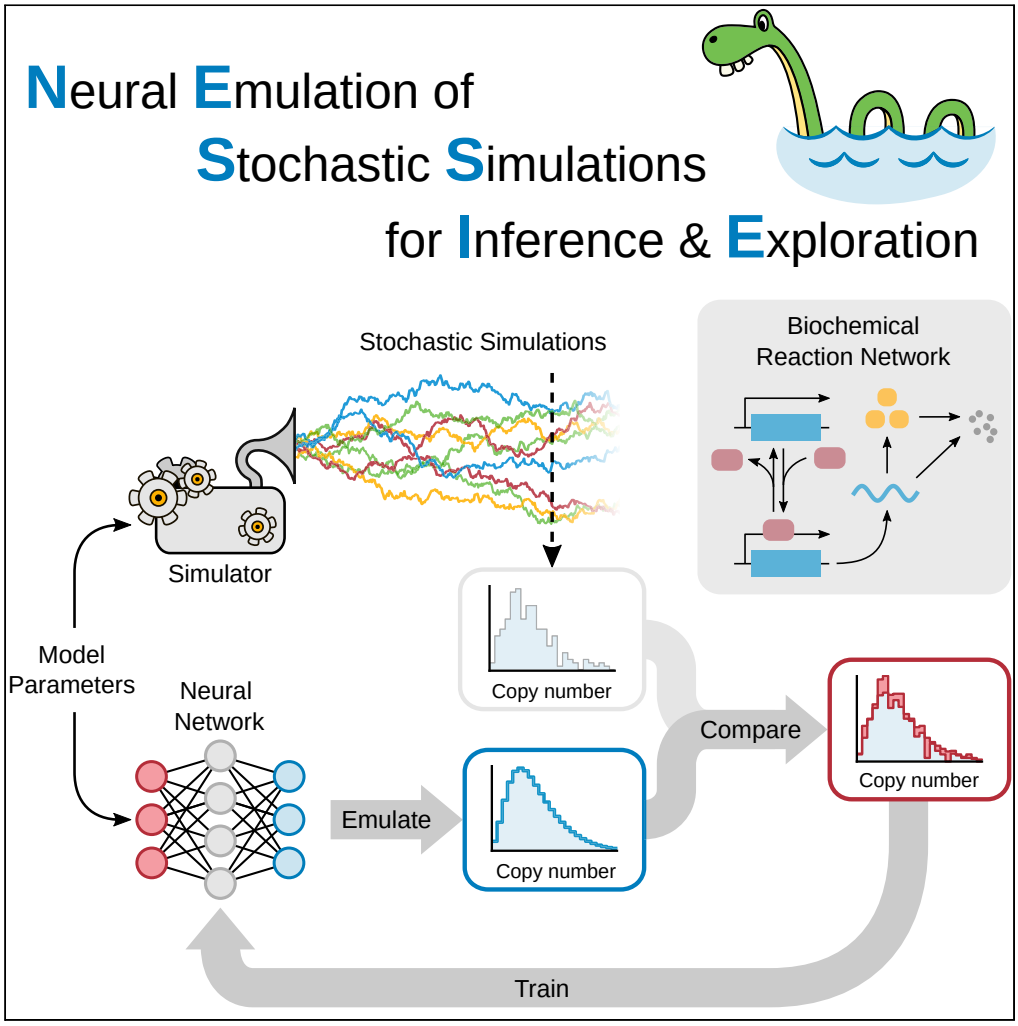
iScience

CellPress
OPEN ACCESS

Article

Approximating solutions of the Chemical Master equation using neural networks



Augustinas Sukys, Kaan Öcal, Ramon Grima

asukys@turing.ac.uk (A.S.)
kaan.ocal@ed.ac.uk (K.Ö.)
ramon.grima@ed.ac.uk (R.G.)

Highlights

We approximate solutions of the Chemical Master Equation using neural networks

Simple networks suffice to learn complex distributions over a wide parameter range

Neural emulation can significantly speed up parameter exploration and inference

Sukys et al., iScience 25, 105010
September 16, 2022
https://doi.org/10.1016/j.isci.2022.105010

iScience

CellPress
OPEN ACCESS

Article

Approximating solutions of the Chemical Master equation using neural networks

Augustinas Sukys,[1,3,4,*] Kaan Öcal,[1,2,4,*] and Ramon Grima[1,5,*]

SUMMARY

The Chemical Master Equation (CME) provides an accurate description of stochastic biochemical reaction networks in well-mixed conditions, but it cannot be solved analytically for most systems of practical interest. Although Monte Carlo methods provide a principled means to probe system dynamics, the large number of simulations typically required can render the estimation of molecule number distributions and other quantities infeasible. In this article, we aim to leverage the representational power of neural networks to approximate the solutions of the CME and propose a framework for the Neural Estimation of Stochastic Simulations for Inference and Exploration (Nessie). Our approach is based on training neural networks to learn the distributions predicted by the CME from relatively few stochastic simulations. We show on biologically relevant examples that simple neural networks with one hidden layer can capture highly complex distributions across parameter space, thereby accelerating computationally intensive tasks such as parameter exploration and inference.

INTRODUCTION

The past decades have seen great progress in our understanding of the complex dynamics that underlie noisy cellular processes, both from an experimental and a theoretical perspective. Modern experimental techniques have shown that mRNA and protein levels can vary enormously at the single-cell level, but building detailed quantitative models that take into account the stochasticity of biochemical systems remains a daunting task. Owing to the numerous challenges involved in describing the stochastic dynamics of these systems, modeling frequently relies on deterministic and small noise approximations which do not paint an accurate picture in many situations. Such simplified descriptions are often insufficient to describe how biochemical networks function in the presence of molecular noise (Maheshri and O'Shea, 2007; McAdams and Arkin, 1999) and do not capture intricate noise-driven phenomena involved in cell fate decision (Elowitz et al., 2002; Choi et al., 2008) and phenotypic regulation (Raser and O'Shea, 2005).

The most commonly used formalism for modeling biochemical reaction networks in a fully stochastic framework is the Chemical Master Equation (CME) (Schnoerr et al., 2017), which describes how the probability distribution over states evolves with time. The CME cannot be solved analytically for most biologically relevant cases, and as the state space is typically infinite, numerical solutions to the CME often involve state space truncation methods such as the Finite State Projection (FSP) (Munsky and Khammash, 2006). However, owing to the combinatorial explosion of the state space in the number of species, using the FSP to solve the CME quickly becomes too computationally intensive for most non-trivial systems (Kazeev et al., 2014; Kazeev and Schwab, 2015; Dinh and Sidje, 2020). A wide variety of other approximation methods exist for the CME (see Schnoerr et al. (2017) for an overview), but these often trade computational efficiency for accuracy and are generally difficult to apply to complex systems involving many species and interactions.

While solving the CME remains challenging, *sampling* realizations of a system is possible thanks to the Stochastic Simulation Algorithm (SSA) (Gillespie, 1976). Many physical quantities such as moments of molecule number distributions can be computed to arbitrary accuracy by repeatedly simulating samples from the system. Nevertheless, the SSA can be prohibitively computationally expensive when many repeated simulations are needed for the accurate estimation of these quantities. As simulations have to be performed anew for all parameters of interest, investigating system properties over time and parameter space with this

[1]School of Biological Sciences, University of Edinburgh, Edinburgh EH9 3JH, UK

[2]School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, UK

[3]The Alan Turing Institute, London NW1 2DB, UK

[4]These authors contributed equally

[5]Lead contact

*Correspondence: asukys@turing.ac.uk (A.S.), kaan.ocal@ed.ac.uk (K.Ö.), ramon.grima@ed.ac.uk (R.G.)

https://doi.org/10.1016/j.isci.2022.105010

approach can quickly become intractable. Furthermore, likelihoods are hard to estimate reliably using Monte Carlo methods, rendering likelihood-based inference particularly difficult (Wilkinson, 2018).

Given the difficulties inherent in solving the CME exactly, it is natural to explore whether we could tackle this problem using neural networks, which in recent years have found diverse applications in the physical and biological sciences (Angermueller et al., 2016; Min et al., 2017; Carleo et al., 2019; Mehta et al., 2019). Their ability to detect patterns and learn complex representations given enough data is particularly useful when combined with simulator-based modeling, where such data can often be generated aplenty. In the context of systems biology, neural network-based approaches have been used to perform parameter inference on deterministic models, accelerate parameter exploration for models described by partial and stochastic differential equations (Wang et al., 2019), and even learn Markovian approximations to non-Markov models, translating them into the CME framework (Jiang et al., 2021), among other applications.

Moreover, a number of recent studies have investigated the use of neural networks to learn various properties of stochastic biochemical reaction networks modeled using the CME (Gupta et al., 2021; Bortolussi and Palmieri, 2018; Repin and Petrov, 2021; Davis et al., 2020; Cairoli et al., 2021). Schnoerr et al. (2017) presented DeepCME, an approach that uses reinforcement learning to estimate summary statistics such as means and variances from stochastic simulations. The model abstraction procedure introduced by Bortolussi and Palmieri (2018) employs Mixture Density Networks (Bishop, 1994) to learn the transition kernel of the CME and has been further extended into a framework providing automated neural network architecture search (Repin and Petrov, 2021). In the same vein, mixture density networks have been used to directly predict the probability distributions characterizing the dynamics of an SIR-type model (Davis et al., 2020). Finally, Cairoli et al. (2021) demonstrate how Generative Adversarial Networks can be trained to generate full trajectories resembling the output of the SSA.

In this article, we introduce Nessie, a framework for the Neural Emulation of Stochastic Simulations for Inference and Exploration, based on using neural networks to learn solutions of the CME from stochastic simulations. Using only a moderate number of simulations of the specified reaction system at different parameter values, we train a neural network to learn the marginal probability distributions predicted by the CME over the whole parameter region of interest. We approximate the target distributions using mixtures of negative binomials, a flexible class of distributions particularly well-suited for this task (Perez-Carrasco et al., 2020; Öcal et al., 2022). Once trained, Nessie becomes a surrogate for the CME that can efficiently and accurately predict the solution of the CME at different times for a wide range of parameters, bypassing the need to use further simulations or expensive approximation techniques to analyze the reaction network in question.

Our work differs from related approaches (Gupta et al., 2021; Bortolussi and Palmieri, 2018; Repin and Petrov, 2021; Davis et al., 2020; Cairoli et al., 2021) in several regards. Unlike Bortolussi and Palmieri (2018) and Repin and Petrov (2021) or Cairoli et al. (2021), we do not aim to learn the transition kernels or the distributions of trajectories, i.e. the dynamics of the chemical system in question, but to capture the *marginal distributions* directly. In this sense, Nessie is also different from DeepCME (Gupta et al., 2021), which focuses on the task of learning summary statistics such as moments of molecule numbers. The relevant expectation values can be computed directly from the distributions predicted by Nessie, and we verify in our examples that Nessie can predict means and variances to a high degree of accuracy. Although our neural network is a variant of mixture density networks, in contrast to Bortolussi and Palmieri (2018) and Repin and Petrov (2021) we do not use continuum approximations based on mixtures of Gaussians, thereby avoiding training and numerical stability issues that can arise in this context (Hjorth, 1999; Bortolussi and Palmieri, 2018; Repin and Petrov, 2021; Goodfellow et al., 2017). As our approach directly targets experimentally observable distributions, it can also be used to perform parameter estimation based on comparing the neural network output with experimental data.

The article is organized as follows. In STAR Methods we summarize the relevant theory on the CME and the basics of artificial neural networks. In Results we describe Nessie, providing an overview of the technical details of our neural network implementation and the workflow we use to predict the marginal probability distributions for a given system. We test our approach on several biologically relevant examples: an autoregulatory feedback loop, a genetic toggle switch involving mRNA and protein dynamics for two genes, a detailed model of mRNA turnover, and a model of the mitogen-activated protein kinase (MAPK) pathway in

S. Cerevisiae. The results indicate that Nessie can learn the dynamics of biochemical reaction networks for a wide range of parameters, allowing us to investigate the physical properties of a given system such as multimodality and parameter identifiability. Furthermore, we demonstrate how Nessie enables us to build on Öcal et al. (2022) and perform efficient parameter inference from population snapshot data, a challenging problem for CME-based models. We conclude by discussing the results and the limitations of our study.

RESULTS

Nessie

Our goal in this article is to learn marginal distributions predicted by the CME for different parameters and measurement times. As such the inputs to our neural network will consist of the chemical reaction network parameters θ and the time t; as these can span several orders of magnitude we log-transform them first. Although we work with fixed initial conditions for each reaction system, this constraint could be relaxed by adding the molecule numbers at time $t = 0$ as extra inputs to the neural network.

We approximate the marginal distribution of interest by a mixture of negative binomials, a flexible parametric class of distributions that have been shown to be very accurate for a large class of reaction networks (Öcal et al., 2022; Perez-Carrasco et al., 2020). Indeed, it is known that single-time marginal distributions predicted by the CME for many different reaction networks can be modeled as a mixture of negative binomials in the presence of timescale separation (Friedman et al., 2006; Shahrezaei and Swain, 2008; Z. Cao and Grima, 2020; Perez-Carrasco et al., 2020; Jia and Grima, 2020). Experimental measurements of mRNA and protein distributions in bacterial, yeast, and mammalian cells show that these often fit well by such mixtures, even when timescale separation is not applicable (Cai et al., 2006; Taniguchi et al., 2010; Singer et al., 2014; Phillips et al., 2021). We remark that a mixture of negative binomials always has a Fano factor (variance over mean) greater than 1, and systems whose Fano factor is significantly less than 1 (see e.g. Braichenko et al. (2021)) would benefit from a different parametric approximation which we shall not consider here.

A mixture of negative binomials can be parameterized as

$$q_\varphi(n) = \sum_{i=1}^{K} w_i \cdot \mathrm{NB}(n; r_i, p_i). \qquad \text{(Equation 1)}$$

where K is the number of mixture components, w_i is the weight of the i-th component and $\mathrm{NB}(r_i, p_i)$ is a negative binomial distribution with parameters (r_i, p_i). The number of components is fixed *a priori* and the weights are subject to the normalization constraint $w_1 + \ldots + w_K = 1$. Our task is, therefore, to learn a mapping $(t, \theta) \mapsto (w_i, r_i, p_i)_{i=1}^{K}$ from the input parameters to those of the output distribution.

Each parameter characterizing the output distribution is represented by a single neuron in the output layer. To respect the constraints on the weights w_i we apply the *softmax* activation function to the corresponding neurons: the outputs are exponentiated, then normalized to sum to 1. For the neurons corresponding to the count parameters r_i we choose exponential activation functions, and for the probabilities p_i we use sigmoid activation functions. The architecture of our neural network is shown in Figure 1.

The number of hidden layers and the number of neurons per hidden layer can have a large impact on the representational power of a neural network (see Background on Neural Networks). The networks we build throughout this article contain only a single hidden layer as we have found such architecture in our case to be easier to train and provide better predictive performance than "deeper" networks (see Hyperparameter Tuning), an observation corroborated in Jiang et al. (2021). We choose the number of neurons in the hidden layer depending on the complexity of the chemical reaction network at hand and use the ReLU activation function as it enables efficient training (Glorot et al., 2011).

In our setup, a single training point consists of an input point $x = (t, \theta)$ and a reference distribution p of target molecules at time t for the specified reaction network with parameters θ, obtained by averaging over a number of SSA trajectories (or by using the FSP). We build the training set by sampling parameter sets θ in the parameter region of interest and running simulations at each θ; this can be conducted in parallel for all parameters. We then use the simulation results at fixed times as training inputs to Nessie, using its ability to interpolate between these times to learn general time-dependent distributions. In order to

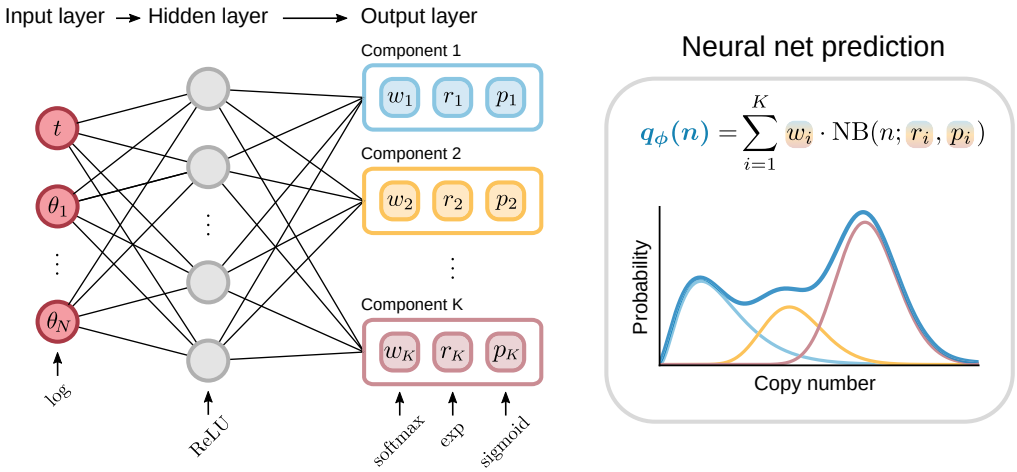


Figure 1. Architecture of Nessie, a simple feedforward neural network with one hidden layer

The input layer takes in the time t and the model parameters Θ, and the output layer returns the corresponding negative binomial mixture components. Activation functions are indicated at the bottom.

ensure that the training data evenly cover the entire parameter region, we use Sobol sequences (Sobol, 1967), which generally provide more uniform coverage than random sampling.

A common method to match distributions in the statistics literature is to minimize the Kullback-Leibler (KL) divergence $D_{\mathrm{KL}}(p \parallel q_\varphi)$, where p is the target distribution and q_φ the prediction; this procedure is mathematically equivalent to maximizing the average log-likelihood under q_φ of a sample drawn from p (Goodfellow et al., 2017). Hence we use the KL divergence as our loss function for each point in the training set:

$$\mathcal{L}(\varphi; x^{(i)}, p^{(i)}) = D_{\mathrm{KL}}(p^{(i)} \| q_\varphi(x^{(i)})). \qquad \text{(Equation 2)}$$

This is equivalent to the cross-entropy, up to the addition of a constant that does not depend on the network weights φ. Computing the mixture of negative binomials q_φ for a given input point is straightforward using Equation 1. The complete workflow for training Nessie is shown in Figure 2.

We note that maximizing the average log-likelihood for a mixture of Gaussians, as is commonly conducted with Mixture Density Networks (MDNs) (Bishop, 1994), can lead to stability issues for more than one component. For example, the neural network can learn to place a Gaussian component at zero with arbitrarily low variance, which will give an arbitrarily high likelihood if 0 occurs anywhere in the training dataset, irrespectively of the overall quality of fit — an example of overfitting (see Training Neural Networks). This is because in the continuous case one deals with probability *densities*, which can become arbitrarily large in contrast to probabilities. We believe that this phenomenon is responsible for some common numerical issues observed e.g. in Goodfellow et al. (2017), Hjorth (1999), and Repin and Petrov (2021). One can attempt to remedy this issue by integrating the density over a finite interval (say, $[-0.5, 0.5]$), or by regularizing the precision of each component (thereby adding hyperparameters to the training procedure). In contrast, mixtures of negative binomials were not prone to overfitting in our experiments and did not require any form of regularization.

In what follows, we quantify the relative accuracy of the trained neural networks by computing the Hellinger distance between the predicted and test distributions. Although the KL divergence is more suited as a loss function (Goodfellow et al., 2017) to train the neural network for its computational efficiency, the Hellinger distance is a bounded metric and a more interpretable measure of the model's predictive performance.

We used Julia (Bezanson et al., 2017) with Flux.jl (Innes. 2018) to implement neural networks. Gradients of the loss function were computed directly by Flux using the built-in Zygote.jl automatic differentiation system (Innes et al., 2019). The training datasets were constructed by defining chemical reaction networks via Catalyst.jl (Loman et al., 2022) and simulating them using DifferentialEquations.jl (Rackauckas and Nie. 2017) (SSA) and FiniteStateProjection.jl (FSP). Minimization of the Hellinger distance in the MAPK inference

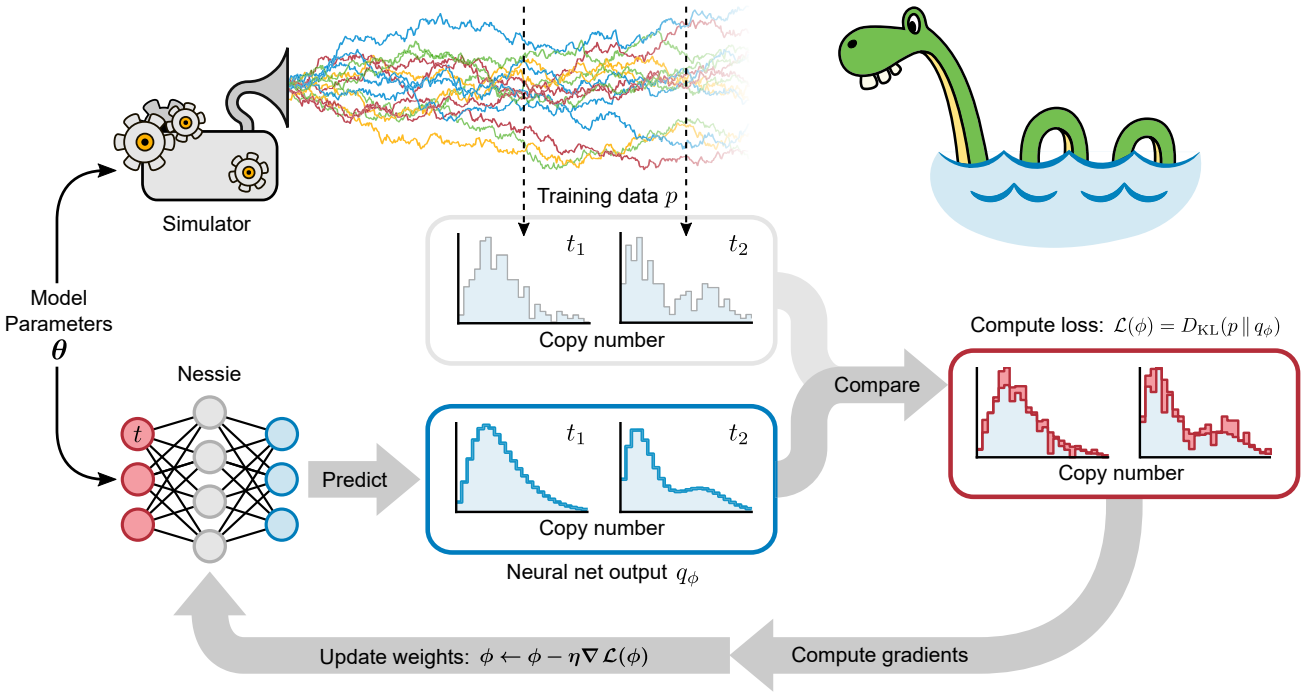


Figure 2. Workflow for training Nessie

Given model parameters $\boldsymbol{\theta}$, the reaction network is simulated repeatedly using the SSA to obtain empirical distributions at training time points $t_1, t_2, \ldots$. These are then compared with the output of the neural network and the total loss $\mathcal{L}$ is computed. In order to decrease the loss, the neural weights φ are updated iteratively via gradient descent until the loss has converged. In the figure η denotes the learning rate (see Training Neural Networks) and $D_{\mathrm{KL}}(p \parallel q_\varphi)$ the KL divergence between the true distribution p and the neural network prediction q_φ.

example was performed using BlackBoxOptim.jl. All numerical experiments were performed on a Intel Xeon Silver 4114 CPU (2.2 GHz) using 16 threads.

Autoregulatory feedback loop

We first consider a simple autoregulatory feedback loop illustrated schematically in Figure 3A, consisting of the following reactions:

$$G_u + P \underset{\sigma_u}{\overset{\sigma_b}{\rightleftharpoons}} G_b,\ G_u \xrightarrow{\rho_u} P,\ G_b \xrightarrow{\rho_b} P,\ P \xrightarrow{1} \varnothing.$$

This system contains a single gene with two promoter states G_u and G_b, each associated with different protein P production rates ρ_u and ρ_b (mRNA dynamics are not modeled explicitly). The feedback is introduced via reversible binding of a protein molecule to the promoter region with binding rate σ_b and unbinding rate σ_u, which causes switching between the two promoter states. Finally, protein degradation is modeled by an effective first-order reaction. This system is a rudimentary example of stochastic self-regulation in a gene: the model functions as a positive feedback loop if $\rho_b > \rho_u$, and a negative feedback loop if $\rho_b < \rho_u$.

Although the CME of the autoregulatory feedback loop has only been solved analytically in the steady-state (Grima et al., 2012), an efficient time-dependent numerical solution can be obtained with the FSP in this specific case as the model contains few molecular species and chemical reactions. Estimating the probability distributions for the autoregulatory feedback loop via the FSP is much faster than using the SSA. This makes the autoregulatory feedback loop an ideal toy model for our initial experiments as we can relatively quickly build arbitrarily large training datasets with the FSP, calibrate the neural network, and probe the performance of Nessie in capturing the marginal distributions of protein numbers.

We use a training set of size 1k, a validation set of size 100 and a test set of size 500, sampled using a Sobol sequence in the parameter region indicated in Table S1. For each datapoint, we take four snapshots at times $t = \{5, 10, 25, 100\}$ and construct the corresponding histograms using the FSP. Our neural network

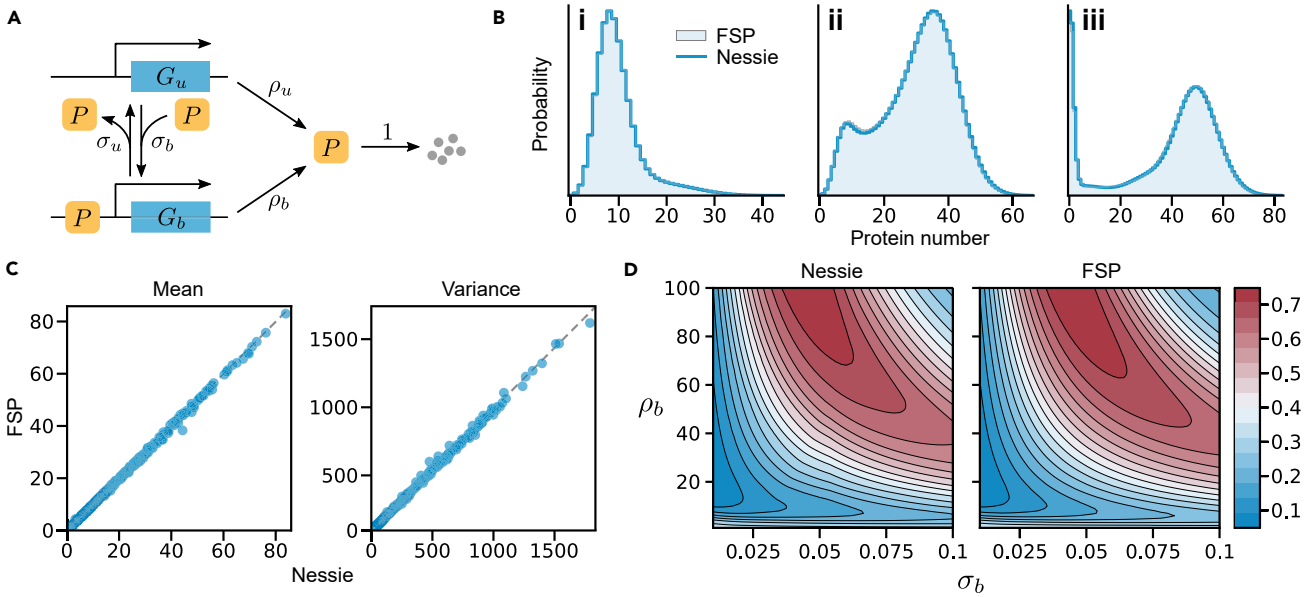


Figure 3. Nessie applied to an autoregulatory genetic feedback loop

(A) Schematic of the reaction network. We assume mass action kinetics for all reactions.

(B) Protein distributions for three different test parameter values (indicated in Table S1). The ground truth distributions were computed using the FSP.

(C) Comparison of true and predicted means and variances of protein numbers at time $t = 100$ for the test set containing 500 parameter values. True means and variances were again computed using the FSP.

(D) Exact and predicted bimodality coefficients as a function of ρ_b and σ_b, where we set $\sigma_u = \rho_u = 1$ and $t = 100$. Here the bimodality coefficients predicted by Nessie closely agree with their ground truth values.

consists of a single hidden layer with 128 neurons and outputs 4 negative binomial mixture components; we use a batch size of 64 for training. More details on the training procedure and the hyperparameter choices are given in the STAR Methods.

In Figure 3B we show the protein distributions for three test parameter sets, comparing the predicted distribution with the FSP results. Our approach provides highly accurate fits for the different distribution shapes obtained at these points in the parameter space, showcasing the flexibility of negative binomial mixtures in approximating the CME of the autoregulatory feedback loop.

Having learned the output distributions for this chemical system we can compute various quantities of interest from these. In Figure 3C, we compare the means and variances of the protein number predicted by Nessie to the true values computed using the FSP for all points in the test dataset. Furthermore, in Figure 3D we analyze the bimodality coefficient as defined by Xu et al. (2016). The bimodality coefficient of a distribution is defined as $1/(\kappa - \gamma^2)$, where κ and γ are the skewness and kurtosis respectively, and is a measure of bimodality with higher values corresponding to strongly bimodal distributions. We see that Nessie provides a good approximation to this quantity and closely matches the FSP results.

Numerically estimating the bimodality coefficient for many different parameters is a computationally intensive task, whereas predicting it using the neural network, once trained on its 1k datapoints, is very quick: using Nessie we can produce the plotted heatmap in 0.03 s, in contrast to the FSP which takes 240s. This example illustrates how we can apply the neural network to rapidly and efficiently analyze large swathes of parameter space and, in turn, to determine the regions of bimodality in the system. Note that in evaluating the model performance we did not take into account the time required to generate the initial training data and to train the neural network, which is given in Table S5. Although this does incur a notable overhead, it becomes largely insignificant in comparison to the computational gains provided by the trained neural network in large parameter exploration studies. Similarly, our approach scales to more complicated chemical reaction networks that are not amenable to study using the FSP, allowing us for example to perform global sensitivity analysis for the genetic toggle switch presented in the next section.

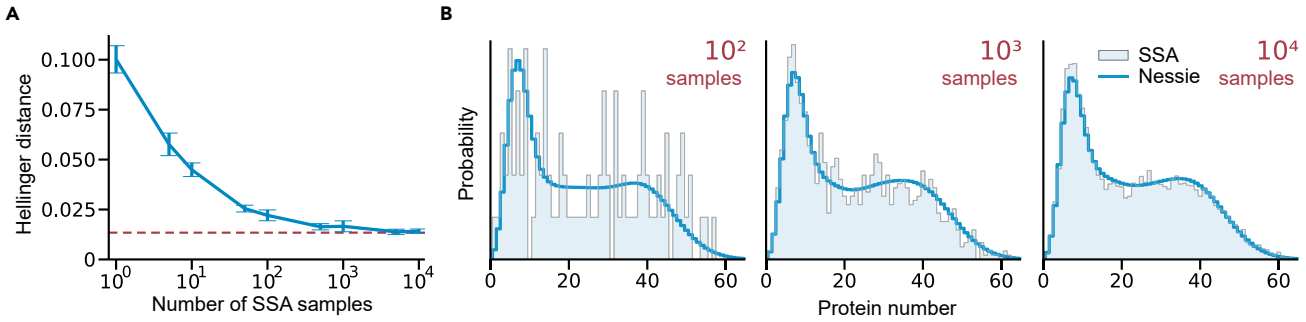


Figure 4. Training Nessie using the SSA

(A) Mean Hellinger distance computed over the validation dataset versus the number of SSA trajectories used to construct the histogram of each training datapoint. Here the validation dataset consists of 100 different parameter values at 4 time snapshots, constructed with the corresponding number of SSA trajectories. The error bars are obtained by averaging over 10 independent Nessie training runs, where for each run we resample the training and validation datasets and train a new model, as conducted for the manual tuning of other hyperparameters discussed in Hyperparameter Tuning and shown in Figure S1. The red dashed line indicates the accuracy of Nessie trained on the FSP data.

(B) Example distributions constructed with 100, 1k, and 10k SSA samples (indicated in red at the top) compared to the neural network predictions. We retrain Nessie for each plot using the respective number of trajectories per training point. Parameter values and ranges are given in Table S1.

Finally, in Figure 4 we consider using the SSA as an alternative to the FSP for constructing the training dataset. As each training histogram is built from a number of SSA samples, we investigate how many samples per training point are required to accurately train the network. Note that the numerical FSP solution is virtually indistinguishable from the exact CME solution as its approximation error can be systematically reduced by increasing the size of the truncated state space (Munsky and Khammash, 2006), and hence it is effectively equivalent to an infinite number of SSA trajectories. We see in Figure 4A that for the autoregulatory feedback loop using as few as 100 trajectories per training point enables Nessie to produce good approximations of the true distributions. In contrast, obtaining similar quality fits using the SSA alone would require two orders of magnitude more samples per parameter, as demonstrated in Figure 4B where we compare the histograms obtained using 100, 1k, and 10k SSA trajectories. One may expect this effect to be amplified for larger system sizes and more widely spread out distributions, where many more simulations are typically needed to obtain smooth histograms.

Genetic toggle switch

For our next experiment, we consider a stochastic model of the genetic toggle switch, one of the first synthetic biological circuits (Gardner et al., 2000). The reaction network, introduced in Thomas et al. (2014), is composed of two mutually interacting genes (which we label *A* and *B*) and takes into account transcription, translation, and the subsequent degradation of the produced mRNA and proteins (see Figure 5A), consisting altogether of the following reactions:

$$G_u^A + P^B \underset{\sigma_u^A}{\overset{\sigma_b^A}{\rightleftharpoons}} G_b^A,\ G_u^B + P^A \underset{\sigma_u^B}{\overset{\sigma_b^B}{\rightleftharpoons}} G_b^B,$$

$$G_u^A \xrightarrow{\rho_u^A} G_u^A + M^A,\ G_b^A \xrightarrow{\rho_b^A} G_b^A + M^A,\ M^A \xrightarrow{\gamma^A} M^A + P^A,\ M^A \xrightarrow{\delta_m^A} \varnothing,\ P^A \xrightarrow{\delta_p} \varnothing$$

$$G_u^B \xrightarrow{\rho_u^B} G_u^B + M^B,\ G_b^B \xrightarrow{\rho_b^B} G_b^B + M^B,\ M^B \xrightarrow{\gamma^B} M^B + P^B,\ M^B \xrightarrow{\delta_m^B} \varnothing,\ P^B \xrightarrow{1} \varnothing$$

$$M^B + P^A \underset{\sigma_u^M}{\overset{\sigma_b^M}{\rightleftharpoons}} M^B P^A \xrightarrow{\delta_{pm}} \varnothing.$$

The translated protein *A* can bind to the promoter region of the gene that produces protein *B*, and vice versa with protein *B* binding to the gene promoter of protein *A*. This results in effective transcriptional regulation: depending on the mRNA production rate associated with each promoter state, the process can either lead to the repression or activation of transcription for each species. In addition, the system contains post-transcriptional regulation mediated by protein *A* binding to the mRNA of species *B* and modulating its translation accordingly (Hafner et al., 2010).

The toggle switch is noticeably more intricate than the autoregulatory feedback loop considered previously. It exhibits rich dynamics highlighted by diverse protein distributions that can be highly multimodal

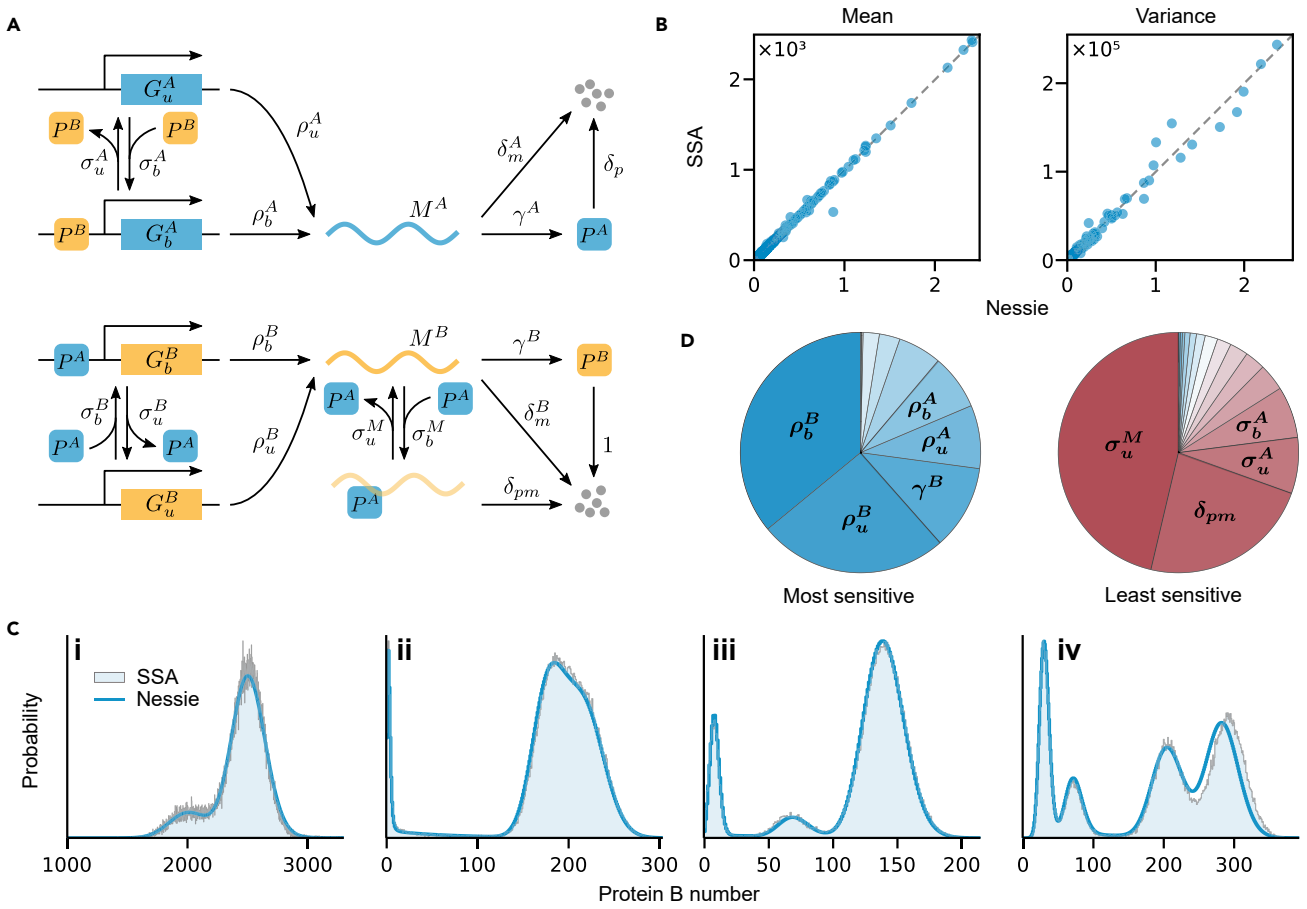


Figure 5. Nessie applied to a genetic toggle switch with post-transcriptional regulation

(A) Schematic of the reaction network (assuming mass action kinetics for all reactions).

(B) Comparison of true and predicted means and variances of the protein *B* numbers for the test set consisting of 1k different parameter values at time t = 100 constructed using 100k SSA trajectories.

(C) Protein *B* distributions for four different test parameter values (specified in Table S2). The SSA distributions were computed by averaging over 100k trajectories.

(D) Sensitivity of the Fano factor of protein *B* to parameter perturbation at time t = 100, where the pie charts show the most and least sensitive parameters. The results are obtained by using Nessie to compute the logarithmic sensitivity of the Fano factor to the 16 model parameters for 100k parameter values drawn from a Sobol sequence covering the training range given in Table S2. We observe that only a few reaction parameters can be identified as typically the most/least sensitive (indicated in bold).

(Thomas et al., 2014). Owing to the considerable number of reaction parameters, the frequent occurrence of high copy numbers (>1000), and the complexity of the observed distributions, studying this system poses significant problems both analytically and computationally. This makes it a good challenge for Nessie.

Our aim is to predict the probability distributions of the target protein *B* in the genetic toggle switch. Note that we could similarly consider the distribution of protein *A* or, alternatively, the neural network architecture itself could be extended to predict both marginal distributions for the two proteins in parallel. If, however, joint distributions are required, the univariate mixture of negative binomials we use needs to be replaced by a multivariate equivalent; we briefly mention possible approaches in the Discussion.

We draw 40k, 100, and 1k parameters for the training, validation, and test datasets respectively using a Sobol sequence in the parameter region indicated in Table S2. For each parameter set we take 8 snapshots at times $t = \{2, 4, 10, 16, 32, 50, 74, 100\}$. The complexity of the system prevents us from using the FSP to construct the reference histograms and hence we resort to the SSA. As discussed previously, a relatively small number of SSA samples can be used to successfully train the neural network. For this reason, we

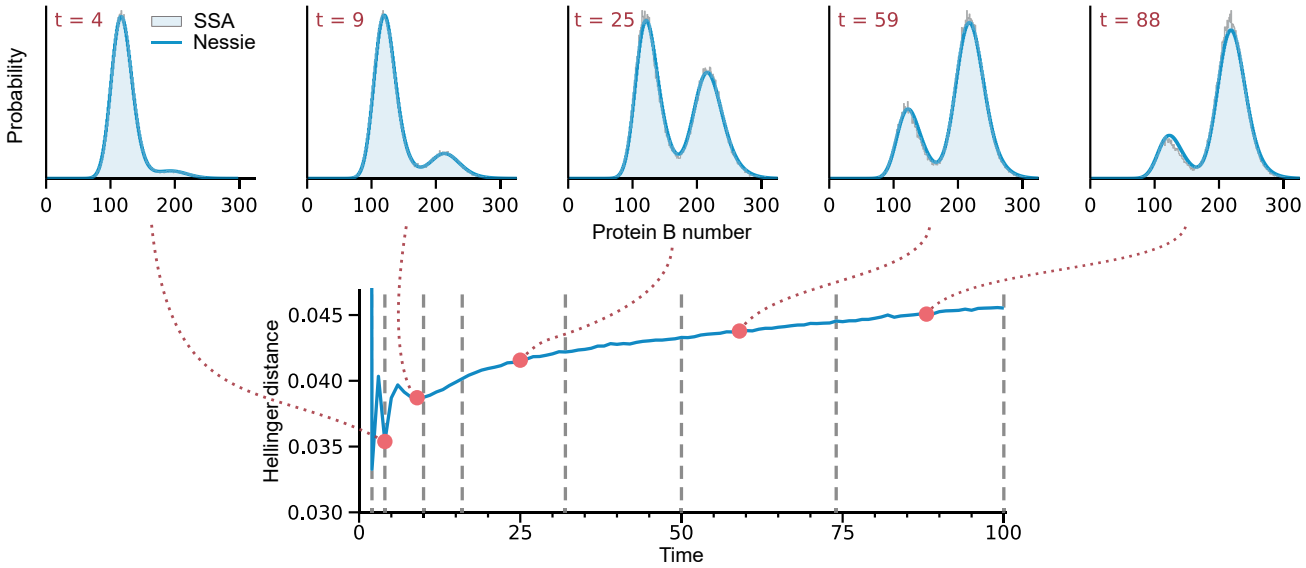


Figure 6. Using Nessie to interpolate and predict protein *B* dynamics in time

Bottom: Mean Hellinger distance computed over the test dataset made up of 1k different parameter values (constructed using 100k SSA trajectories) evaluated at times $t = \{1, 2, \ldots, 100\}$. The vertical gray dashed lines indicate the time snapshots used for training the neural network, showing that the predictive error does not notably increase between the training points. Top: Time evolution of the protein distribution predicted by Nessie and the SSA (averaged over 100k trajectories) for an example parameter set, demonstrating Nessie's ability to accurately capture the time evolution of the protein *B* distribution.

use 1k simulations for each training datapoint and 100k simulations for the validation and test data (in order to ensure a more accurate comparison to the true distributions). In this case, we use a neural network with a single hidden layer of 1024 neurons and 6 output mixture components and fix the batch size to 1k for the training procedure (see Training Neural Networks for more details). The remainder of our setup is the same as in the previous example. The data generation and neural network training times are given in Table S5.

In Figure 5B we verify that the moments (means and variances) of the protein *B* numbers predicted by Nessie closely match those computed using the SSA for all test datapoints. Furthermore, in Figure 5C, we compare the predicted protein distributions to the true distributions constructed by averaging over 100k SSA realizations. Notably, the trained neural network is able to reconstruct the complex distributions and provides a good approximation to the CME solution of the genetic toggle switch. The promising performance of Nessie highlights the usefulness of neural emulators for dealing with stochastic chemical reaction networks that go beyond the more tractable examples that are typically studied in the literature; we demonstrate this further by applying Nessie to a detailed model of mRNA turnover in the next section.

Next, we explore the sensitivity of the genetic toggle switch to noise. The Fano factor, which is defined as the ratio of the variance of molecule numbers to the mean molecule number, is a commonly used measure of deviations from Poisson noise and the extent of transcriptional/translational bursting. We investigate how the Fano factor of the protein *B* number changes on parameter perturbation. Using the trained neural network we have computed the logarithmic sensitivity (Ingalls, 2008) of the Fano factor of protein *B* to all reaction parameters over a wide parameter range and identified the most and least sensitive parameters on average, as shown in the pie charts in Figure 5D. In particular, we can identify a few parameters that are the most or least sensitive to noise in the majority of cases. For example, for over 60% of the parameter space, the mRNA production rates ρ_u^B and ρ_b^B are the most sensitive, and hence tweaking these parameters is usually the optimal way to control the fluctuations in the protein *B* number. Note that performing such global sensitivity analysis is highly computationally expensive using the SSA, whereas with Nessie it can be approximated within minutes, making it possible to significantly accelerate further parameter exploration studies.

Although we trained Nessie with distribution snapshots only at a few fixed time points, an obvious question of interest is whether the neural network can capture the temporal dynamics of the chemical system over its whole trajectory. In Figure 6 we plot the Hellinger distance between the predicted and true (SSA)

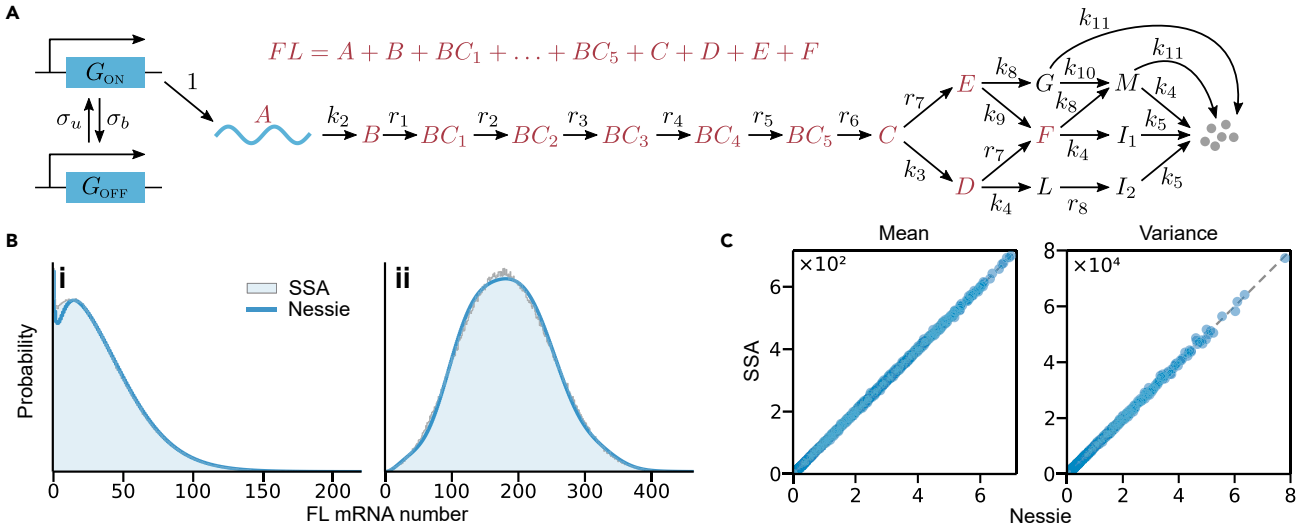


Figure 7. Nessie applied to a detailed model of mRNA turnover

(A) Schematic of the reaction network (assuming mass action kinetics for all reactions). We are interested in modeling the sum of full-length mRNA segments (the FL mRNA number), highlighted in red. Note that the mRNA fragment G is not related to the gene states G_{ON} and G_{OFF} (the notation is kept consistent with the original model (D. Cao and Parker, 2001)).

(B) FL mRNA number distributions for two different test parameter values. The SSA distributions were computed by averaging over 1M trajectories.

(C) Comparison of true and predicted means and variances of the FL mRNA numbers for the test set consisting of 1k different parameter values at time t = 500 constructed using 100k SSA trajectories. Parameter values and ranges are given in Table S3.

distributions at times $t = \{1, 2, \ldots, 100\}$ averaged over the 1k test parameter sets. Remarkably, the predictive accuracy is largely similar throughout the whole time range, indicating that the neural network is able to effectively interpolate between the time points it has seen during training. Note that the worse performance at $t < 2$ is expected as we do not train on the initial transient during which the dynamics rapidly evolve from the initial condition.

Model of mRNA turnover

In this section we consider a detailed model of eukaryotic mRNA turnover, first proposed by D. Cao and Parker (2001) and consisting of the following reactions:

$$G_{ON} \underset{\sigma_u}{\overset{\sigma_b}{\rightleftharpoons}} G_{OFF}, G_{ON} \xrightarrow{1} A, A \xrightarrow{k_2} B,$$
$$B \xrightarrow{r_1} BC_1, BC_1 \xrightarrow{r_2} BC_2, BC_2 \xrightarrow{r_3} BC_3, BC_3 \xrightarrow{r_4} BC_4, BC_4 \xrightarrow{r_5} BC_5, BC_5 \xrightarrow{r_6} C,$$
$$C \xrightarrow{r_7} E, C \xrightarrow{k_3} D, E \xrightarrow{k_8} G, E \xrightarrow{k_9} F, D \xrightarrow{r_7} F, D \xrightarrow{k_4} L,$$
$$G \xrightarrow{k_{10}} M, G \xrightarrow{k_{11}} \varnothing, F \xrightarrow{k_8} M, F \xrightarrow{k_4} I_1, L \xrightarrow{r_8} I_2,$$
$$M \xrightarrow{k_{11}} \varnothing, M \xrightarrow{k_4} \varnothing, I_1 \xrightarrow{k_5} \varnothing, I_2 \xrightarrow{k_5} \varnothing.$$

This reaction network contains a single gene with two states G_{ON} and G_{OFF}, which in the active state G_{ON} produces nuclear mRNA molecules A that are then degraded in a complex downstream pathway. The Nuclear mRNA A is transported to the cytoplasm where it undergoes deadenylation followed by decapping and exonucleolytic degradation either in 3′→5′ or 5′→3′ direction (Cao and Parker, 2001), modeled as a complex sequence of first-order reactions (see Figure 7A). Note that although in the original model mRNA production is described as occurring constitutively, we have extended it to include gene state switching in order to account for transcriptional bursting (Suter et al., 2011).

The mRNA degradation model contains even more species and reaction parameters than the genetic toggle switch presented in the Main Text, and approximating it using our framework is a further example of how Nessie can be applied to study large reaction networks. Our focus here is on predicting the probability distributions of the total number of full-length mRNA segments (FL) that have not yet been exposed to exonucleolytic degradation, given by the sum $A + B + BC_1 + \ldots + BC_5 + C + D + E + F$.

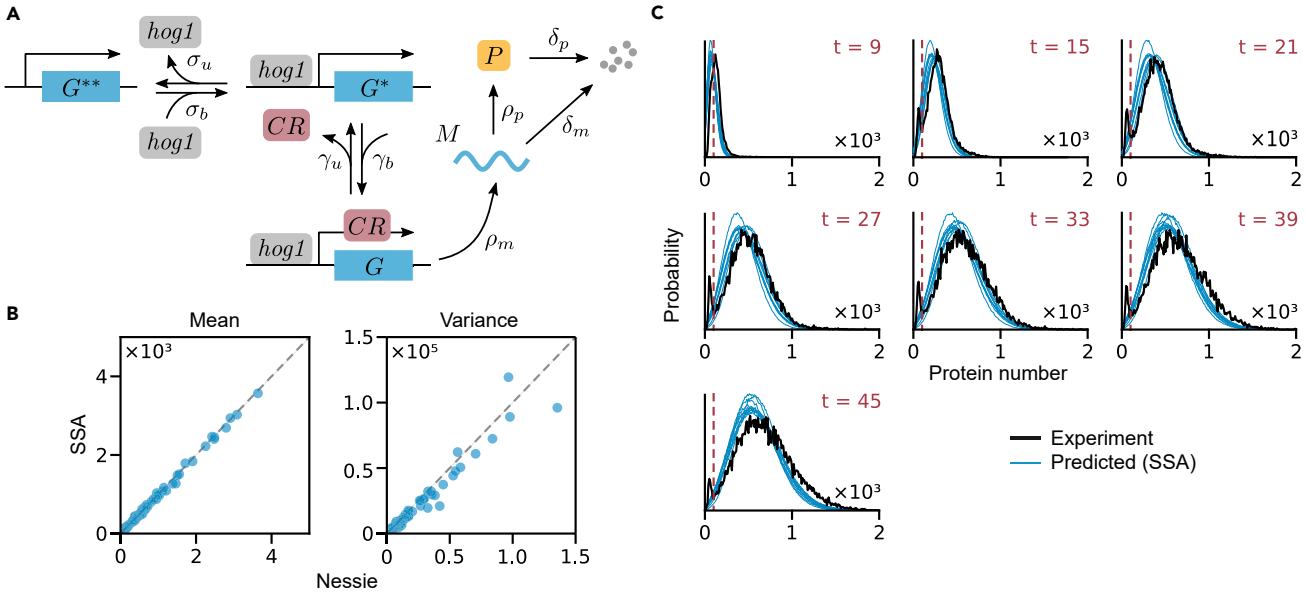


Figure 8. Nessie applied to a model of the MAPK pathway

(A) Schematic of the reaction network (see supplemental information for details). The hog1 concentration over time is taken from Zechner et al. (2012).

(B) True and predicted means and variances of the protein distribution for the test set consisting of 100 different parameter values at time $t = 27$ constructed using 100k SSA trajectories.

(C) Comparison of the experimental distribution (black) with those predicted by the CME for the parameters inferred using Nessie (blue). To probe parameter uncertainty we performed 10 independent estimation rounds. SSA distributions were computed by averaging over 1M trajectories. Parameter values and ranges are given in Table S4.

We draw 100k, 100, and 1k parameters for the training, validation, and test datasets respectively using a logarithmic Sobol sequence in the parameter region indicated in Table S3 For each parameter set we take 8 snapshots at times $t = \{100, 230, 360, 500, 620, 750, 880, 1000\}$. We generate 1k SSA trajectories for each training datapoint and 100k simulations for the validation and test data. The number of hidden neurons is 1024 neurons, and we used 4 output mixture components, using a batch size of 1k for the training procedure. The times required to generate the data and train the neural network are given in Table S5.

In Figure 7B we show that the predicted FL mRNA distributions closely match the true distributions constructed using 1M SSA realizations. Furthermore, in Figure 7C, the means and variances of the FL mRNA numbers predicted by Nessie accurately compare with those computed using the SSA thus demonstrating that Nessie can perform well even in larger-scale applications.

Mitogen-activated protein kinase pathway

We finally apply Nessie to a biological model of the MAPK pathway in *S. Cerevisiae* with the aim of inferring system parameters using experimental data from Zechner et al. (2012). The reaction network can be seen in Figure 8A and is modified from Zechner et al. (2012), removing extrinsic noise contributions. It consists of the following reactions:

$$G^{**} \underset{\eta_u}{\overset{\eta_b(t)}{\rightleftharpoons}} G^*, G^* + CR \underset{\sigma_u}{\overset{\sigma_b}{\rightleftharpoons}} G, G \xrightarrow{\rho_m} G + M, M \xrightarrow{\rho_p} M + P, M \xrightarrow{\delta_m} \varnothing, P \xrightarrow{\delta_p} \varnothing.$$

where $\eta_b(t)$ depends on the current Hog1 concentration, which was measured experimentally, via the formula

$$\eta_b(t) = \frac{V_{\max}(\mathrm{hog1}(t) + b)^h}{K_m^h + (\mathrm{hog1}(t) + b)^h}$$

The number of ribosomes and chromatin remodellers in Zechner et al. (2012) were treated as constant and absorbed into the reaction rates ρ_p.

This reaction network describes the pSTL1 gene and includes activation owing to a time-varying hog1 signal, chromatin remodeling, transcription, and translation. When yeast is subjected to external osmotic pressure, activation of the MAPK signaling cascade results in doubly phosphorylated hog1 molecules entering the nucleus. These bind to the pSTL1 promoter, which is initially in an inactive state (G^{**}). Upon binding of hog1 to the gene, subject to chromatin structure remodeling via the chromatin remodeling complex (CR), starts transcribing mRNA, which after translocation into the cytosol is translated into protein. Zechner et al. (2012) used flow cytometry to measure protein number distributions for the MAPK pathway, for which they proposed the above model. Protein distributions predicted by this model tend to be bimodal with a sharp peak at 0, as depending on the parameters a sizeable fraction of cells never starts transcribing mRNA before the hog1 signal decays.

The data measured by Zechner et al. (2012) consist of intensity measurements (in arbitrary units, AU) at times $t = \{3, 9, 15, 21, 27, 33, 39, 45\}$ (in minutes) after salt was added to the solution to induce osmotic shock in the cells, which triggers the MAPK pathway. As the fluorescence intensity per protein (I/P) was not measured in these experiments we assumed a value of 1AU per protein and rounded the estimated protein number to the nearest integer (Zechner et al. (2012) noted that identifying all parameters is not possible from these experiments).

Observing that the experimental distributions at most times had a peak near 0 whose width was consistent across time points, we binned all observations less than 100AU, the approximate width of the peaks. The observed peaks are best explained by measurement noise that does not allow us to identify the exact protein numbers in the low copy number regime. Binning in this case, while potentially losing some information, renders the procedure more reliable than the alternative of discarding observations below the threshold (Fu et al., 2022; Chen et al., 2022). As measurements at time $t = 3$ were almost entirely below the threshold we discarded that time point for inference purposes.

Our goal is to find parameters consistent with the data by minimizing the discrepancy between the experimentally observed distributions and the model output as predicted by Nessie. The training, validation, and test sets consist of 15k, 250, and 100 points, respectively, which were generated from a logarithmic Sobol sequence in the parameter region indicated in Table S4, chosen around the maximum a posteriori estimates reported by Zechner et al. (2012). We use 1k simulations for each training datapoint and 100k simulations for the validation and test data. The size of the hidden layer was set to 2048 and the number of mixture components was 5. As a significant fraction of the simulated trajectories had 0 proteins we added a sixth component that was set to be a Dirac delta at 0; this was performed by adding a single output neuron predicting the weight of this peak. To speed up training we split the procedure into two rounds, first training with 10% of the training data for 100 rounds and then with the entire training set for 400. We used a batch size of 1k throughout. Figure 8B shows that the means and variances predicted by the resulting neural network are close to those obtained using the SSA.

Once Nessie is trained we estimate the model parameters by minimizing the sum of Hellinger distances between the experimental distributions and Nessie's predictions, treating all observations below 100 AU as lying in one bin as discussed above. As the outputs of the neural network are fully differentiable with respect to its inputs, including the model parameters, we can perform minimization using any standard gradient-based algorithm. This is similar to the training procedure discussed in Section 2, except that we fix the neural weights and vary the model parameters instead. To evaluate the accuracy of the estimation scheme we ran the SSA at the predicted parameters, verifying that the results match the experimentally observed distribution (see Figure 8C for the results using 10 estimated parameter sets). As can be seen in the figure our results do not reproduce the peak near 0 found in the experimental input, and an extensive parameter search did not lead to any parameters which exactly reproduce the experimental data. We, therefore, suspect that the model we used does not fully describe the dynamics of hog1-mediated gene expression and that obtaining better results will require a more detailed model than the one we are using.

While neural networks could be used to perform maximum likelihood estimation or Bayesian inference (Lueckmann et al., 2018), we did not pursue likelihood-based approaches in this article. Owing to large number of datapoints (over 100k), any small approximation error in the distributions predicted by Nessie will get amplified by several orders of magnitude: as the likelihoods for each datapoint add up to form the total likelihood of the data, the errors in the likelihood will, too. This leads to highly fluctuating

likelihood values for similar parameters that are an artifact of the neural approximation and not present in the true model. This results in the predicted posterior being concentrated tightly around one parameter set where these fluctuations result in a marginally higher likelihood for the experimental data than the others, and the resulting uncertainty estimates reflect the approximation error incurred by Nessie instead of true parameter uncertainty. Such concentration of the estimated posterior owing to randomness is a common problem in using MCMC with "tall data" (Bardenet et al., 2017), where estimating likelihoods for large datasets becomes very difficult.

As fitting parameters using Hellinger distances does not directly provide uncertainty information, we estimated uncertainty by repeatedly fitting parameters to the experimental data; Table S4 shows the results of 10 fits. As can be seen in Figure 8B, these results produce similar distributions under the CME, yet some parameters such as σ_b and δ_p are spread over an order of magnitude. Such parameter unidentifiability is common with the type of experimental data measured in biological experiments and should be taken into account when interpreting results. In particular, the Hill coefficient, which was allowed to range from 1 to 10, could not be narrowed down within this range.

Once the network is trained, globally optimizing the Hellinger distance within the targeted parameter region takes a few minutes. Our approach should, therefore, be particularly suited for scenarios where distribution data is available for many copies of one network, e.g. when using a single gene expression model to analyze many different genes in an organism.

DISCUSSION

In this article, we presented Nessie, a framework that allows us to train neural networks on simulation data to accurately estimate the solution of the CME for various biological systems. Our approach is scalable to complex nonlinear reaction networks with over a dozen parameters that exhibit diverse, multimodal dynamics across parameter space. We illustrated the performance of Nessie on four examples: a well-studied autoregulatory feedback loop, a complex genetic toggle switch, a detailed model of mRNA turnover, and the MAPK pathway in *S. Cerevisiae*. The latter models pose significant challenges both analytically and computationally owing to the number of species and reactions involved, yet Nessie allows us to efficiently emulate them and analyze their properties, with applications for parameter exploration and estimation. Although the models we have tested in this work were all Markovian, the simulator-based nature of our approach makes it suitable for non-Markovian models including delays, see e.g. Barrio et al. (2006) and Leier and Marquez-Lago (2015).

Nessie can be particularly useful in rapidly exploring large swathes of parameter space, for example, to perform a local or global sensitivity analysis. This has many uses, e.g. to guide the tuning of parameters to find desired phenotype (Feng et al., 2004) for the design of optimal experiments (Zak et al., 2003), to provide insights into the robustness and fragility tradeoff in genetic regulatory mechanisms (Stelling et al., 2004) and to find those parameters which most influence the size of transcriptional noise (Cao et al., 2020). We note that performing any such analysis using the standard stochastic simulation methods like the SSA can be prohibitively computationally expensive (Gunawan et al., 2005).

Following methodology similar to that proposed by Lueckmann et al. (2018), Nessie can be used to fit models to data by matching experimentally observed distributions to those predicted by the neural network, as demonstrated in the case of the MAPK pathway model where we recovered model parameters that are most consistent with experimental observations. We remark, however, that this approach has to be used with care in the context of likelihood-based inference owing to small approximation errors in the likelihood being amplified in the presence of many datapoints. In order to be reliable any such approach, including Bayesian inference, must take into account the bias introduced by the choice of approximation. This could be conducted e.g. by placing a prior over network weights and treating them as unobserved variables, sampling from the resulting Bayesian neural network using Hamiltonian Monte Carlo methods.

As discussed in the Introduction, our approach differs from other studies that use neural networks to predict the dynamics of stochastic biochemical systems (Gupta et al., 2021; Bortolussi and Palmieri, 2018; Repin and Petrov, 2021; Davis et al., 2020; Cairoli et al., 2021). As all of these approaches try to learn different things, comparing them directly is not straightforward and is further complicated by the sensitivity of neural networks to architecture and hyperparameter choices (Goodfellow et al., 2017). An advantage of Nessie as

presented in this article is that it requires relatively little setup in terms of hyperparameter optimization. Owing to its architectural simplicity, a very limited amount of tuning is required, without requiring automated neural architecture search techniques (Repin and Petrov, 2021), and we provide a detailed discussion of the relevant hyperparameter and training considerations in the STAR Methods. We hope that this will enable the interested reader to quickly deploy and apply Nessie to their favorite reaction network.

Limitations of the study

Although Nessie can relatively accurately interpolate in time between the training snapshots for such models as the genetic toggle switch, its performance may be inadequate when applied to systems with complex oscillatory behavior. This is a general limitation of our approach, which uses a simple feedforward network and therefore may not be able to efficiently represent oscillating functions. To remedy this, besides the recently proposed generative adversarial network-based approach in Cairoli et al. (2021) one could consider more sophisticated neural network architectures such as recurrent neural networks (Goodfellow et al., 2017) and universal differential equations (Rackauckas and Nie, 2017). This would allow us to extract temporal features such as power spectra and first passage times, which, while difficult to measure experimentally, have been shown to provide a wealth of information about the system and significantly aid in model discrimination (Jia and Grima, 2021; Szavits-Nossan and Grima, 2022; Iyer-Biswas and Zilman, 2016).

While in this article we have focused on the task of learning one-dimensional marginal distributions predicted by the CME, Nessie can also be extended to capture joint distributions. One way to implement this could be by replacing the mixture of univariate negative binomials with a mixture of independent negative multinomials or alternative multivariate distributions. Such generalization only requires updating the output layer to correctly represent the parameters of the new mixture. However, we expect the computational cost of training such a network to greatly increase as the state space (and hence the number of required gradient computations) grows exponentially. The construction of training datasets may also become significantly more expensive, as generating sufficiently smooth multi-dimensional histograms with the SSA may require many more trajectories than in the one-dimensional case. Another way to learn multivariate distributions was recently proposed by Gorin et al. (2022) and consist of learning the distribution of one species conditioned on the number of another species, whose marginal distribution is known beforehand (or can be learned). Exploring this and other possible ways to efficiently approximate joint distributions using Nessie remains an interesting avenue for future research.

STAR★METHODS

Detailed methods are provided in the online version of this paper and include the following:

- KEY RESOURCES TABLE
- RESOURCE AVAILABILITY
  - Lead contact
  - Materials availability
  - Data and code availability
- METHOD DETAILS
  - Chemical master equation
  - Background on neural networks
  - Training neural networks
  - Hyperparameter tuning

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.isci.2022.105010.

ACKNOWLEDGMENTS

This work was supported by the Alan Turing Institute Doctoral Studentship (under the EPSRC grant EP/N510129/1) for A. S., the EPSRC Center for Doctoral Training in Data Science (EPSRC grant EP/L016427/1), and the University of Edinburgh for K. Ö. and a Leverhulme Trust grant (Grant No. RPG-2020-327) for R. G. The authors would like to thank Christoph Zechner for the courtesy of sharing his data for the MAPK pathway model.

AUTHOR CONTRIBUTIONS

Conceptualization, R.G.; Methodology, A.S., K.Ö., and R.G.; Software, A.S. and K.Ö.; Analysis, A.S. and K.Ö.; Writing – Original Draft, A.S., and K.Ö.; Writing – Review & Editing, A.S., K.Ö., and R.G.; Supervision, R.G.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: May 2, 2022
Revised: June 13, 2022
Accepted: August 18, 2022
Published: September 16, 2022

REFERENCES

Angermueller, C., Pärnamaa, T., Parts, L., and Stegle, O. (2016). Deep learning for computational biology. Mol. Syst. Biol. *12*, 878. https://doi.org/10.15252/msb.20156651.

Baldi, P., Sadowski, P., and Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. Nat. Commun. *5*, 4308. https://doi.org/10.1038/ncomms5308.

Bardenet, R., Doucet, A., and Holmes, C. (2017). On Markov chain Monte Carlo methods for tall data. J. Mach. Learn. Res. *18*, 1–43.

Barrio, M., Burrage, K., Leier, A., and Tian, T. (2006). Oscillatory regulation of Hes1: discrete stochastic delay modelling and simulation. PLoS Comput. Biol. *2*. e117-14. https://doi.org/10.1371/journal.pcbi.0020117.

Bergstra, J., and Bengio, Y. (2012). Random search for hyper-parameter optimization. J. Mach. Learn. Res. *13*, 25. https://doi.org/10.5555/2188385.2188395.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V.B. (2017). Julia: a fresh approach to numerical computing. SIAM Rev. Soc. Ind. Appl. Math. *59*, 65–98. https://doi.org/10.1137/141000671.

Bishop, C.M. (1994). Mixture Density Networks. https://publications.aston.ac.uk/id/eprint/373/1/NCRG_94_004.pdf.

Bortolussi, L., and Palmieri, L. (2018). Deep abstractions of chemical reaction networks. In Computational Methods in Systems Biology, *11095*, M. Češka and D. Šafránek, eds. Computational Methods in Systems Biology (Springer International Publishing), pp. 21–38. https://doi.org/10.1007/978-3-319-99429-1_2.

Bottou, L., Curtis, F.E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. SIAM Rev. Soc. Ind. Appl. Math. *60*, 223–311. https://doi.org/10.1137/16M1080173.

Braichenko, S., Holehouse, J., and Grima, R. (2021). Distinguishing between models of mammalian gene expression: telegraph-like models versus mechanistic models. J. R. Soc. Interface *18*, 20210510. https://doi.org/10.1098/rsif.2021.0510.

Cai, L., Friedman, N., and Xie, X.S. (2006). Stochastic protein expression in individual cells at the single molecule level. Nature *440*, 358–362. https://doi.org/10.1038/nature04599.

Cairoli, F., Carbone, G., and Bortolussi, L. (2021). Abstraction of markov population dynamics via generative adversarial nets. International Conference on Computational Methods in Systems Biology (Springer), pp. 19–35. https://doi.org/10.1007/978-3-030-85633-5_2.

Cao, D., and Parker, R. (2001). Computational modeling of eukaryotic mRNA turnover. RNA *7*, 1192–1212. https://doi.org/10.1017/S1355838201010330.

Cao, Z., and Grima, R. (2020). Analytical distributions for detailed models of stochastic gene expression in eukaryotic cells. Proc. Natl. Acad. Sci. USA *117*, 4682–4692. https://doi.org/10.1073/pnas.1910888117.

Cao, Z., Filatova, T., Oyarzún, D.A., and Grima, R. (2020). A stochastic model of gene expression with polymerase recruitment and pause release. Biophys. J. *119*, 1002–1014. https://doi.org/10.1016/j.bpj.2020.07.020.

Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., and Zdeborová, L. (2019). Machine learning and the physical sciences. Rev. Mod. Phys. *91*, 045002. https://doi.org/10.1103/RevModPhys.91. 045002.

Chen, L., Zhu, C., and Jiao, F. (2022). A generalized moment-based method for estimating parameters of stochastic gene transcription. Math. Biosci. *345*, 108780. https://doi.org/10.1016/j.mbs.2022.108780.

Choi, P.J., Cai, L., Frieda, K., and Xie, X.S. (2008). A stochastic single-molecule event triggers phenotype switching of a bacterial cell. Science *322*, 442–446. https://doi.org/10.1126/science.1161427.

Davis, C.N., Hollingsworth, T.D., Caudron, Q., and Irvine, M.A. (2020). The use of mixture density networks in the emulation of complex epidemiological individual-based models. PLoS Comput. Biol. *16*. e1006869. https://doi.org/10.1371/journal.pcbi.1006869.

Dinh, T., and Sidje, R.B. (2020). An adaptive solution to the Chemical Master Equation using quantized tensor trains with sliding windows. Phys. Biol. *17*, 065014. https://doi.org/10.1088/1478-3975/aba1d2.

Elowitz, M.B., Levine, A.J., Siggia, E.D., and Swain, P.S. (2002). Stochastic gene expression in a single cell. Science *297*, 1183–1186. https://doi.org/10.1126/science.1070919.

Feng, X.-J., Hooshangi, S., Chen, D., Li, G., Weiss, R., and Rabitz, H. (2004). Optimizing genetic circuits by global sensitivity analysis. Biophys. J. *87*, 2195–2202. https://doi.org/10.1529/biophysj.104.044131.

Friedman, N., Cai, L., and Xie, X.S. (2006). Linking stochastic dynamics to population distribution: an analytical framework of gene expression. Phys. Rev. Lett. *97*, 168302. https://doi.org/10.1103/PhysRevLett.97.168302.

Fu, X., Patel, H.P., Coppola, S., Xu, L., Cao, Z., Lenstra, T.L., and Grima, R. (2022). Quantifying how posttranscriptional noise and gene copy number variation bias transcriptional parameter inference from mRNA distributions. Preprint at bioRxiv. https://doi.org/10.1101/2021.11.09.467882.

Gardner, T.S., Cantor, C.R., and Collins, J.J. (2000). Construction of a genetic toggle switch in Escherichia coli. Nature *403*, 339–342. https://doi.org/10.1038/35002131.

Gillespie, D.T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J. Comput. Phys. *22*, 403–434. https://doi.org/10.1016/0021-9991(76)90041-3.

Gillespie, D.T. (2007). Stochastic simulation of chemical kinetics. Annu. Rev. Phys. Chem. *58*, 35–55. https://doi.org/10.1146/annurev.physchem.58.032806.104637.

Glorot, X., and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics JMLR Workshop and Conference Proceedings, pp. 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In 14th International Conference on Artificial Intelligence and Statistics, pp. 315–323.

Goodfellow, I., Bengio, Y., Courville, A., and Bach, F. (2017). Deep Learning (MIT Press).

Gorin, G., Carilli, M., Chari, T., and Pachter, L. (2022). Spectral neural approximations for models of transcriptional dynamics. Preprint at bioRxiv. https://doi.org/10.1101/2022.06.16.496448.

Grima, R., Schmidt, D.R., and Newman, T.J. (2012). Steady-state fluctuations of a genetic feedback loop: an exact solution. J. Chem. Phys. *137*, 035104. https://doi.org/10.1063/1.4736721.

Gunawan, R., Cao, Y., Petzold, L., and Doyle, F.J., III (2005). Sensitivity analysis of discrete stochastic systems. Biophys. J. *88*, 2530–2540. https://doi.org/10.1529/biophysj.104.053405.

Gupta, A., Schwab, C., and Khammash, M. (2021). DeepCME: a deep learning framework for computing solution statistics of the Chemical Master Equation. PLoS Comput. Biol. *17*, e1009623. https://doi.org/10.1371/journal.pcbi.1009623.

Hafner, M., Landthaler, M., Burger, L., Khorshid, M., Hausser, J., Berninger, P., Rothballer, A., Ascano, M., Jr., Jungkamp, A.C., Munschauer, M., et al. (2010). Transcriptome-wide identification of RNA-binding protein and microRNA target sites by PAR-CLIP. Cell *141*, 129–141. https://doi.org/10.1016/j.cell.2010.03.009.

Hjorth, L. (1999). Regularisation of mixture density networks. In 9th International Conference on Artificial Neural Networks, *1999* (IEEE), pp. 521–526. https://doi.org/10.1049/cp:19991162.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. Neural Network. *4*, 251–257. https://doi.org/10.1016/0893-6080(91)90009-T.

Ingalls, B. (2008). Sensitivity analysis: from model parameters to system behaviour. Essays Biochem. *45*, 177–193. https://doi.org/10.1042/bse0450177.

Innes, M. (2018). Flux: elegant machine learning with Julia. J. Open Source Softw. *3*, 602. https://doi.org/10.21105/joss.00602.

Innes, M., Edelman, A., Fischer, K., Rackauckas, C., Saba, E., Shah, V.B., and Tebbutt, W. (2019). A differentiable programming system to bridge machine learning and scientific computing. Preprint at arXiv. https://doi.org/10.48550/ARXIV. 1907.07587.

Iyer-Biswas, S., and Zilman, A. (2016). First-passage processes in cellular biology. Adv. Chem. Phys. 261–306. https://doi.org/10.1002/9781119165156.ch5.

Jia, C., and Grima, R. (2020). Small protein number effects in stochastic models of autoregulated bursty gene expression. J. Chem. Phys. *152*, 084115. https://doi.org/10.1063/1.5144578.

Jia, C., and Grima, R. (2021). Frequency domain analysis of fluctuations of mRNA and protein copy numbers within a cell lineage: theory and experimental validation. Phys. Rev. X *11*, 021032. https://doi.org/10.1103/PhysRevX.11.021032.

Jiang, Q., Fu, X., Yan, S., Li, R., Du, W., Cao, Z., Qian, F., and Grima, R. (2021). Neural network aided approximation and parameter inference of non-Markovian models of gene expression. Nat. Commun. *12*, 2618. https://doi.org/10.1038/s41467-021-22919-1.

Kazeev, V., and Schwab, C. (2015). Tensor approximation of stationary distributions of chemical reaction networks. SIAM J. Matrix Anal. Appl. *36*, 1221–1247. https://doi.org/10.1137/130927218.

Kazeev, V., Khammash, M., Nip, M., and Schwab, C. (2014). Direct solution of the Chemical Master Equation using quantized tensor trains. PLoS Comput. Biol. *10*, e1003359. https://doi.org/10.1371/journal.pcbi.1003359.

Keskar, N., Nocedal, J., Tang, P., Mudigere, D., and Smelyanskiy, M. (2017). On large-batch training for deep learning: generalization gap and sharp minima. In 5th International Conference on Learning Representations.

Kingma, D.P., and Ba, J. (2014). Adam: a method for stochastic optimization. Preprint at arXiv. https://doi.org/10.48550/ARXIV. 1412.6980.

Leier, A., and Marquez-Lago, T.T. (2015). Delay chemical master equation: direct and closed-form solutions. Proc. Math. Phys. Eng. Sci. *471*, 20150049. https://doi.org/10.1098/rspa.2015.0049.

Loman, T., Ma, Y., Ilin, V., Gowda, S., Korsbo, N., Yewale, N., Rackauckas, C.V., and Isaacson, S.A. (2022). Catalyst: fast biochemical modeling with Julia. Preprint at bioRxiv. https://doi.org/10.1101/2022.07.30.502135.

Lueckmann, J.-M., Bassetto, G., Karaletsos, T., and Macke, J.H. (2018). Likelihood-free inference with emulator networks. In 1st Symposium on Advances in Approximate Bayesian Inference, *16*.

Maheshri, N., and O'Shea, E.K. (2007). Living with noisy genes: how cells function reliably with inherent variability in gene expression. Annu. Rev. Biophys. Biomol. Struct. *36*, 413–434. https://doi.org/10.1146/annurev.biophys. 36.040306.132705.

McAdams, H.H., and Arkin, A. (1999). It's a noisy business! Genetic regulation at the nanomolar scale. Trends Genet. *15*, 65–69. https://doi.org/10.1016/S0168-9525(98)01659-X.

Mehta, P., Wang, C.H., Day, A.G.R., Richardson, C., Bukov, M., Fisher, C.K., and Schwab, D.J. (2019). A high-bias, low-variance introduction to Machine Learning for physicists. Phys. Rep. *810*, 1–124. https://doi.org/10.1016/j. physrep.2019.03.001.

Min, S., Lee, B., and Yoon, S. (2017). Deep learning in bioinformatics. Brief. Bioinform. *18*, 851–869. https://doi.org/10.1093/bib/bbw068.

Munsky, B., and Khammash, M. (2006). The finite state projection algorithm for the solution of the Chemical Master Equation. J. Chem. Phys. *124*, 044104. https://doi.org/10.1063/1.2145882.

Öcal, K., Gutmann, M.U., Sanguinetti, G., and Grima, R. (2022). Inference and uncertainty quantification of stochastic gene expression via synthetic models. J. R. Soc. Interface *19*, 20220153. https://doi.org/10.1098/rsif.2022.0153.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). PyTorch: an imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, *32*, H.M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E.B. Fox, and R. Garnett, eds, pp. 8024–8035.

Perez-Carrasco, R., Beentjes, C., and Grima, R. (2020). Effects of cell cycle variability on lineage and population measurements of messenger RNA abundance. J. R. Soc. Interface *17*, 20200360. https://doi.org/10.1098/rsif. 2020.0360.

Phillips, N.E., Hugues, A., Yeung, J., Durandau, E., Nicolas, D., and Naef, F. (2021). The circadian oscillator analysed at the single-transcript level. Mol. Syst. Biol. *17*, e10135. https://doi.org/10.15252/msb.202010135.

Prechelt, L. (2012). Early stopping – but when? In Neural Networks: Tricks of the Trade, Second edition, G. Montavon, G.B. Orr, and K.-R. Müller, eds. (Springer), pp. 53–67. https://doi.org/10.1007/978-3-642-35289-8_5.

Rackauckas, C., and Nie, Q. (2017). DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia. J. Open Res. Softw. *5*, 15. https://doi.org/10.5334/jors.151.

Raser, J.M., and O'Shea, E.K. (2005). Noise in gene expression: origins, consequences, and control. Science *309*, 2010–2013. https://doi.org/10.1126/science.1105891.

Repin, D., and Petrov, T. (2021). Automated deep abstractions for stochastic chemical reaction networks. Inf. Comput. *281*, 104788. https://doi.org/10.1016/j.ic.2021.104788.

Schnoerr, D., Sanguinetti, G., and Grima, R. (2017). Approximation and inference methods for stochastic biochemical kinetics - a tutorial review. J. Phys. A Math. Theor. *50*, 093001. https://doi.org/10.1088/1751-8121/aa54d9.

Shahrezaei, V., and Swain, P.S. (2008). Analytical distributions for stochastic gene expression. Proc. Natl. Acad. Sci. USA *105*, 17256–17261. https://doi.org/10.1073/pnas.0803850105.

Singer, Z.S., Yong, J., Tischler, J., Hackett, J.A., Altinok, A., Surani, M.A., Cai, L., and Elowitz, M.B. (2014). Dynamic heterogeneity and DNA methylation in embryonic stem cells. Mol. Cell *55*, 319–331. https://doi.org/10.1016/j.molcel.2014.06.029.

Sobol, I.M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. USSR Comput. Math. Math. Phys. *7*, 86–112. https://doi.org/10.1016/0041-5553(67)90144-9.

Stelling, J., Gilles, E.D., and Doyle, F.J. (2004). Robustness properties of circadian clock architectures. Proc. Natl. Acad. Sci. USA *101*, 13210–13215. https://doi.org/10.1073/pnas.0401463101.

Suter, D.M., Molina, N., Gatfield, D., Schneider, K., Schibler, U., and Naef, F. (2011). Mammalian genes are transcribed with widely different bursting kinetics. Science *332*, 472–474. https://doi.org/10.1126/science. 1198817.

Szavits-Nossan, J., and Grima, R. (2022). Steady-state distributions of nascent RNA for general

initiation mechanisms. Preprint at bioRxiv. https://doi.org/10.1101/2022.03.30.486441.

Taniguchi, Y., Choi, P.J., Li, G.-W., Chen, H., Babu, M., Hearn, J., Emili, A., and Xie, X.S. (2010). Quantifying E.coli proteome and transcriptome with single-molecule sensitivity in single cells. Science *329*, 533–538. https://doi.org/10.1126/science.1188308.

Thomas, P., Popović, N., and Grima, R. (2014). Phenotypic switching in gene regulatory networks. Proc. Natl. Acad. Sci. USA *111*, 6994–6999. https://doi.org/10.1073/pnas.1400049111.

Van Kampen, N. (2007). Stochastic Processes in Physics and Chemistry, Third edition (Elsevier).

Wang, S., Fan, K., Luo, N., Cao, Y., Wu, F., Zhang, C., Heller, K.A., and You, L. (2019). Massive computational acceleration by using neural networks to emulate mechanism-based biological models. Nat. Commun. *10*, 4354. https://doi.org/10.1038/s41467-019-12342-y.

Wilkinson, D.J. (2018). Stochastic Modelling for Systems Biology, Third edition (Chapman and Hall/CRC).

Xu, H., Skinner, S.O., Sokac, A.M., and Golding, I. (2016). Stochastic kinetics of nascent RNA. Phys. Rev. Lett. *117*, 128101. https://doi.org/10.1103/PhysRevLett.117.128101.

Zak, D.E., Gonye, G.E., Schwaber, J.S., and Doyle, F.J. (2003). Importance of input perturbations and stochastic gene expression in the reverse engineering of genetic regulatory networks: insights from an identifiability analysis of an in silico network. Genome Res. *13*, 2396–2405. https://doi.org/10.1101/gr.1198103.

Zechner, C., Ruess, J., Krenn, P., Pelet, S., Peter, M., Lygeros, J., and Koeppl, H. (2012). Moment-based inference predicts bimodality in transient gene expression. Proc. Natl. Acad. Sci. USA *109*, 8340–8345. https://doi.org/10.1073/pnas.1200161109.

STAR★METHODS

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
MAPK pathway cytometry data	Zechner et al. (2012)	
Software and algorithms		
Nessie framework	In this paper (and https://github.com/augustinas1/Nessie)	https://doi.org/10.5281/zenodo.6978865
Julia v1.7	https://julialang.org/	Bezanson et al. (2017)
Flux.jl v0.13.3	https://github.com/FluxML/Flux.jl	Innes (2018)
Zygote.jl v0.6.4	https://github.com/FluxML/Zygote.jl	Innes et al. (2019)
Catalyst.jl v10.8.0	https://github.com/SciML/Catalyst.jl	Loman et al. (2022)
DifferentialEquations.jl v7.1.0	https://github.com/SciML/DifferentialEquations.jl	Rackauckas and Nie (2017)
FiniteStateProjection.jl v0.2.0	https://github.com/kaandocal/FiniteStateProjection.jl	
BlackBoxOptim.jl v0.6.1	https://github.com/robertfeldt/BlackBoxOptim.jl	

RESOURCE AVAILABILITY

Lead contact

Further information and requests should be directed to and will be fulfilled by the lead contact, Ramon Grima (ramon.grima@ed.ac.uk).

Materials availability

This study did not generate new unique reagents.

Data and code availability

- The neural network training data can be generated using the original code, or it can be shared by the lead contact upon request. The MAPK Pathway inference example uses data from Zechner et al. (2012) which can be obtained from the authors.
- All original code has been deposited at https://github.com/augustinas1/Nessie and is publicly available as of the date of publication. The DOI is listed in the key resources table.
- Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

METHOD DETAILS

Chemical master equation

This section aims to provide a brief review of the CME and its use to model stochastic reaction networks in biology. We refer to Schnoerr et al. (2017) and Van Kampen (2007) for a readable and comprehensive treatment of the theory.

A biochemical reaction network consists of species X_i ($i = 1,\ldots,N$) and R reactions of the form

$$a_{1j}\,X_1 + \ldots + a_{Nj}\,X_N \rightarrow b_{1j}\,X_1 + \ldots + b_{Nj}\,X_N, \qquad \text{(Equation 3)}$$

where a_{ij} and b_{ij} respectively denote the numbers of reactant and product molecules of species i in the chemical reaction j. The stoichiometric matrix is defined as $S_{ij} = b_{ij} - a_{ij}$ (the net change in the number of molecules of species X_i when reaction j occurs). The state of the system is determined by the state vector $n = (n_1,\ldots,n_N)$, where n_i is the number of molecules of species X_i present in the system.

Starting with initial conditions $P(n, t = 0) = P_0(n)$, the time evolution of the probability distribution over the states $P(n, t)$ is described by the CME:

$$\frac{dP(n,t)}{dt} = \sum_{j=1}^{R} \left[\rho_j(n - S_j, t)P(n - S_j, t) - \rho_j(n, t)P(n, t)\right], \quad \text{(Equation 4)}$$

where $\rho_j(n, t)$ is the propensity function of reaction j and S_j is the j^{th} column of the stoichiometric matrix S. The propensity $\rho_j(n, t)$ is the rate at which reaction j occurs when the system is in state n at time t; more formally, $\rho_j(n, t)dt$ is the probability that reaction j will take place in the infinitesimally short time interval $(t, t + dt)$ (Gillespie, 2007).

Background on neural networks

A neural network learns a mapping f between inputs x and outputs $y = f(x)$ via a parametric approximation f_φ such that $f_\varphi(x) \approx y$. The basic building block of a neural network is a single neuron, which performs the mapping $x \mapsto g(x \cdot w + b)$ for a weight vector w, a bias b and a nonlinear activation function g. Several neurons arranged in parallel form a layer, and their outputs can be treated as inputs to another layer of neurons. By combining several layers in a row one gets a standard feedforward neural network, illustrated in Figure 1. Here the first layer is called the input layer, the last layer is the output layer and the layers in between are hidden layers. For a comprehensive introduction to neural networks and deep learning we refer to Goodfellow et al. (2017).

Using activation functions with each neuron enables neural networks to learn complicated nonlinear mappings. Commonly used activation functions in the Machine Learning community are sigmoid functions, Rectified Linear Units (ReLUs) (Glorot et al., 2011) and variants thereof (Goodfellow et al., 2017). For these activation functions one can show that a feedforward neural network with a single hidden layer and a sufficient number of neurons acts as a universal approximator, i.e. it is able to represent any sufficiently smooth function (Hornik, 1991) to arbitrary accuracy. In theory, a better approximation could be achieved using a deep neural network with multiple hidden layers, which can compose simpler functions into increasingly more complex ones. Such "deep" neural networks, often combining a variety of architectures more complex than a simple feedforward neural network, can outperform their shallow counterparts on difficult tasks such as Natural Language Processing, Computer Vision and others (Goodfellow et al., 2017), which has led to a surge of interest in Deep Learning in recent years (Angermueller et al., 2016; Min et al., 2017; Carleo et al., 2019; Mehta et al., 2019; Baldi et al., 2014). As we will see, however, a single hidden layer is enough for our purposes.

The network parameters φ (weights and biases for each neuron) that minimize the discrepancy between the mapping f_φ represented by the neural network and the true mapping f, are not known and have to be learned from data. This is commonly achieved by constructing a labeled set of training data $\mathcal{D}$ containing $N \gg 1$ different input-output pairs, and minimizing a *loss function* $\mathcal{L}(\varphi; \mathcal{D})$ that measures the deviation between the neural mapping f_φ and the target:

$$\mathcal{L}(\varphi; \mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\varphi; x^{(i)}, y^{(i)}), \quad \text{(Equation 5)}$$

where $(x^{(i)}, y^{(i)})$ denotes the i^{th} example pair in the training dataset. The most appropriate loss function depends on the type of data and the task the neural network is trying to perform: common examples are L2 distances for regression, cross-entropy for classification and negative log-likelihoods for inference problems (Goodfellow et al., 2017).

As the loss function is often highly nonlinear and nonconvex, minimizing it with respect to the network parameters φ is a difficult task, most often performed using iterative gradient-based optimizers (Bottou et al., 2018). This requires the loss function to be differentiable with respect to the weights. Computing the gradients of the loss function is usually done using the backpropagation algorithm (Goodfellow et al., 2017), which is implemented in most common deep learning frameworks such as Flux (Innes, 2018) or PyTorch (Paszke et al., 2019). Once the gradients have been computed one can use an optimization algorithm such as stochastic gradient descent or Adam (Kingma and Ba, 2014) to minimize the loss. Note that the training set is generated once and reused for every gradient descent iteration.

In practice the behavior of a neural network and its training procedure are determined by a number of hyperparameters, such as the number and size of hidden layers, the activation functions applied on each layer, the choice of the optimizer (as well as its associated parameters) and the convergence criterion. There is no universal formula for determining the best hyperparameter choices for each task, and hence one has to resort to heuristics and hyperparameter tuning to find the best setup, which can be one of the most time-consuming aspects of training complicated neural networks. We discuss these practical considerations in connection to our approach below in the STAR Methods.

Note that minimizing the loss function over the training data does not guarantee that the neural network will be able to *generalize*, i.e. accurately learn the mapping for previously unobserved inputs. For this reason, the network's generalization ability is usually evaluated on a separate validation dataset made up of examples that are not included in the training data (Goodfellow et al., 2017). Comparison of the loss on the training and validation datasets during the training procedure allows us to perform effective hyperparameter tuning. Finally the predictive performance of the trained network can be accurately measured on a separate test dataset consisting of yet another set of input examples.

Training neural networks

We train our neural networks using the Adam optimizer (Kingma and Ba, 2014), one of the most popular optimization algorithms for this purpose. The gradients of the loss function with respect to the network parameters φ are calculated over minibatches of m training points, which are then used to update φ using the optimizer (Goodfellow et al., 2017). One training *epoch* is completed by iterating over all minibatches in the training dataset and hence performing many gradient steps (which can lead to faster convergence). Before training we initialize the network weights using the Glorot Uniform method (Glorot and Bengio, 2010).

The batch size and the learning rate are two optimizer hyperparameters that may significantly affect the results of the training procedure. We adjust these hyperparameters and define our stopping criterion using heuristic arguments outlined below.

Batch Size: It has been noted that large batch size m may reduce the model's ability to generalize, whereas small m can lead to more reliable results (Keskar et al., 2017). In our experiments we observed that very small batch sizes did little to improve results while significantly increasing training time. To balance these observations, we usually choose m to be 2-10% the size of the training dataset, which consistently gave good performance.

Learning Rate: The learning rate η of the optimizer controls the step size of each gradient update and should be chosen appropriately: too low a choice can lead to slow convergence, whereas a large learning rate can overshoot the target minimum. In our experiments we usually initialize $\eta = 0.01$ and decrease it as training progress. Namely, we periodically monitor the loss function over the validation dataset and halve η if the loss has improved by less than 0.5% over the last 25 epochs on average.

Stopping Criterion: The training procedure is terminated after η has been decreased 5 times, which usually indicates that optimization has stalled. We found this stopping criterion to work well for our examples as training beyond this point did not often lead to significant improvements in accuracy.

The learning rate decay described above is similar to early stopping (Prechelt, 2012), a regularization technique that helps to prevent overfitting of the training dataset. Overfitting occurs when the neural network learns, or "memorizes", particular features of the training dataset that are not representative of the model as a whole, and loses its ability to generalize to unseen data. This can often be detected by an increase in the validation loss together with a monotonically decreasing training loss. While a number of popular regularization strategies can be used to prevent this, such as L2 regularization or dropout (Goodfellow et al., 2017), we did not find overfitting to be an issue in our experiments. We conjecture that this is due to the rigid nature of the negative binomial distribution which, unlike a Gaussian, cannot overfit single datapoints away from 0.

Hyperparameter tuning

As noted above, training a neural network effectively requires finding a good architecture and hyperparameters. Beyond manual tuning this is classically done using black-box optimization methods such as grid

search, random search or Bayesian Optimization (Bergstra and Bengio, 2012); the related work by Repin and Petrov (2021) uses a more recent differentiable architecture search. For deep neural networks such approaches can be very computationally expensive depending on the number of hyperparameters. In contrast, our approach based on a single hidden layer can feasibly be tuned manually, and using the autoregulatory feedback loop as a testbed we can obtain intuition about the effect of each hyperparameter for our problem.

Hidden layers

The number and structure of the hidden layers in a neural network greatly affect its capacity, i.e. its ability to represent sufficiently complex functions. In Figure S1A (supplemental information) we plot the Hellinger distance between the true and predicted distributions for a single hidden layer with different numbers of neurons. We observe that the network's capacity quickly grows with the number of neurons, reaching peak accuracy at about 128 neurons. Increasing the number further does not have a measurable effect on our model's performance beyond increasing training time.

Network depth

In Figure S1B we compare the predictive performance for neural networks with multiple hidden layers of different sizes. The results suggest that shallow neural networks consisting of a single hidden layer are as effective as deeper ones for our purposes, while being easier to set up and train. Our experiments with the other systems (not shown) corroborated this observation.

Number of components

Another important hyperparameter is the number of negative binomial components in the output mixture. In Figure S1C we see that at least 4 components are needed to obtain good approximations for the autoregulatory feedback loop. This is not unexpected as the protein distributions of the autoregulatory feedback loop can be distinctly bimodal in certain parameter regimes, as shown in Figure 3. The maximum number of modes in models of gene networks (that we focus in this study) is usually given by the number of gene states times the total number of gene copies (Thomas et al., 2014), and this acts as an effective lower bound for choosing the number of mixture components. However, increasing the number of mixture components beyond this lower bound may be required to improve the accuracy of the neural network further, as is the case for the autoregulatory feedback loop.

Although adding extra components does not lead to overfitting, as observed by Öcal et al. (2022) it increases the training time and can also make the network more prone to mistakenly predicting too many modes in the solution for certain parameter regimes, which can be regarded as an unphysical artifact. A viable option to find the optimal number of mixture components would be to initialize the neural network with a relatively high number of components and implement L1 regularization (Goodfellow et al., 2017) in order to sparsify the output mixture during training.

Dataset size

Having enough training samples is essential to train a neural network in a way that allows it to learn to generalize. In Figure S1D we show how increasing the size of the training dataset improves the performance of our network. Although relatively small datasets are sufficient for achieving good accuracy, in this case we have only five neural network inputs. Therefore, due to the curse of dimensionality, significantly larger datasets may be needed for effective training on chemical systems involving more reaction parameters. Adding more training points when the validation loss is significantly higher than the training loss is an effective way to determine the appropriate size.

Number of simulations

The number of simulations at each training point also affects the accuracy of the fit. In general we suggest considering more SSA samples if the neural network does not provide a good fit on the training data. As fitting negative binomial mixtures to samples has a strong regularizing effect, the number of simulations per training point is generally much less than what is required to get an accurate histogram from samples (see Figure 4).