Deep Abstractions of Chemical Reaction Networks

Luca Bortolussi[1] and Luca Palmieri[1,2]

[1] DMG, University of Trieste lbortolussi@units.it
[2] SISSA, Trieste, contact@lpalmieri.com

Abstract. Multi-scale modelling of biological systems, for instance of tissues composed of millions of cells, are extremely demanding to simulate, even resorting to HPC facilities, particularly when each cell is described by a detailed model of some intra-cellular pathways and cells are coupled and interacting at the tissue level. Model abstraction can play a crucial role in this setting, by providing simpler models of intra-cellular dynamics that are much faster to simulate so to scale better the analysis at the tissue level. Abstractions themselves can be very challenging to build ab-initio. A more viable strategy is to learn them from single cell simulation data.
In this paper, we explore this direction, constructing abstract models of chemical reaction networks in terms of Discrete Time Markov Chains on a continuous space, and learning transition kernels using deep neural networks. This allows us to obtain accurate simulations, greatly reducing the computational burden.

Keywords: deep learning, chemical reaction networks, model abstraction, stochastic simulation

1 Introduction

Computational modelling is a central ingredient in the quest for understanding and predicting the dynamics of complex biological systems [8]. A wide range of interesting biological processes can be modelled as a complex network of biochemical reactions. These reactions take place inside cells, which are themselves part of networks of intercellular interaction. This is the case, for instance, of tumoral tissues [11]. In silico modelling plays a central role also for studying such multi-scale systems.

At the intra-cellular level, there are fairly established techniques to model and simulate biochemical reaction networks, taking into account the intrinsic variability and noise due to the small number of molecules involved and a certain degree of randomness in their distribution [8]. The most prominent approach to analyse such models is stochastic simulation, starting from the well known Gillespie algorithm [1], and moving to more efficient but approximate methods like tau-leaping [6] and hybrid simulation [9].

When dealing with a multiscale system, it would be desirable to explicitly model the detailed intracellular mechanics. Unfortunately, simulations times become quickly unfeasible when working with large number of cells 10^5 or 10^6 cells, even using state of the art HPC infrastructures and techniques and approximate simulation algorithms.
Models of phenomena happening at the tissue level at this scale of complexity require a considerable reduction of single cell simulation times, simplifying cellular models. This calls model abstraction into play.

While manual crafting of abstract models is always possible, a more scalable approach is to learn abstract models from single-cell simulation data. The basic idea is to start from a suitable number of simulated system trajectories, using available simulation algorithms, and then learn a simpler probabilistic model from such data. Such model should allow us to generate approximate trajectories, possibly only for a subset of variables involved in the inter-cellular processes, in a significantly faster way that the original detailed model, still retaining a reasonable accuracy.

Related work. This idea was employed by Liu et al. in [13] to approximate an ODE dynamic and was further refined by Palaniappan et al. in [26] to deal with a stochastic dynamics. In this work, the authors select a subset of relevant variables and discretise them using information theoretic tools, and then build an approximate model based on a dynamic Bayesian network (DBN). Palaniappan et al. were able to perform accurate simulations using their abstract DBN, reducing simulation times by an order of magnitude compared to the original model. An alternative approach is that of [25], in which authors learn a simplified model of the bacteria chemotaxis mechanisms in bacteria exploiting Gaussian Processes (GP). In particular, they consider a fast equilibrating internal process, and use GP to model the choice of movement strategy as a function of the environmental state.

Contributions In this paper, we focus on the abstraction procedure, starting from [26]. Their approach is mostly driven by information theoretic considerations, which is extremely effective in isolating the core components needed to perform an accurate approximation of the original process. Moreover, [26] and [13] models are discrete in time and space: the domain of each tracked chemical species is partitioned in a certain number of subintervals and, at each time step, the system may change its state by jumping to another node of the discrete state space. The jumping probabilities, properly factored according to the DBN topology, have to be stored in memory. This becomes troublesome when a high number of chemical species is involved or when a high level of resolution on the state of certain variables is required. In this case, the size of the discrete state space is doomed to explode.

Instead of working with a DBN, our insight is to resort to a different probabilistic model: a discrete-time Markov chain on a *continuous state space*, bypassing the difficulties arising from the state space discretization. A Markov chain

$\{\eta_k\}_{k\in\mathbb{N}}$ is completely determined once its transition kernel has been specified, i.e. a function which maps the previous state s_n of the chain to the probability distribution on the state space describing the possible outcomes for η_{n+1} conditioned on $\eta_n = s_n$. Transition kernels are the continuous equivalent of a transition matrix for a Markov chain with a finite number of states.

Our proposal is to model transition kernels as *probability mixtures*, which are weighted combinations of a certain number of elementary probability distributions components (normal, log-normal, etc.). Sampling from a mixture is fast, while the model itself is extremely flexible and can be used to approximate fairly complicated probability distributions. Our components are chosen from the exponential family and are thus completely determined by a fixed number of parameters. Everything then boils down to an optimization problem: we need to properly tune the number of components and their parameters in order to produce a good transition kernel which maximizes the likelihood of our data.

Real world chemical reaction networks involve a high number of different species, which means that our components are high dimensional probability distributions: the parameter space we want to explore is expected to be complicated. Furthermore, parameters will depend on the previous state η_n visited by the discrete abstraction, hence they have to be modelled as (continuous) functions of this state.

We tackled this supervised learning problem using Deep Neural Networks (NN). In particular, we exploit Mixture Density Networks, [2], and use different NN body architectures . We provide a software implementation of the described approach as a reusable Python library, and show on some case studies the effectiveness of our method, capable of reducing the computational complexity of several orders of magnitude.

Paper Structure The paper is organised as follows: In Section 2, we introduce the relevant background notions, while in Section 3 we discuss the abstraction procedure. Section 4 is devoted to present the implementation, and Section 5 to experimental evaluation. Conclusions are drawn in Section 6.

2 Background

Chemical reaction networks Chemical Reaction Networks (CRNs) are the standard formalism to describe dynamical models of biological systems. Under the well-stirred assumption, they can be interpreted stochastically as a Continuous Time Markov Chain (CTMC) [8, 1, 27] on a discrete state space S. We denote by $\mathcal{X} = \{X_1, \ldots, X_n\}$ the chemical species involved in our CRN and by $\eta_t = (\eta_{t,1}, \ldots, \eta_{t,n}) \in S = \mathbb{N}^n$ the state vector at time t: $\eta_{t,i}$ is the number of X_i molecules in the system at time t.

The dynamics is encoded by a set $\mathcal{R} = \{R_1, \ldots, R_m\}$ of reactions, each being a tuple (f_{R_i}, ν_i). f_{R_i} is the *propensity function*, and gives the rate at which reaction R_i fires, while ν_i is the update vector: the firing of reaction R_i changes the state from η_t to $\eta_t + \nu_i$.

$\mathbb{P}(\eta_t = s \,|\eta_{t_0} = s_0) = \mathbb{P}_{s_0}(\eta_t = s)$ is the probability of finding our system in state s at time t given that it was in state s_0 at time t_0. It satisfies a system of ODEs known as *Chemical Master Equation* (CME):

$$\partial_t \mathbb{P}_{s_0}(\eta_t = s) = \sum_{i=1}^{h} [\mathbb{P}_{s_0}(\eta_t = s - \nu_i) f_{R_i}(s - \nu_i) - \mathbb{P}_{s_0}(\eta_t = s) f_{R_i}(s)] \qquad (1)$$

Solving the CME numerically is very challenging (see [27] for further details), and typically CRNs are simulated. The most commonly used simulation algorithm is Gillespie's SSA [1].

Running example: the SIR model The SIR epidemiological model, describing the spread of an infectious disease that grants immunity to those who recover from the acute phase. Despite not being properly a molecular system, it can be modelled as a CRN. The SIR model describes a population of N individuals divided in three mutually exclusive groups: **S**usceptible, **I**nfected and **R**ecovered/**R**emoved. The system state vector is given by $\eta_t = (S_t, I_t, R_t)$, each component standing for the total number of individuals in the corresponding population group. The interactions considered by the SIR model are the following:

$$R_1 : \ \mathrm{S} + \mathrm{I} \longrightarrow 2\,\mathrm{I} \qquad (\textit{infection}) \qquad (2)$$

$$R_2 : \ \mathrm{I} \longrightarrow \mathrm{R} \qquad (\textit{recovery/death}) \qquad (3)$$

The propensity functions are of mass action type [8]: $f_{R_1}(S_t, I_t, R_t) = k_1 \frac{I_t S_t}{N}$ and $f_{R_2}(S_t, I_t, R_t) = k_2 I_t$, with corresponding update vectors $\nu_1 = (-1, +1, 0)$ and $\nu_2 = (0, -1, +1)$. The ratio k_1/k_2 is called basic reproduction number and its value is strongly connected with the overall behaviour of the system: minor outbreak, serious outbreak, pandemic (see [10]). The SIR model dynamic is well understood and we shall use it as testing ground for our abstraction protocol.

Neural Networks Machine learning can be loosely defined as a collection of algorithms and techniques employed to **automatically** tune (*learn*) an analytical model from a (possibly huge) set of data [23, 5]. Learned models are used to perform a wide range of different tasks: image recognition, time series analysis, machine translation, speech recognition, etc.

Artificial neural networks (NN) date back to the 50s and have recently conquered the spotlight thanks to an impressive set of achievements [23]. The basic (*feed-forward*) NN architecture can be represented as a weighted directed graph where each vertex stands for a node (or *neuron*). It can be divided in three blocks: an input layer, a set of hidden layers[3] and an output layer. Data

[3] A node is said to be hidden if it does not belong to the input or the output layer. A layer is, roughly, a collection of nodes at the same depth level with respect to the input layer. A layer composed of hidden nodes is called hidden layer.

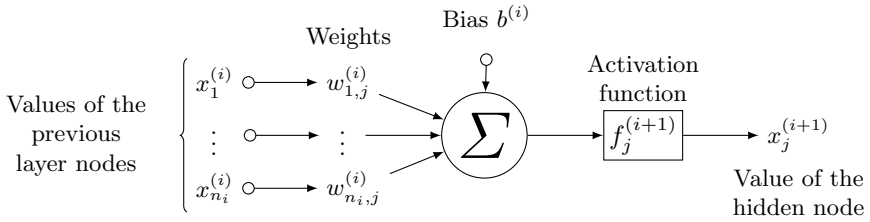


Fig. 1: A visual summary of what happens at hidden and output nodes.

are fed to the input layer and flow through the network undergoing a certain number of (non-linear) transformations determined by hidden block design. The transformed data are then returned to the user through the output layer. This procedure is called *forward propagation*. The value of an output node is thus the result of the stacked application of linear transformations and a (non-linear) function. Despite their mathematical simplicity, NNs can approximate any measurable function from the input to the output nodes with arbitrary precision [23].

In a supervised machine learning problem we are given a training dataset: each input data is associated with the desired output. Training a NN is a (stochastic) optimization problem: we try to minimize an error function, which is computed using the values of the output nodes, with respect to the NN weights and biases.

Mixture Density Networks A typical regression approach would try to learn the value of η_t using η_{t-1} as input minimizing a sum-of-squares error function. This corresponds, implicitly, to the task of learning the mean of a multivariate normal variable via maximum likelihood estimation (see section 5.2 of [5]). This scheme, however, does not capture information about the distribution of η_t given η_{t-1}, and is going to perform particularly poorly if the system exhibits a certain degree of multimodality.

To circumvent such problem, we are going to learn the parameters needed to specify a mixture distribution: these are going to be our output nodes, as anticipated in Section 1. The error function associated with the output layer is naturally the negative log-likelihood of the mixture density distribution. This type of NNs are called Mixture Density Networks (MDN). All the models in this work are MDNs. A detailed explanation of MDNs can be found in [2], where they made their first appearance. MDNs are also well explained in section 5.6 of Bishop's book, [5].

3 MDN-based abstraction procedure

Model abstraction Let $\{\eta_t\}_{t\geq 0}$ be a CTMC describing a CRN system with state space $S = \mathbb{N}^m$. To construct our abstraction, we assume to be interested only in the behaviour of the model in a grid of time points at a fixed temporal

distance. Hence, we fix a time step Δt and an initial time instant $t_0 \in \mathbb{R}$ and define

$$\tilde{\eta}_i := \eta_{t_0 + i\Delta t} \qquad \forall i \in \mathbb{N}. \tag{4}$$

The stochastic process $\{\tilde{\eta}_i\}_i$, thanks to the Markov property enjoyed by CTMCs, is a time-homogeneous Discrete Time Markov Chain (DTMC) with transition kernel

$$K_d(s \mid s_0) = \mathbb{P}(\eta_{\Delta t} = s \mid \eta_0 = s_0) \tag{5}$$

for all $s, s_0 \in S$.

Our model abstraction procedure introduces two approximations:

1. The state space $S = \mathbb{N}^m$ is embedded into the continuous space $\tilde{X} = \mathbb{R}^m_{\geq 0}$. The abstract model takes values in $\tilde{X}$.
2. The kernel K_d is approximated by a new kernel $K(x \mid x_0)$ taking values in the continuous space $\tilde{X}$.

In constructing the approximate kernel $K(x \,|\, x_0)$, Rather than trying to preserve the full behaviour of the process, we restrict our attention to a time-bounded **reward function** $r : S^M \to T$ from S^M to an arbitrary space T (i.e. $\mathbb{R}$, $\mathbb{N}$, $\mathbb{B}$, or $\mathbb{R}^k$). Here M is an upper bound on the duration of discrete time trajectories we consider to evaluate the reward; we indicate time-bounded trajectories by $\tilde{\eta}_{[0,M]}$. Such a function r can be a projection, thus monitoring the number of molecules belonging to a certain subset of chemical species at a certain time step, or it can take Boolean values in $\mathbb{B} = \{0, 1\}$, representing the truth of a linear temporal property, for example checking if the system has entered into a dangerous region. Note that $r(\tilde{\eta}_{[0,M]})$ is a probability distribution on T.

The second ingredient we need is a way to measure the error introduced by the abstract model, i.e. how much the abstract distribution differs from $r(\tilde{\eta}_{[0,M]})$. This can be accomplished by fixing a distance among distributions. In our experiments, we rely on the L1 norm, typically used to measure the goodness of fit:

$$d(X, Y) := \int_{\mathbb{R}^k} |p_X(z) - p_Y(z)| \; d\, z \tag{6}$$

This metric will be practically evaluated statistically, resulting in the so called histogram distance [7]. We now give a formal definition of model abstraction.

Definition 1. *Let $\eta = \{\eta_i\}_{i=0}^M$ be a discrete time stochastic process over an arbitrary state space S, with $M \in \mathbb{N}_+$ a time horizon, and let $r : S^M \to T$ be the associated reward function. An abstraction of (η, r) is a quadruple $(S', p, r', A = \{A_i\}_{i=0}^M)$ where:*

- *$\bar{S}$ is the **abstract state space**;*
- *$p : S \to \bar{S}$ is the **abstraction function**;*
- *$\bar{r} : \bar{S}^M \to T$ is the **abstract reward**;*
- *$\bar{\eta} = \{\bar{\eta}_i\}_{i=0}^M$ is the abstract discrete time stochastic process over $\bar{S}$.*

Let $\varepsilon > 0$. $\bar{\eta}$ is said to be ε-close to η with respect to d if, for almost any $s_0 \in S$,

$$d\big(r(\eta_{[0,M]}), \bar{r}(\bar{\eta}_{[0,M]})\big) < \varepsilon \qquad \textit{conditioned on } \; \eta_0 = s_0,\, \bar{\eta}_0 = p(s_0) \tag{7}$$

It is common enough to choose a projection over a subset of chemical species as abstraction function p - as in [26], possibly identified by information theoretic criteria to be those most influencing the reward of interest. Alternatively, we could follow [13] and use a projection over a certain number of sub-regions of the original state space in order to get a finite abstract state space $\bar{S}$. Equation 7 is typically experimentally verified simulating a sufficiently high number of trajectories from both the original system η and the abstraction $\bar{\eta}$ starting from a common initial setting. There is no way to ensure, with this experimental procedure, that equation 7 holds for almost every s_0 in S. What can be done, instead, is to choose a high number of different initial settings which were not in the training set and check if the condition holds for them - the classical training/validation procedure which is used in Machine Learning to estimate the generalization error of a model.

Dataset Generation We build our model abstraction reframing the situation as a supervised learning problem. Choose N random starting states $\{s_0^{(j)}\}_{j=1}^N$ from (a finite subset of) $\tilde{X}$. For each $s_0^{(j)}$ run a simulation from t_0 to $t_1 := t_0 + \Delta t$. Denote by $\eta_{t_1}^{(j)}$ the system state at time t_1 for each one of these simulations. Define: $x^{(j)} := s_0^{(j)}$ and $y^{(j)} := \eta_{t_1}^{(j)}$ for all $j \in \{1, \dots, N\}$.

We have thus built $\mathcal{D} := \{(x^{(j)}, y^{(j)})\}_{j=1}^N$, where each $y^{(j)}$ is a sample from the probability distribution $\mathbb{P}(\eta_{\Delta t} \mid \eta_0 = x^{(j)})$. We can as well simulate trajectories from t_0 to $t_h := t_0 + h\Delta t$, $h \in \mathbb{N}_+$. It is then sufficient to extract the system state at time instants $\{t_0, t_0 + \Delta t, \dots, t_0 + h\Delta t\}$ in order to consider consecutive datapoints $\left(\eta_{t_0+i\Delta t}, \eta_{t_0+(i+1)\Delta t}\right)$, $i \in \{0, \dots, h-1\}$ as an (x, y) pair, like we described above.

Model training Fix a parametrized family of mixture distributions $\mathcal{M}$. Let g_θ be a MDN with $g_\theta(x) \in \mathcal{M}$ for each feature vector x, where θ are the network weights. g_θ is trained on the dataset $\mathcal{D}$, the simulation data, to learn the desired approximation K of K_d:

$$K_d(s \mid s_0) = \mathbb{P}(\eta_{\Delta t} = s \mid \eta_0 = s_0) \approx \mathbb{P}(g_\theta(s_0) \in B_s) := K(B_s \mid s_0) \tag{8}$$

where $B_s := \left\{ x \in \tilde{X} \;\middle|\; \|x - s\|_\infty < \frac{1}{2} \right\}$ is the ball with respect to the infinity norm of radius $1/2$, centred in s. Training g_θ has a cost: neural networks are computationally intensive models. Nonetheless, once the network has been tuned, its evaluation is extremely fast, considering that all modern deep learning frameworks provide a GPU implementation.

Abstract Model Simulation In order to simulate the abstract model, we just need to sample up to time horizon $M > 0$ from the approximate kernel K, starting from the initial state s_0 and initial time t_0. While sampling, we do not restrict to be in the discrete state space S, but rather simulate trajectories

on the continuous state space $\tilde{X}$. Each timestep of our simulations has thus a fixed computational cost, and requires us to evaluate g_θ at the current state x and draw a sample from the resulting distribution. This means that, choosing Δt equal to the timescale of interest, we can simulate arbitrary long trajectories at the needed level of time resolution without wasting computational resources. This algorithm can be easily employed in a multiscale setting: we just need to train g_θ once, while a high number of agents can be simulated in parallel levering the computational power of one or more GPUs.

4 Implementation

Our implementation is in Python, and builds on several available tools and libraries, developed in the research communities of deep learning and computational systems biology. Integrating tools of different communities in a common pipeline has not been straight-forward and a considerable amount of work is required to design a robust experimental setup. This was the main reason behind the development of **StochNet**, which hides the difficulties of such integration to the user, which can then focus on the design and tuning of models. The library, at the present stage of development, offers the following functionalities through a high-level API:

- Concurrent simulation of CRN models starting from different initial states using SSA/τ-leaping;
- A wide collection of implemented random variables to be used on their own or in mixtures to approximate complex probability distributions. Mixtures can be build using random variables from different families, as long as their samples have the same dimensionality;
- Seamless integration with Keras and Tensorflow to train and deploy Mixture Density Networks - there is no low-level scripting required to define, train and run a complete model;
- Simulation of trajectories from a Mixture Density Networks model using a GPU-based concurrent sampling strategy;
- A ready-to-run fault-tolerant pipeline to manage long-running numerical experiments (abstraction building). The pipeline can be easily customized and extended with custom computational tasks.

Most of these functionalities are built on top of existing Python packages: Tensorflow ([18]) and Keras ([20]) for neural networks, Gillespy ([22]) as an interface to StochKit 2.0 ([14]) for CRN simulations and Luigi ([15]) as work-flow manager.

StochNet is currently hosted on GitHub, where we are going to provide a detailed package documentation: https://github.com/LukeMathWalker/StochNet.

5 Experimental Results

In this section, we validate our approach on two case studies: the simple SIR model, and a more computationally intensive genetic network. Our focus is in the accuracy of the abstract model and on its computational efficiency.

Experimental setting To perform all the simulations described in the next sections we used a desktop personal computer, equipped with an AMD Ryzen 1700 (3GHz), 8 cores CPU, an Nvidia GeForce GTX 1080Ti, 11GB, 3'584 CUDA cores GPU, and 32GB DDR4 of RAM.

Data preparation Neural network convergence is enhanced if each component of the training dataset has zero mean and unit variance (cfr. Section 4.3 in [3] or Chapter 12 in [24]). We have thus followed this established procedure for all our datasets. No other forms of data cleaning or preprocessing have been performed.

5.1 SIR model

We start by presenting experimental results on the SIR model, first discussing the MDN architecture we used to build the approximate kernel, and then presenting the experimental results.

MDN architecture Considering the simple dynamic of the SIR model we opted for a fairly straight-forward architecture: our final setup uses a single hidden layer composed of 150 units with a ReLU[4] activation function; the output layer is designed to learn the parameters of a mixture of 2 multivariate normal random variable with a diagonal covariance matrix. We used the Adam algorithm (with default settings) to optimize our loss function (cfr. [16]).
In order to avoid overfitting we introduced two forms of regularization:

- **Max-norm regularization**([17]);
- **Early stopping** (cfr. Section 7.8 of [24]).

For the SIR model we constrained the weights euclidean norm to be below 3.
Each epoch used 30′000 training datapoints, processed in batches of 32 datapoints each. Our early stopping patience was set to 6 epochs and we evaluated the validation error on 5′000 held-out validation datapoints.
The number of nodes in the hidden layer, the number of normal random variables in the mixture, the weight max-norm and the usage of other regularization techniques (such as Dropout [17] or gaussian noise injection) have been compared using the loss on the validation dataset as a measure of the generalization error.
It is worth to point out that even though the system dynamic is essentially unimodal in the chosen time step Δt, we experienced a significant worsening in the

[4] Rectified Linear Unit: $f(x) = \max\{0, x\}$, cfr. [12].

Task	Time: SIR	Time: GeneNet
GenerateDataset (training)	35 s	2,328 s (∼ 39 min)
GenerateDataset (validation)	35 s	2,319 s (∼ 39 min)
FormatDataset (training)	0.015 s	0.07 s
FormatDataset (validation)	0.015 s	0.07 s
GenerateHistogramData (training)	45 s	26,650 s (∼ 7 h & 24 min)
GenerateHistogramData (validation)	36 s	27,299 s (∼ 7 h & 30 min)
TrainNN	31 s	1,337 s (∼ 22 min)

Table 1: Execution time required to complete each step of the abstraction pipeline for the SIR model and the gene network model.

MDN performances when using a single multivariate normal random variable as output schema, with several training attempts failed due to a divergent behaviour in the loss minimization procedure. This never happened when using a mixture of two normal random variables: the possibility to distribute "errors" on two different gaussians seems to stabilize the learning phase, leading to more consistent results. This is consistent with current practice in deep learning, which prefers a high model-capacity coupled with strong regularisation.

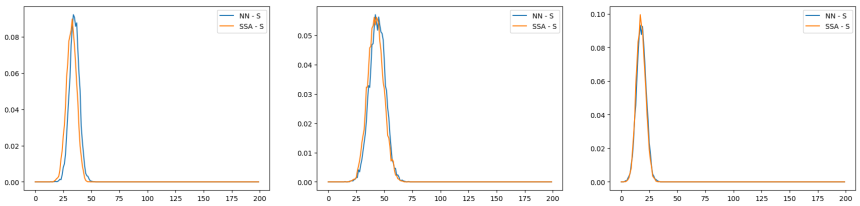
Results The execution time of whole pipeline takes slightly more than 3 minutes, of which roughly the 16% is used to train the MDN - see Table 1.

Our MDN has been trained to predict the system state after 0.5 units of simulation time. We computed the mean histogram distance on 1-step and 5-steps predictions between the MDN and the SSA algorithm with respect to the projection on the S population. A sample of the histograms after 5 steps can be seen in Figure 2. The mean histogram (over 25 different initial conditions) after 1 step is 0.3 and after 5 steps is 0.33.

We can see from the figures that the MDN model captures the system dynamic quite accurately - the mode is almost always perfectly aligned with the SSA mode, while the reconstructed variance is sometimes slightly defective or excessive. Even though 5-steps performances are slightly worse than their respective single-step analogues, we do not observe any significant error-propagation phenomenon in place, considering the level of resolution we are using for our histograms (200 bins on $[0, 200]$). Nonetheless we cannot affirm that the MDN distribution is consistently indistinguishable from the SSA distribution: the upper-bound on the mean of the self-distance using 200 bins and $10'000$ samples (cfr. [7]) is $\sim$0.16 with a variance upper-bound of $\sim$0.015; the lowest mean histogram distance achieved by our MDN on the training histogram dataset, with single-step simulations, is 0.24 (cfr. **??**).

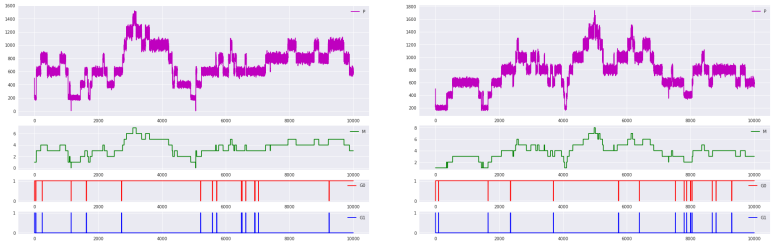
5.2 Gene Regulatory Network

Introduction Stochasticity plays a prominent role in a CRN dynamics whenever some key chemical species in the network have low molecular counts: adding



(a) (b) (c)

Fig. 2: Comparison of SSA and MDN histograms after 5 simulation steps over 3 different initial settings sampled from the validation dataset. $K = 200$ equally sized bins are being used. The mean histogram distance, over 25 different initial settings, is 0.33.



(a) (b)

Fig. 3: Two sample trajectories for the gene regulatory network.

or removing a single copy might result in significant fluctuations. A typical example of this behaviour can be observed in a simple self-regulated gene network [19]: a single gene G is transcribed to produce copies of a mRNA signal molecule M, which are in turn translated into copies of a protein P; P acts as a repressor with respect to G - it binds to a DNA-silencer region, inhibiting gene transcription. In other words, the gene activity is regulated through a negative-feedback loop, a common pattern in biological systems. The system chemical equations are the following:

$$R_1: \ \mathrm{G}^{\mathrm{ON}} \xrightarrow{\mathrm{k_{prodM}}} \mathrm{G}^{\mathrm{ON}} + \mathrm{M} \qquad (\textit{transcription}) \qquad (9)$$

$$R_2: \ \mathrm{M} \xrightarrow{\mathrm{k_{prodP}}} \mathrm{M} + \mathrm{P} \qquad (\textit{translation}) \qquad (10)$$

$$R_3: \ \mathrm{G}^{\mathrm{ON}} + \mathrm{P} \xrightarrow{\mathrm{k_{act}}} \mathrm{G}^{\mathrm{OFF}} \qquad (\textit{protein binding}) \qquad (11)$$

$$R_4: \ \mathrm{G}^{\mathrm{OFF}} \xrightarrow{\mathrm{k_{deact}}} \mathrm{G}^{\mathrm{ON}} + \mathrm{P} \qquad (\textit{protein unbinding}) \qquad (12)$$

$$R_5: \ \mathrm{M} \xrightarrow{\mathrm{k_{degM}}} \varnothing \qquad (\textit{mRNA degradation}) \qquad (13)$$

$$R_6: \ \mathrm{P} \xrightarrow{\mathrm{k_{degP}}} \varnothing \qquad (\textit{protein degradation}) \qquad (14)$$

k_{prodM}	k_{prodP}	k_{act}	k_{deact}	k_{degM}	k_{degP}
350	300	1	166	0.001	1.5

Table 2: Reaction rates used in our simulations of the gene regulation network.

All propensity functions are assumed to be of mass-action type, with the name of their respective rate specified on the reaction arrow. The system dynamic varies significantly with respect to the choice of reaction rates. The parameters values for our simulations are reported in Table 2 while Figure 3 shows some of the simulated system trajectories: the systems exhibits several well-separated *stable configurations* which are roughly determined by the number of available mRNA molecules. It is worth mentioning that, on a smaller scale, each stable point is actually noisy - we can in fact observe a high number of small amplitude oscillations. These characteristics can be easily detected looking at the probability density function of $\mathbb{P}_{s_0}(\eta_t)$, for $t > t_0$: we have 5 or 6 distinguishable modes, with a certain amount of gaussian noise affecting each one of them (see Figure 5).

Approximating this system dynamics with a normal regression NN is an impossible task, considering that the system dynamic is definitely not unimodal: this makes of this model a perfect testing ground for our MDNs.

MDN architecture To model the dynamic of the gene regulation network we devised a more sophisticated architecture than for the SIR case, see Figure 4. We are no longer dealing with a shallow neural network: our architecture uses two hidden layers. The learning capabilities of deep neural networks are significantly higher, even though depth increases the chance of overfitting and introduces the vanishing/exploding gradient problem, an issue affecting the minimization of the loss function (cfr. [4]).

ReLU activation functions avoid the problem of exploding gradients (ReLU derivative is either 0 or 1) but they still do not solve the issue of vanishing gradients. A variety of different strategies have been devised (changing weight initialization, gradient clipping, etc.): we chose to follow in the footsteps of [21] authors - the first to introduce the ResNet architecture. Their proposal is strikingly simple: instead of stacking hidden layers directly on top of each other we use *skip-connections* to regularize the network behaviour. In other words, instead of learning a map of the form $NN(x) = f_1(x)$ we try to learn $NN(x) = f_1(x)+x$. If $H(x)$ is the *true* function we are trying to fit, then f_1 is actually trying to approximate $H(x)-x$ which is called *residual function*. Despite of its simplicity, this adjustment allowed [21] authors to train effectively neural networks with 152 hidden layers. We do not need such extreme configurations, but deeper architectures can help improve performance of our approximations.

In terms of regularisation, we constrained the euclidean norm of layer weights to be below 3, we used early stopping with 6 epochs of patience and we injected gaussian noise (0 mean, 0.01 variance) between the input layer and the first hidden layer. Noise addition is another common regularisation technique which tries to force the network to learn a more robust representation, i.e. a mapping robust enough to be insensible to small perturbation of the inputs. Each epoch

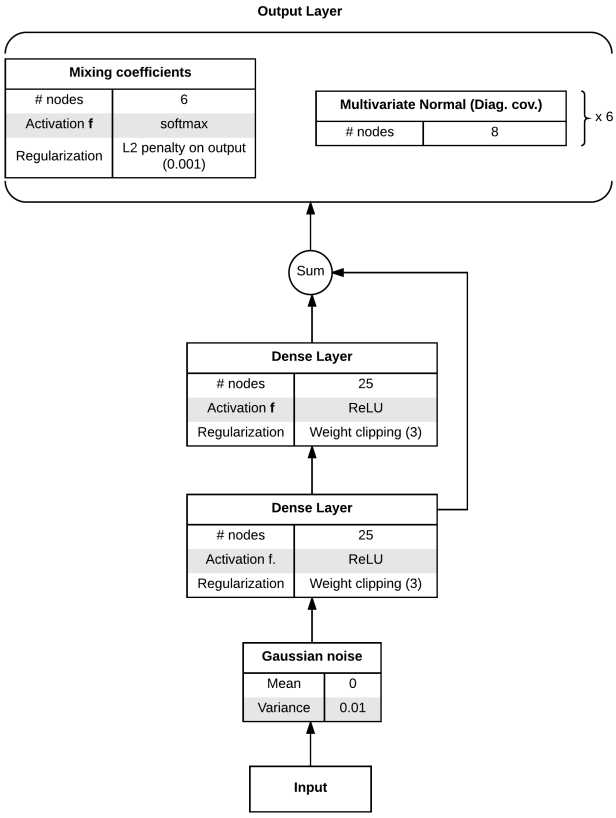
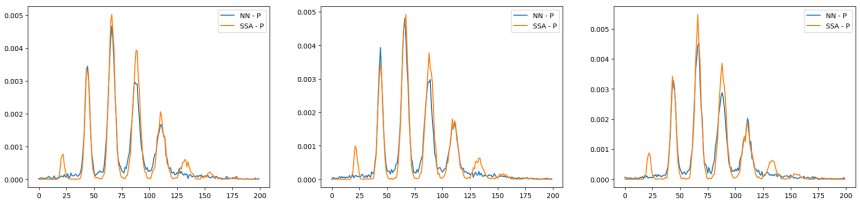


Fig. 4: Architecture of the MDN trained for the gene regulation network.

used 30′000 training datapoints, which were processed in batches of 32 datapoints each. We evaluated the validation error at the end of each epoch using 5′000 held-out validation datapoints. All model hyper-parameters have been tuned using the loss on the validation dataset as a measure of the generalisation error.

Results Executing the whole pipeline requires ∼16 hours and 30 minutes: the training of the MDN is responsible for a mere 2% of the overall computational cost - see Table 1. Our MDN has been trained to predict the system state after 400.0 units of simulation time. We computed the mean histogram distance on 1-step and 50-steps predictions between the MDN and the SSA algorithm with respect to the projection on the protein P, just like we did for the SIR model. A sample of the resulting histograms for the 50 step case can be seen in Figure 5. The mean histogram distance after 1 step is 0.34, stabilising to 0.28 after 50 steps. Although the distributions are not distinguishable (mean histogram self distance is XXXX), the true and the approximate distributions are quite similar, and the main qualitative characteristics of the process are well

Missing data: please fiii



(a) (b) (c)

Fig. 5: Comparison of SSA and MDN histograms of species P after 50 steps of simulation time over 3 different initial settings sampled from the validation dataset. $K = 200$ equally sized bins are being used. The mean histogram distance, over 25 different initial settings, is 0.28.

Algorithm	**SIR - Time**	**GRN - Time**
SSA	40 s	∼60'650 s ($\sim$ 17 h)
τ-leaping	NA	∼10'750 s ($\sim$ 3 h)
MDN (trained)	7 s	35 s

Table 3: Time required to simulate 10′000 trajectories of the SIR and of the gene regulatory network (GRN) for 25 different initial settings with endtime 5 for SIR and 10′000 for the GRN. For SIR: The MDN returns a datapoint every 0.5 units of simulation time. for GRN: The MDN and τ-leaping return a datapoint every 400 units of simulation time.

captured. In particular, the MDN model is able to identify quite consistently the four modes associated with the highest probabilities, with a quite accurate reconstruction of the respective variances (prone to be overestimated, more than underestimated). The two rarest modes, instead, are usually ignored.

The speed gain, though, is quite remarkable (see Table 3): simulating 10′000 trajectories for 25 different initial settings with endtime 10′000 takes $\sim$ 17 hours using SSA, while the MDN model is capable of doing it in 35 seconds - it is roughly 1730 times faster than SSA. We also compared our MDN approach to τ-leaping. Using $\tau = 400.0$ in order to achieve the greatest possible speed-up, τ-leaping produces trajectories that are indistinguishable from SSA-generate trajectories, achieving a mean histogram distance of ~ 0.1. However, it took τ-leaping ~ 3 hours to generate 10′000 trajectories for 25 different initial settings with endtime 10′000: almost 6 times faster than SSA but still 300 slower than our MDN abstraction.

6 Conclusions

In the paper we presented a pipeline to build abstract models in order to increase simulation efficiency of Biochemical Reaction Networks. Our approach leverages recent advances in theory and tools for deep learning, approximating a CTMC

as a Discrete Time process whose transition kernel is learned from simulation data using Mixture Density Neural Networks.

In the paper, we have shown that the method has a significant potential: we have been able to capture with significant accuracy the qualitative behaviour of a genuinely multimodal CRN without introducing any kind of prior knowledge into our procedure. The achieved speed-up is impressive and it would enable, on similar systems, to actually perform multiscale simulations with population of 10^6 or 10^7 cells. Future work is mostly on performing further experiments and studies in this direction, and integrating these approaches for multi-scale models.

Moreover, even though MDNs are quite an old model (they were first introduced in 1991) they have not seen wide adoption so far. There is not a consistent record of publications in the ML literature trying to specifically address the limitations and the best practices concerning these models. It is thus worthwhile to further research MDNs on their own, trying to work out the best way to design and train these neural networks, in particular for the applications we are concerned with.

References

[1] Daniel T. Gillespie. "Exact stochastic simulation of coupled chemical reactions". In: *The journal of physical chemistry* 81.25 (1977), pp. 2340–2361. (Visited on 11/28/2013).

[2] Christopher M Bishop. *Mixture density networks*. Tech. rep. NCRG/94/004. Neural Computing Research Group, Aston University, 1994.

[3] Yann LeCunn et al. "Neural Networks: Tricks of the Trade". In: Springer, 1998. Chap. Efficient Backprop, pp. 9–50. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf.

[4] S. Hochreiter et al. "A Field Guide to Dynamical Recurrent Neural Networks". In: ed. by S. C. Kremer and J. F. Kolen. IEEE Press, 2001. Chap. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[5] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer-Verlag New York, 2006.

[6] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. "Efficient step size selection for the tau-leaping simulation method". In: *The Journal of Chemical Physics* 124.4 (2006), p. 044109. DOI: 10.1063/1.2159468.

[7] Yang Cao and Linda Petzold. "Accuracy limitations and the measurement of errors in the stochastic simulation of chemically reacting systems". In: *Journal of Computational Physics* 212.1 (Feb. 2006), pp. 6–24. DOI: https://doi.org/10.1016/j.jcp.2005.06.012.

[8] D. J. Wilkinson. *Stochastic Modelling for Systems Biology*. Chapman & Hall, 2006.

[9] J. Pahle. "Biochemical simulations: stochastic, approximate stochastic and hybrid approaches". In: *Briefings in Bioinformatics* 10.1 (2008), pp. 53–64. DOI: 10.1093/bib/bbn050.

[10] Priscilla E. Greenwood and Luis F. Gordillo. "Mathematical and Statistical Estimation Approaches in Epidemiology". In: Springer Netherlands, 2009. Chap. Stochastic Epidemic Modeling, pp. 31–52.

[11] T. S. Deisboeck et al. "Multiscale Cancer Modeling". In: *Annual Review of Biomedical Engineering* 13.1 (2011), pp. 127–155. ISSN: 1523-9829, 1545-4274. DOI: 10.1146/annurev-bioeng-071910-124729.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* 15 (2011), pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[13] Bing Liu, David Hsu, and P.S. Thiagarajan. "Probabilistic approximations of ODEs based bio-pathway dynamics". In: *Theoretical Computer Science* 412 (2011), pp. 2188–2206.

[14] Kevin R. Sanft et al. "StochKit2: software for discrete stochastic simulation of biochemical systems with events". In: *Bioinformatics* 27.17 (Sept. 2011), pp. 2457–2458. DOI: https://dx.doi.org/10.1093%2Fbioinformatics%2Fbtr401.

[15] Erik Bernhardsson and Elias Freider. *Luigi*. 2012. URL: https://github.com/spotify/luigi.

[16] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2014). URL: https://arxiv.org/abs/1412.6980.

[17] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958.

[18] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[19] Chiara Bodei et al. "On the impact of discreteness and abstractions on modelling noise in gene regulatory networks". In: *Computational Biology and Chemistry* 56 (2015), pp. 98–108. DOI: 10.1016/j.compbiolchem.2015.04.004.

[20] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[21] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (Dec. 2015). URL: https://arxiv.org/abs/1512.03385.

[22] John H. Abel et al. "GillesPy: A Python Package for Stochastic Model Building and Simulation". In: *IEEE* (Sept. 2016), pp. 35–38. DOI: https://doi.org/10.1109/LLS.2017.2652448.

[23] I. Goodfellow, Y. Bengio, and A. Courvilleet. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[24] I. Goodfellow, Y. Bengio, and A. Courvilleet. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[25] Michalis Michaelides, Jane Hillston, and Guido Sanguinetti. "Statistical Abstraction for Multi-scale Spatio-Temporal Systems". In: *Quantitative Evaluation of Systems, QEST 2017,* 2017, pp. 243–258. DOI: 10.1007/978-3-319-66335-7_15.

[26] S. K. Palaniappan et al. "Abstracting the dynamics of biological pathways using information theory: a case study of apoptosis pathway". en. In: *Bioinformatics* (2017). ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/btx095.

[27] D. Schnoerr, G. Sanguinetti, and R. Grima. "Approximation and inference methods for stochastic biochemical kinetics - a tutorial review". In: *Journal of Physics A: Mathematical and Theoretical* 50.9 (2017), p. 093001. ISSN: 1751-8113, 1751-8121. DOI: 10.1088/1751-8121/aa54d9. (Visited on 04/20/2017).