Probabilistic programming for mechanistic models using PyMC3

Sanmitra Ghosh

March 6, 2018

Computational Biology Group,
Department of Computer Science,
University of Oxford
sanmitra.ghosh@cs.ox.ac.uk

Table of contents

Bayesian inference in mechanistic models

Bayesian inference summary

A generative mechanistic model: $X \sim \mathcal{M}_\theta$, $\mathcal{M}_\theta$ being an ODE system

$$\frac{dX}{dt} = f(X, \boldsymbol{\theta}). \tag{1}$$

Having observed some data $\boldsymbol{y}$ we define a likelihood $p(\boldsymbol{y}|X, \boldsymbol{\theta})$ and combine that with a prior distribution $p(\boldsymbol{\theta})$ to obtain the posterior as:

$$p(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|X, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\boldsymbol{y}|X, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}}, \tag{2}$$

and the marginal likelihood or the evidence of model $\mathcal{M}_\theta$ as:

$$p(\boldsymbol{y}|\mathcal{M}_\theta) = \int p(\boldsymbol{y}|X, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}. \tag{3}$$

- $p(\boldsymbol{y}|\mathcal{M}_\theta)$ is intractable and thus we resort to sampling.

Bayesian inference challenges

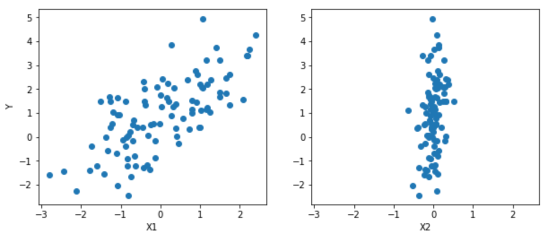
Conceptually MCMC is very simple. However, a modeller needs to spend some time in implementing and tuning MCMC to get useful results.

Implementing advanced MCMC schemes and/or estimating normalizing constants require further experience and training in computational statistics.

- Probabilistic programming libraries such as STAN [1] and PyMC3 [2] can make things considerably easier.

Probabilistic programming in PyMC3

Lets automate inference: Linear regression



The generative model (likelihood):

$$\begin{aligned} Y &= \mathcal{N}(\mu, \sigma^2) \\ \mu &= \alpha + \beta_1 X_1 + \beta_2 X_2 \end{aligned} \tag{4}$$

The priors:

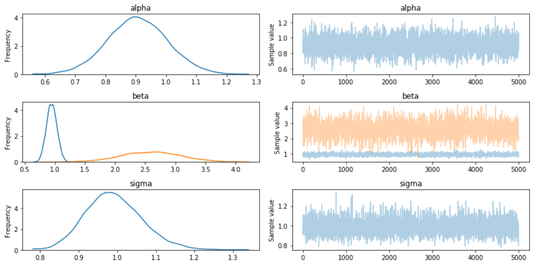
$$\begin{aligned} \alpha &= \mathcal{N}(0, 100) \\ \beta_i &= \mathcal{N}(0, 100) \\ \sigma &= |\mathcal{N}(0, 1)| \end{aligned} \tag{5}$$

Lets automate inference: PyMC3

```python
import pymc3 as pm


basic_model = pm.Model()
with basic_model:

    # Priors for unknown model parameters
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=10, shape=2)
    sigma = pm.HalfNormal('sigma', sd=1)

    # Expected value of outcome
    mu = alpha + beta[0]*X1 + beta[1]*X2

    # Likelihood (sampling distribution) of observations
    Y_obs = pm.Normal('Y_obs', mu=mu, sd=sigma, observed=Y)
```

Lets automate inference: PyMC3...

```
with basic_model:
    # draw 5000 posterior samples
    trace = pm.sample()
pm.traceplot(trace)
```

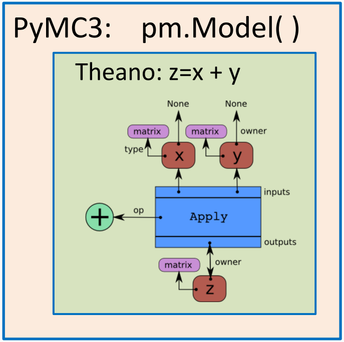


- Supported sampler: HMC [3], NUTS [4](default), Metropolis, SMC [5], Slice [6].
- PyMC3 doesn't support ODEs out-of-the-box. But there is a hack!

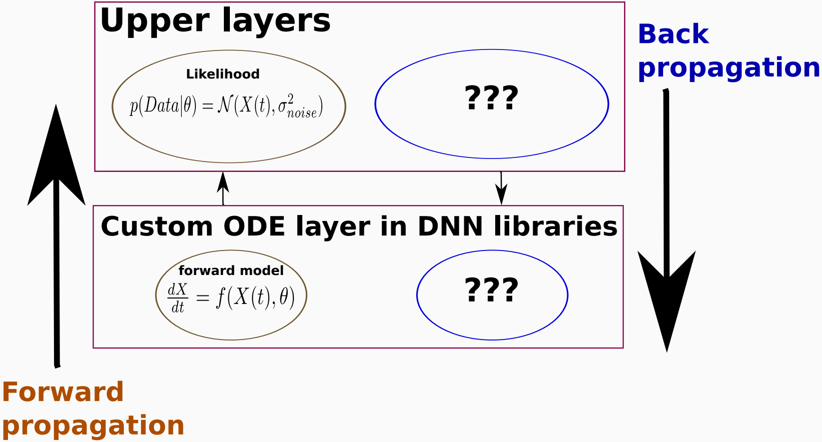
Probabilistic Models and AutoDiff

The PyMC3 under the hood

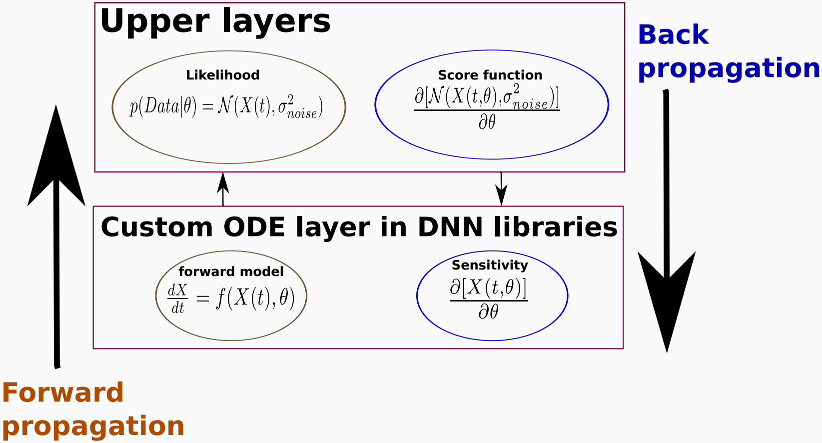
PyMC3 compiles the model into a Theano [7] program. Theano rolls out a symbolic computation graph which is compiled as a C program. Theano is used as a backend to facilitate auto-differentiation to calculate gradients of likelihood. This is required to run the Hamiltonian MC, NUTS sampler.



ODE layer in Theano, TensorFlow etc.



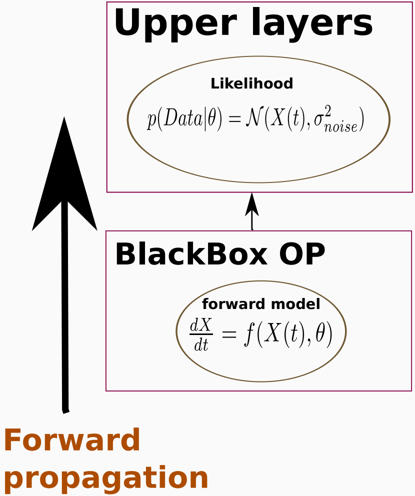
ODE layer Backprop



- Simply apply Backprop, the Deep learning trick:

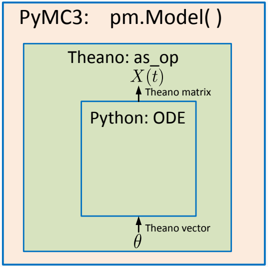
$$\frac{\partial[\mathcal{N}(X(t,\theta))]}{\partial\theta} = \frac{\partial[\mathcal{N}(X(t,\theta))]}{\partial X(t,\theta)}^T \cdot \frac{\partial X(t,\theta)}{\partial\theta} \tag{6}$$

Or use a Black-Box layer



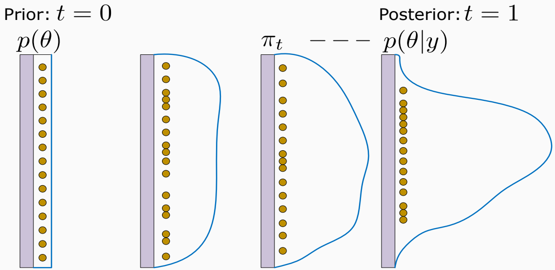
Theano custom operation

We can define a custom Theano operation as a black-box wrapper over a python ODE solver. We just need to tell Theano the input, $\boldsymbol{\theta}$ and output, $\boldsymbol{X}$ data types.



- We can't use auto-differentiation and thus NUTS sampler and Variational Inference. However, we can use Metropolis and the powerful SMC sampler.

SMC in PyMC3: Transitional Markov Chain Monte Carlo



Sample from a series of intermediate distributions to reach the target. Intermediate distributions are given by

$$\pi_t = p(\mathbf{y}|\boldsymbol{\theta})^t p(\boldsymbol{\theta}), \tag{7}$$

where $0 < t < 1$ is used to anneal the likelihood surface.

- As a by product this algorithm returns an estimate of the marginal likelihood $p(\mathbf{y}) = \int p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})$.

Variational inference

The Bayesian setup once again;

$$p(\theta|Data) = \frac{p(Data|\theta)p(\theta)}{p(Data)} \tag{8}$$

In variation inference we approximate $p(\theta|Data)$ with an approximate pdf $q(\theta|\lambda)$, say a Normal density where $q(\theta|\lambda) = \mathcal{N}(\theta|\lambda)$.

- Unlike MCMC we don't sample from $p(\theta|Data)$, we just find the parameters λ that maximise a chosen distance metric $||$ between $q(\theta|\lambda)||p(\theta|Data)$ through gradient ascent.
- Thus our expensive sampling based inference is turned into a significantly faster deterministic optimisation.

PyMC3 for ODEs

Let me walk you through a simple example in jupyter.

Github: Probabilistic Programming for Mechanistic Models

Conclusion

STAN vs PyMC3

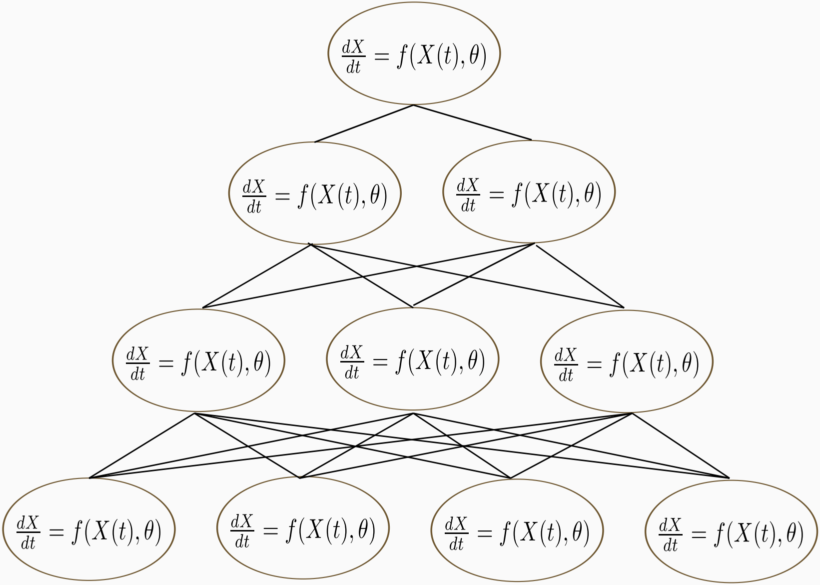
PyMC3

- User can choose any solver. Not limited to ODEs only.
- SMC can explore the posterior surface efficiently. But computational cost is much higher.
- Probably the easiest software for model selection.
- Considerably slower than STAN.

STAN

- One can only use the default ODE solver and NUTS sampler.
- For many well specified statistical models NUTS is the fastest sampler.
- Estimating marginal likelihood is difficult.
- Gradient based sampling is not the best idea for sloppy and/or unidentifiable models.
- Implemented in C++, so good for HPC applications.

Aside: Deep Dynamical Net



Thank you.

References I

- A. Gelman, D. Lee, and J. Guo, "Stan: A probabilistic programming language for bayesian inference and optimization," *Journal of Educational and Behavioral Statistics*, vol. 40, no. 5, pp. 530–543, 2015.
- A. Patil, D. Huard, and C. J. Fonnesbeck, "Pymc: Bayesian stochastic modelling in python," *Journal of statistical software*, vol. 35, no. 4, p. 1, 2010.
- S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid monte carlo," *Physics letters B*, vol. 195, no. 2, pp. 216–222, 1987.
- M. D. Hoffman and A. Gelman, "The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.

References II

- J. Ching and Y.-C. Chen, “Transitional markov chain monte carlo method for bayesian model updating, model class selection, and model averaging,” *Journal of engineering mechanics*, vol. 133, no. 7, pp. 816–832, 2007.
- R. M. Neal, “Slice sampling,” *Annals of statistics*, pp. 705–741, 2003.
- J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, “Theano: A cpu and gpu math compiler in python,” in *Proc. 9th Python in Science Conf*, 2010, pp. 1–7.