RxInfer: A Julia package for reactive real-time Bayesian inference

Dmitry Bagaev, Albert Podusenko, and Bert de Vries

1 Technical University of Eindhoven

Summary

Bayesian inference realizes optimal information processing through a full commitment to reasoning by probability theory. The Bayesian framework is positioned at the core of modern AI technology for applications such as speech and image recognition and generation, medical analysis, robot navigation, and more. The framework describes how a rational agent should update its beliefs when new information is revealed by the agent’s environment. Unfortunately, perfect Bayesian reasoning is generally intractable, since calculations of (often) very high-dimensional integrals are required for many models of interest. As a result, a number of numerical algorithms for approximating Bayesian inference have been developed and implemented in probabilistic programming packages. Successful methods include the Laplace approximation (Gelman et al., 2015), variants of Monte Carlo (MC) sampling (Salimans et al., n.d.), Variational Inference (VI) (Blei et al., 2017), Automatic-Differentiation Variational Inference (ADVI) (Kucukelbir et al., 2017), and Black-Box Variational Inference (BBVI) (Bamler & Mandt, 2017).

We present RxInfer.jl, which is a Julia (Jeff Bezanson et al., 2012; J. Bezanson et al., 2017) package for real-time variational Bayesian inference based on reactive message passing in a factor graph representation of the model under study (Bagaev & Vries, 2021). RxInfer.jl provides access to a powerful model specification language that translates a textual description of a probabilistic model into a corresponding factor graph representation. In addition, RxInfer.jl supports hybrid variational inference processes, where different Bayesian inference methods can be combined in different parts of the model, resulting in a straightforward mechanism to trade off accuracy for computational speed. The underlying implementation relies on a reactive programming paradigm and supports by design the processing of infinite asynchronous data streams. In the proposed framework, the inference engine reacts to new data and automatically updates relevant posteriors.

Over the past few years, the inference methods in this package have been tested on many advanced probabilistic models, resulting in several publications in highly ranked journals such as Entropy (Podusenko, Kouw, et al., 2021; Şenöz et al., 2021), Frontiers (Podusenko, Erp, Kouw, et al., 2021), and conferences such as MLSP-2021 (Podusenko, Erp, Bagaev, et al., 2021), EUSIPCO-2022 (Erp & Vries, 2022; Podusenko et al., 2022) and SiPS (Nguyen et al., 2022).

Statement of need

Many important AI applications, including audio processing, self-driving vehicles, weather forecasting, and extended-reality video processing, and others require continually solving an inference task in sophisticated probabilistic models with a large number of latent variables. Often, the inference task in these applications must be performed continually and in real time in response to new observations. Popular MC-based inference methods, such as the No U-Turn Sampler (NUTS) (Hoffman & Gelman, 2011) or Hamiltonian Monte Carlo (HMC)
sampling (Brooks et al., 2011), rely on computationally heavy sampling procedures that do not scale well to probabilistic models with thousands of latent states. Therefore, MC-based inference is practically not suitable for real-time applications. While the alternative variational inference method (VI) promises to scale better to large models than sampling-based inference, VI requires the derivation of gradients of the “variational Free Energy” cost function. For large models, manual derivation of these gradients might not be feasible, while automated “black-box” gradient methods do not scale either because they are not capable of taking advantage of sparsity or conjugate pairs in the model. Therefore, while Bayesian inference is known as the optimal data processing framework, in practice, real-time AI applications rely on much simpler, often ad hoc, data processing algorithms.

Solution proposal

We present RxInfer.jl, a package for processing infinite data streams by real-time Bayesian inference in large probabilistic models. RxInfer.jl implements variational Bayesian inference as a variational Constrained Bethe Free Energy (CBFE) functional optimization process (Şenöz et al., 2021). The underlying inference engine derives its speed from taking advantage of both statistical independencies and conjugate pairings of variables in the factor graph. Inference proceeds continually by an automated reactive message passing process on the graph, where each message carves away a bit of the variational Free Energy cost function. Very often, closed-form message computation rules are available for specific nodes and node combinations, leading to much faster inference than sampling-based inference methods, and additionally enables hierarchical composition of different models without need for extra derivations. These properties distinguish RxInfer.jl from other popular Bayesian inference libraries in Julia, such as Turing.jl (Ge et al., 2018), Stan.jl (Stan Development Team, 2022; Stan.jl Development Team, 2022), and others, which are not designed to run inference continually in response to new observations in real-time.

Overview of functionality

RxInfer.jl is an open source package, available at https://github.com/biaslab/RxInfer.jl, and enjoys the following features:

- A user-friendly specification of probabilistic models. Through Julia macros, RxInfer.jl is capable of automatically transforming a textual description of a probabilistic model to a factor graph representation of that model.
- A hybrid inference engine. The inference engine supports a variety of well-known message passing-based inference methods such as belief propagation, structured and mean-field variational message passing, expectation propagation, expectation maximization, and conjugate-computation variational inference (CVI) (Akbayrak et al., 2022).
- A customized trade-off between accuracy and speed. For each location (node and edge) in the graph, RxInfer.jl allows a custom specification of the inference constraints on the variational family of distributions in the CBFE optimization procedure. This enables the use of different Bayesian inference methods at different locations of the graph, leading to an optimized trade-off between accuracy and speed.
- Support for real-time processing of infinite data streams. RxInfer.jl is based on a reactive programming paradigm that enables asynchronous data processing as soon as data arrives.
- Support for large static data sets. The package is not limited to real-time processing of data streams and also scales well to batch processing of large data sets and large probabilistic models that can include hundreds of thousands of latent variables (Bagaev, 2021).
- RxInfer.jl is extensible. The public API defines a straightforward and user-friendly way to extend the built-in functionality with custom nodes and message update rules.

The inference procedure is auto-differentiable with external packages, such as ForwardDiff.jl (Revels et al., 2016) or ReverseDiff.jl.

The inference engine supports different types of floating-point numbers, such as Float32, Float64, and BigFloat.

A large collection of examples is available at https://biaslab.github.io/RxInfer.jl/stable/examples/overview/.

Example usage

In this section, we show a small example based on Example 3.7 in Sarkka (Särkkä, 2013), where the goal is to track in real-time the state (angle and velocity) of a simple pendulum system. The differential equations for a simple pendulum can be written as a special case of a continuous-time nonlinear dynamic system where the hidden state \( x(t) \) is a two-dimensional vector \( \begin{bmatrix} x(1) \\ x(2) \end{bmatrix} \equiv \begin{bmatrix} \alpha \\ v \end{bmatrix} \) with \( \alpha \) and \( v \) being the angle and velocity, respectively, and the state transition function \( f(x) = \begin{bmatrix} x(1) + x(2) \Delta t \\ x(2) - g \cdot \sin(x(1)) \Delta t \end{bmatrix} \). For more detailed derivations we refer interested reader to Särkkä (2013).

We use the RxInfer’s @model macro to specify the probabilistic model. We use the @meta macro to specify an approximation method for the nonlinearity in the model, the @constraints macro to define constraints for the variational distributions in the Bethe Free Energy optimization procedure, and the @autoupdates macro to specify how to update priors about the current state of the system. Finally, we use the rxinference function to execute the inference process, see Figure 1. The inference process runs in real time and takes 162 microseconds on average per observation on a single CPU of a regular office laptop (MacBook Pro 2018, 2.6 GHz Intel Core i7).

```julia
# `g` is the gravitational constant
f(x) = [x[1] + x[2] * Δt, x[2] - g * sin(x[1]) * Δt]

# We use the `@model` macro to define the probabilistic model
@model function pendulum()

    # Define reactive inputs for the `prior`
    # of the current angle state
    prior_mean = datavar(Vector{Float64})
    prior_cov = datavar(Matrix{Float64})

    previous_state ~ MvNormal(mean = prior_mean, cov = prior_cov)

    # Use `f` as state transition function
    state ~ f(previous_state)

    # Assign a prior for the noise component
    noise_shape = datavar(Float64)
    noise_scale = datavar(Float64)
    noise ~ Gamma(shape = noise_shape, scale = noise_scale)

    # Define reactive input for the `observation`
    observation = datavar(Float64)
```

# We observe only the first component of the state
observation ~ Normal(mean = dot([1.0, 0.0], state), precision = noise)
end

@constraints function pendulum_constraint()
    # Assume the `state` and the `noise` are independent
    q(state, noise) = q(state)q(noise)
end

@meta function pendulum_meta()
    # Use the `Linearization` approximation method
    # around the nonlinear function `f`
    f() -> Linearization()
end

function pendulum_experiment(observations)
    # The `@autoupdates` structure defines how to update
    # the priors for the next observation
    autoupdates = @autoupdates begin
        prior_mean = mean(q(state))
        prior_cov = cov(q(state))
        noise_shape = shape(q(noise))
        noise_scale = scale(q(noise))
    end

    results = rxinference(
        model = pendulum(),
        constraints = pendulum_constraint(),
        meta = pendulum_meta(),
        autoupdates = autoupdates,
        data = (observation = observations,),
        initmarginals = (  
            # We assume a relatively good prior for the very first state
            state = MvNormalMeanPrecision([0.5, 0.0], [100.0 0.0; 0.0 100.0]),
            # And we assign a vague prior for the noise component
            noise = Gamma(1.0, 100.0)
        ),
        # We indicate that we want to keep a history of estimated
        # states and the noise component
        historyvars = (state = KeepLast(), noise = KeepLast()),
        keephistory = length(observations),
        iterations = 5,
        # We perform 5 VMP iterations on each observation
        autostart = true
    )

    return results
end
Figure 1: The inference results for the pendulum example. X-axis represents time $t$ (in seconds). Y-axis represents the current angle of the pendulum (in radians) at time $t$. Real (unobserved) signal is shown in blue line. Observations are shown as orange dots. The inference results are shown as green line with area, which represents posterior uncertainty (one standard deviation). The inference process runs in real time and takes 162 microseconds on average per observation on a single CPU of a regular office laptop (MacBook Pro 2018, 2.6 GHz Intel Core i7).

Acknowledgments

The authors gratefully acknowledge contributions and support from colleagues in the BIASlab in the Department of Electrical Engineering at the University of Eindhoven of Technology.

References


