Probabilistic Programming

Goals

▶ General Bayesian modeling
▶ Decouple models and inference
▶ User writes model, supplies data
▶ PPL does inference

Example: Eight Schools Model

Originally due to Rubin, 1981 (co-inventor of EM algorithm, also foundational model in causal statistical analysis)

From Bayesian Data Analysis, section 5.5 (Gelman et al. 2013):

A study was performed for the Educational Testing Service to analyze the effects of special coaching programs for SAT-V (Scholastic Aptitude Test-Verbal) in each of eight high schools. The outcome variable in each study was the score on a special administration of the SAT-V, a standardized multiple choice test administered by the Educational Testing Service and used to help colleges make admissions decisions; the scores can vary between 200 and 800, with mean about 500 and standard deviation about 100. The SAT examinations are designed to be resistant to short-term efforts directed specifically toward improving performance on the test; instead they are designed to reflect knowledge acquired and abilities developed over many years of education. Nevertheless, each of the eight schools in this study considered its short-term coaching program to be very successful at increasing SAT scores. Also, there was no prior reason to believe that any of the eight programs was more effective than any other or that some were more similar in effect to each other than to any other.
The Eight Schools Model

Models effect of coaching on SAT-V scores at 8 schools

\[
\mu \sim \mathcal{N}(0, 5) \\
\log \tau \sim \mathcal{N}(5, 1) \\
\theta_i \sim \mathcal{N}(\mu, \tau) \quad i = 1, \ldots, 8 \\
y_i \sim \mathcal{N}(\theta_i, \sigma_i) \quad i = 1, \ldots, 8
\]

Unobserved:

- \( \mu \): across-school mean effect size
- \( \tau \): across-school variance
- \( \theta_i \): school \( i \) effect

Observed:

- \( y_i \): estimated effect for school \( i \)
- \( \sigma_i \): standard error estimate for \( y_i \)

\[
y = [28, 8, -3, 7, -1, 1, 18, 12] \\
\sigma = [15, 10, 16, 11, 9, 11, 10, 18]
\]
Named for Stanislaw Ulam, co-inventor of Monte Carlo method

**JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION**

September 1949  
Volume 44

**THE MONTE CARLO METHOD**

Nicholas Metropolis and S. Ulam  
Los Alamos Laboratory

We shall present here the motivation and a general description of a method dealing with a class of problems in mathematical physics. The method is, essentially, a statistical approach to the study of differential equations, or more generally, of integro-differential equations that occur in various branches of the natural sciences.
Eight Schools in Stan

- See demo
- Browse stan examples

Example Posterior

"Unconstraining"

How Stan Works

- Model compiled into C++ black box for $\log p(z, x)$
- Autodiff to get $\nabla_z \log p(z, x)$
- Run variant of HMC (with robust hyperparameter tuning)
- Automatic variable transformations to deal with constraints (e.g., upper/lower bounds)
  - all variables transformed to unconstrained space for inference
  - uses change-of-variable formula, adds Jacobian terms to log-density
Strengths and Weaknesses

Strengths
- Flexible: full user control over joint density
- Model is compiled and optimized for efficiency, stability
- Fully automatic HMC inference
- Automatic diagnostics

Weaknesses
- Log-density only, can’t simulate from model like some other PPLs
- Difficult to compose, write modular programs
- No discrete latent variables
NumPyro

PPL where model is Python program. Uses program tracing to obtain log-density, JAX autodiff framework for gradients

NumPyro

Probabilistic programming with NumPy powered by JAX for autograd and JIT compilation to GPU/TPU/CPU.

Docs and Examples | Forum

Similar: Pyro (PyTorch instead of JAX), Edward2, TensorFlow Probability

Eight Schools in NumPyro

def eight_schools(J, sigma, y=None):
    mu = numpyro.sample('mu', dist.Normal(0, 5))
    tau = numpyro.sample('tau', dist.LogNormal(5, 1))
    with numpyro.plate('J', J):
        theta = numpyro.sample('theta', dist.Normal(mu, tau))
        y = numpyro.sample('obs', dist.Normal(theta, sigma), obs=y)

How NumPyro Works

A NumPyro model is a Python program. It can be used in two ways:

1. Run program to sample from model
2. Trace program to compute log-density (for fixed data, any values for latent variables)

Effect handlers used to determine behavior of numpyro.sample statements to switch between two modes

Other aspects (variable transformations, etc.) similar to Stan. Main goal is to get function to compute log \( p(z, x) \) and \( \nabla_z \log p(z, x) \).
**Probabilistic Programming**

**Stan**

**NumPyro**

**Broader Outlook**

---

**Strengths and Weaknesses**

**Strengths**

- Flexible user control over model
- Very fast. JIT compiled for efficiency using JAX
- Fully automatic HMC inference
- Automatic diagnostics
- Python software development practices, modularity, testing

**Weaknesses**

- Somewhat more verbose syntax
- Advanced programming framework may be challenging for some users, hard to debug
- No discrete latent variables

---

**Other PPLs**

Stan and NumPyro both tailored to HMC; currently very popular/effective. There are many other PPLs

- Pyro, TensorFlow Probability, Edward2 (similar to NumPyro, not as HMC-focused)
- BUGS, WinBUGS, JAGS: stats-community PPLs based on Gibbs sampling, very popular predecessors to Stan
- PyMC3: nice full-featured Python library; model definition similar to NumPyro; multiple inference algorithms
- Infer.Net: factor graphs, variational message passing, TrueSkill (XBox Live)
- Research PPLs: Anglican, Gen, etc. Very general models; may sacrifice fast, automatic, specialized inference algorithms.

---

**COVID-19 Forecasting With NumPyro**

- In March–May 2020 I created a COVID-19 forecasting model called “UMass-MechBayes” with NumPyro
- We submitted forecasts to the COVID-19 Forecast Hub from April 2020 to February 2022
- UMass-MechBayes was the most accurate individual model for forecasting COVID-19 deaths out of dozens of teams (PNAS, 2022). Only the ensemble model submitted to CDC was more accurate overall.
- Nothing revolutionary about the model → win for Bayesian inference, probabilistic programming
- Especially important: modular development, fast inference in NumPyro. Enabled rapid cycle of development, testing, validation, deployment at scale.