

DeepCME: A deep learning framework for solving the Chemical Master Equation

Ankit Gupta^{*1}, Christoph Schwab^{†2}, and Mustafa Khammash^{‡1}

¹Department of Biosystems Science and Engineering, ETH Zürich, Mattenstrasse 26, 4058 Basel, Switzerland.

²Seminar für Angewandte Mathematik, ETH Zürich, 8092 Zürich, Switzerland.

June 5, 2021

Abstract

Stochastic models of biomolecular reaction networks are commonly employed in systems and synthetic biology to study the effects of stochastic fluctuations emanating from reactions involving species with low copy-numbers. For such models, the Kolmogorov’s forward equation is called the chemical master equation (CME), and it is a fundamental system of linear ordinary differential equations (ODEs) that describes the evolution of the probability distribution of the random state-vector representing the copy-numbers of all the reacting species. The size of this system is given by the number of states that are accessible by the chemical system, and for most examples of interest this number is either very large or infinite. Moreover, approximations that reduce the size of the system by retaining only a finite number of important chemical states (e.g. those with non-negligible probability) result in high-dimensional ODE systems, even when the number of reacting species is small. Consequently, accurate numerical solution of the CME is very challenging, despite the linear nature of the underlying ODEs. One often resorts to estimating the solutions via computationally intensive stochastic simulations.

The goal of the present paper is to develop a novel deep-learning approach for solving high-dimensional CMEs by reformulating the stochastic dynamics using Kolmogorov’s backward equation. The proposed method leverages superior approximation properties of Deep Neural Networks (DNNs) and is algorithmically based on reinforcement learning. It only requires a moderate number of stochastic simulations (in comparison to typical simulation-based approaches) to train the “policy function”. This allows not just the numerical approximation of the CME solution but also of its sensitivities to all the reaction network parameters (e.g. rate constants). We provide four examples to illustrate our methodology and provide several directions for future research.

Keywords: stochastic reaction networks; Chemical Master Equation; Deep Learning; Neural Networks, Kolmogorov’s backward equation

Mathematical Subject Classification (2010): 60J22; 60J27; 60H35; 65C40; 92E20

1 Introduction

Stochastic modelling in systems and synthetic biology has become an indispensable tool to quantitatively understand the intrinsically noisy dynamics within living cells [1]. Intracellular reaction networks typically involve low copy-number species in reactions that fire intermittently at random time, as opposed to continuously. Hence, deterministic models of such networks based on Ordinary Differential Equations (ODEs) fail to capture the essential properties of the system, and stochastic models become necessary [2].

*ankit.gupta@bsse.ethz.ch

†christoph.schwab@sam.math.ethz.ch

‡mustafa.khammash@bsse.ethz.ch

Among the most widely used stochastic models are continuous-time Markov chains (CTMCs) whose states represent the copy-numbers of all species involved in the Chemical Reaction Network (CRN) [3]. If the number of species in the CRN is n , the Markov chain evolves over a discrete, possibly infinite, state-space $\mathfrak{X} \subset \mathbb{N}_0^n$ comprising all accessible states. In most applications, the key object of interest is the probability distribution $p(t)$ of the random state $X(t) \in \mathfrak{X}$ at time t . This probability distribution evolves in time t according to Kolmogorov’s forward equation that is more famously known in the chemical literature as the Chemical Master Equation (CME) (see, e.g., [4], and (2.7)). The CME is a system of coupled, deterministic ODEs describing the rates of inflow and outflow of probability at each state in the state-space \mathfrak{X} . For even very small examples of CRNs, \mathfrak{X} can be very large or infinite, and hence the CME cannot be solved directly despite the linear nature of its constituent ODEs. Hence, one typically estimates CME solutions numerically either by simulating the CTMC trajectories with numerical methods like the Stochastic Simulation Algorithm (SSA) [5] or the Next Reaction Method [6], or one models (parts of) the CME asymptotically in various parameter regimes, such as the large copy-number limit, or the large systems limit (see, e.g., [3, 7] and the references therein). Then, Fokker-Planck PDEs govern the evolution of the limiting densities. Solutions of these PDEs are known to admit DNN approximations which are free from the “Curse of Dimensionality” (CoD), see e.g. [8] and the references there.

The main drawback of simulation-based solvers is that obtaining statistically precise estimates of the CME solution can be very cumbersome, due to the high cost of CTMC trajectory simulation. This led to the development of the *Finite State Projection* (FSP) method [9] that approximately solves the CME by truncating the state-space to a finite, tractable size. The FSP has been successfully used in many important biological studies with stochastic reaction network models. Over time, numerous algorithmic improvements to the original FSP method have been made, using advanced techniques such as Krylov Subspace approximations [10] or Tensor-Train representations [11]. Despite these advances, the scope of FSP’s applicability is still fairly limited because of the CoD inherent to the CME for complex CRNs: the dimension of the copy-number space of a large number of species involved in the CRN can be potentially prohibitive. With the algorithmic complexity of deterministic solution methods of the CME scaling exponentially with the number of species n , the CoD obviates the efficient numerical treatment of the CME for complex CRNs. In spite of these drawbacks, the SSA combined with FSP and its variants have emerged as the methodology of choice during the past decades for the computational exploration of complex CRNs in systems biology. This is mainly due to the lack of computational schemes that can effectively deal with CoD.

In the past decade, with the ubiquitous emergence of possibly massive, noisy data from natural biological CRNs, and the possibility of *engineering synthetic biological CRNs*, the question of efficient numerical simulation of CRNs has become pivotal. Indeed several tasks in computational biology strongly depend on the availability of *scalable, efficient computational tools to analyse large CRNs*. These include structure and parameter identification in large CRNs, assimilation of observable data into CRN models, Bayesian estimation of non-observable quantities of interest conditional on CRNs, among many others.

Recently, in the context of high-dimensional partial differential equations (PDEs), *deep-learning based numerical approaches* have been found highly effective in dealing with the CoD in these settings and appear efficient in numerical approximation of PDE solutions with high dimensional state and parameter spaces [12, 13, 14]. We refer to the survey [8] and the references therein. Importantly, several types of PDEs considered in these studies also arise from various asymptotic scalings (large copy-numbers, large systems limits) of large CRNs. (E.g. [15, 4, 16, 7]). Furthermore, DNNs have been shown to be at least as expressive as certain tensor-structured formats from numerical multi-linear algebra which for the CME were developed in [11], see [17].

Motivated by these advances and observations, in this paper we develop and explore corresponding deep-learning approaches for the efficient numerical solution of CMEs and for the related tasks of parameter estimation, and inference.

Before detailing our approach, we remark that leveraging Machine Learning (ML) based approaches for the numerical treatment of complex CRNs is, in our view, natural and critical: CRNs being themselves networks, any viable computational approach should, in some sense, mimic this structure in order to accommodate the complexity of CRNs. This is in line with our previous work on *tensor network based computational methods* (e.g. [11, 18]). On the other hand, ML-based computational methodologies for data assimilation and quantitative prediction of complex systems is currently undergoing intense development. We therefore expect that corresponding advances in computational ML, such as progress in interpretability and training methods for DNNs, will entail corresponding

methodological advances in the exploration of large, complex CRNs in biological systems engineering.

Next we briefly describe our ML approach to solving the CME. Instead of estimating the probability $p(t, x)$ for each state $x \in \mathfrak{X}$, one is often interested in learning the expectation of suitable real-valued functions g , referred to as the *output* function, under this probability distribution. Therefore, one is interested in the input-output map that associates an initial density $\{p(0, x_0) : x_0 \in \mathfrak{X}\}$ to¹

$$\mathbb{E}(g(X(t))) = \sum_{x \in \mathfrak{X}} g(x)p(t, x). \quad (1.1)$$

For example, if $g(x_1, \dots, x_n) = x_i^m$, for some $m \in \mathbb{N}_0$ and $i \in \{1, \dots, n\}$, then the output to be estimated is the m -th moment of the random copy-number of the i -th species at time t , i.e.

$$\mathbb{E}(g(X(t))) = \mathbb{E}(X_i^m(t)).$$

Another relevant example is when $g(x) = \mathbb{1}_A(x)$, the indicator function for some subset $A \subset \mathfrak{X}$ defined as

$$\mathbb{1}_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise.} \end{cases}$$

Then the output is the probability of the state $X(t)$ being in set A at time t

$$\mathbb{E}(g(X(t))) = \mathbb{P}(X(t) \in A).$$

One method of choice to numerically approximate the map $p(0, \cdot) \mapsto \mathbb{E}(g(X(t)))$ is by the SSA and its variants (e.g. [5, 19, 20, 21] and the references there) combined with ensemble averaging. Generally, this approach mandates a large number of sample paths, to achieve Monte-Carlo convergence to reasonable accuracy for $\mathbb{E}(g(X(t)))$ at fixed $t > 0$. In the present paper, we propose DeepCME, a *deep neural network* based methodology to emulate the above-mentioned map. Also in the present approach path simulation is required, during the DNN training phase. However, we find that the number of paths to achieve DNN training generally is lower than by direct use of Monte-Carlo estimator combined with SSA; accuracy is achieved through the generalization properties of DNNs rather than through approximation admissible sets of initial densities.

As is by now well-known in ML, an essential ingredient in DNN based approaches to emulate high-dimensional maps is the mathematical setup of suitable loss-functions which determine the training process. In DeepCME, we propose a particular loss function which is inspired by other, recent approaches in computational finance (e.g. [12] and the references therein). Specifically, using Kolmogorov’s backward equation, Kurtz’s random time change formulation [22] and Ito’s formula for jump processes, we identify an equation that the output quantity of interest $\mathbb{E}(g(X(t)))$ along with some “policy map” $\mathcal{V}(t, X(t))$ must *uniquely* satisfy for each stochastic trajectory $(X(t))_{t \geq 0}$ almost surely. Minimising a “loss” function that measures the error in this equation, allows us to train a deep neural network to learn the policy map and the quantity of interest $\mathbb{E}(g(X(t)))$ in a reinforcement learning framework. Remarkably, this approach also yields the sensitivities of the quantity of interest $\mathbb{E}(g(X(t)))$ w.r.t. all model parameters. Estimating these parametric sensitivities is important for many applications, but it is considered a difficult problem towards which a lot of research effort has recently been directed [23, 24, 25, 26, 27, 28, 29, 30, 31].

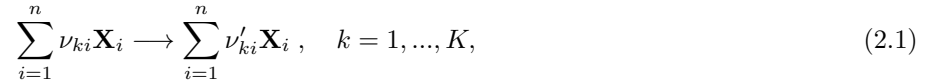
This paper is organised as follows. In Section 2 we provide some background on the CTMC model of a reaction network. In Section 3 we present our main results that allow us to cast the problem of solving a CME into the reinforcement learning framework. In Section 4 we describe our deep-learning approach and its implementation in TensorFlow. In Section 5 we illustrate this approach with four examples. Finally, in Section 6 we conclude and present directions for future research.

¹In the case $\#(\mathfrak{X}) = \infty$ this sum is formal for now. We later will indicate some sufficient conditions for this sum to be well-defined. Applying state-space truncation schemes e.g. [9], we may assume that $\#(\mathfrak{X}) < \infty$ holds with a small error in the estimated expectation, which renders the summation finite.

2 Preliminaries

2.1 The stochastic model

We start by describing the *continuous-time Markov chain* (CTMC) model of a reaction network. Consider a network with n species, denoted by $\mathbf{X}_1, \dots, \mathbf{X}_n$, that participate in K reactions of the form



where ν_{ki} (resp. ν'_{ki}) is the number of molecules of species \mathbf{X}_i consumed (resp. produced) by reaction k . The system's state $x = (x_1, \dots, x_n) \in \mathbb{N}_0^n$ at any time is the vectors of copy-numbers of the n species. As time advances, this state gets displaced by the *stoichiometric* vector $\zeta_k = (\nu'_{k1} - \nu_{k1}, \dots, \nu'_{kn} - \nu_{kn})$ when reaction k fires, and this event occurs at rate $\lambda_k(x)$ where $\lambda_k : \mathbb{N}_0^n \rightarrow [0, \infty)$ is the propensity function for reaction k . Commonly λ_k is given by mass-action kinetics [3])

$$\lambda_k(x_1, \dots, x_n) = c_k \prod_{i=1}^n \binom{x_i}{\nu_{ki}}, \quad (2.2)$$

with $c_k > 0$ being the associated rate constant.

There are many ways to formally specify the CTMC representing a reaction network. One way is through its generator, which is an operator that captures the rate of change of the probability distribution of the process (see Chapter 4 in [22]). It is given by

$$\mathbb{A}f(x) = \sum_{k=1}^K \lambda_k(x) (f(x + \zeta_k) - f(x)), \quad (2.3)$$

for any f that is a bounded real-valued function on the state-space $\mathfrak{X} \subset \mathbb{N}_0^n$ of the Markov chain. The state-space \mathfrak{X} is assumed to be nonempty and closed under the reaction dynamics, i.e. if $x \in \mathfrak{X}$ and $\lambda_k(x) > 0$ then $(x + \zeta_k)$ is also in \mathfrak{X} .

Another way to specify the CTMC is via Kurtz's random time change representation (see Chapter 6 in [22]).

$$X(t) = X(0) + \sum_{k=1}^K R_k(t) \zeta_k, \quad (2.4)$$

where $R_k(t)$ is a counting process that counts the number of firings of reaction k in the time-period $[0, t]$. As is customary in trajectory-simulation (e.g. [3, 19]) which will also be required by us for DNN training, we express it in terms of an independent unit rate Poisson process Y_k as e.g. [21].

$$R_k(t) = Y_k \left(\int_0^t \lambda_k(X(s)) ds \right). \quad (2.5)$$

With this representation in place, we consider the CTMC $(X(t))_{t \geq 0}$ on the canonical probability space generated by the independent unit Poisson processes Y_1, \dots, Y_K .

2.2 Kolmogorov's forward and backward equations

Let $(X(t))_{t \geq 0}$ be the CTMC representing reaction dynamics with some initial state $X(0) \in \mathfrak{X}$. For any state $x \in \mathfrak{X} \subset \mathbb{N}_0^n$, let

$$p(t, x) = \mathbb{P}(X(t) = x) \quad (2.6)$$

be the probability that the CTMC is in state x at time t . These probabilities evolve deterministically in time according to Kolmogorov's forward equation, more widely known as the *Chemical Master Equation* (CME) [3, 4]. The CME is the following system of deterministic linear ODEs

$$\frac{dp(t, x)}{dt} = \sum_{k=1}^K p(t, x - \zeta_k) \lambda_k(x - \zeta_k) - p(t, x) \sum_{k=1}^K \lambda_k(x), \quad (2.7)$$

for each $x \in \mathfrak{X}$. Note that the number of ODEs in this CME system is equal to $\#(\mathfrak{X})$, the number of elements in \mathfrak{X} , which is typically exorbitantly large or even infinite.

Consider an *output* function $g : \mathfrak{X} \rightarrow \mathbb{R}$ such that

$$\mathbb{E}(|g(X(T))|) < \infty \quad (2.8)$$

for some finite time horizon $T > 0$. The Kolmogorov's backward equation [32] describes the evolution of the martingale (w.r.t. the filtration generated by $(X(t))_{t \geq 0}$)

$$V_g(t, x) = \mathbb{E}(g(X(T)) | X(t) = x) \quad (2.9)$$

in the time interval $[0, T]$, and it is given by

$$\frac{\partial}{\partial t} V_g(t, x) = -\mathbb{A} V_g(t, x) = - \sum_{k=1}^K \lambda_k(x) (V_g(t, x + \zeta_k) - V_g(t, x)), \quad (2.10)$$

with the terminal condition $V_g(T, x) = g(x)$, $x \in \mathfrak{X}$. The backward equation (2.9) will play a key role in our development of a deep learning approach for estimating quantities of the form $\mathbb{E}(g(X(T)))$.

In the case where the state-space \mathfrak{X} is finite, i.e. $\#(\mathfrak{X}) = m < \infty$, we can enumerate it as $\mathfrak{X} = \{x^{(1)}, \dots, x^{(m)}\}$. Then the CTMC generator \mathbb{A} in (2.10) can be expressed as the $m \times m$ transition rate matrix $Q = [Q_{ij}]$ given by²

$$Q_{ij} = \begin{cases} -\sum_{k=1}^K \lambda_k(x^{(i)}) & \text{if } i = j \\ \lambda_k(x^{(i)}) & \text{if } x^{(j)} = x^{(i)} + \zeta_k \text{ for some } k \\ 0 & \text{otherwise.} \end{cases}$$

Viewing $p(t)$ as the vector $p(t) = (p(t, x^{(1)}), \dots, p(t, x^{(m)})) \in [0, 1]^{\#(\mathfrak{X})}$, we can express the CME (2.7) as

$$\frac{dp}{dt} = Q^\top p(t), \quad t \geq 0. \quad (2.11)$$

Here, $Q_{ij}^\top := Q_{ji}$, $i, j \in 1 : m$. CME (2.11) admits the closed-form solution

$$p(t) = \exp(tQ^\top) p(0) \quad \text{for any } t \geq 0. \quad (2.12)$$

Similarly, viewing $V_g(t)$ as the vector $(V_g(t, x^{(1)}), \dots, V_g(t, x^{(m)})) \in \mathbb{R}^{\#(\mathfrak{X})}$, the backward equation (2.10) can be solved as

$$V_g(t) = \exp(-Q(T-t)) g \quad \text{for any } t \in [0, T] \quad (2.13)$$

where g denotes the vector $g = (g(x^{(1)}), \dots, g(x^{(m)})) \in \mathbb{R}^{\#(\mathfrak{X})}$. We are interested in networks where $m = \#(\mathfrak{X})$ is extremely large or infinite. Then, numerically computing matrix exponentials in (2.12) or in (2.13) is not an option.

2.3 Parametric sensitivity analysis

Now consider the situation where the propensity functions depend on a scalar parameter θ (like reaction rate constant for mass-action kinetics, temperature etc.). Denoting the θ -dependent CTMC as $(X_\theta(t))_{t \geq 0}$, it is often of interest to compute the parametric sensitivity

$$S_\theta(g, T) = \frac{\partial}{\partial \theta} \mathbb{E}(g(X_\theta(T))), \quad (2.14)$$

²We assume for convenience that all stoichiometry vectors (ζ_k -s) are distinct.

of the observed output $\mathbb{E}(g(X_\theta(T)))$ at time T . Such sensitivity values are important for many applications and their direct calculation is generally impossible but a number of simulation-based approaches have recently been developed to provide efficient numerical estimation of these sensitivity values; we mention only [23, 24, 25, 26, 27, 28, 29, 30, 31].

Theorem 3.3 in [29] proves that

$$S_\theta(g, T) = \sum_{k=1}^K \mathbb{E} \left(\int_0^T \frac{\partial \lambda_k(X_\theta(t), \theta)}{\partial \theta} (V_g(t, X_\theta(t) + \zeta_k) - V_g(t, X_\theta(t))) dt \right), \quad (2.15)$$

where $V_g(t, x)$ is defined by (2.9) with $X(\cdot)$ replaced by $X_\theta(\cdot)$. The main difficulty in using this formula for computing sensitivities is that the function

$$\Delta_k V_g(t, x) := V_g(t, x + \zeta_k) - V_g(t, x) \quad (2.16)$$

is unknown and hence it must be estimated “on the fly” by numerically generating auxiliary paths [29]. In the method we develop in this paper we shall “learn” (i.e., emulate by ML techniques) this function using deep neural networks. This would provide a simple direct way to estimate the parameter sensitivity via formula (2.15). This approach would in fact yield sensitivities w.r.t. to *all* model parameters in one shot, unlike what is afforded by existing sensitivity estimation approaches.

3 Main Results

In this section we state and prove the key result which our deep learning approach depends on. Recall that our goal is to estimate $\mathbb{E}(g(X(t)))$ (see (1.1)) which is same as $V_g(0, x_0)$ (see (2.9)) if the initial state of the CTMC is $X(0) = x_0$. Also recall the random time-change representation (2.4) and the definition of the reaction counting process R_k from (2.5). Henceforth we shall denote the *centred* version of this process as

$$\tilde{R}_k(t) := Y_k \left(\int_0^t \lambda_k(X(s)) ds \right) - \int_0^t \lambda_k(X(s)) ds, \quad k = 1, \dots, K. \quad (3.1)$$

This centred process is a local martingale w.r.t. the filtration $\mathcal{F}_X(t)$ generated by $(X(t))_{t \geq 0}$ (see Chapter 1 in [33]).

We now state an assumption that we require for our approach.

Assumption 3.1 (Non-explosivity of the CTMC) *Let $(X(t))_{t \geq 0}$ be the CTMC given by (2.4) with deterministic initial condition $X(0) = x_0$. Let $\mathfrak{X} \subset \mathbb{N}_0^q$ denote the state-space of this CTMC and let $\mathcal{F}_X(t)$ be the filtration it generates. If τ_M is the $\mathcal{F}_X(t)$ -stopping time defined by*

$$\tau_M = \inf\{t \geq 0 : \|X(t)\| \geq M\},$$

then $\tau_M \rightarrow \infty$ almost surely as $M \rightarrow \infty$.

Remark 3.2 *There are a number of works in the literature that provide sufficient conditions for this non-explosivity condition to hold, subject to the form of the propensity functions (see for example in [34, 35, 36] and the references therein). Under the no-explosion assumption a probability distribution $p(t)$ satisfying the CME exists uniquely (see e.g. Lemma 1.23 in [33]).*

We next present the main result on which our deep learning approach is based.

Theorem 3.3 (Expected output and policy map characterization) *Suppose Assumption 3.1 holds for the CTMC $(X(t))_{t \geq 0}$ and the output function $g : \mathfrak{X} \rightarrow \mathbb{R}$ satisfies (2.8). Let \mathcal{Y} be a real number and let $\mathcal{V}(t, x) = (\mathcal{V}_1(t, x), \dots, \mathcal{V}_K(t, x))$ be a measurable \mathbb{R}^K -valued function on $[0, T] \times \mathfrak{X}$ such that the following relation holds almost surely*

$$g(X(T)) = \mathcal{Y} + \sum_{k=1}^K \int_0^T \mathcal{V}_k(t, X(t)) d\tilde{R}_k(t). \quad (3.2)$$

Then \mathcal{Y} and $\mathcal{V}(t, x)$ exist uniquely and they can be identified as

$$\mathcal{Y} = \mathbb{E}(g(X(T))) \quad \text{and} \quad \mathcal{V}(t, x) = (\Delta_1 V_g(t, x), \dots, \Delta_K V_g(t, x)) \quad (3.3)$$

where $V_g(t, x)$ is given by (2.9) and the difference operator Δ_k is as in (2.16).

Remark 3.4 (Connection to our deep learning approach) Before we prove this theorem, we briefly describe how this result translates into our deep learning approach whose details will be provided in Section 4. Using the modified next reaction method [6], one can easily generate trajectories of the CTMC $(X(t))_{t \geq 0}$ along with the associated centred reaction counting processes $(\tilde{R}_1(t), \dots, \tilde{R}_K(t))_{t \geq 0}$. For each such trajectory, relation (3.2) can be interpreted in terms of known and unknown quantities as

$$\underbrace{g(X(T))}_{\text{known}} = \underbrace{\mathcal{Y}}_{\text{unknown}} + \sum_{k=1}^K \int_0^T \underbrace{\mathcal{V}_k(t, X(t))}_{\text{unknown}} \underbrace{d\tilde{R}_k(t)}_{\text{known}}. \quad (3.4)$$

We represent the unknown map $(t, x) \mapsto \mathcal{V}(t, x)$ by a deep neural network (DNN) and consider unknown \mathcal{Y} as an optimisation variable. Then by minimising a “loss” function $\mathcal{L}(\mathcal{Y}, \mathcal{V})$ that measures the discrepancy in relation (3.4) we try to recover the optimal values of \mathcal{Y} and \mathcal{V} that are given by (3.3). This allows us to estimate the quantity of interest $\mathbb{E}(g(X(T)))$ (as \mathcal{Y}) and also its parametric sensitivities by substituting $\mathcal{V}_k(t, x)$ for $\Delta_k V_g(t, x)$ in (2.15).

Proof.[Proof of Theorem 3.3] We prove this result in two steps. We first show that \mathcal{Y} and $\mathcal{V}(t, x)$ given by (3.3) satisfy (3.2) almost surely. Then, we prove that if another such pair $(\hat{\mathcal{Y}}, \hat{\mathcal{V}}(t, x))$ satisfying (3.2) exists we must necessarily have $\hat{\mathcal{Y}} = \mathcal{Y}$ and $\hat{\mathcal{V}}(t, x) = \mathcal{V}(t, x)$.

Applying Ito’s formula for jump Markov processes to $V_g(t, X(t))$ we obtain

$$V_g(T, X(T)) = V_g(0, x_0) + \int_0^T \frac{\partial}{\partial t} V_g(t, X(t)) dt + \sum_{k=1}^K \int_0^T \Delta_k V_g(t, X(t)) d\tilde{R}_k(t).$$

Using Kolmogorov’s backward equation (2.10) and simplifying we get

$$V_g(T, X(T)) = V_g(0, X(0)) + \sum_{k=1}^K \int_0^T \Delta_k V_g(t, X(t)) d\tilde{R}_k(t). \quad (3.5)$$

Noting that $V_g(T, X(T)) = g(X(T))$ and $V_g(0, X(0)) = \mathbb{E}(g(X(T)))$ we see that (3.2) holds with \mathcal{Y} and $\mathcal{V}(t, x)$ given by (3.3).

Now let $(\hat{\mathcal{Y}}, \hat{\mathcal{V}}(t, x))$ be another pair satisfying (3.2), i.e.

$$g(X(T)) = \hat{\mathcal{Y}} + \sum_{k=1}^K \int_0^T \hat{\mathcal{V}}_k(t, X(t)) d\tilde{R}_k(t).$$

We subtract (3.5) from this equation to obtain

$$\Delta \hat{\mathcal{Y}} + \sum_{k=1}^K \int_0^T \Delta \hat{\mathcal{V}}_k(t, X(t)) d\tilde{R}_k(t) = 0 \quad (3.6)$$

where

$$\Delta \hat{\mathcal{Y}} = \hat{\mathcal{Y}} - \mathbb{E}(g(X(T))) \quad \text{and} \quad \Delta \hat{\mathcal{V}}_k(t, X(t)) = \hat{\mathcal{V}}_k(t, X(t)) - \Delta_k V_g(t, X(t)).$$

Note that

$$m(t) = \Delta \hat{\mathcal{Y}} + \sum_{k=1}^K \int_0^t \Delta \hat{\mathcal{V}}_k(s, X(s)) d\tilde{R}_k(s)$$

is a local martingale and if τ_M is the stopping time defined in Assumption 3.1, then the stopped process $m(t \wedge \tau_M)$ is a martingale, where $a \wedge b := \min\{a, b\}$. Applying Doob's maximal inequality [22] on the submartingale $|m(t \wedge \tau_M)|$ we obtain

$$\mathbb{E} \left[\left(\sup_{0 \leq t \leq T \wedge \tau_M} |m(t)| \right)^2 \right] \leq 4\mathbb{E}(m(T \wedge \tau_M)^2). \quad (3.7)$$

Note that terms on both sides of the inequality are monotonically increasing in M . This monotonicity is obvious for the term on the l.h.s. . For the term on the r.h.s. it follows from the conditional Jensen's inequality and from the martingale property

$$\begin{aligned} \mathbb{E}(m(T \wedge \tau_{M+1})^2) &= \mathbb{E} \left[\mathbb{E} (m(T \wedge \tau_{M+1})^2 | \mathcal{F}_X(T \wedge \tau_M)) \right] \\ &\geq \mathbb{E} \left[(\mathbb{E} (m(T \wedge \tau_{M+1}) | \mathcal{F}_X(T \wedge \tau_M)))^2 \right] \\ &= \mathbb{E}(m(T \wedge \tau_M)^2). \end{aligned}$$

Letting $M \rightarrow \infty$ and using the monotone convergence theorem on both sides of (3.7) we obtain

$$\mathbb{E} \left[\left(\sup_{0 \leq t \leq T} |m(t)| \right)^2 \right] \leq 4\mathbb{E}(m(T)^2)$$

where we have used the fact that $\tau_M \rightarrow \infty$ as $M \rightarrow \infty$ due to Assumption 3.1. Relation (3.6) informs us that $m(T) = 0$ almost surely and hence

$$\mathbb{E} \left[\left(\sup_{0 \leq t \leq T} |m(t)| \right)^2 \right] = 0.$$

This is sufficient to conclude that $\Delta \widehat{\mathcal{Y}} = 0$ and $\Delta \widehat{\mathcal{V}}_k(t, X(t)) = 0$ for any $t \in [0, T]$.

As this holds for any CTMC trajectory $(X(t))_{t \geq 0}$, we must have $\Delta \widehat{\mathcal{V}}_k(t, x) = 0$ for any $(t, x) \in [0, T] \times \mathfrak{X}$. This completes the proof of this theorem. \square

4 DeepCME: Deep Learning Formulation for CME

In this section we detail our deep learning method for solving CME, referred to as *DeepCME*. We have computationally implemented this method using the machine learning library `TensorFlow` [37].

As outlined in Remark 3.4, our approach is based on the almost sure relationship established in Theorem 3.3. Even though this result was presented for a single output function $g(x)$, it can be easily extended for a vector-valued function $g(x) = (g_1(x), \dots, g_R(x))$ by considering the unknown variable \mathcal{Y} as a R -dimensional vector and the unknown map $\mathcal{V}(t, x)$ as taking values in the space of $R \times K$ matrices. Such an extension is useful because in most applications one is interested in estimating multiple statistical properties (like means, variances, covariances etc.) of the CME solution $p(T)$.

We now define the ‘‘loss’’ function $\mathcal{L}(\mathcal{Y}, \mathcal{V})$ that measures the discrepancies in the R almost sure relations given by (3.4). Let $\mathbf{L} : \mathbb{R}^R \rightarrow [0, \infty)$ be the following continuously differentiable function

$$\mathbf{L}(x_1, \dots, x_R) = \sum_{i=1}^R \phi \left(\frac{x_i}{\Delta_i} \right)$$

where $\Delta = (\Delta_1, \dots, \Delta_R)$ is a vector of positive threshold values and

$$\phi(x) = \begin{cases} x^2 & \text{if } |x| < 1 \\ 2|x| - 1 & \text{otherwise.} \end{cases}$$

We define the loss function as

$$\mathcal{L}(\mathcal{Y}, \mathcal{V}) = \mathbb{E} \left[\mathbf{L} \left(g(X(T)) - \mathcal{Y} - \sum_{k=1}^K \int_0^T \mathcal{V}_k(t, X(t)) d\tilde{R}_k(t) \right) \right], \quad (4.1)$$

where the expectation is estimated by computing the sample mean over a finite *batch* of “training” trajectories. During the training process this loss function is minimised in order to learn the optimal \mathcal{Y} , which estimates our quantities of interest

$$\mathbb{E}(g(X(T))) = (\mathbb{E}(g_1(X(T))), \dots, \mathbb{E}(g_R(X(T)))) ,$$

and the optimal matrix-valued map $\mathcal{V}(t, x)$, commonly called the “policy” map in the parlance of reinforcement learning. This policy map will enable us to estimate sensitivities of the quantities of interest w.r.t. all the model parameters as discussed in Section 2.3. The threshold values $\Delta = (\Delta_1, \dots, \Delta_R)$ help in neutralising the disparities in the relative magnitudes of the estimated quantities and the discrepancies in the corresponding almost sure relations. The loss function minimisation is performed with *stochastic gradient descent* (SGD) algorithm that makes use of the automatic differentiation routines that are built in `TensorFlow`. Differentiability properties of the function \mathbf{L} which defines the loss function are important for convergence of the SGD iterations.

Next we discuss the structure of the deep neural network (DNN) which encodes the matrix-valued policy map $(t, x) \mapsto \mathcal{V}(t, x)$. Recall from Section 2.2 that if the state-space is finite then $V_g(t, x)$ can in principle be found by exponentiating the transition rate matrix Q multiplied with $(T - t)$ (see (2.13)). Hence, if $\lambda = \lambda_1 + i\lambda_2$ is an eigenvalue of Q , then on the associated eigenspace we would expect that the dependence of $V_g(t, x)$ on time t is given by $e^{\lambda(T-t)} = e^{\lambda_1(T-t)}(\cos(\lambda_2(T-t)) + i \sin(\lambda_2(T-t)))$. Motivated by this rationale, rather than passing the time-values t directly as inputs to the DNN that encodes $\mathcal{V}(t, x)$, we shall pass *temporal features* of the form

$$\mathcal{T}(t) = (e^{\lambda_{11}(T-t)}, \dots, e^{\lambda_{r1}(T-t)}, \sin(\lambda_{12}(T-t) + \psi_1), \dots, \sin(\lambda_{r2}(T-t) + \psi_r)), \quad (4.2)$$

where $\lambda_{11}, \dots, \lambda_{r1}, \lambda_{12}, \dots, \lambda_{r2}$ are $2r$ trainable variables that represent the r dominant eigenvalues of the generator of the CTMC. Additionally, r trainable variables ψ_1, \dots, ψ_r are included to represent ‘phase shifts’. Problem-specific temporal features, like the ones we consider, have been successfully employed in existing deep learning methods for ODE-based reaction network models (see, e.g., [38] and the references therein). Note that the mapping between time t and the temporal features $\mathcal{T}(t)$ is one-to-one and hence no information is lost by substituting time inputs with temporal features.

We encode the policy map $(t, x) \mapsto \mathcal{V}(t, x)$ as fully connected *feedforward* neural network whose architecture is schematically shown in Figure 1.

A feedforward deep neural network (DNN for short) consists of an input layer, L hidden layers and an output layer. Mathematically, DNNs Φ considered here are determined by a tuple

$$\Phi = ((T_1, \rho_1), \dots, (T_{L+1}, \rho_{L+1})) , \quad (4.3)$$

affine transformations $T_\ell : \mathbb{R}^{N_{\ell-1}} \rightarrow \mathbb{R}^{N_\ell}$, ie., $T_\ell(x) = W_\ell x + b_\ell$, $\ell = 1, \dots, L+1$, with *weight matrices* $W_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$, and *bias vectors* $b_\ell \in \mathbb{R}^{N_\ell}$. As mentioned, in the presently considered DNNs, the input layer takes the temporal features $\mathcal{T}(t)$ and the state vector $x = (x_1, \dots, x_n)$.

The nonlinearities $\rho_\ell : \mathbb{R}^{N_\ell} \rightarrow \mathbb{R}^{N_\ell}$ in (4.3) act on vectors in \mathbb{R}^{N_ℓ} component-wise, with possibly different activations at each layer. The number $L+1$ denotes the *number of layers* (sometimes referred to as depth) of the DNN Φ , and L denotes the *number of hidden layers* of DNN Φ .

With the DNN Φ , we associate a *realization*, i.e., a map

$$R(\Phi) : \mathbb{R}^{N_0} \rightarrow \mathbb{R}^{N_{L+1}} , \text{ where } R(\Phi) := \rho_{L+1} \circ T_{L+1} \circ \dots \circ \rho_1 \circ T_1 .$$

The relation between Φ and its realisation $R(\Phi)$ as a map is not one-to-one.

The goal of DNN approximations is to provide a parsimonious surrogate map for many-parametric, input-output maps.

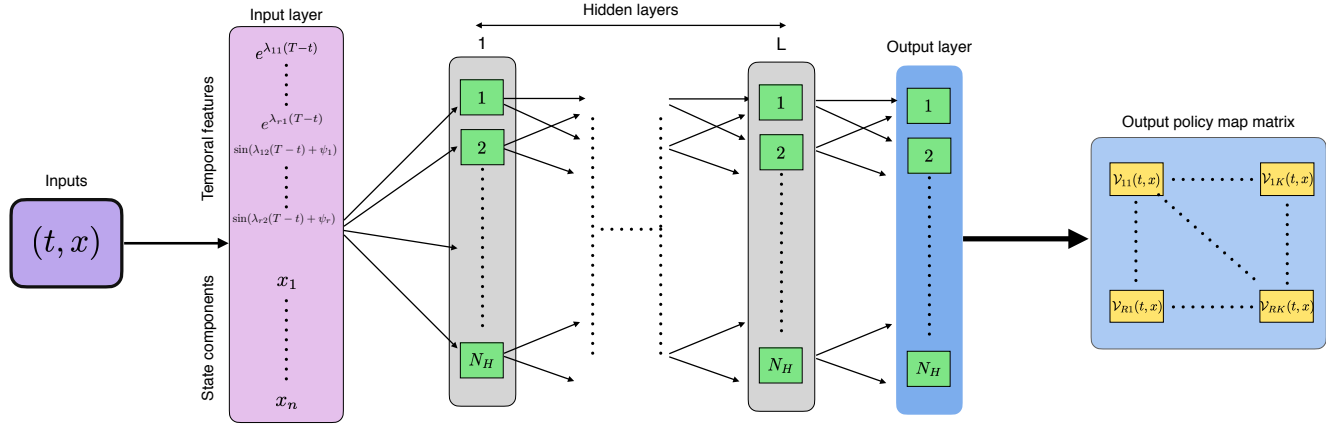


Figure 1: **Architecture of the neural network:** DNN architecture to encode the matrix-valued map $(t, x) \mapsto \mathcal{V}(t, x)$. The inputs (t, x) are passed to an input layer, which leaves the state values x unchanged but activates a dictionary of *temporal features* (4.2). The resulting output is propagated through a DNN with L fully connected hidden layers, and an additional output layer with each layer having N_H nodes. For simplicity, we assume no sparsity in the weight matrices and the bias vectors of these layers. In the final step, the output from the output layer is cast into the policy-map matrix $\mathcal{V}(t, x)$ corresponding to inputs (t, x) .

The input layer transforms the time-value t into temporal features (4.2) but leaves the state vector $x = (x_1, \dots, x_n)$ unchanged. For the layers, we assume fixed width, i.e., that each layer consists of N_H nodes (including the output layer). We also assume that no activation is applied at the output layer, i.e. ρ_{L+1} is the identity function,

and all activations in the hidden layers are equal, i.e. for $\ell = 1, \dots, L$ and for $i = 1, \dots, N_H$, $\varrho = (\rho_\ell)_i : \mathbb{R} \rightarrow \mathbb{R}$. In the ensuing numerical examples, we employ the so-called *ReLU-activation* for the hidden layers, which is given by

$$\varrho(x) := \text{ReLU}(x) = \max\{x, 0\}, \quad x \in \mathbb{R}. \quad (4.4)$$

Remark 4.1 *More generally, for $k \in \mathbb{N}$, we may choose the activations $\varrho^k(x)$, observing that increasing the value of k increases differentiability of realizations of the DNN Φ . This may be of relevance in cases where the diffusion limits for large copy number counts of particular species imply higher smoothness of the map $x \mapsto p(t, x)$.*

To numerically evaluate the loss function, we require simulated trajectories of the form $(X(t), \tilde{R}(t))_{t \geq 0}$ where X denotes the CTMC and each $\tilde{R} = (\tilde{R}_1, \dots, \tilde{R}_K)$ is the vector of centred reaction counting processes defined by (3.1). Such trajectories can be easily generated with Anderson's modified next reaction method [6]. We discretise the time-interval $[0, T]$ as

$$0 = t_0 < t_1 < \dots < t_J = T$$

and for each simulated trajectory we compute \mathcal{Y}_j iteratively as

$$\mathcal{Y}_j = \mathcal{Y}_{j-1} + \mathcal{V}(t_{j-1}, X(t_{j-1}))(\tilde{R}(t_j) - \tilde{R}(t_{j-1})) \quad \text{for } j = 1, \dots, J,$$

with $\mathcal{Y}_0 = \mathcal{Y}$. Here each \mathcal{Y}_j is a $R \times 1$ vector, $\mathcal{V}(t_{j-1}, X(t_{j-1}))$ is a $R \times K$ matrix and $(\tilde{R}(t_j) - \tilde{R}(t_{j-1}))$ is a $K \times 1$ vector. Following this scheme we can compute $\mathcal{Y}_j^{(q)}$ for the q -th simulated trajectory $(X^{(q)}(t), \tilde{R}^{(q)}(t))_{t \geq 0}$. With M i.i.d such trajectories, the *loss function* (4.1) can be estimated as

$$\hat{\mathcal{L}}(\mathcal{Y}, \mathcal{V}) := \frac{1}{M} \sum_{q=1}^M \mathbf{L} \left(g(X^{(q)}(T)) - \mathcal{Y}_J^{(q)} \right). \quad (4.5)$$

Here, we made use of (3.6).

Remark 4.2 In the loss function (4.1) and its MC estimate (4.5), one could add a sparsity-promoting regularization term, in which case (4.1) would become

$$\widehat{\mathcal{L}}(\mathcal{Y}, \mathcal{V}) := \frac{1}{M} \sum_{q=1}^M \mathbf{L} \left(g(X^{(q)}(T)) - \mathcal{Y}_J^{(q)} \right) + \mu \mathcal{P}(\Phi). \quad (4.6)$$

Here, $\mu \geq 0$ is a penalty parameter and $\mathcal{P}(\Phi)$ promotes sparsity in weights W_ℓ and biases b_ℓ comprising Φ . In the numerical experiments to be reported, we did not use this device.

Remark 4.3 When the time-interval $[0, T]$ is large, instead of using a single DNN to approximate the policy map $(t, x) \mapsto \mathcal{V}(t, x)$, it may be beneficial to employ multiple temporal DNNs that are uniformly distributed in the time-interval $[0, T]$. All these DNNs have the same structure, as shown in Figure 1. If N_T such DNNs are employed, then we use the m -th DNN to represent the policy map $(t, x) \mapsto \mathcal{V}(t, x)$ for $t \in [(m-1)\delta, m\delta]$ where $m = 1, \dots, N_T$ and $\delta = T/N_T$. Distributing DNNs across time would reduce the complexity of the policy map (as a function of time t) that needs to be learned. This is helpful in scenarios where the stochastic dynamics has intricate temporal features, such as oscillations.

5 Examples

We now present four examples to illustrate our *DeepCME* method. All these examples are reaction networks with n species, denoted by $\mathbf{X}_1, \dots, \mathbf{X}_n$, and $2n$ reactions. By varying n , we shall investigate how the *DeepCME* method performs as the network gets larger and compare its performance with simulation based methods.

In all the examples, we assume that all the species have zero copy-numbers initially, and we consider two output functions $g_1(x) = x_n$ and $g_2(x) = x_n^2$ whose expectation is to be estimated under the probability distribution given by the CME solution at time $T = 1$. In other words, we shall use *DeepCME* to estimate the first two moments of the copy-number of species \mathbf{X}_n at time T , viz.

$$\mathbb{E}(g_1(X(T))) = \mathbb{E}(X_n(T)) \quad \text{and} \quad \mathbb{E}(g_2(X(T))) = \mathbb{E}(X_n^2(T)). \quad (5.1)$$

We shall compare these estimates to those obtained by simulating 1000 CTMC trajectories with SSA [5]. Our method *DeepCME* also yields estimates of the sensitivities of the estimated moments (5.1) w.r.t. all model parameters. We plot these estimates and compare them with the estimates obtained via the simulation-based *Bernoulli Path Algorithm* (BPA) [29]. These latter estimates are based on a sample of size 1000 and for each sample BPA requires generation of a certain number of auxiliary paths (see Section 2.3) which we set to be 10 in our examples.

In all the examples, we encode the policy map $(t, x) \mapsto \mathcal{V}(t, x)$ as a DNN with $L = 2$ hidden layers and $N_H = 4$ nodes per layer (see Figure 1), irrespective of the number of species n . The activation function for all hidden layer nodes is $\text{ReLU}(x)$ (see (4.4)) and we choose $r = 1$ for the temporal features (4.2) to transform the time-values. For loss function computation, we partition the time-interval $[0, T]$ into $J = 50$ equal size time-increments.

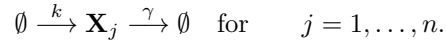
The neural network is trained with a training batch of 100 trajectories generated a priori with the modified next reaction method [6] (see Section 4), and another such batch of 100 trajectories is used for validation. We display the loss function for the validation trajectories to track the training process. For facilitate comparison across network sizes, we normalise all the loss function trajectories to be one at the start of training. Finally, note that the definition of our loss function (4.1) depends on certain threshold values $\Delta = (\Delta_1, \Delta_2)$. We choose these values as

$$\Delta_j = 1 + |\widehat{\mu}_j| + 2\widehat{\sigma}_j$$

where $\widehat{\mu}_j$ (resp. $\widehat{\sigma}_j$) denotes the sample mean (resp. standard deviation) of the values of the output function g_j for the trajectories in the training batch.

5.1 Independent birth death network

As our first example (see Figure 2(A)), we consider a network of n species that are all undergoing independent birth-death reactions



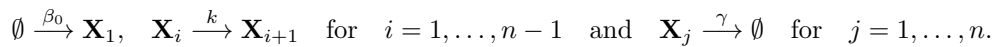
We set $k = 10$ and $\gamma = 1$. The propensity functions obey mass-action kinetics and are hence affine functions of the state variable x .

For $n = 5, 10, 20$ many species, we apply DeepCME to this reaction network by training the neural network for 10'000 SGD iterations. Based on the trained neural network, we compute estimates of the first two moments (5.1) and their sensitivities to both the model parameters k and γ . We also estimate these quantities with simulation-based methods (SSA and BPA) with 1000 samples, and since the propensity functions are linear we can compute these quantities exactly as well. In plots shown in Figure 2(D-F), we compare the estimates from all these approaches for various values of n . Observe that DeepCME is in general quite accurate in estimating both the moments and their parametric sensitivities, but there are a few cases where the error is significant (e.g. sensitivity w.r.t. γ for $\mathbb{E}(X_n^2(T))$). These errors can in principle be reduced by employing a larger neural network to encode the policy map. In our experience, these errors were also reduced in some cases by including a sparsity promoting term in the loss function (see Remark 4.2) but the result was highly sensitive to the relative weight (i.e. parameter μ in (4.6)) of this term (data not shown).

The *central processing unit* (CPU) time required by DeepCME and simulation-based methods for obtaining moment and sensitivity estimates are plotted in Figure 2(B). Note that the CPU time for simulation-based methods grows linearly with the network size n , but for DeepCME this growth is sub-linear owing to the fixed structure of the underlying neural network. Despite this fixed structure, the loss function trajectories for DeepCME are similar for all n (see Figure 2(C)), indicating that the training process has low dependence on the number of species, probably because the species are evolving independently.

5.2 Linear signalling cascade

Our second example is a linear cascade with n -species (see Figure 3(A)), where species \mathbf{X}_i catalyses the production of species \mathbf{X}_{i+1} . The $2n$ reactions are given by



We set $\beta_0 = 10$, $k = 5$ and $\gamma = 1$. As in the previous example, all the propensity functions obey mass-action kinetics and are hence affine functions of the state.

For number of species $n = 2, 5, 10$, we apply DeepCME to this reaction network by training the neural network for 10'000 SGD iterations. Then we compute the moment estimates (5.1) and their sensitivities to all the model parameters. These quantities are also estimated with simulation-based methods (SSA and BPA) with 1000 samples, and as with the previous example, the linearity of the propensity functions enables us to compute these quantities exactly as well. In the plots shown in Figure 3(D-F), we compare the estimates from all these approaches for various values of n . Observe that DeepCME is accurate in estimating the moments but some of the parameter sensitivity estimates are not very accurate (e.g. sensitivity w.r.t. k for $\mathbb{E}(X_n^2(T))$) and $n = 20$. This is probably because the neural network we use is not sufficiently “deep” for large n and the training process is less successful, as indicated by the loss function trajectories shown in Figure 3(C). The CPU times for DeepCME and simulation-based methods are plotted in Figure 3(B), and as expected they show sub-linear growth w.r.t. n for the former but linear growth for the latter.

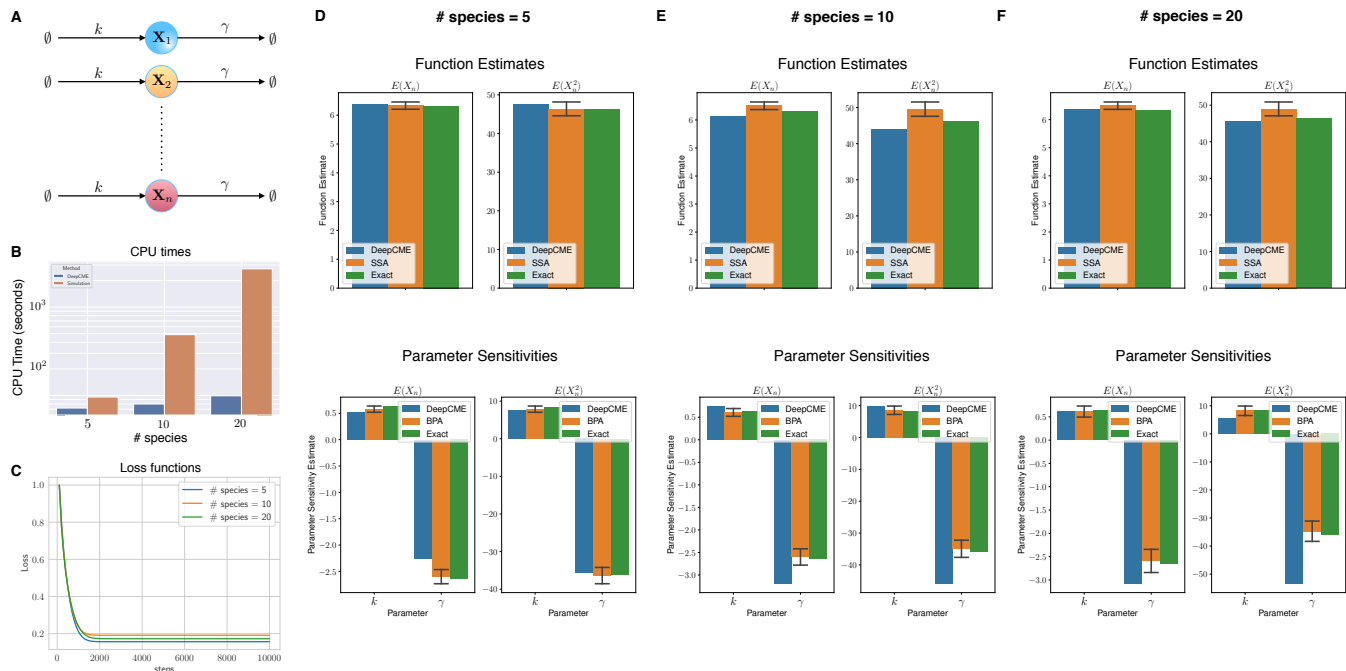


Figure 2: **Independent birth death network:** (A) Depicts the network with n species and $2n$ reactions with mass-action kinetics. (B) The CPU times are shown for training the DNN for different values of n (denoted as # species), and for comparison the time needed to generate 1000 SSA trajectories is also indicated. (C) Plots the loss function w.r.t. training steps for various n values. Panels (D-F) show estimates for the function values ($\mathbb{E}(X_n(T))$ and $\mathbb{E}(X_n^2(T))$) and the parameter sensitivities. The estimates with simulation based methods (SSA for function estimates and BPA for parameter sensitivities) are shown as 95% confidence intervals with 1000 samples.

5.3 Nonlinear signalling cascade

We now consider a variant of the network in Example 2 where the catalytic production of species X_{i+1} by species X_i is non-linear (see Figure 4(A)) and is given by an activating Hill propensity with a basal rate

$$\mathcal{H}(x) = b + \frac{k_m x_i^H}{k_0 + x_i^H} \quad (5.2)$$

where $b = 1$, $k_m = 100$, $k_0 = 10$ and $H = 1$. Other reactions have mass-action kinetics as in Example 2, with the same rate constants $\beta_0 = 10$ and $\gamma = 1$.

For number of species $n = 2, 5, 10$, we apply DeepCME to this reaction network by training the neural network for 10'000 SGD iterations. Then we compute the moment estimates (5.1) and their sensitivities to all the model parameters. These quantities are also estimated with simulation-based methods (SSA and BPA) with 1000 samples, and unlike previous examples we cannot compute these quantities exactly due to nonlinear propensities. In the plots shown in Figure 4(D-F), we compare the estimates from DeepCME and simulation-based approaches for various values of n . Observe that DeepCME is reasonably accurate in estimating the moments and their parametric sensitivities for all values of n . The success of the training process is shown by the loss function profiles shown in Figure 4(C). Note that these loss functions increase monotonically with n and this is consistent with the observation that errors in DeepCME-estimated quantities increases with n (see Figure 4(D-F)). The CPU times for DeepCME and simulation-based methods are displayed in Figure 4(B), and as in previous examples they show sub-linear growth w.r.t. n for the former but linear growth for the latter.

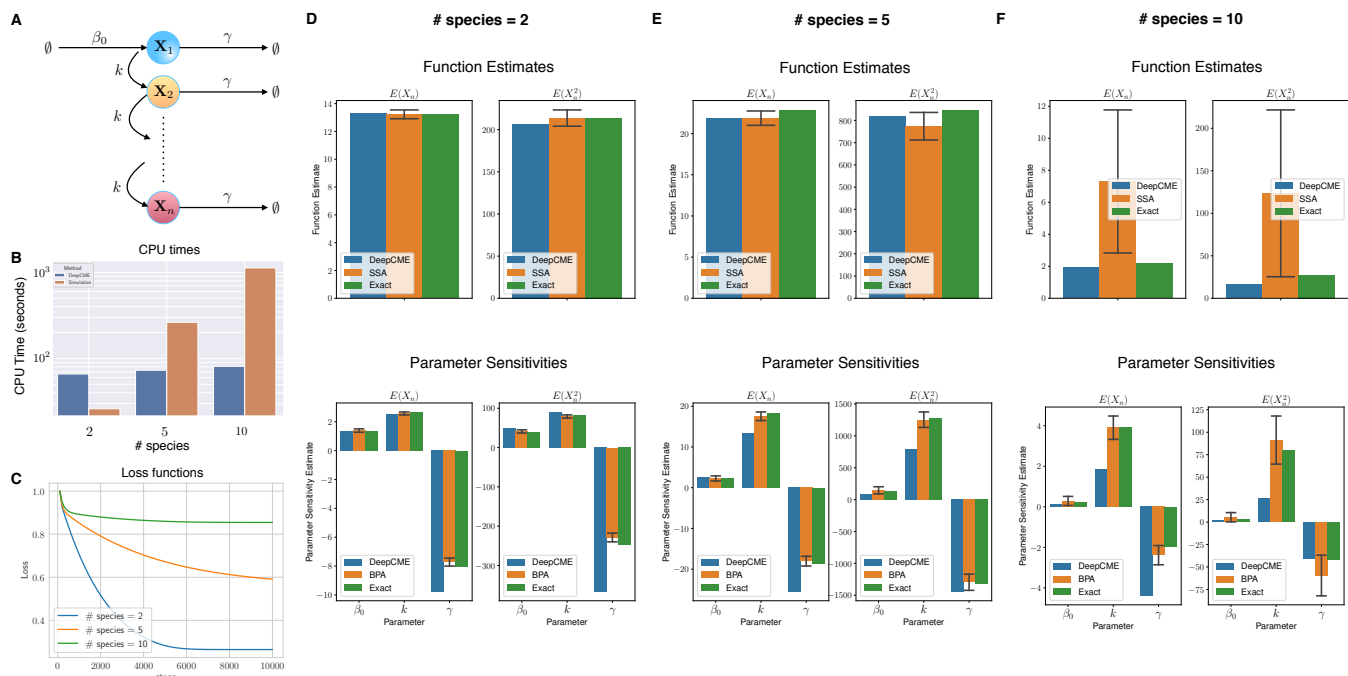


Figure 3: **Linear signalling cascade:** (A) Depicts the network with n species and $2n$ reactions with mass-action kinetics. (B) The CPU times are shown for training the DNN for different values of n (denoted as # species), and for comparison the time needed to generate 1000 SSA trajectories is also indicated. (C) Plots the loss function w.r.t. training steps for various n values. Panels (D-F) show estimates for the function values ($\mathbb{E}(X_n(T))$ and $\mathbb{E}(X_n^2(T))$) and the parameter sensitivities. The estimates with simulation based methods (SSA for function estimates and BPA for parameter sensitivities) are shown as 95% confidence intervals with 1000 samples.

5.4 Linear signalling cascade with feedback

Lastly we consider another variant of the network in Example 2 where there is negative feedback to the production of X_1 from species X_n (see Figure 5(A)) which is given by a repressing Hill function with a basal rate

$$\mathcal{H}(x) = b + \frac{k_m}{k_0 + x_n^H}, \quad (5.3)$$

where $b = 1$, $k_m = 100$, $k_0 = 10$ and $H = 1$. Other reactions have mass-action kinetics as in Example 2, with the same rate constants $k = 5$ and $\gamma = 1$. Due to the presence of feedback, oscillations can arise in the dynamics and to better represent this temporal dependence of the policy map we encode it with $N_T = 5$ identical DNNs (see Remark 4.3).

For number of species $n = 2, 5, 10$, we apply DeepCME to this reaction network by training the neural network for 10'000 SGD iterations. Then we compute the moment estimates (5.1) and their sensitivities to all the model parameters, and we also estimate these quantities with simulation-based methods (SSA and BPA) using 1000 samples. In the plots shown in Figure 5(D-F), we compare the estimates from both these approaches for various values of n . Observe that DeepCME is quite accurate in estimating the moments for $n = 2, 5$ and the parametric sensitivities for only $n = 2$. For $n = 5, 10$ only the sensitivities for $\mathbb{E}(X_n(T))$ are accurate but the sensitivities for $\mathbb{E}(X_n^2(T))$ are not accurate with our neural network architecture. The loss function trajectories are shown in Figure 5(C). The CPU times for DeepCME and simulation-based methods are plotted in Figure 5(B), and they show a similar growth pattern as our earlier examples.

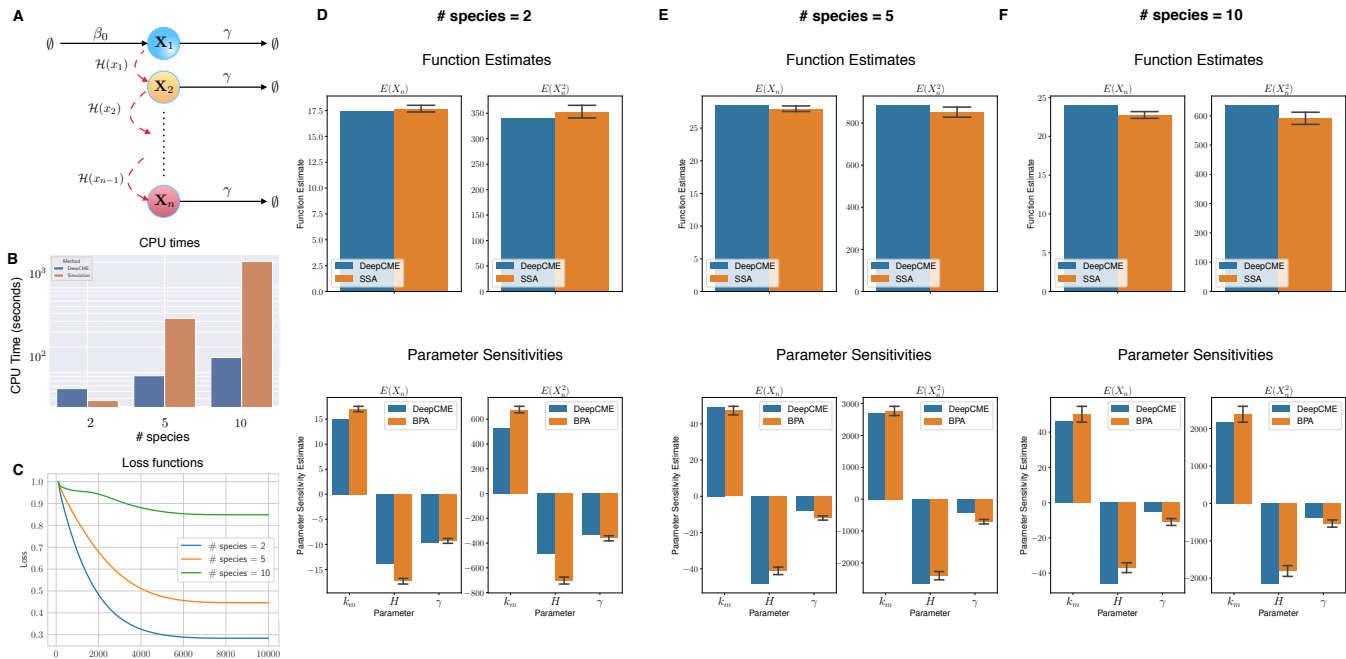


Figure 4: **Nonlinear signalling cascade:** (A) Depicts the network with n species and $2n$ reactions. The reactions shown with red dashed-arrow have propensities given by a nonlinear activating Hill function (5.2). Other reactions have mass-action kinetics. (B) The CPU times are shown for training the DNN for different values of n (denoted as # species), and for comparison the time needed to generate 1000 SSA trajectories is also indicated. (C) Plots the loss function w.r.t. training steps for various n values. Panels (D-F) show estimates for the function values ($\mathbb{E}(X_n(T))$ and $\mathbb{E}(X_n^2(T))$) and the parameter sensitivities (only the significant sensitivities are shown). The estimates with simulation based methods (SSA for function estimates and BPA for parameter sensitivities) are shown as 95% confidence intervals with 1000 samples.

6 Conclusion

Over the past couple of decades, stochastic reaction network models have become increasingly popular as a modelling paradigm for noisy intracellular processes. Many consequential biological studies have experimentally highlighted the random dynamical fluctuations within living cells, and have employed such stochastic models to quantify the effects of this randomness in shaping the phenotype at both the population and the single-cell levels [39]. As experimental technologies continue to improve at a rapid pace, it is urgent to develop computational tools that are able to bring larger and more realistic systems within the scope of stochastic modelling and analysis.

The central object of interest in stochastic reaction network models is a high-dimensional system of linear ODEs, called the *Chemical Master Equation* (CME). Numerical solutions to the CME are difficult to obtain and commonly used simulation-based schemes to estimate the solutions often require an inordinate amount of computational time, even for moderately-sized networks. Inspired by the recent success of machine learning approaches in solving high-dimensional PDEs [13], our goal in this paper is to devise a similar strategy, based on deep reinforcement learning to numerically solve CMEs. We develop such a method, called *DeepCME*, and we illustrate it with a number of examples. The neural network we train in DeepCME not only provides estimates for the CME solution but also its sensitivities w.r.t. all model parameters without any extra effort. Such parametric sensitivities are important for many applications, such as evaluating a network’s robustness properties [40] or identifying its critical components [41], but they are even more difficult to estimate than solutions to the CME [23, 24, 25, 26, 27, 28, 29, 30, 31].

Our work opens up several directions for future research. The machine-learning based computational framework and the mathematical formulation which we provide allows to deploy and transfer strong ML methodologies to the quantitative analysis and to data assimilation into complex CRNs. The present, basic approach can be

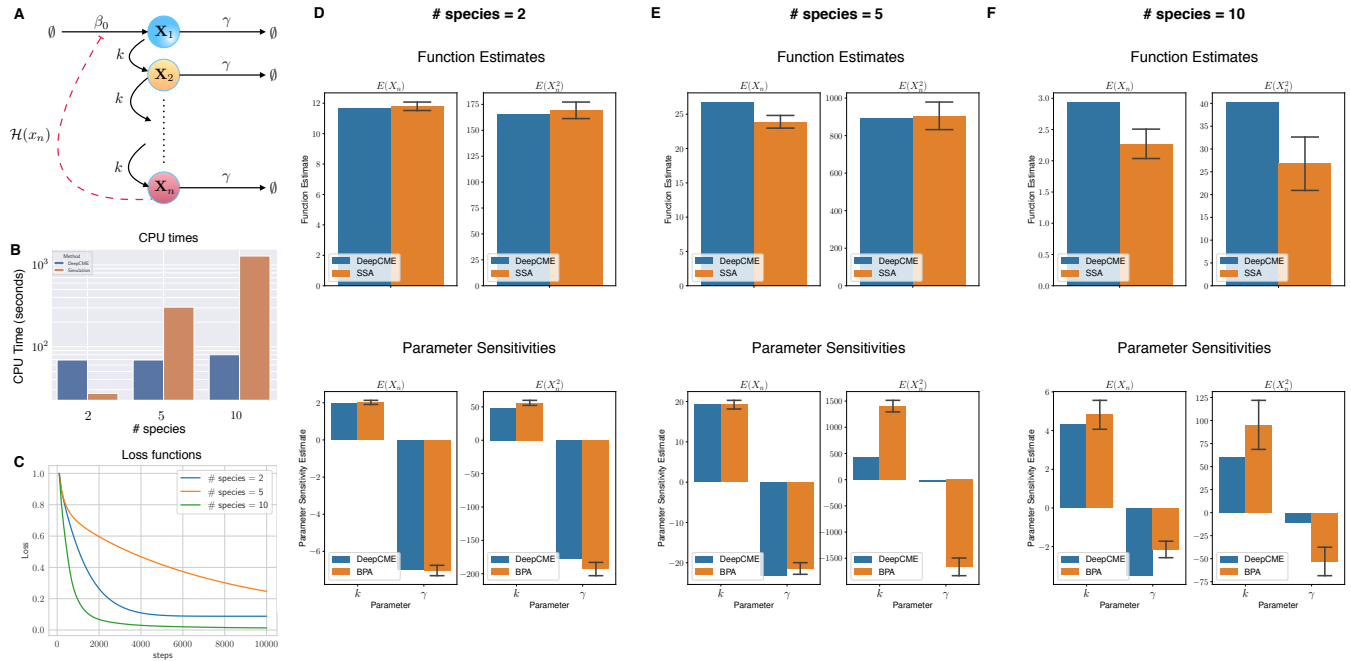


Figure 5: **Linear signalling cascade with feedback:** (A) Depicts the network with n species and $2n$ reactions. The reaction shown with red dashed-arrow has propensity given by a nonlinear repressing Hill function (5.3). All other reactions have mass-action kinetics. (B) The CPU times are shown for training the DNN for different values of n (denoted as # species), and for comparison the time needed to generate 1000 SSA trajectories is also indicated. (C) Plots the loss function w.r.t. training steps for various n values. Panels (D-F) show estimates for the function values ($\mathbb{E}(X_n(T))$ and $\mathbb{E}(X_n^2(T))$) and the parameter sensitivities (only the significant sensitivities are shown). The estimates with simulation based methods (SSA for function estimates and BPA for parameter sensitivities) are shown as 95% confidence intervals with 1000 samples.

improved/extended in a number of ways.

Firstly, it needs to be investigated how the architecture of the neural network can be optimally selected, for improved convergence of the training process, based on the CRN model.

Secondly, our framework requires a few ‘exact’ stochastic simulations of the dynamics, which could still be computationally infeasible for many large biological networks, especially if they are multiscale in nature [42, 43]. It might be possible to improve efficiency replacing exact simulations with τ -leaping simulations [20], multi-level schemes [21] or with simulations based on reduced models for multiscale networks [16, 44, 42, 43]. Incorporating such approaches for generating training trajectories would make our approach computationally feasible for much larger networks than those considered here. In particular, multi-level schemes based on coupling techniques simulations [21] would allow one to construct a lower variance estimator for the loss function (4.5) which would in turn benefit the accuracy and the convergence of the training process. In the context of multiscale networks, identifying the appropriate copy-number scalings that give rise to reduced models with simpler dynamics is a highly specialised task requiring careful theoretical analysis [16]. However our approach can be extended to “learn” these scaling factors during the training process by including them as trainable subnetworks into the ML feature space and employing them to scale the state-vectors in the input layer of the DNNs (see Figure 1). It is quite possible that incorporating these scaling factors would enhance the expressivity of the DNN.

Thirdly, the parameter sensitivities that we compute in our method could be employed in an ‘outer’ gradient descent method with the purpose of inferring model parameters by matching the computed statistics of CME solution with experimental data [45].

On the theoretical front, greater mathematical effort is required to understand how deep reinforcement-learning

approaches can help in circumventing the curse of dimensionality inherent to CMEs.

Finally, the architecture of the DNNs may include feature spaces comprising *parametric dictionaries of motifs*, which are adjusted during training to the reaction rates and to the kinetics of the CRN under consideration.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme grant agreement no. 743269 (CyberGenetics project).

References

- [1] Michael B. Elowitz, Arnold J. Levine, Eric D. Siggia, and Peter S. Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002. [1](#)
- [2] Harley H. McAdams and Adam Arkin. Stochastic mechanisms in gene expression. *Proc. Natl. Acad. Sci., Biochemistry*, 94:814–819, 1997. [1](#)
- [3] D.A. Anderson and T.G. Kurtz. Continuous time Markov chain models for chemical reaction networks. In H. Koepl, G. Setti, M. di Bernardo, and D. Densmore, editors, *Design and Analysis of Biomolecular Circuits*. Springer-Verlag, 2011. [1](#), [2.1](#), [2.1](#), [2.2](#)
- [4] N. G. van Kampen. A power series expansion of the master equation. *Canadian Journal of Physics*, 39(4):551–567, 1961. [1](#), [2.2](#)
- [5] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977. [1](#), [1](#), [5](#)
- [6] David F Anderson. A modified next reaction method for simulating chemical systems with time dependent propensities and delays. *The Journal of chemical physics*, 127(21):214107, 2007. [1](#), [3.4](#), [4](#), [5](#)
- [7] Derya Altıntan and Heinz Koepl. Hybrid master equation for jump-diffusion approximation of biomolecular reaction networks. *BIT*, 60(2):261–294, 2020. [1](#)
- [8] F. Hornung, A. Jentzen, and D. Salimova. Space-time deep neural network approximations for high-dimensional partial differential equations. Technical Report 2020-35, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2020. [1](#)
- [9] B. Munsky and M. Khammash. The finite state projection algorithm for the solution of the chemical master equation. *Journal of Chemical Physics*, 124(4), 2006. [1](#), [1](#)
- [10] Shev MacNamara, Kevin Burrage, and Roger B Sidje. Multiscale modeling of chemical kinetics via the master equation. *Multiscale Modeling & Simulation*, 6(4):1146–1168, 2008. [1](#)
- [11] Vladimir Kazeev, Mustafa Khammash, Michael Nip, and Christoph Schwab. Direct solution of the chemical master equation using quantized tensor trains. *PLoS Comput Biol*, 10(3):e1003359, 03 2014. [1](#)
- [12] E Weinan, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017. [1](#), [1](#)
- [13] Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018. [1](#), [6](#)
- [14] Jan Hermann, Zeno Schätzle, and Frank Noé. Deep-neural-network solution of the electronic Schrödinger equation. *Nature Chemistry*, 12(10):891–897, Sep 2020. [1](#)

- [15] Thomas G. Kurtz. Strong approximation theorems for density dependent Markov chains. *Stochastic Processes Appl.*, 6(3):223–240, 1977/78. [1](#)
- [16] Hye-Won Kang and Thomas G. Kurtz. Separation of time-scales and model reduction for stochastic reaction networks. *Ann. Appl. Probab.*, 23(2):529–583, 2013. [1](#), [6](#)
- [17] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. *29th Annual Conference on Learning Theory (COLT)*, 2016. [1](#)
- [18] Ankit Gupta, Jan Mikelson, and Mustafa Khammash. A finite state projection algorithm for the stationary solution of the chemical master equation. *The Journal of Chemical Physics*, 147(15):154101, 2017. [1](#)
- [19] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001. [1](#), [2.1](#)
- [20] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics*, 124(4), 2006. [1](#), [6](#)
- [21] David F Anderson and Desmond J Higham. Multilevel monte carlo for continuous time markov chains, with applications in biochemical kinetics. *Multiscale Modeling & Simulation*, 10(1):146–179, 2012. [1](#), [2.1](#), [6](#)
- [22] S. N. Ethier and T. G. Kurtz. *Markov processes : Characterization and Convergence*. Wiley Series in Probability and Mathematical Statistics: Probability and Mathematical Statistics. John Wiley & Sons Inc., New York, 1986. [1](#), [2.1](#), [2.1](#), [3](#)
- [23] D. Anderson. An efficient finite difference method for parameter sensitivities of continuous time markov chains. *SIAM: Journal on Numerical Analysis*, 50, 2012. [1](#), [2.3](#), [6](#)
- [24] M. Rathinam, P. W. Sheppard, and M. Khammash. Efficient computation of parameter sensitivities of discrete stochastic chemical reaction networks. *Journal of Chemical Physics*, 132, 2010. [1](#), [2.3](#), [6](#)
- [25] P. W. Sheppard, M. Rathinam, and M. Khammash. A pathwise derivative approach to the computation of parameter sensitivities in discrete stochastic chemical systems. *Journal of Chemical Physics*, 136, 2012. [1](#), [2.3](#), [6](#)
- [26] S. Plyasunov and A.P. Arkin. Efficient stochastic sensitivity analysis of discrete event systems. *Journal of Computational Physics*, 221:724–738, 2007. [1](#), [2.3](#), [6](#)
- [27] A. Gupta and M. Khammash. Unbiased estimation of parameter sensitivities for stochastic chemical reaction networks. *SIAM Journal on Scientific Computing*, 35(6), 2013. [1](#), [2.3](#), [6](#)
- [28] Ankit Gupta and Mustafa Khammash. An efficient and unbiased method for sensitivity analysis of stochastic reaction networks. *Journal of The Royal Society Interface*, 11(101), 2014. [1](#), [2.3](#), [6](#)
- [29] Ankit Gupta, Muruhan Rathinam, and Mustafa Khammash. Estimation of parameter sensitivities for stochastic reaction networks using tau-leap simulations. *SIAM Journal on Numerical Analysis*, 56(2):1134–1167, 2018. [1](#), [2.3](#), [2.3](#), [5](#), [6](#)
- [30] Ankit Gupta and Mustafa Khammash. Sensitivity analysis for stochastic chemical reaction networks with multiple time-scales. *Electron. J. Probab*, 2014. [1](#), [2.3](#), [6](#)
- [31] Ankit Gupta and Mustafa Khammash. Sensitivity analysis for multiscale stochastic reaction networks using hybrid approximations. *Bulletin of Mathematical Biology*, 81(8):3121–3158, 2019. [1](#), [2.3](#), [6](#)
- [32] J. R. Norris. *Markov chains*, volume 2 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 1998. Reprint of 1997 original. [2.2](#)
- [33] David F Anderson and Thomas G Kurtz. *Stochastic analysis of biochemical systems*. Springer, 2015. [3](#), [3.2](#)
- [34] Ankit Gupta, Corentin Briat, and Mustafa Khammash. A scalable computational framework for establishing long-term behavior of stochastic reaction networks. *PLoS Comput Biol*, 10(6):e1003669, 06 2014. [3.2](#)

- [35] Muruhan Rathinam. Moment growth bounds on continuous time markov processes on non-negative integer lattices. *Quarterly of Applied Mathematics*, pages 347–364, 2015. 3.2
- [36] Stefan Engblom. On the stability of stochastic jump kinetics. *arXiv preprint arXiv:1202.3892*, 2012. 3.2
- [37] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016. 4
- [38] Alireza Yazdani, Lu Lu, Maziar Raissi, and George Em Karniadakis. Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS computational biology*, 16(11):e1007575, 2020. 4
- [39] Avigdor Eldar and Michael B Elowitz. Functional roles for noise in genetic circuits. *Nature*, 467(7312):167–173, 2010. 6
- [40] Jörg Stelling, Ernst Dieter Gilles, and Francis J Doyle. Robustness properties of circadian clock architectures. *Proceedings of the National Academy of Sciences*, 101(36):13210–13215, 2004. 6
- [41] Xiao-jiang Feng, Sara Hooshangi, David Chen, Genyuan Li, Ron Weiss, and Herschel Rabitz. Optimizing genetic circuits by global sensitivity analysis. *Biophysical journal*, 87(4):2195–2202, 2004. 6
- [42] Y. Cao, D.T. Gillespie, and L.R. Petzold. The slow-scale stochastic simulation algorithm. *Journal of Chemical Physics*, 122(1):1–18, 2005. 6
- [43] Weinan E, Di Liu, and Eric Vanden-Eijnden. Nested stochastic simulation algorithms for chemical kinetic systems with multiple time scales. *J. Comput. Phys.*, 221(1):158–180, January 2007. 6
- [44] Benjamin Hepp, Ankit Gupta, and Mustafa Khammash. Adaptive hybrid simulations for multiscale stochastic reaction networks. *The Journal of chemical physics*, 142(3):034118, 2015. 6
- [45] Brian Munsky, Brooke Trinh, and Mustafa Khammash. Listening to the noise: random fluctuations reveal gene network parameters. *Molecular systems biology*, 5(1):318, 2009. 6