
      W_0 (foreach 10 []
             (foreach 1 [] (sample weight-prior)))
      W_1 (foreach 10 []
             (foreach 10 [] (sample weight-prior)))
      W_2 (foreach 1 []
             (foreach 10 [] (sample weight-prior)))

      b_0 (foreach 10 []
             (foreach 1 [] (sample weight-prior)))
      b_1 (foreach 10 []
             (foreach 1 [] (sample weight-prior)))
      b_2 (foreach 1 []
             (foreach 1 [] (sample weight-prior)))

      x   (mat-transpose [[1] [2] [3] [4] [5]])
      y   [[1] [4] [9] [16] [25]]
      h_0 (mat-tanh (mat-add (mat-mul W_0 x)
                              (mat-repmat b_0 1 5)))
      h_1 (mat-tanh (mat-add (mat-mul W_1 h_0)
                              (mat-repmat b_1 1 5)))
      mu  (mat-transpose
             (mat-tanh (mat-add (mat-mul W_2 h_1)
                                (mat-repmat b_2 1 5))))]
(foreach 5 [y_r y
            mu_r mu]
   (foreach 1 [y_rc y_r
               mu_rc mu_r]
      (observe (normal mu_rc 1) y_rc)))
[W_0 b_0 W_1 b_1])
```

Listing 5. Gaussian Mixture Model

```
(let [data [1.1 2.1 2.0 1.9 0.0 -0.1 -0.05]
      likes (foreach 3 []
                     (let [mu (sample (normal 0.0 10.0))
                           sigma (sample (gamma 1.0 1.0))]
                       (normal mu sigma)))
      pi (sample (dirichlet [1.0 1.0 1.0]))
      z-prior (discrete pi)
      z (foreach 7 [y data]
          (let [z (sample z-prior)
                _ (observe (get likes z) y)]
            z))]
  (= (first z) (second z)))
```

Listing 6. Sprinkler

```
(let [sprinkler true
      wet-grass true
      is-cloudy (sample (flip 0.5))

      is-raining (if (= is-cloudy true )
                    (sample (flip 0.8))
                    (sample (flip 0.2)))
      sprinkler-dist (if (= is-cloudy true)
                        (flip 0.1)
                        (flip 0.5))
      wet-grass-dist (if (and (= sprinkler true)
                              (= is-raining true))
                        (flip 0.99)
                        (if (and (= sprinkler false)
                                 (= is-raining false))
                          (flip 0.0)
                          (if (or (= sprinkler true)
                                  (= is-raining true))
                            (flip 0.9))))]
  (observe sprinkler-dist sprinkler)
  (observe wet-grass-dist wet-grass)
  is-raining)
```