DynamicPPL: Stan-like Speed for Dynamic Probabilistic Models

Mohamed Tarek, Kai Xu, Martin Trapp, Hong Ge, and Zoubin Ghahramani

1University of New South Wales at Canberra; 2Pumas-AI Inc.; 3University of Edinburgh; 4Graz University of Technology; 5University of Cambridge; 6Google Research

1. Introduction

Practical Bayesian inference for probabilistic models with dynamic dimensionality and variable types is a challenging problem in probabilistic programming languages (PPLs). General-purpose PPLs enable users to introduce new model parameters anywhere in a model definition, freely influence control flows with random variables, and often eliminate the need to specify variable types. Such flexibility often leads to a more productive probabilistic modelling workflow – models are easy to read and write. Such increased productivity is critical for a broader adoption of Bayesian modelling and approximate inference. In this poster, we present DynamicPPL, which enables a significant speedup of inference in Turing – a general-purpose PPL implemented in the Julia programming language. Our approach leverages incremental tracing and type specialisation. We show through an extensive set of benchmarks that the proposed approach significantly accelerates inference in dynamic models (often an order of magnitude faster than a previous version of Turing) and achieves state-of-the-art performance for static models that is close to and sometimes better than Stan’s (which is implemented in C++).

The main contribution of this poster is the DynamicPPL Julia package, a user-friendly and modular PPL frontend that makes use of incremental tracing and type specialization of model parameters in probabilistic programs allowing Turing to achieve speeds close to and sometimes faster than Stan even for static models while efficiently supporting a family of dynamic models.

2. Overview of TuringLang Organisation

3. Dynamic vs Static Typing

- Dynamic PPLs lack the random variables’ type information at compile time.
- Type information of variables is paramount to many compiler optimizations required to generate efficient machine code; Python is slower than C.
- The Julia programming language is a fast dynamically typed programming language; its design and complexity allow it to behave both as a dynamic programming language as well as a static one in different contexts, thus combining the advantages of both paradigms.

4. Design Principles

- User-friendly syntax
- Interoperability with the Julia ecosystem
  - Distributions.jl for distributions
  - ForwardDiff.jl, Tracker.jl, ReverseDiff.jl and Zygote.jl for automatic differentiation
  - Flux.jl for neural networks
  - Optim.jl for optimization
  - CUDA.jl for GPU data parallelism
  - DifferentialEquations.jl for differential equation solvers
  - Memoization.jl for memoising expensive functions in dynamic Gibbs sampling
  - Arbitrary Julia code allowed including custom distributions
- Performance: comparable performance to Stan

5. Trace Type Specialisation

Trace type specialisation in DynamicPPL + Turing
1. Run the model using UntypedVarInfo
2. Specialise the container types in the trace creating a TypedVarInfo
3. Run the rest of inference using a trace specialised for the model
4. (WIP) Incrementally expand the trace type as needed using a MixedVarInfo

The use of TypedVarInfo allows for optimized code generation for:
- Static models
- A class of dynamic models where the random variable containers’ names and types are static, but the number of random variables and their distributions can be dynamic.

The MixedVarInfo work will enable efficient code generation of fully dynamic models, where type information can also be dynamic, with 0 overhead for the currently efficient classes of models. New type information can get exploited incrementally to re-generate efficient code using the Julia compiler.

6. Benchmarks

- Static HMC benchmarks of Turing against Stan using AdvancedHMC.jl + ReverseDiff.jl from Turing.jl. Time is in seconds.

7. Conclusion

1. Incremental tracing and type specialization facilitates the acceleration of inference in dynamic PPLs such as Turing.
2. DynamicPPL is a modular, high-performance, dynamic PPL frontend implementation with a user-friendly and extensible interface.
3. In the future, we hope to extend DynamicPPL and Turing to support more realistic modeling, chain-structure Gibbs sampling and Infer.net-style message passing.

8. References