Automatic Guide Generation for Stan via NumPyro

GUILLAUME BAUDART, Inria Paris, École normale supérieure – PSL University
LOUIS MANDEL, 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 $p(z \mid \mathbf{x})$ [Blei et al. 2016]. Members of the family Q are characterized by the values of the *variational parameters* θ. The fitness of a candidate is measured using the Kullback-Leibler (KL) divergence from the true posterior, which VI aims to minimize:

$$q_{\theta^*}(z) = \operatorname*{argmin}_{\theta \in \Theta} \mathrm{KL}\Big(q_\theta(z) \parallel p(z \mid \mathbf{x})\Big).$$

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. 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, 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); }
```

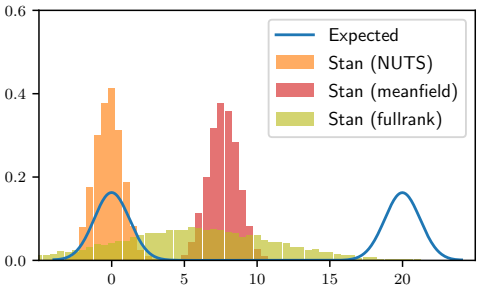

Fig. 1. Multimodal example in Stan, and graph of expected posterior distribution and histograms of the inferred posterior distributions using Stan NUTS and Stan ADVI with the mean-field and full-rank algorithms.

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 last histogram 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 illustrate our runtime.

```
1  from stannumpyro import NumPyroModel
2  import numpyro.infer.autoguide as autoguide
3  from numpyro.infer import Trace_ELBO
4  from numpyro.optim import Adam
5  from jax.random import PRNGKey
6
7  numpyro_model = NumPyroModel("multimodal.stan")
8  guide = autoguide.AutoBNAFNormal(numpyro_model.get_model())
9  svi = numpyro_model.svi(Adam(step_size=0.0005), Trace_ELBO(), guide)
10 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

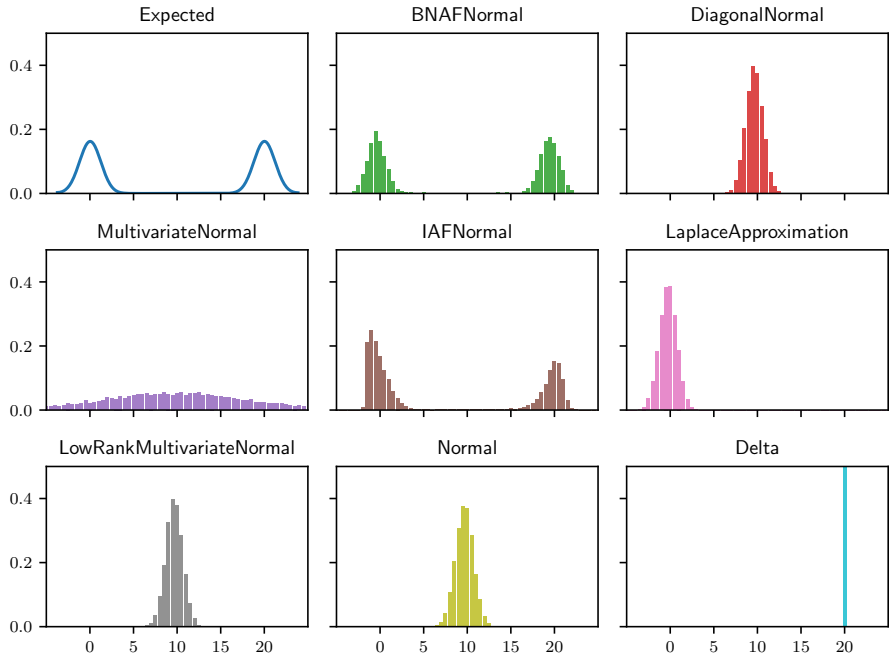

Fig. 2. Inference results using NumPyro autoguides on the model of Figure 1 after 100,000 inference steps with 10,000 samples.

NumPyro offers eight different heuristics to synthesize a guide from the model:[3] `AutoBNAFNormal`, `AutoDiagonalNormal`, `AutoMultivariateNormal`, `AutoIAFNormal`, `AutoLaplaceApproximation`, `AutoLowRankMultivariateNormal`, `AutoNormal`, and `AutoDelta`. 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 Stan[4] — appear

[3] http://num.pyro.ai/en/stable/autoguide.html
[4] https://github.com/stan-dev/performance-tests-cmdstan

Table 1. Relative errors w.r.t. the reference samples of PosteriorDB. We report the maximum relative error across parameters in green if it is below 0.3, a ✗ indicates a runtime error.

Dataset - Model	BNAFNormal	Delta	DiagonalNormal	IAFNormal	LaplaceApproximation	LowRankMultivariateNormal	MultivariateNormal	Normal	Stan MeanField	Stan FullRank
arK-arK	0.26	0.44	0.09	0.22	0.38	0.06	0.13	0.17	✗	0.33
arma-arma11	✗	0.26	✗	✗	0.16	✗	✗	✗	✗	✗
sblri-blr	1.96	0.50	0.70	2.46	0.38	0.64	0.76	0.64	✗	✗
sblrc-blr	1.05	0.51	0.18	1.52	0.40	0.55	1.58	0.27	✗	✗
dogs-dogs	0.10	0.12	0.13	0.16	0.11	0.05	0.08	0.13	0.38	0.80
dogs-dogs_log	✗	0.12	0.07	✗	0.12	0.03	0.05	0.07	0.93	✗
earnings-earn_height	0.30	24.65	23.88	✗	24.26	24.21	24.08	24.55	✗	18.71
eight_schools-eight_schools_centered	0.13	1.13	0.65	0.07	1.13	0.52	0.60	0.62	1.17	1.39
eight_schools-eight_schools_noncentered	0.07	1.13	0.22	0.07	13.03	0.17	0.18	0.26	0.22	0.21
garch-garch11	2.68	1.15	2.68	2.71	0.74	1.80	1.68	2.69	0.81	0.34
bball_drive_event_0-hmm_drive_0	0.28	290.68	0.29	✗	0.42	0.43	0.27	0.10	0.48	14.31
bball_drive_event_1-hmm_drive_1	0.53	0.59	0.52	0.77	768.80	0.52	0.63	0.48	9.68	4.59
hmm_example-hmm_example	0.16	0.37	0.46	✗	0.25	0.05	0.30	0.36	1.73	0.55
kidiq-kidscore_interaction	0.62	0.63	2.47	0.82	0.49	2.46	2.21	2.47	2.61	✗
kidiq_with_mom_work-kidscore_interaction_c	0.73	48.31	46.66	0.23	46.42	48.03	49.68	46.71	44.70	56.46
kidiq_with_mom_work-kidscore_interaction_c2	0.67	35.30	34.10	0.33	33.91	34.35	35.09	34.13	✗	93.98
kidiq_with_mom_work-kidscore_interaction_z	0.70	48.03	46.41	0.44	46.15	48.49	49.75	46.45	2.87	48.65
kidiq_with_mom_work-kidscore_mom_work	0.65	16.47	15.94	0.17	15.79	16.05	16.04	15.95	1.94	7.23
kidiq-kidscore_momhs	0.19	16.61	15.84	✗	15.66	15.96	16.02	15.83	6.76	2.18
kidiq-kidscore_momhsiq	0.27	0.17	2.44	0.25	0.12	2.28	1.82	2.44	✗	4.07
kidiq-kidscore_momiq	0.10	0.15	2.20	✗	0.12	2.13	1.75	2.21	3.28	3.65
kilpisjarvi_mod-kilpisjarvi	2.12	1.99	1.97	✗	1.95	1.98	7.00	1.98	✗	✗
earnings-log10earn_height	0.12	0.09	0.17	✗	0.08	0.15	0.56	0.19	✗	✗
earnings-logearn_height	0.13	0.09	1.08	✗	0.08	1.11	1.21	1.08	✗	✗
earnings-logearn_height_male	0.25	0.11	4.22	0.16	0.08	4.06	3.74	4.21	1.33	10.73
earnings-logearn_interaction	0.56	0.12	5.52	0.81	0.11	5.55	5.99	5.51	✗	✗
earnings-logearn_interaction_z	0.67	0.12	0.27	0.42	0.10	0.10	0.27	0.15	1.90	1.28
earnings-logearn_logheight_male	0.45	0.12	1.22	0.54	0.09	1.21	1.17	1.22	0.66	✗
mesquite-logmesquite	0.15	0.93	0.03	0.22	0.80	0.05	0.06	0.05	✗	0.11
mesquite-logmesquite_logva	0.34	0.65	0.06	0.37	0.52	0.05	0.05	0.05	0.14	0.05
mesquite-logmesquite_logvas	0.14	0.93	0.13	0.27	0.80	0.11	0.06	0.13	✗	0.09
mesquite-logmesquite_logvash	0.22	0.84	0.08	0.26	0.72	0.06	0.08	0.05	✗	✗
mesquite-logmesquite_logvolume	0.32	0.45	0.13	✗	0.33	0.11	0.13	0.04	0.22	0.08
low_dim_gauss_mix-low_dim_gauss_mix	0.68	0.13	0.22	0.75	0.08	0.21	0.23	0.27	63.08	62.71
mesquite-mesquite	0.16	8.12	11.02	6.39	7.89	11.80	11.25	10.94	23.32	17.18
nes1988-nes	0.18	0.27	0.06	0.33	0.24	0.33	0.06	0.11	0.88	2.09
nes1972-nes	0.19	0.21	0.05	0.33	0.18	0.38	0.05	0.15	0.27	1.30
nes1984-nes	0.16	0.23	0.04	0.30	0.20	0.36	0.04	0.14	1.25	1.31
nes1980-nes	0.14	0.29	0.03	0.19	0.25	0.31	0.03	0.13	1.05	0.74
nes2000-nes	0.12	0.37	0.04	0.19	0.32	0.25	0.03	0.12	1.01	1.28
nes1976-nes	0.19	0.24	0.05	0.32	0.21	0.35	0.06	0.13	2.61	1.89
nes1992-nes	0.18	0.23	0.05	0.27	0.21	0.35	0.05	0.13	2.19	0.85
nes1996-nes	0.17	0.25	0.04	0.25	0.21	0.34	0.05	0.14	1.29	2.54
Average	0.46	11.72	5.30	0.71	22.89	5.43	5.59	5.32	6.16	11.30
Successes	25	21	22	16	20	14	21	22	4	5
Mismatches	16	22	20	16	23	28	21	20	25	27
Errors	2	0	1	11	0	1	1	1	14	11

in green. On this set of benchmarks, `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. 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, the Stan runtime relies on an adaptive stepsize 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

Guillaume Baudart, Javier Burroni, Martin Hirzel, Louis Mandel, and Avraham Shinnar. 2021. Compiling Stan to Generative Probabilistic Languages and Extension to Deep Probabilistic Programming. In *PLDI*. ACM.

Guillaume Baudart, Martin Hirzel, and Louis Mandel. 2018. Deep Probabilistic Programming Languages: A Qualitative Study. *CoRR* abs/1804.06458 (2018).

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *J. Mach. Learn. Res.* 20 (2019), 28:1–28:6.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. 2016. Variational Inference: A Review for Statisticians. *CoRR* abs/1601.00670 (2016).

Bradley P Carlin and Thomas A Louis. 2008. *Bayesian methods for data analysis*. CRC Press.

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A probabilistic programming language. *Journal of Statistical Software* 76, 1 (2017), 1–37. https://doi.org/10.18637/jss.v076.i01

Andrew Gelman and Jennifer Hill. 2006. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press. https://doi.org/10.1017/CBO9780511790942

Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. 2013. *Bayesian data analysis*. Chapman and Hall/CRC.

Matthew D. Hoffman and Andrew Gelman. 2014. The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.* 15, 1 (2014), 1593–1623.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. 2017. Automatic Differentiation Variational Inference. *J. Mach. Learn. Res.* 18 (2017), 14:1–14:45.

Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2020. Towards verified stochastic variational inference for probabilistic programs. *PACMPL* 4, POPL (2020), 16:1–16:33. https://doi.org/10.1145/3371084

Du Phan, Neeraj Pradhan, and Martin Jankowiak. 2019. Composable Effects for Flexible and Accelerated Probabilistic Programming in NumPyro. *CoRR* abs/1912.11554 (2019).

Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. 2017. Deep Probabilistic Programming. In *ICLR (Poster)*.

Aki Vehtari and Måns Magnusson. 2020. PosteriorDB: a database with data, models and posteriors. In *Stan Conf.* https://github.com/stan-dev/posteriordb

[5] https://discourse.mc-stan.org/t/intermediate-between-mean-field-and-full-rank-advi