High Dimensional Bayesian ~~Workflow~~ in Pyro Pipelines

Fritz Obermeyer @ Broad Institute

Build →

1. Clean the data
2. Create a generative model
3. Create an initialization heuristic
4. Sanity check via MAP or mean field VI
5. Reparametrize the model
6. Customize the variational family
7. Sample via reparameterized HMC

← Validate & Backtrack

Why do we build?

- 7. Sample via reparameterized HMC — high quality samples, but expensive
- 6. Customize the variational family — pyro's autoguides support preconditioning/NeuTra; account for correlations, multimodality
- 5. Reparametrize the model — improves geometry, helps with convergence; e.g. decentering, Haar wavelet, auxiliary vars
- 4. Sanity check via MAP or mean field VI — warning: MAP is incompatible with reparameterization
- 3. Create an initialization heuristic — fast & robust, but MAP doesn't work with reparam; needed to avoid NANs
- 2. Create a generative model — speeds up model iteration
- 1. Clean the data — PPLs make it easy to change the model

What does validation mean in high-dimensional pyro models?

- type errors
- shape errors
- NANs
- OOM, memory footprint
- slow ops
- parameter divergence
- loss of numerical precision
- slow convergence
- bugs in custom components
- pipeline rot
- corrupt data
- statistical issues

} software workflow

} Bayesian workflow (see Bob Carpenter's poster)

What could go wrong?

7. Sample via reparameterized HMC
6. Customize the variational family
5. Reparametrize the model
4. Sanity check via MAP or mean field VI
3. Create an initialization heuristic
2. Create a generative model
1. Clean the data

After feedback from scientists, you decide to **add a new latent variable** to the model…

Problem: when the model changes everything above it must change :(

Solutions: HMC, MAP, and mean field VI are model agnostic :) pyro's autoguides, reparameterizers, and initializers provide model-adaptive strategies. You may need one-line changes to initialization & custom variational families.

What could go wrong?

7. Sample via reparameterized HMC
6. Customize the variational family
5. Reparametrize the model
4. Sanity check via MAP or mean field VI
3. Create an initialization heuristic
2. Create a generative model
1. Clean the data

While validating your posterior distribution, you decide to **change coordinate systems** using a pyro.reparam effect…

Problem: Changing coordinates breaks your previous work creating an initialization heuristic: the old init values are in the old coordinate system, triggering more work :(

Solution: Each of Pyro's reparameterizers handles transformation of init values into the new space :) This is a lot of code, but improves user experience.

What could go wrong?

7. Sample via reparameterized HMC
6. Customize the variational family
5. Reparametrize the model
4. Sanity check via MAP or mean field VI
3. Create an initialization heuristic
2. Create a generative model
1. Clean the data

To account for batch effects in recently updated data, you **make one latent variable more local**, moving it inside a pyro.plate…

Problem: your pyro autoguide increases memory use from 10MB to 40GB, no longer fitting on a GPU :(

Solution: autoguides are composable, so you can manually split the model and use a separate autoguide on each half, dropping a posterior dependency :)

How can things go right?

7. Sample via reparameterized HMC
6. Customize the variational family
5. Reparametrize the model
4. Sanity check via MAP or mean field VI
3. Create an initialization heuristic
2. Create a generative model
1. Clean the data

You got to the last step, and run statistical tests...

Problem: the model looks awful :(

Solution: Automatically search through model architectures :) pyro makes it easy to create adaptive strategies for autoguides, reparameterizers, and init strategies. Use these strategies and fast VI to search through a wide class of models.