DARE
ARC Training Centre in
Data Analytics for Resources & Environments

NumPyro PPL

NumPyro is a lightweight probabilistic programming library that provides a NumPy backend for Pyro. We rely on JAX for automatic differentiation and JIT compilation to GPU / CPU.

What is a PPL?

- Provide a framework for defining probabilistic models
  - Distributions: A large range of distribtions which can be used to define priors for the parameters of your model.
  - Define a likelihood function
  - Define model form (e.g., $y = ax + b$)
- Inference algorithms
  - Algorithms for automatically inferring the posterior distribution of model parameters given some observed data
  - MCMC - NUTS, HMC, MH
  - Variational inference.

DARE
ARC Training Centre in
Data Analytics for Resources & Environments

Quick detour into Jax

- JAX is Autograd and XLA, brought together for high-performance machine learning research. It can automatically differentiate native Python and NumPy functions. It can differentiate through loops, branches, recursion, and closures, and it can take derivatives of derivatives of derivatives.
- Other PPLs rely on similar autodiff libraries to enable gradient based MCMC.
  - Tensorflow Probability <--> TensorFlow
  - Pyro <--> PyTorch
  - PyMC3 <--> Aesara (used to be Theano)
- JAX

```
import jax.numpy as jnp
# Use almost any numpy function as normal e.g.,
abc = jnp.exp(4)
```

- Using XLA you can compile your functions end-to-end with `jit`

DARE
ARC Training Centre in
Data Analytics for Resources & Environments

Example problem - rating curve

- Model rating curves as:

$$Q = aH^b$$

- H is the water level (m)
- Q is the discharge (m^3/s)
- in log space:

$$log(Q) = log\ \ a\ \ + b * log(H)$$

DARE
ARC Training Centre in
Data Analytics for Resources & Environments

The flow of a NumPyro project

- Load the data and apply any transformations
- Define the model

```
def model(input, observed=None):
    # Define priors of the model
    parameter = numpyro.sample("parameter", dist.Normal(0, 1))
    sigma = numpyro.sample("sigma", dist.HalfCauchy(1))
    # Define the model itself
    pred = input * parameter
    # Construct the log-likehood
    numpyro.sample("obs", dist.Normal(pred, sigma), obs=observed)
```

- Sample the priors to check these
- Inference - in our case we will use the NUTS sampler
- Visualisation
  - We will use arviz to store the fruits of our sampling endeavours in a netcdf
  - arviz then contains handing functions for exploring our MCMC traces.

DARE
ARC Training Centre in
Data Analytics for Resources & Environments

Important functions

- `numpyro.sample("parameter",dist.Normal(0, 1))` - sample the prior for paramaeter from $\mathcal{N}(0, 1)$
- `numpyro.deterministic("mu",mu)` - store the output of a value that is not directly sampled (e.g., model output)
- Prior predictive:
  - `Predictive(model, num_samples=100)(rng_key_, water_level=X)`
  - pass the model and the number of samples from the prior, then call with random key and model inputs
- Posterior predictive:
  - `Predictive(model, samples)(rng_key_, water_level=X)`
  - same call with samples from the MCMC sampling provided
- `hpdi(post_samples, ci)`: Computes "highest posterior density interval" (HPDI) which is the narrowest interval with probability mass `ci`. Used to show our credible interval.

DARE
ARC Training Centre in
Data Analytics for Resources & Environments

Finer points

- JAX's psuedo random number generator relies on a key generated with a seed like `rng_key_ = random.PRNGKey(1234)`.
  - We pass this key along to the MCMC sampler and our predictive sampling functions.
  - Need to be careful and read the docs if you're dealing directly with it in JAX