Automatic Guide Generation for Stan via NumPyro
Guillaume Baudart, Louis Mandel

To cite this version:
Guillaume Baudart, Louis Mandel. Automatic Guide Generation for Stan via NumPyro. PROBPROG 2021 - Third International Conference on Probabilistic Programming, Oct 2021, Virtual, United States. hal-03401708

HAL Id: hal-03401708
https://hal.archives-ouvertes.fr/hal-03401708
Submitted on 25 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Automatic Guide Generation for Stan via NumPyro

GUILLAUME BAUDART, Inria Paris, École normale supérieure – PSL University
LOUIS MANDEL, MIT-IBM Watson AI Lab, IBM Research

Stan is a very popular probabilistic language with a state-of-the-art HMC sampler but it only offers a limited choice of algorithms for black-box variational inference. In this paper, we show that using our recently proposed compiler from Stan to Pyro, Stan users can easily try the set of algorithms implemented in Pyro for black-box variational inference. We evaluate our approach on PosteriorDB, a database of Stan models with corresponding data and reference posterior samples. Results show that the eight algorithms available in Pyro offer a range of possible compromises between complexity and accuracy. This paper illustrates that compiling Stan to another probabilistic language can be used to leverage new features for Stan users, and give access to a large set of examples for language developers who implement these new features.

1 MOTIVATION

The Stan probabilistic language [Carpenter et al. 2017] enjoys broad adoption by the statistics and social sciences communities [Carlin and Louis 2008; Gelman and Hill 2006; Gelman et al. 2013]. A Stan program defines a function from data and parameters to the value of a special variable target that represents the log-density of the model. Given the observed data, the posterior distribution of the parameters can then be inferred using specialized inference algorithms, e.g., NUTS [Hoffman and Gelman 2014] (No U-Turn Sampler), an optimized Hamiltonian Monte Carlo (HMC) algorithm that is the preferred inference method for Stan.

Pyro [Bingham et al. 2019] and its JAX-based counterpart NumPyro [Phan et al. 2019] on the other hand are generative probabilistic languages. They describe generative models, i.e., stochastic procedures that simulate the data generation process. Generative PPLs are increasingly used in machine-learning research and are rapidly incorporating new ideas, such as Stochastic Gradient Variational Inference (VI), or Bayesian neural networks in what is now called Deep Probabilistic Programming [Baudart et al. 2018; Bingham et al. 2019; Tran et al. 2017].

Variational Inference tries to find the member \( q_{\theta^*}(z) \) of a family \( Q = \{ q_{\theta}(z) \}_{\theta \in \Theta} \) of simpler distributions that is the closest to the true posterior \( p(z \mid x) \) [Blei et al. 2016]. Members of the family \( Q \) are characterized by the values of the variational parameters \( \theta \). The fitness of a candidate is measured using the Kullback-Leibler (KL) divergence from the true posterior, which VI aims to minimize:

\[
q_{\theta^*}(z) = \arg\min_{\theta \in \Theta} \text{KL} \left( q_{\theta}(z) \parallel p(z \mid x) \right).
\]

Pyro natively supports variational inference and lets users define the family \( Q \) (the variational guide) alongside the model. However, manually defining a correct and efficient guide is complex and error prone [Lee et al. 2020]. Stan, on the other hand, offers ADVI [Kucukelbir et al. 2017], an implementation of black-box VI where guides are automatically synthesized from the model using a mean-field or a full-rank approximation. But, ADVI is only efficient for a subclass of models.

As an intermediate solution, Pyro developers recently introduced a zoo of autoguides: variational guides that are automatically synthesized from the model using different heuristics [Webb et al. 2019]. Users can now try a range of synthesized guides on a given model before attempting to craft their own.

Authors’ addresses: Guillaume Baudart, Inria Paris, École normale supérieure – PSL University; Louis Mandel, MIT-IBM Watson AI Lab, IBM Research.
parameters {
    real cluster; real theta; }

model {
    real mu;
    cluster ~ normal(0, 1);
    if (cluster > 0) mu = 20;
    else mu = 0;
    theta ~ normal(mu, 1);
}

We recently proposed new backends for the Stanc3 Compiler to Pyro and NumPyro\(^1\) and showed how to extend Stan with support for explicit variational guides [Baudart et al. 2021]. In this paper, we show that our compiler and runtime can be used to test NumPyro autoguides on Stan models, and evaluate our approach on PosteriorDB a database of Stan models with corresponding data, and reference posterior samples [Vehtari and Magnusson 2020].

2 EXAMPLE

The multimodal example shown in Figure 1 is a mixture of two Gaussian distributions with different means but identical variances. This example is particularly challenging for NUTS. Using multiple chains, NUTS finds the two modes, but the chains do not mix and the relative densities are incorrect. This is a known limitation of HMC.\(^2\) This model is also challenging for Stan’s ADVI since the synthesized guide cannot approximate multi-modal distribution as illustrated in the histograms of Figure 1 (by default, the runtime detects that ADVI does not converge and throws an exception).

Using our compiler from Stan to NumPyro we can try Pyro autoguides on the same example. The following code illustrates our runtime.

```python
from stannumpyro import NumPyroModel
import numpyro.infer.autoguide as autoguide
from numpyro.infer import Trace_ELBO
from numpyro.optim import Adam
from jax.random import PRNGKey

numpyro_model = NumPyroModel("multimodal.stan")
guide = autoguide.AutoBNAFNormal(numpyro_model.get_model())
svi = numpyro_model.svi(Adam(step_size=0.0005), Trace_ELBO(), guide)
svi.run(PRNGKey(0), {}, num_steps=10000, num_samples=10000)
```

The file multimodal.stan contains the Stan program of Figure 1. From this file, we create a NumPyroModel object which compiles the model and loads the NumPyro code (line 7). We then synthesize a guide for the model using the AutoBNAFNormal heuristic (line 8). Following NumPyro’s API, to launch the inference, we first create a SVI object from an optimizer (Adam), a loss function (Trace_ELBO), and the guide (line 9). Given a random seed PRNGKEY(0), and the input data (here an empty dictionary since the model has no `data` block), the run method first computes `num_steps` optimization steps and then draws `num_samples` samples from the posterior distribution (line 10).

\(^1\)https://github.com/deepppl/stanc3
\(^2\)https://mc-stan.org/users/documentation/case-studies/identifying_mixture_models.html
NumPyro offers eight different heuristics to synthesize a guide from the model:

- AutoBNAFNormal (Block Neural Autoregressive Flow [Cao et al. 2019]),
- AutoDiagonalNormal (Stan’s mean-field),
- AutoMultivariateNormal (Stan’s full-rank),
- AutoIAFNormal (Inverse Autoregressive Flow [Kingma et al. 2016]),
- AutoLaplaceApproximation (quadratic approximation),
- AutoLowRankMultivariateNormal [Ong et al. 2017],
- AutoNormal (similar to AutoDiagonalNormal), and
- AutoDelta (MAP estimates).

Figure 2 clearly shows that these heuristics yield different results on the example of Figure 1. Only two of them successfully identify the two modes.

3 EVALUATION

PosteriorDB [Vehtari and Magnusson 2020] provides reference samples for 49 pairs (model, dataset). Due to missing functions in our implementation of the standard library (e.g., ODE solvers), we cannot test six models. For each of the remaining 43 models, we run 100,000 inference steps (using Adam(step_size=0.0005)) and generate 10,000 samples from the posterior distribution.

To evaluate inference accuracy, for each parameter $x$ in the posterior distribution we compute the relative error w.r.t. to the reference samples. For multidimensional parameters we compute the error for every component.

$$
err = \frac{|\text{mean}(x_{\text{ref}}) - \text{mean}(x)|}{\text{stddev}(x_{\text{ref}})}
$$
The evaluation code is available at https://github.com/deepppl/evaluation-autoguide. Table 1 summarizes the results. For each model, we report the maximal relative error across parameters. Relative errors below 0.3 — the criteria used by regression tests for Stan⁴ — appear in green. On this set of benchmarks, the autoguides relying on normalizing flows AutoBNAFNormal and AutoIAFNormal outperform other autoguides. AutoDelta can be used with all models without raising runtime errors, but it only computes a MAP estimate. AutoLaplaceApproximation also runs without error on all models but returns less accurate distributions based on a MAP estimations. Runtime errors correspond to NaN values. Overall, the autoguides demonstrate a range of possible compromises between complexity and accuracy.

For comparison, we also report the results of Stan ADVI with the mean-field and full-rank algorithms. Compared to NumPyro, instead of running a fixed number of iterations, the Stan runtime relies on an adaptive optimization sequence and throws an exception when the algorithm fails to converge.

4 CONCLUSION

In this paper, we showed that by compiling Stan models to NumPyro, Stan users now have access to a large variety of automatically synthesized guides for variational inference. As illustrated on 43 benchmarks from PosteriorDB, these new guides offer new compromises between complexity and accuracy compared to Stan ADVI full-rank and mean-field algorithms.⁵

This paper illustrates that compiling Stan to another probabilistic language can be used to leverage new features for Stan users, and give access to a large set of examples for language developers who implement these new features.

ACKNOWLEDGMENTS

We would like to thank Eli Bingham, Fritz Obermeyer, and Du Phan for suggesting us this application of the Stan to NumPyro compiler.

REFERENCES


⁴https://github.com/stan-dev/performance-tests-cmdstan
⁵https://discourse.mc-stan.org/t/intermediate-between-mean-field-and-full-rank-advi


