StochNetV2: A Tool for Automated Deep Abstractions for Stochastic Reaction Networks

Denis Repin, Nhat-Huy Phung, and Tatjana Petrov(✉)

Department of Computer and Information Sciences, University of Konstanz, Konstanz, Germany
den.ne.repin@gmail.com, tatjana.petrov@uni-konstanz.de

Abstract. We present a toolbox for stochastic simulations with CRN models and their (automated) deep abstractions: a mixture density deep neural network trained on time-series data produced by the CRN. The optimal neural network architecture is learnt along with learning the transition kernel of the abstract process. Automated search of the architecture makes the method applicable directly to any given CRN, which is time-saving for deep learning experts and crucial for non-specialists. The tool was primarily designed to efficiently reproduce simulation traces of given complex stochastic reaction networks arising in systems biology research, possibly with multi-modal emergent phenotypes. It is at the same time applicable to any other application domain, where time-series measurements of a Markovian stochastic process are available by experiment or synthesised with simulation (e.g. are obtained from a rule-based description of the CRN).

1 Introduction

Predicting stochastic cellular dynamics as emerging from the mechanistic models of molecular interactions is a long-standing challenge in systems biology: low-level chemical reaction network (CRN) models give rise to a highly-dimensional continuous-time Markov chain (CTMC) which is computationally demanding and often prohibitive to analyse in practice. Deep abstractions of CRN models, proposed in [2], use deep learning to replace this CTMC with a discrete-time continuous state-space process, by training a mixture density deep neural network with traces sampled at regular time intervals (which can be obtained either by simulating a given CRN or as time-series data from experiment). Deep abstractions are dramatically cheaper to execute, while preserving the

TP's research is supported by the Ministry of Science, Research and the Arts of the state of Baden-Württemberg, and the DFG Centre of Excellence 2117 'Centre for the Advanced Study of Collective Behaviour' (ID: 422037984), DR's research is supported by Young Scholar Fund (YSF), project no. $P83943018FP430\_/18$ and by the 'Centre for the Advanced Study of Collective Behaviour'. The authors would like to thank to Luca Bortolussi for inspiring discussions on the topic.

M. Gribaudo et al. (Eds.): QEST 2020, LNCS 12289, pp. 27–32, 2020.
https://doi.org/10.1007/978-3-030-59854-9_4

statistical features of the training data. The abstraction accuracy improves with the amount of training data. However, the overall quality of the method will also depend on the choice of neural network architecture. In practice, the modeller has to find the suitable architecture manually, through a trial-and-error cycle. In [8], we proposed to learn the optimal neural network architecture along with learning the transition kernel of the abstract process [3,7]. A similar idea has been recently employed for emulating epidemiological spread [4]; However, this work has focused on a single, uni-modal model of epidemics and only stationary regime, while our method is generic - applicable to any given CRN.

In this paper, we present StochNetV2Toolbox[1]- a tool for MDN-based deep abstractions of CRNs. Deep abstractions provide time-series trajectories which abstract the trajectories of the original CRN. Abstract models are implemented with neural networks, which predict a distribution for sampling the next system state. Moreover, StochNetV2Toolbox allows to, in addition to the initial state, parametrise the neural network with the kinetic rates *(as a part of the input).* The method is described in [8]. For illustration purposes, the tool includes a functionality for simulating multiple CRN instances on a spatial grid, where CRNs communicate via a subset of shared species which are diffused across neighbouring grid nodes.

2 Tool Architecture and Functionality

StochNetV2is implemented with four entities: `CRN_model`, `Dataset`, `StochNet`, and `Trainer` (see Fig. 1 for an overview). Two latter classes each have two different implementations: (i) a static implementation, used for standard deep abstractions as suggested in [2], and (ii) a dynamic implementation, used for *automated* deep abstractions, where the architecture of the neural network is learn along with the kernel of the process [8].

The general workflow proceeds in the following steps: (1) define a `CRN_model`, (2) produce trajectories, (3) create dataset from trajectories, (4) configure `StochNet`, (5) train it with `Trainer`, (6) produce trajectories. Finally, the user has the option to simulate multiple CRN instances on a spatial grid with class `Grid runner`.

2.1 CRN Models

The module contains base and example classes defining CRN models. These models can be simulated with Gillespie algorithm provided by `gillespy2` package. CRN models are used as a source of synthetic data to train and evaluate abstract models. An instance of `CRN_model` class can

- generate randomized initial concentrations (populations),
- generate randomized reaction rates,

[1] The tool name makes it transparent that the tool was inspired by [2] called 'StochNet'.

– set initial concentrations and reaction rates,
– produce trajectories.

A new CRN model should be inherited from `BaseCRNModel` class and implement all abstract methods. Several example models are provided (e.g. `SIR, Bees, Gene, X16`). In general, SBML models (Systems Biology Markup Language) can be imported, but it should be noted that the variability of the SBML format makes automated imports practically tedious, and for most models some pre-processing is required, e.g. editing reaction rates formulas, rewriting reversible reactions as two separate reactions, etc. see `BaseSBMLModel` and `EGFR` classes for examples.

2.2 Dataset

The `dataset` module implements functions and classes for creation and operations over trajectories data. It supports shuffling and applying pre-processing functions (such as adding noise) on-the-fly.

2.3 StochNet (Static)

`StochNet` class implements an interface for an abstract model. It is wrapped around a neural network (Mixture Density Network) which can be trained on simulation datasets and then used to produce trajectories. MDN consists of two parts: body (neural network extracting features of input state) and mixture (probability distribution with parameters depending on the extracted features). StochNet is initialized with body and mixture configuration files (config-file examples in `stochnet_v2/examples/configs`). A set of pre-defined building blocks for the body-part can be found in `stochnet_v2/static_classes/nn_bodies` file, which provides flexibility in the sense that custom building blocks can be added by the user. The supported distributions (the 'components' we use) for the mixture part can be found in `stochnet_v2/static_classes/top_layers`.

2.4 Training (Static)

Once `StochNet` is initialized, it can be trained with `Trainer`. When training is finished, all necessary files are saved to the model folder. A saved model can be loaded to produce trajectories at any time.

2.5 NASStochNet (Dynamic)

`NASStochnet` is an extension of the StochNet class. Instead of designing the body-part of MDN, it takes only a few hyper-parameters, such as an overall *depth* (number of layers) and *width* (number of neurons in layers). It starts with an over-parameterized probabilistic meta-model, which (by sampling so-called architecture parameters) represents many architectures at once. During training, the set of preferred layers, their order, and inter-connections are optimized automatically for given data.

2.6 Training (Architecture Search)

For the Architecture Search, training consists of two stages: (I) search for optimal configuration. This stage is a two-level optimisation, i.e. we run two separate optimisation procedures in altering manner for several epochs each: (main) update network parameters - weights in layers, (arch) update architecture parameters - weights of candidate operations in mix-layer, (II) fine-tuning of the found architecture after all redundancies are pruned.

After the search and fine-tuning stages, all necessary files are saved to the model folder, and the model can be loaded for simulations. Either `StochNet` or `NASStochNet` can be used to load trained model and run simulations.

2.7 Grid Runner

`GridRunner` implements a simulation of multiple CRN instances on a (spatial) grid with communication via spreading a subset of species across neighboring grid nodes. `GridRunner` is initialized with a model and `GridSpec`, which specifies a grid. Then, `GridRunner` stores state values for every model instance which can be updated by either in-node (one forward step of the model in every node) reactions, or on-grid interactions (diffusion of shared species across the grid).

2.8 Luigi Workflow Manager

The workflow is wrapped with the `luigi` library designed for running complex pipelines of inter-dependent tasks. Alternatively to manually run the above commands, one can fill a luigi configuration file, and it will run the whole sequence of tasks taking care of the right order and pre-requisites for every task.

3 Implementation

StochNetV2is written in Python 3 and uses Python libraries, mainly `tensorflow`, `gillespy2`, `luigi` and dependencies thereof. Source code of the tool with previously published Jupyter notebooks can be downloaded from GitHub - https://github.com/dennerepin/StochNetV2.

4 Evaluation and Applications

To evaluate the quality of abstract models, we compare distributions (histograms) of species of interest (e.g. Fig. 2a). For this, we simulate many trajectories of the original model starting from a set of random initial settings. Then an evaluation script runs from the same initial settings (example runtime comparison given in Fig. 2b). The evaluation script saves: (1) overall average value of histogram distance, (2) plots of species histograms after different number of steps, (3) plots of average (over different settings) distance between histograms produced by original and abstract model after different number of time-steps.

The case studies in the toolbox include applications in systems biology and collective behavior, such as the model of signaling pathway (EGFR [5], challenging multi-modal gene regulatory models (e.g. from [9]), and a reaction-based model of collective defence in honeybees (see [6] and GitHub page for details - https://github.com/dennerepin/StochNetV2[2]).

5 Related Tools

The original idea of using Mixture Density Networks was proposed in [2], which is followed by a theoretical work [1]. We are not aware of other tools using deep learning to abstract stochastic CRNs.

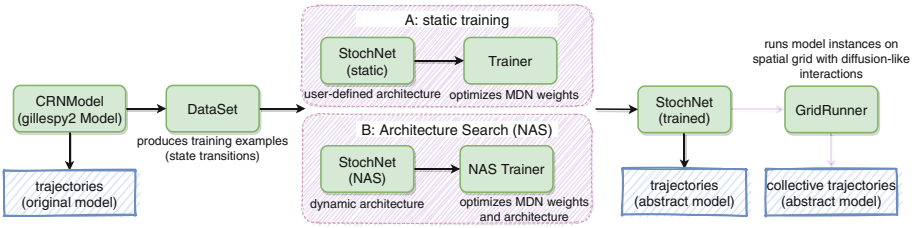

Fig. 1. Main components and workflow.

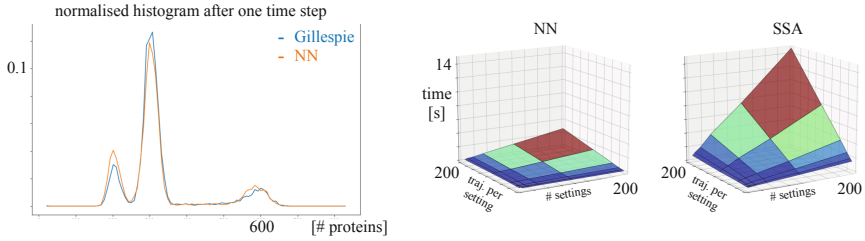

Fig. 2. (left) X40 case study from [9]: histograms of protein P_2 concentration after 1 time step and (right) the comparison of simulation run-time with NN and SSA, wrt. the number of initial settings and trajectories per setting.

References

1. Bortolussi, L., Cairoli, F.: Bayesian abstraction of Markov population models. In: Parker, D., Wolf, V. (eds.) QEST 2019. LNCS, vol. 11785, pp. 259–276. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30281-8_15

[2] While we performed specific performance evaluation, e.g. in Fig. 2 and [8], a systematic scalability analysis is beyond the scope of this tool presentation.

2. Bortolussi, L., Palmieri, L.: Deep abstractions of chemical reaction networks. In: Češka, M., Šafránek, D. (eds.) CMSB 2018. LNCS, vol. 11095, pp. 21–38. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99429-1_2
3. Cai, H., Zhu, L., Han, S.: ProxylessNAS: direct neural architecture search on target task and hardware. CoRR abs/1812.00332 (2018). http://arxiv.org/abs/1812.00332
4. Davis, C.N., Hollingsworth, T.D., Caudron, Q., Irvine, M.A.: The use of mixture density networks in the emulation of complex epidemiological individual-based models. PLoS Comput. Biol. **16**(3), 1–16 (2020). https://doi.org/10.1371/journal.pcbi.1006869
5. Feret, J., Henzinger, T., Koeppl, H., Petrov, T.: Lumpability abstractions of rule-based systems. Theoret. Comput. Sci. **431**, 137–164 (2012)
6. Hajnal, M., Nouvian, M., Šafránek, D., Petrov, T.: Data-informed parameter synthesis for population Markov chains. In: Češka, M., Paoletti, N. (eds.) HSB 2019. LNCS, vol. 11705, pp. 147–164. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28042-0_10
7. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=S1eYHoC5FX
8. Petrov, T., Repin, D.: Automated deep abstractions for stochastic chemical reaction networks. arXiv preprint arXiv:2002.01889 (2020)
9. Plesa, T., Erban, R., Othmer, H.G.: Noise-induced mixing and multimodality in reaction networks. Eur. J. Appl. Math. **30**(5), 887–911 (2019)