sbi: A toolkit for simulation-based inference

Alvaro Tejero-Cantero[e, 1]**, Jan Boelts**[e, 1]**, Michael Deistler**[e, 1]**,**
Jan-Matthis Lueckmann[e, 1]**, Conor Durkan**[e, 2]**, Pedro J. Gonçalves**[1, 3]**,**
David S. Greenberg[1, 4]**, and Jakob H. Macke**[1, 5, 6]

e Equally contributing authors **1** Computational Neuroengineering, Department of Electrical and Computer Engineering, Technical University of Munich **2** School of Informatics, University of Edinburgh **3** Neural Systems Analysis, Center of Advanced European Studies and Research (caesar), Bonn **4** Model-Driven Machine Learning, Centre for Materials and Coastal Research, Helmholtz-Zentrum Geesthacht **5** Machine Learning in Science, University of Tübingen **6** Empirical Inference, Max Planck Institute for Intelligent Systems, Tübingen

Summary

Scientists and engineers employ stochastic numerical simulators to model empirically observed phenomena. In contrast to purely statistical models, simulators express scientific principles that provide powerful inductive biases, improve generalization to new data or scenarios and allow for fewer, more interpretable and domain-relevant parameters. Despite these advantages, tuning a simulator's parameters so that its outputs match data is challenging. Simulation-based inference (SBI) seeks to identify parameter sets that a) are compatible with prior knowledge and b) match empirical observations. Importantly, SBI does not seek to recover a single 'best' data-compatible parameter set, but rather to identify all high probability regions of parameter space that explain observed data, and thereby to quantify parameter uncertainty. In Bayesian terminology, SBI aims to retrieve the posterior distribution over the parameters of interest. In contrast to conventional Bayesian inference, SBI is also applicable when one can run model simulations, but no formula or algorithm exists for evaluating the probability of data given parameters, i.e. the likelihood.

We present `sbi`, a PyTorch-based package that implements SBI algorithms based on neural networks. `sbi` facilitates inference on black-box simulators for practising scientists and engineers by providing a unified interface to state-of-the-art algorithms together with documentation and tutorials.

Motivation

Bayesian inference is a principled approach for determining parameters consistent with empirical observations: Given a prior over parameters, a stochastic simulator, and observations, it returns a posterior distribution. In cases where the simulator likelihood *can* be evaluated, many methods for approximate Bayesian inference exist (e.g., Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953; Baydin et al., 2019; Graham & Storkey, 2017; Le, Baydin, & Wood, 2017; Neal, 2003). For more general simulators, however, evaluating the likelihood of data given parameters might be computationally intractable. Traditional algorithms for this 'likelihood-free' setting (Cranmer, Brehmer, & Louppe, 2020) are based on Monte-Carlo rejection (Pritchard, Seielstad, Perez-Lezaun, & Feldman, 1999; Sisson, Fan, & Tanaka, 2007), an approach known as *Approximate Bayesian Computation* (ABC). More recently, algorithms based on neural networks have been developed (Greenberg, Nonnenmacher, & Macke, 2019; Hermans, Begy, & Louppe, 2020; Lueckmann et al., 2017; Papamakarios & Murray, 2016;

Papamakarios, Sterratt, & Murray, 2019). These algorithms are not based on rejecting simulations, but rather train deep neural conditional density estimators or classifiers on simulated data. To aid in effective application of these algorithms to a wide range of problems, `sbi` closely integrates with PyTorch and offers state-of-the-art neural network-based SBI algorithms (Greenberg et al., 2019; Hermans et al., 2020; Papamakarios et al., 2019) with flexible choice of network architectures and flow-based density estimators. With `sbi`, researchers can easily implement new neural inference algorithms, benefiting from the infrastructure to manage simulators and a unified posterior representation. Users, in turn, can profit from a single inference interface that allows them to either use their own custom neural network, or choose from a growing library of preconfigured options provided with the package.

Related software and use in research

We are aware of several mature packages that implement SBI algorithms. `elfi` (Lintusaari et al., 2018) is a package offering BOLFI, a Gaussian process-based algorithm (Gutmann & Corander, 2016), and some classical ABC algorithms. The package `carl` (Louppe, Cranmer, & Pavez, 2016) implements the algorithm described in Cranmer, Pavez, & Louppe (2015). Two other SBI packages, currently under development, are `hypothesis` (Hermans, 2019) and `pydelfi` (Alsing, 2019). `pyabc` (Klinger, Rickert, & Hasenauer, 2018) and `ABCpy` (Dutta, Schoengens, Onnela, & Mira, 2017) are two packages offering a diversity of ABC algorithms.

`sbi` is closely integrated with PyTorch (Paszke et al., 2019) and uses `nflows` (Durkan, Bekasov, Papamakarios, & Murray, 2019) for flow-based density estimators. `sbi` builds on experience accumulated developing `delfi` (mackelab.org, 2017), which it succeeds. `delfi` was based on `theano` (Al-Rfou et al., 2016) (development discontinued) and developed both for SBI research (Greenberg et al., 2019; Lueckmann et al., 2017) and for scientific applications (Gonçalves et al., 2019). The `sbi` codebase started as a fork of `lfi` (Durkan, 2020), developed for Durkan et al. (2020).

Description

`sbi` currently implements three families of neural inference algorithms:

- Sequential Neural *Posterior* Estimation (SNPE) trains a deep neural density estimator that directly estimates the posterior distribution of parameters given data. Afterwards, it can sample parameter sets from the posterior, or evaluate the posterior density on any parameter set. Currently, SNPE-C (Greenberg et al., 2019) is implemented in `sbi`.

- Sequential Neural *Likelihood* Estimation (SNLE) (Papamakarios et al., 2019) trains a deep neural density estimator of the likelihood, which then allows to sample from the posterior using e.g. MCMC.

- Sequential Neural *Ratio* Estimation (SNRE) (Durkan et al., 2020; Hermans et al., 2020) trains a classifier to estimate density ratios, which in turn can be used to sample from the posterior e.g. with MCMC.

The inference step returns a `NeuralPosterior` object that represents the uncertainty about the parameters conditional on an observation, i.e. the posterior distribution. This object can be sampled from —and if the chosen algorithm allows, evaluated— with the same API as a standard PyTorch probability distribution.

An important challenge in making SBI algorithms usable by a broader community is to deal with diverse, often pre-existing, complex simulators. `sbi` works with any simulator as long as it can be wrapped in a Python callable. Furthermore, `sbi` ensures that custom simulators

work well with neural networks, e.g. by performing automatic shape inference, standardizing inputs or handling failed simulations. To maximize simulator performance, `sbi` leverages vectorization where available and optionally parallelizes simulations using `joblib` (Varoquaux, 2008). Moreover, if dimensionality reduction of the simulator output is desired, `sbi` can use a trainable summarizing network to extract relevant features from raw simulator output and spare the user manual feature engineering.

In addition to the full-featured interface, `sbi` provides also a *simple* interface which consists of a single function call with reasonable defaults. This allows new users to get familiarized with simulation-based inference and quickly obtain results without having to define custom networks or tune hyperparameters.

With `sbi`, we aim to support scientific discovery and computational engineering by making Bayesian inference applicable to the widest class of models (simulators with no likelihood available), and practical for complex problems. We have designed an open architecture and adopted community-oriented development practices in order to invite other machine-learning researchers to join us in this long-term vision.

Acknowledgements

References

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., et al. (2016). Theano: A python framework for fast computation of mathematical expressions. *arXiv*, arXiv–1605.

Alsing, J. (2019). pydelfi: Density estimation likelihood-free inference. *GitHub repository*. https://github.com/justinalsing/pydelfi; GitHub.

Baydin, A. G., Shao, L., Bhimji, W., Heinrich, L., Meadows, L., Liu, J., Munk, A., et al. (2019). Etalumis: Bringing probabilistic programming to scientific simulators at scale. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1–24). doi:10.1145/3295500.3356180

Cranmer, K., Brehmer, J., & Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*. doi:10.1073/pnas.1912789117

Cranmer, K., Pavez, J., & Louppe, G. (2015). Approximating likelihood ratios with calibrated discriminative classifiers. *arXiv preprint arXiv:1506.02169*.

Durkan, C. (2020). Lfi. *GitHub repository*. https://github.com/conormdurkan/lfi; GitHub.

Durkan, C., Bekasov, A., Papamakarios, G., & Murray, I. (2019). nflows: Normalizing flows in PyTorch. *GitHub repository*. https://github.com/bayesiains/nflows; GitHub.

Durkan, C., Murray, I., & Papamakarios, G. (2020). On contrastive learning for likelihood-free inference. *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of machine learning research, *98*.

Dutta, R., Schoengens, M., Onnela, J.-P., & Mira, A. (2017). ABCpy: A user-friendly, extensible, and parallel library for approximate bayesian computation. In *Proceedings of the platform for advanced scientific computing conference* (pp. 8:1–8:9). doi:10.1145/3093172.3093233

Gonçalves, P. J., Lueckmann, J.-M., Deistler, M., Nonnenmacher, M., Öcal, K., Bassetto, G., Chintaluri, C., et al. (2019). Training deep neural density estimators to identify mechanistic models of neural dynamics. *bioRxiv*, 838383. doi:10.1101/838383

Graham, M. M., & Storkey, A. J. (2017). Asymptotically exact inference in differentiable generative models. *Electronic Journal of Statistics*, *11*(2), 5105–5164. doi:10.1214/17-EJS1340SI

Greenberg, D., Nonnenmacher, M., & Macke, J. (2019). Automatic posterior transformation for likelihood-free inference. In *Proceedings of the 36th international conference on machine learning*, Proceedings of machine learning research (Vol. 97, pp. 2404–2414). PMLR.

Gutmann, M. U., & Corander, J. (2016). Bayesian optimization for likelihood-free inference of simulator-based statistical models. *The Journal of Machine Learning Research*, *17*(1), 4256–4302.

Hermans, J. (2019). Hypothesis. *GitHub repository*. https://github.com/montefiore-ai/hypothesis; GitHub.

Hermans, J., Begy, V., & Louppe, G. (2020). Likelihood-free MCMC with approximate likelihood ratios. In *Proceedings of the 37th international conference on machine learning*, Proceedings of machine learning research (Vol. 98). PMLR.

Klinger, E., Rickert, D., & Hasenauer, J. (2018). pyABC: Distributed, likelihood-free inference. *Bioinformatics*, *34*(20), 3591–3593. doi:10.1093/bioinformatics/bty361

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference compilation and universal probabilistic programming. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017*, *54*.

Lintusaari, J., Vuollekoski, H., Kangasrääsiö, A., Skytén, K., Järvenpää, M., Marttinen, P., Gutmann, M. U., et al. (2018). ELFI: Engine for likelihood-free inference. *The Journal of Machine Learning Research*, *19*(1), 643–649.

Louppe, G., Cranmer, K., & Pavez, J. (2016). carl: A likelihood-free inference toolbox. *Journal of Open Source Software*, *1*(1), 11. doi:10.21105/joss.00011

Lueckmann, J.-M., Goncalves, P. J., Bassetto, G., Öcal, K., Nonnenmacher, M., & Macke, J. H. (2017). Flexible statistical inference for mechanistic models of neural dynamics. In *Advances in neural information processing systems 30* (pp. 1289–1299).

mackelab.org. (2017). DELFI: Density estimation likelihood-free inference. *GitHub repository*. https://github.com/mackelab/delfi; GitHub.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. doi:10.1063/1.1699114

Neal, R. M. (2003). Slice sampling. *The Annals of Statistics*, *31*(3), 705–741. doi:10.1214/aos/1056562461

Papamakarios, G., & Murray, I. (2016). Fast ϵ-free inference of simulation models with bayesian conditional density estimation. In *Advances in neural information processing systems 29* (pp. 1028–1036).

Papamakarios, G., Sterratt, D., & Murray, I. (2019). Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. *Proceedings of Machine Learning Research*, Proceedings of machine learning research, *89*, 837–848.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035).

Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., & Feldman, M. W. (1999). Population growth of human Y chromosomes: A study of Y chromosome microsatellites. *Molecular biology and evolution*, *16*(12), 1791–1798. doi:10.1093/oxfordjournals.molbev.a026091

Sisson, S. A., Fan, Y., & Tanaka, M. M. (2007). Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, *104*(6), 1760–1765. doi:10.1073/pnas.0607208104

Varoquaux, G. (2008). Joblib. *GitHub repository*. https://github.com/joblib/joblib; GitHub.