TURING.JL

https://github.com/yebai/Turing.jl

A fresh approach to probabilistic programming in Julia

Kai Xu JuliaCon, 22 June 2017

Hong Ge & Zoubin Ghahramani

Machine Learning Group, University of Cambridge

OUTLINE

- What is probabilistic programming?
- Why Julia?
- Key features
- How Turing.jl works?
- Live demo(s)
- Q&A

WHAT IS PROBABILISTIC PROGRAMMING?

- Probabilistic programming is general-purpose programming with intrinsic support of non-deterministic statements.
- Such programming languages are called probabilistic programming languages (PPLs).
- One of the main applications of PPLs is probabilistic modelling, a popular field of machine learning.

- Two types of PPLs
  - Standalone:
    - BUGS
    - Stan
  - Embedded-in:
    - Probabilistic C in C
    - Anglican in Clojure
    - Edward in Python
    - WebPPL in JS
    - Turing.jl in ★Julia★

$$\sigma^2 \sim \mathit{Inv\text{-}Gamma}(2,3)$$
$$\mu \sim \mathit{Normal}(0,\sigma)$$
$$x_1, x_2 \sim \mathit{Normal}(\mu,\sigma)$$

```
@model gdemo(x) = begin
  σ² ~ InverseGamma(2, 3)
  μ  ~ Normal(0, sqrt(σ²))
  x[1] ~ Normal(μ, sqrt(σ²))
  x[2] ~ Normal(μ, sqrt(σ²))
  σ², μ
end
```

Code 1: Gaussian model with conjugate priors

Looking closely, we can see the probablistic features we mentioned before in Turing.jl ...

```julia
using Turing

@model gdemo(x) = begin
  σ² ~ InverseGamma(2, 3)
  μ  ~ Normal(0, sqrt(σ²))
  x[1] ~ Normal(μ, sqrt(σ²))
  x[2] ~ Normal(μ, sqrt(σ²))
  σ², μ
end

modelf = gdemo([1.5, 2])
alg1 = PG(50, 1000)
chn1 = sample(modelf, alg)
alg2 = HMC(1000, 0.2, 3)
chn2 = sample(modelf, alg)
```

Code 2: Gaussian model with conjugate priors (inference steps breakdown)

- Non-deterministic?
  - Distributions
- Language?
  - `@model` macro
  - ~ notaion (macro)
- Machine learning?
  - *Bayesian inference* by sampling: SMC, PG, HMC, NUTS, Gibbs ...

WHY JULIA?

- Rich collections of statistical libraries
  - StatsFuns.jl, Distributions.jl, Mamba.jl
- Meta-programming
  - Turing's compiler, i.e. `@model` and `@~`
- Coroutines
  - Particle Gibbs implementation
- Automatic differentiation (AD)
  - ForwardDiff.jl provides an easy-to-use `Dual` type
  - Hamiltonian Monte Carlo implementation
  - Generic typing help AD work with distributions

KEY FEATURES

- Universal probabilistic programming with an ituitive modelling interface embedded in Julia
- Support of models with discrete variables and stochastic control flows by particle filtering
- HMC sampler for differentiable distributions
- ❗ Compositional MCMC interface
- ❗ Resumption of MCMC chains
- ☐ More novel samplers, other inference methods, sampling from user-defined models, ...

❗ = new releases, ☐ = next steps

$$\mu_k \sim Normal(0, 25), \sigma_k^2 \sim Inv\text{-}Gamma(2, 3), k = 1 \ldots K$$
$$z_i \sim Cat(1/K), x_i \sim Normal(\mu_{z_i}, \sigma_{z_i}), i = 1 \ldots N$$

```
using Turing

@model MoG(x, K) = begin
    μ = Vector{Real}(K)
    σ² = Vector{Real}(K)
    for i = 1:K
        μ[i] ~ Normal(0, 25)
        σ²[i] ~ InverseGamma(2, 3)
    end
    N = length(x); z = tzeros(Int, N)
    for i = 1:N
        z[i] ~ Categorical(1/K*ones(K))
        x[i] ~ Normal(μ[z[i]], sqrt(σ²[z[i]]))
    end
    z, σ², μ
end

modelf = MoG([1.1, 1.0, 0.9, 2.1, 2.2], 2)
alg = Gibbs(1000, PG(50, 1, :z), HMC(1, 0.2, 3, :μ, :σ²))
chn = sample(modelf, alg)
```

Code 3: Compositional MCMC interface

```julia
using Turing

@model somemodel(...) = begin
  ...
end

modelf = somemodel(...)
chn1 = sample(modelf, HMC(1000, 1, 0.2, 3; save_state=true))
chn2 = sample(modelf, NUTS(1000, 0.65; resume_from=chn1))
```

Code 3: Resumption of MCMC chains

HOW TURING.JL WORKS?

Key ML techniques

- Bayesian inference
  - General framework for probabilistic modelling
- Sampling
  - Particle filtering
  - Markov chain Monte Carlo

Key Julia techniques

- Coroutines
  - Particle based samplers
- Automatic differentiation
  - Gradient based samplers

Key system components

- Model *defined by* users
  - Normal Julia program with modelling operations
- Sampler *specified by* users
  - Need to interact with model
- `VarInfo`
  - Key data structure
  - Enable interactions between models and samplers
  - Users don't see it
- Samples *returned to* users
  - Embedded in `Chain` type

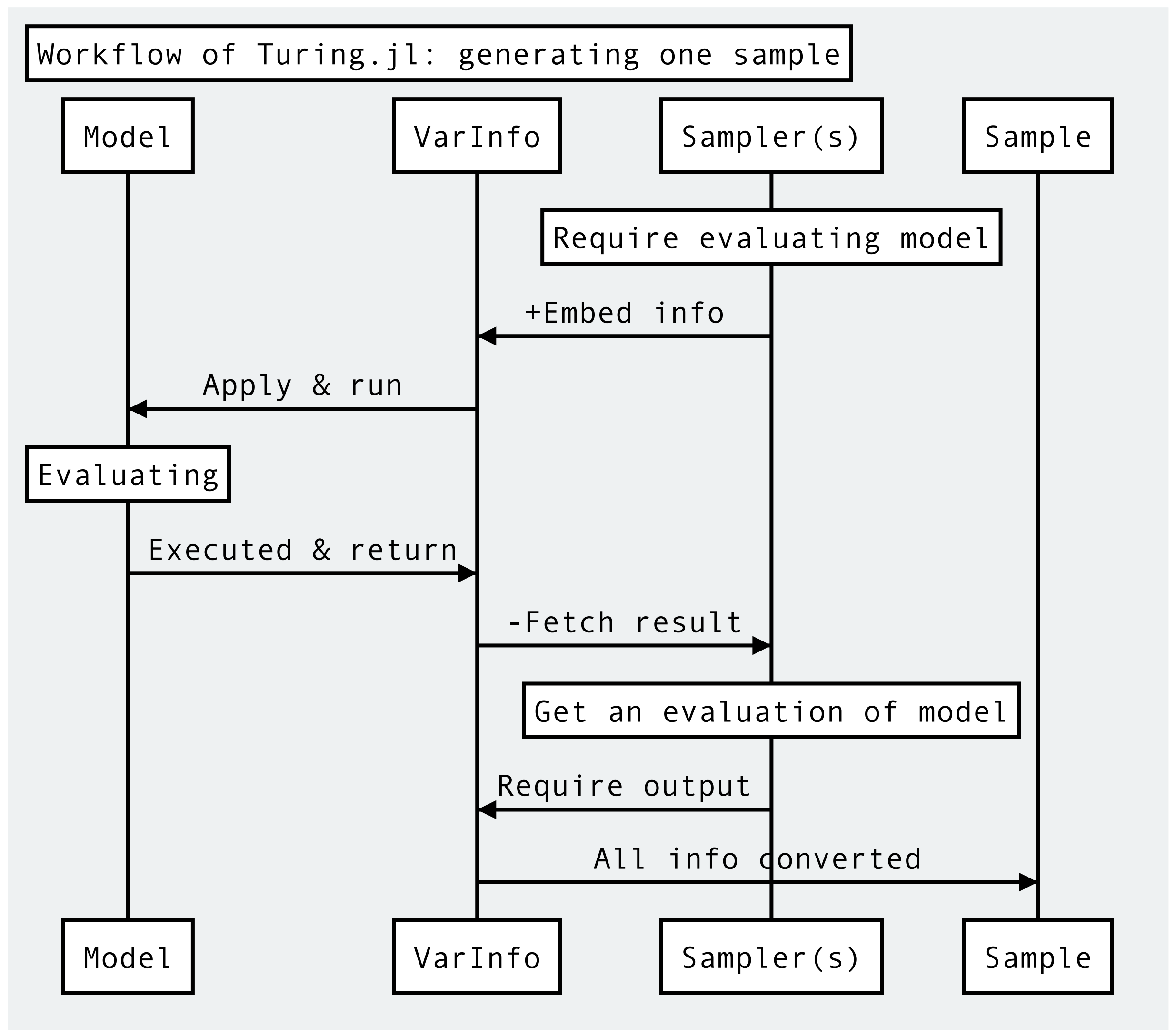

Figure 1: Workflow of Turing.jl: generating one sample

HOW PARTICLE FILTERING WORKS IN TURING.JL

- IS, SMC and PG represent each particle running in parallel using a coroutine
- Each particle is a "live" copy of the model, i.e. an ongoing execution of the model
  - Samplers need to duplicate or kill particles
  - SMC and PG do model evaluation sequentially for each observation
    - Resuming and pausing a particle is necessary
- Lead to state-of-the-art SMC and PG performance

HOW HMC WORKS IN TURING.JL

1. HMC algorithm needs the unormalized posterior of the model and the gradient of variables
2. Execution of the model program with variables set as `ForwardDiff.Dual` gives both
   - Unormalized posterior = real part of log-joint
   - Gradient of variables = dual parts of log-joint
3. Produce a sample candidate by simulating Hamiltonian dynamics with the leapfrog algorithm
4. Accept or reject the sample candidate

LIVE DEMO

Time to see a live Turing.jl program ...

Q&A

Any question?

THANK YOU FOR LISTENING 😄