Probabilistic Programming in Julia: New Inference Algorithms

Kai Xu Engineering Department University of Cambridge

Introduction

Probabilistic Programming (PP) provides a very flexible framework to define probabilistic models and to learn models from data. This frees researchers from programming probablistic models and inference algorithms by hand and enables them to focus more on designing a suitable model by their insights.

In PP, Bayesian inference is automated and inference is done by general inference algorithms such as **Monte Carlo Markov Chain** (MCMC) algorithms, which together make it possible to learn any model defined by users.

This project is aimed to produce a new inference algorithm, **Hamiltonian Monte Carlo** (HMC) under a PP framework called **Turing.jl** currently under developed in the Engineering Department.

Detailed objectives are listed below.

1. Implement HMC in Julia
2. Implement common distributions in Julia
3. Implement PP framework for HMC
4. Evaluate HMC against other inference algorithms in Turing.jl and other PPLs

Bayesian Inference

Bayesian inference uses Bayes' rule to compute the posterior distribution of model parameters by incorporating prior knowledge to the likelihood of data, which is given by

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{\int_\theta P(D|\theta)P(\theta)} \propto P(D|\theta)P(\theta) \qquad (1)$$

, where D is the data set, θ is the model parameter, $P(\theta)$ is the prior, $P(D|\theta)$ is the likelihood of data and $P(\theta|D)$ is the posterior distribution.

References

[1] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Automating Inference

Models in Turing.jl are defined using three syntax **param**, **observe** and **predict**, where **param** defines the prior distribution, **observe** defines the likelihood of data and **predict** defines the parameters to sample.

Inference is then automated by

1. Transform the defined model to the posterior distribution of model parameters;
2. Run inference algorithms on the posterior distribution

Metaprogramming

Keywords **param**, **observe** and **predict** are **marcos** in Julia, which can be expanded to a section of code. Here the model defined can be transformed to a function `f()` representing the posterior distribution $P(\theta|D)$ in Equation 1.

Hamiltonian Monte Carlo

HMC is then used to generate samples of model parameters from the posterior function `f()`, where the gradient information is used to accelerate the process of sampling.

The plot in Figure 1 compares HMC against normal Metropolis-Hasting when sampling from a multivariate Gaussian.

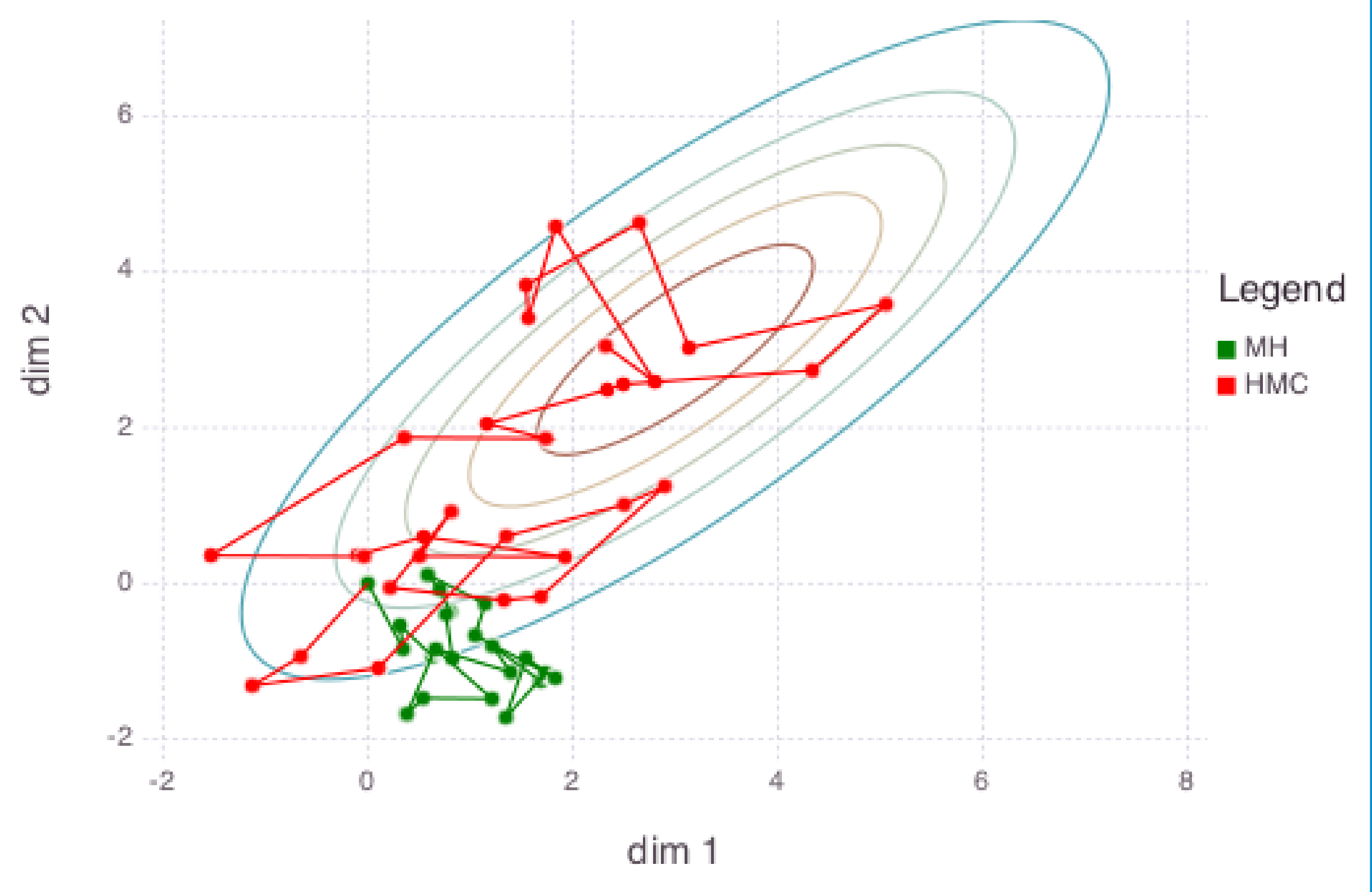

Figure 1: Samples from Gaussian using HMC and MH

Contact Information

Email kx216@cam.ac.uk
Phone +44 770 771 9899

Example 1 - Univariate Gaussian

A univariate Gaussian with conjugate priors and data $D = \{1, 1.6, 1, 1.1, 0.9, 1.3, 0.6\}$ can be defined and learnt by the code below.

```
D = [1,1.6,1,1.1,0.9,1.3,0.6]
@model gauss begin
  @param σ ~ InverseGamma(2, 3)
  @param μ ~ Normal(0, √σ)
  for x in D
    @observe x ~ Normal(μ, √σ)
  end
  @predict μ σ
end
samples = sample(gauss, HMC(500))
```

The traces of μ and σ are shown in Figure 2.

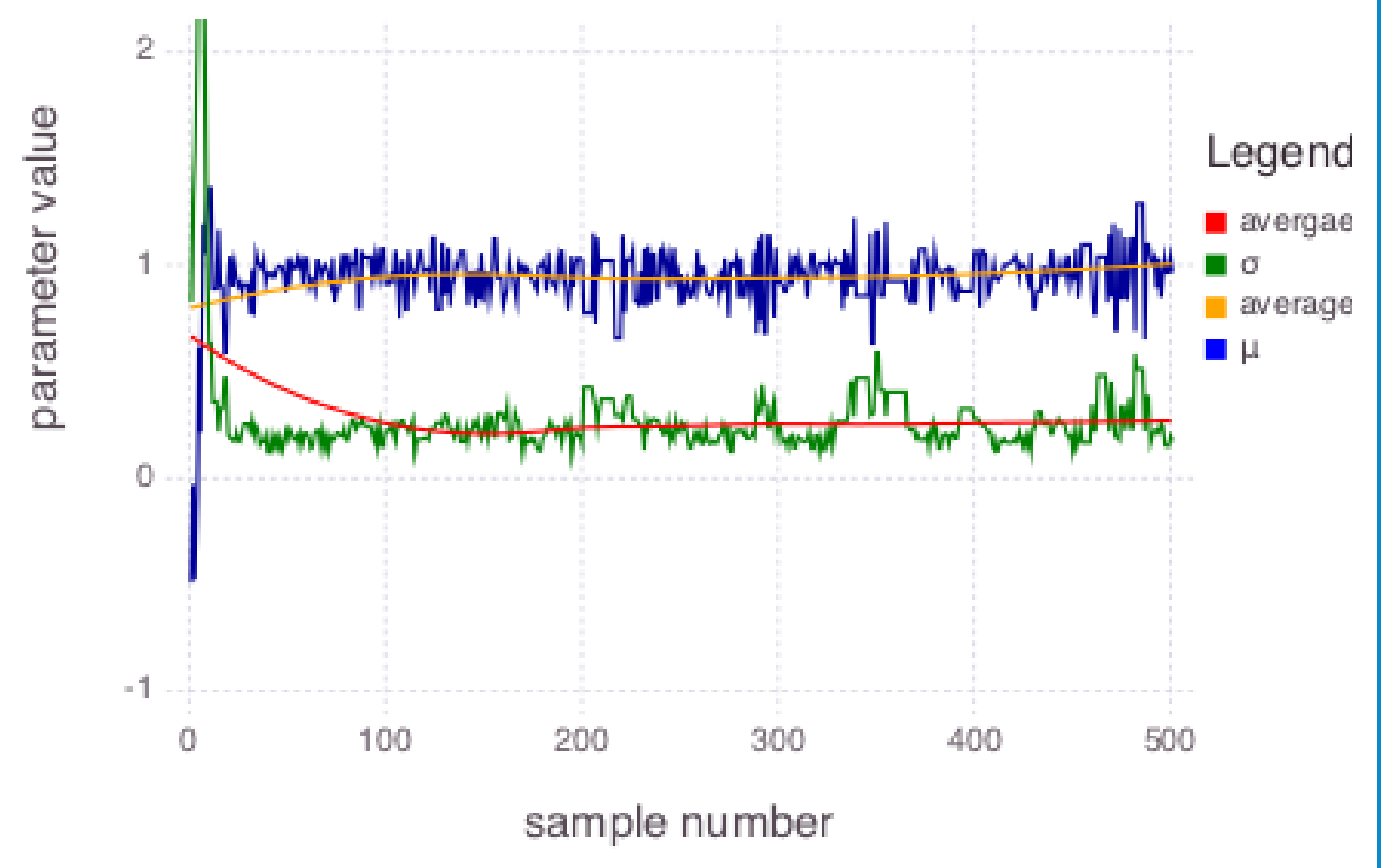

Figure 2: Trace of μ and σ

Example 2 - Bayesian Neural Network

PP framework can also be used to train neural networks by interpreting the loss function as likelihood and the regularisation term as prior.

To train a Bayesian neural network (BNN) with structure in Figure 3, learning the exclusive-or function, the program on the right can be used.

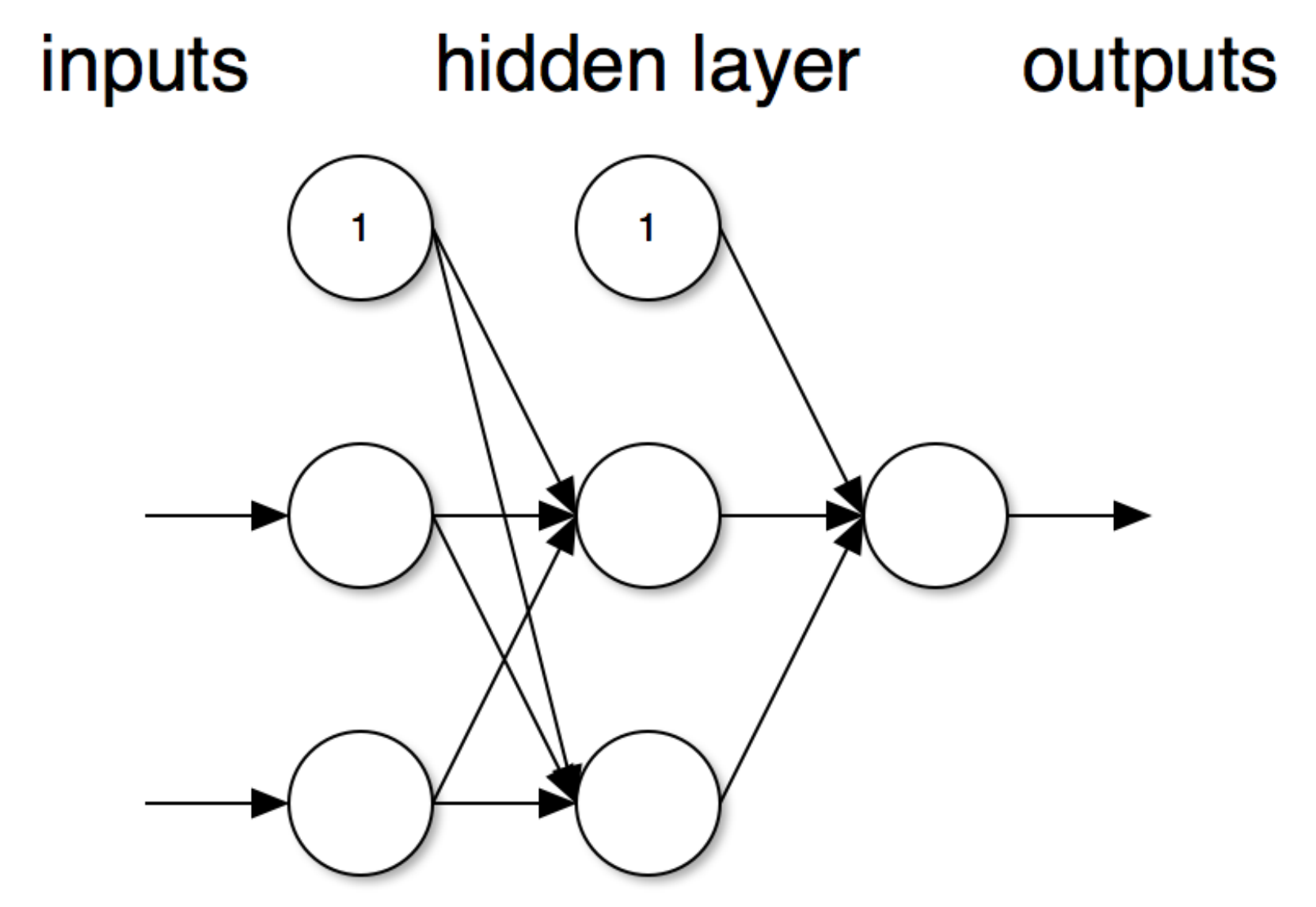

Figure 3: Structure of the Bayesian neural network

```
xs = [[0;0]; [0;1]; [1;0]; [1;1]]
ts = [0; 1; 1; 0]
@model bnn begin
  weights = TArray(Float64, 9)
  @param σ ~ InverseGamma(2, 3)
  for w in weights
    @param w ~ Normal(0, √σ)
  end
  for i in 1:4
    y = nn(xs[i], weights)
    @observe ts[i] ~ Bernoulli(y)
  end
  @predict weights
end
samples = sample(bnn, HMC(500))
```

Progress and Next Step

By the day of poster session, HMC sampler has been implemented in Julia and a wrapper of common distributions is also available. Some handwritten models were used to test these two components. In addition, the student is also contributing to the documentation work.

The next step for the project is to implement the compiler for HMC in the PP framework. When this is done, it will be embedded into Turjing.jl, working with existing inference algorithms. More evaluations on the performance of HMC and Turing.jl will be conducted.