Learning complex stochastic models with invertible neural networks: a likelihood-free Bayesian approach

Stefan T. Radev[1], **Ulf K. Mertens**[1], **Andreas Voss**[1], **Lynton Ardizzone**[2], **and Ullrich Köthe**[2]

[1]Institute of Psychology, Heidelberg University, Hauptstr. 47-51, 69117 Heidelberg, Germany; [2]Heidelberg Collaboratory for Image Processing (HCI), Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany

This manuscript was compiled on July 16, 2019

Mathematical models of complex processes are ubiquitous throughout the sciences. As the processes under study and the models describing them become increasingly complex, parameter estimation with standard Bayesian and frequentist methods can quickly become intractable. To address this issue, we propose a novel method for likelihood-free inference based on invertible neural networks. The method is capable of performing full Bayesian inference on large datasets by training the networks on simulated data to learn a probabilistic mapping between parameters and data. The method is independent of any particular data representation, as it incorporates a summary network trained to embed the observed data into fixed-size vectors in a data-driven way. This makes the method applicable to various scenarios where standard inference techniques fail. We demonstrate the utility of the method on a toy model with known analytic posterior and on example models from population dynamics, epidemiology, cognitive science and genetics. We argue that our method provides a general framework for building reusable parameter estimation machines for any process model from which data can be simulated.

Deep learning | Invertible networks | Bayesian inference | Parameter estimation | Stochastic models

Mathematical models are formal descriptions of scientific theories. In its most abstract form, a mathematical model is specified by a set of parameters $\boldsymbol{\theta}$ and a forward model q mimicking the process by which manifest data $\boldsymbol{x}$ arise from latent parameters:

$$\boldsymbol{x} = q(\boldsymbol{\theta}) \quad [1]$$

The functional form of q, which can represent an arbitrarily complex process by an arbitrarily complicated expression, is usually guided by a well-founded theoretical framework. For instance, q can be a stochastic differential equation describing the dynamics of single neurons in the brain, or a step-by-step biological algorithm controlling the rate of gene expression in certain cells. Thus, it is only within the context of a theory that a meaningful interpretation in terms of some mechanism can be attached to the parameters of a mathematical model. Examples of mathematical models can be found in various scientific domains, for instance, genetics (1, 2), cognitive science (3, 4), neuroscience (5, 6), population dynamics (7, 8), epidemiology (9, 10), to name just a few.

An important goal of mathematical modeling is recovering parameters of the model from observed data. Idealized parameter estimation involves computing the inverse (backward) model $\boldsymbol{\theta} = q^{-1}(\boldsymbol{x})$ exactly. Unfortunately, this inverse rarely exists, and estimating parameters remains computationally inconvenient in a wide range of settings.

Parameter estimation becomes notably difficult when the forward model in Eq.1 does not provide a closed-form for the *likelihood function* (3, 11, 12). This poses great difficulties for Bayesian and frequentist methods alike, since both depend on the numerical evaluation of a likelihood function as a proxy for assessing model fit to data. Even if a likelihood function is available in closed-form, inference may be prohibitively slow for real-world applications. In this case, enforcing simplifying distributional assumptions (i.e., independence or Gaussian assumptions) can increase computational speed, but can also lead to model misspecifications and dramatically incorrect estimates. Therefore, there is a need for accurate and computationally convenient likelihood-free estimation methods.

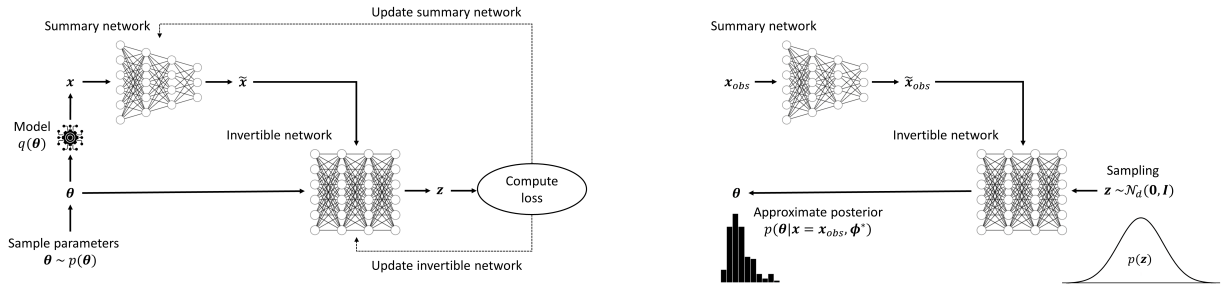
Significance Statement

Describing complex stochastic processes with parametric models lies at the heart of science. Simulating models given a set of parameters is relatively easy with the aid of modern computers, but inferring model parameters from observed data can often be a challenging endeavor. We combine recent advances in deep learning and Bayesian inference into a powerful method for building reusable parameter estimation networks applicable to various types of models and data encountered in different research fields.

Please provide details of author contributions here.

This research was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation; grant number GRK 2277 "Statistical Modeling in Psychology")

[2]To whom correspondence should be addressed. E-mail: stefan.radev@psychologie.uni-heidelberg.de



(a) Training phase

(b) Inference phase

Fig. 1. Graphical illustration of the method. **(a)** During the training phase, the summary and the invertible network are trained on simulated data from the model and updated after each batch of simulations; **(b)** During the inference phase, the true posterior of the model parameters is approximated from real data using the trained networks. Thus, knowledge about the relationship between parameters and data (the mathematical model) is compactly encoded within the weights of the two networks. The trained networks can then be shared and used across researchers working on the same model.

Likelihood-free methods bypass the above problems by resorting to a simulation-based approach to parameter estimation and model selection (3, 13). A subset of likelihood-free methods includes approximate Bayesian computation (ABC) methods, which aim at preserving the advantages of Bayesian data analysis even when the likelihood function is intractable or practically impossible to compute (11, 13, 14). ABC methods approximate the likelihood function by repeatedly sampling parameters from a pre-specified prior distribution $p(\boldsymbol{\theta})$ and then simulating multiple datasets by running the forward model $q(\boldsymbol{\theta})$ using the sampled parameters. Correspondingly, the core ingredients of ABC methods are a prior on $\boldsymbol{\theta}$, and a generative model $q(\boldsymbol{\theta})$, usually specified as a function code in a general-purpose programming language (11, 15).

Performing approximate inference comes at the cost of incurring additional approximation error, which accumulates on top of the irreducible estimation error. Within the context of approximate inference, the most common manifestations of approximation error include: *i)* imprecise form of the posterior; *ii)* imprecise posterior moments; *iii)* under- or overestimation of uncertainty. Different approximation methods usually involve multiple trade-offs between minimizing approximation error and keeping computational time within reasonable bounds (3, 16).

To address the shortcomings of traditional methods, ideas from machine learning and deep learning research have recently entered the field of likelihood-free inference (5, 6, 17–22). The most common approach has been to cast the problem of parameter estimation as a supervised learning task. In this setting, a large dataset of the form $\boldsymbol{D} = \{h(\boldsymbol{x}^{(i)}), \boldsymbol{\theta}^{(i)}\}_{i=1}^{n}$ is created by repeatedly sampling from $p(\boldsymbol{\theta})$ and simulating an artificial dataset $\boldsymbol{x}$ by running $q(\boldsymbol{\theta})$ with the sampled parameters. Usually, the dimensionality of the simulated data is reduced by computing summary statistics with a fixed summary function $h(\boldsymbol{x})$. Then, a supervised learning algorithm $f(h(\boldsymbol{x}); \boldsymbol{\phi}) = \widehat{\boldsymbol{\theta}}$ with learnable parameters $\boldsymbol{\phi}$ (e.g., linear regression, random forest, neural network) is trained on the simulated data to output an estimate of the true data generating parameters. Thus, $f(h(\boldsymbol{x}); \boldsymbol{\phi})$ essentially attempts to "learn" the intractable inverse model $\boldsymbol{\theta} = q^{-1}(\boldsymbol{x})$.

Inspired by previous machine learning approaches (5, 6, 19–22), the current work proposes a novel and universal likelihood-free method capable of performing full Bayesian inference on any mathematical process model from which simulations can be obtained. It treats parameter inference as a task of inverting the forward model in Eq.1 and achieves this by drawing on the modern framework of deep probabilistic modeling for tackling intractable posteriors (23–26). The method integrates two separate deep neural networks modules (detailed in the **Methods** section; see Figure 1) trained jointly on simulated data: a *summary network* and an *invertible network*.

The *summary network* is responsible for learning the most informative summary statistics directly from data. It should be designed to follow the functional and probabilistic symmetries inherent in the data, e.g. an invariant network for *i.i.d.* data (27), a recurrent network (28) or a convolutional network (29) for data with temporal or spatial dependencies. Ideally, it should also be independent of the number of observations. The computation of summary statistics is a crucial aspect in likelihood-free inference. Previous approaches typically used hand-crafted summary statistics tailored to the specific application (14, 20). However, in many applications, finding an optimal set of summary statistics is far from obvious. The summary network eliminates the need to manually decide on and compute a fixed set of summary statistics and, thus, makes the method independent of the format or the size of the data.

The *invertible network* is responsible for learning the posterior of the model parameters given the observed and summarized data. It is based on the recently developed flow-based architecture (24–26). Flow-based methods provide exact latent-variable inference and log-likelihood evaluation when operating at optimum. In the **Methods** section, we show that our method maximizes the posterior over model parameters directly when cast in the context of likelihood-free inference. Furthermore, flow-based methods are capable of approximating high-dimensional distributions (e.g., the pixels of an image). Once trained with a sufficient amount of simulated data, the invertible network can perform rapid Bayesian inference on large datasets from a given research domain.

The joint training of a summary network and an invertible network results in a powerful and universal parameter estimation machine capable of estimating complex mathematical models in various scientific domains (Figure 1). Moreover, the method addresses many of the limitations of previous likelihood-free methods. First, it involves no costly MCMC or rejection sampling, which makes inference lightning fast, once the networks have been trained. Second, it involves no fixed summary statistics, but instead learns the most informative representation of the data in an end-to-end manner. Third, the method is fully Bayesian, as it directly learns the posterior over model parameters and thus allows for the quantification of uncertainty, which is a crucial requirement in parameter estimation (30, 31). Last, the trained networks can be shared and reused by multiple researches within a scientific domain, thus removing the need for wasteful computations and fitting a separate model for each and every dataset. This pooling of computational resources across researchers is an important step forward in mathematical modeling (19).

To illustrate the utility of the new method, we first apply it to a toy Bayesian regression model with known posterior. Then, we present applications to intractable models from cognitive science, population dynamics, epidemiology, and genetics and demonstrate state-of-the art parameter recovery. Across the examples, we introduce multiple tools to validate the performance of our method. The outline of the remaining manuscript is as follows: The **Methods** section introduces the main building blocks of the new method and summarizes the main steps in pseudocode. The **Results** section presents the various applications of the model to real-world research domains. Finally, the **Discussion** section lists the advantages of the current method, treats some potential pitfalls and explores future research vistas. Python code and simulation scripts for all current applications are freely available as Jupyter notebooks at https://github.com/stefanradev93/cINN and as a small library based on *TensorFlow* (32) for creating and training custom invertible networks with GPU support, along with some validation tools.

Methods

Notation. In the following, we denote observed or simulated univariate datasets from the mathematical model of interest as $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, and multivariate datasets as $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_1, ..., \boldsymbol{x}_n)$. The parameters of a mathematical model are represented as a vector $\boldsymbol{\theta} = (\theta_1, \theta_2, ..., \theta_d)$, and all trainable parameters of the invertible and summary neural networks as $\boldsymbol{\phi} = (\boldsymbol{\phi}_{inv}, \boldsymbol{\phi}_{sum})$. The number of parameters of a mathematical model will be denoted as d, and the number of observations in $\boldsymbol{x}$ or $\boldsymbol{X}$ as n.

Deep Probabilistic Modeling. Our method draws on major advances in modern deep probabilistic modeling, also referred to as deep generative modeling (23, 24, 27, 33). A hallmark idea in deep probabilistic modeling is to handle intractable target probability distributions by sampling from simpler distributions (e.g., Gaussian or uniform distributions) and transforming these samples via learned complex non-linear transformations. Most popular deep probabilistic models entail two phases. During the *training phase*, a transformation from the simple to the desired target distribution is learned by optimizing a cost function via backpropagation (see Figure 1a). During the *inference phase*, samples from the target distribution are obtained by sampling from the simple distribution and applying the transformation learned during the training phase (see Figure 1b). Using this approach, recent applications of deep probabilistic models have achieved unprecedented results on extremely high-dimensional and intractable problems (e.g., complex data distributions such as images, music, or text) (24, 25, 27).

In the context of mathematical modeling and Bayesian inference, the target distribution is the posterior distribution of model parameters $p(\boldsymbol{\theta}|\boldsymbol{x})$ capturing the uncertainty about the numerical values of parameters. We can leverage the fact that most mathematical models are generative in nature and as such can be used to perform multiple simulations of the process of interest. By specifying a prior distribution over the model parameters $p(\boldsymbol{\theta})$, one can generate arbitrarily large datasets of the form $\boldsymbol{D} = \{\boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(i)}\}_{i=1}^{n}$ and use a deep generative model to learn a probabilistic mapping from data to parameters. Thus, during the inference phase, the model generates samples $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, ..., \boldsymbol{\theta}^{(L)}$ from the observed data $\boldsymbol{x}_{obs}$, approximating the target posterior $p(\boldsymbol{\theta}|\boldsymbol{x} = \boldsymbol{x}_{obs})$.

In the current work, we propose to implement and use a conditional invertible neural network (cINN) architecture. Previously, INNs have been successfully employed to model data from astrophysics and medicine(23). We adapt the model to suit the task of parameter estimation in the context of mathematical modeling (see Figure 1 for a full graphical illustration of the method) and develop a reusable probabilistic architecture for full Bayesian likelihood-free inference on complex mathematical models.

The Affine Coupling Block. The basic building block of a cINN is the affine coupling block (ACB, see Figure 2a) (23, 24, 26). Each cACB consists of four separate fully connected neural networks denoted as $s_1(\cdot), s_2(\cdot), t_1(\cdot), t_2(\cdot)$. An ACB is specifically designed to be invertible, which means that in addition to a parametric mapping $f_{\boldsymbol{\phi}_{inv}} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ it also learns the inverse mapping $f^{-1}_{\boldsymbol{\phi}_{inv}} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ "for free". Denoting the input vector of $f_{\boldsymbol{\phi}_{inv}}$ as $\boldsymbol{u}$ and the output vector as $\boldsymbol{v}$, it follows that $f(\boldsymbol{u}; \boldsymbol{\phi}_{inv}) = \boldsymbol{v}$ and $f^{-1}(\boldsymbol{v}; \boldsymbol{\phi}_{inv}) = \boldsymbol{u}$. Invertibility is achieved by splitting the input vector into two parts $\boldsymbol{u} = (\boldsymbol{u}_1, \boldsymbol{u}_2)$ and performing the following operations on the split input:

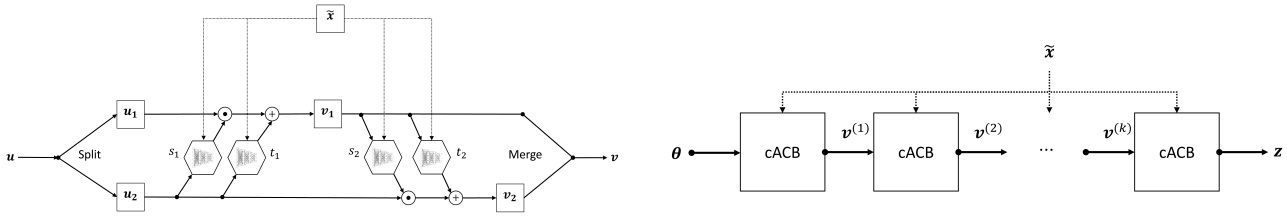
$$\boldsymbol{v}_1 = \boldsymbol{u}_1 \odot \exp(s_1(\boldsymbol{u}_2)) + t_1(\boldsymbol{u}_2) \quad [2]$$

$$\boldsymbol{v}_2 = \boldsymbol{u}_2 \odot \exp(s_1(\boldsymbol{v}_1)) + t_1(\boldsymbol{v}_1) \quad [3]$$

The outputs $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2)$ are then concatenated again and passed to the next ACB. The inverse operation is given by:

$$\boldsymbol{u}_2 = (\boldsymbol{v}_2 - t_2(\boldsymbol{v}_1)) \odot \exp(-s_2(\boldsymbol{v}_1)) \quad [4]$$

$$\boldsymbol{u}_1 = (\boldsymbol{v}_1 - t_1(\boldsymbol{u}_2)) \odot \exp(-s_1(\boldsymbol{u}_2)) \quad [5]$$



(a) The cACB **(b)** A chain of cACBs

Fig. 2. A diagram of the conditional version of the affine coupling block (cACB). **(a)** Each cABC consists of four internal networks performing the invertible operations described in the text; **(b)** In practice, we chain multiple cACBs to obtain higher representational capacity. Each cACB layer uses a fixed permutation to ensure that information about each parameter is encoded in each latent dimension of $\boldsymbol{z}$.

An additional property of this design, which becomes relevant later for optimization, is that the operations of the ACB have tractable, and cheaply computable Jacobians (strictly upper or lower triangular matrices). Furthermore, the internal networks $s_1(\cdot), s_2(\cdot), t_1(\cdot), t_2(\cdot)$ can be represented by arbitrarily complex neural networks, which themselves need not be invertible, since they are only ever evaluated in the forward direction during both the forward and the inverse pass through the ACB. To ensure that the model is powerful enough to represent complicated distributions, we chain multiple ACBs, so that the output of each ACB becomes the input of the next (see Figure 2b). In this way, the whole chain remains invertible from the first input to the last output and can be viewed as a single function parameterized by trainable parameters $\boldsymbol{\phi}_{inv}$.

In our applications, the input to the first ACB is the parameter vector $\boldsymbol{\theta}$, and the output of the final ACB, denoted hitherto as $\boldsymbol{z}$, is encouraged to follow a d-dimensional spherical Gaussian via optimization (described in detail later), that is, $p(\boldsymbol{z}) = \mathcal{N}_d(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I})$. Fixed permutation matrices are used before each ACB to ensure that each axis of the latent space encodes information from all components of $\boldsymbol{\theta}$. In order to take into account the observed or simulated data $\boldsymbol{x}$, each of the internal networks of each ACB is augmented to take $\boldsymbol{x}$ as an additional input - $s_1(\cdot, \boldsymbol{x}), s_2(\cdot, \boldsymbol{x}), t_1(\cdot, \boldsymbol{x}), t_2(\cdot, \boldsymbol{x})$ - so a complete pass through the entire invertible chain can be expressed as:

$$f(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{\phi}_{inv}) = \boldsymbol{z} \quad [6]$$

together with the inverse operation:

$$f^{-1}(\boldsymbol{z}; \boldsymbol{x}, \boldsymbol{\phi}_{inv}) = \boldsymbol{\theta} \quad [7]$$

This process can be interpreted as follows: the forward pass maps data-generating parameters to $\boldsymbol{z}$-space using conditional information of $\boldsymbol{x}$, while the inverse pass maps data points from $\boldsymbol{z}$-space to the data-generating parameters of interest using the same conditional information provided by the data. In the next section, we describe the optimization procedure used to match the outputs of $f^{-1}(\boldsymbol{z}; \boldsymbol{x}, \boldsymbol{\phi}_{inv})$ to the posterior $p(\boldsymbol{\theta}|\boldsymbol{x})$.

Summary Network. Since in practice the conditioning data set $\boldsymbol{x}$ can have variable number of input points (e.g., trial sizes, time points) and exhibit various redundancies, the cINN can profit from some form of dimensionality reduction applied to the data. Ideally, we want to avoid hand-crafted summary statistics, and instead learn the most informative summary statistics directly from data. Therefore, instead of feeding the raw simulated or observed data to each ACB, we pass the data through an additional summary network to obtain a fixed-sized vector of learned summary statistics $\tilde{\boldsymbol{x}} = h(\boldsymbol{x}; \boldsymbol{\phi}_{sum})$ and learn the parameters of the summary network h jointly with those of the cINN chain via backpropagation. Thus, the current method remains completely end-to-end and is capable of generalizing to data sets of variable input size and structure.

Learning the Posterior. The cINN learns to approximate the posterior of model parameters by optimizing a maximum likelihood (ML) criterion. Broadly speaking, the goal of ML estimation is to find a set of parameters which maximize the probability of the data under a parametric model. In our case, we are interested in maximizing the expectation over all possible neural network parameters with respect to the parameters of the mathematical model:

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\, \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\boldsymbol{x})} \left[p(\boldsymbol{\phi}|\boldsymbol{\theta}, \boldsymbol{x}) \right] \quad [8]$$

Applying Bayes' rule to the posterior over all neural network parameters we obtain:

$$p(\boldsymbol{\phi}|\boldsymbol{\theta}, \boldsymbol{x}) \propto p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{\phi}) p(\boldsymbol{\phi}) \quad [9]$$

Note, that by maximizing Eq.9 we are maximizing the posterior over model parameters of interest $p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{\phi})$. Thus, it remains to find a tractable expression for Eq.8 to be minimized by backpropagation given a finite number of simulated samples from the model. To this end, we recall that we can relate the pdf of $\boldsymbol{\theta}$ to that of $\boldsymbol{z}$ via the change of variable theorem:

$$p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{\phi}) = p(\boldsymbol{z}) \left| \det \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}} \right) \right| \quad [10]$$

$$= p\left(f(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{\phi}) \right) \left| \det \left(\frac{\partial f}{\partial \boldsymbol{\theta}} \right) \right| \quad [11]$$

where $\partial f/\partial \boldsymbol{\theta} = \boldsymbol{J}_f$ is the Jacobian of the learned transformation $f(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{\phi})$ with respect to the input. Both terms in Eq.11 are now tractable, since we have previously defined $\boldsymbol{z}$ as following a spherical unit Gaussian, that is, $p(\boldsymbol{z}) = (2\pi)^{-2/d} \exp(\|\boldsymbol{z}\|_2^2)$ and the determinant of the Jacobian is easily computed as $s_1(\boldsymbol{u}_2, \boldsymbol{x}) + s_2(\boldsymbol{v}_1, \boldsymbol{x})$ due to eqs. 2 and 3. We can now formulate the ML loss as the Monte-Carlo approximation of the negative logarithm of Eq.8 for a batch of size m:

$$\mathcal{L}(\boldsymbol{\phi}) = -\frac{1}{m}\sum_{i=1}^{m} \log\left(p(\boldsymbol{\phi}|\boldsymbol{\theta}^{(i)}, \boldsymbol{x}^{(i)})\right) \quad [12]$$

$$= -\frac{1}{m}\sum_{i=1}^{m} \log\left(p(\boldsymbol{\theta}^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\phi})p(\boldsymbol{\phi})\right) \quad [13]$$

$$= -\frac{1}{m}\sum_{i=1}^{m} \left(\frac{\left\|f(\boldsymbol{\theta}^{(i)}; \boldsymbol{x}^{(i)}, \boldsymbol{\phi})\right\|_2^2}{2} + \log\left|\det\left(\boldsymbol{J}_f^{(i)}\right)\right|\right) + \tau\|\boldsymbol{\phi}\|_2^2 \quad [14]$$

where we place a Gaussian prior over the neural network parameters with $\tau \equiv 1/\sigma^2$, corresponding to a standard $L2$-regularization.

Minimizing Eq.14 can be interpreted as searching for the optimal neural network parameters $\boldsymbol{\phi}^*$ which maximize the probability of model parameters $\boldsymbol{\theta}$ given data $\boldsymbol{x}$. This is exactly the probability we are concerned with in Bayesian inference. Note that our formulation maximizes the posterior of model parameters directly, in contrast to variational methods which optimize a lower bound on the posterior (22, 33). Once the backpropagation algorithm has settled to a local minimum of the ML loss, one can easily obtain samples from the approximate posterior $p(\boldsymbol{\theta}|\boldsymbol{x} = \boldsymbol{x}_{obs}, \boldsymbol{\phi} = \boldsymbol{\phi}^*)$, based on an observed dataset $\boldsymbol{x}_{obs}$, by repeatedly sampling $\boldsymbol{z}^{(l)} \sim \mathcal{N}_d(\boldsymbol{0}, \boldsymbol{I})$, and then passing $\boldsymbol{z}^{(l)}$ in reverse to the cINN in order to compute $\boldsymbol{\theta}^{(l)} = f^{-1}(\boldsymbol{z}^{(l)}; \boldsymbol{x}_{obs}, \boldsymbol{\phi}^*)$ for $l = 1, ..., L$. Figure 1b illustrates the inference phase of the method. It is worth noting that sampling a large number of parameter values from the approximate posterior takes a negligible amount of time, since it only requires a single pass through the cINN in reverse.

Training the Networks. We train all cINNs and summary networks described in this paper jointly via backpropagation. For all following experiments, we use the Adam optimizer with a starter learning rate of 10^{-3} and an exponential decay rate of .95. We set the weight regularization parameter τ to a value of 10^{-5}. With these settings, we perform 50 000 to 100 000 iterations (network updates) for each of the examples in this paper, and report the results on the trained network. Note, that we did not perform an extensive search for optimal values of the cINN hyperparameter, but use a cINN with 10 ACBs all examples in this paper (see **SI** for details of the cINNs and summary networks). All networks were implemented in Python using the *TensorFlow* library (32) and trained on a single-GPU machine equipped with NVIDIA® GTX1060 graphics card. Regarding the data generation step, we use two training approaches.

The first approach follows the classical approximate Bayesian computation approach to create a large *reference table* or grid of the form $\boldsymbol{D} = \{\boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(i)}\}_{i=1}^{n}$. The reference table is then used as training data for the neural network and training continues for a pre-specified number of epochs through the entire reference table. A separate validation dataset is eventually used to assess the performance of the network. Training on large pre-simulated datasets separates the simulation from the training phase but can cause large memory overhead, as the reference table must be stored on disk and then loaded in chunks or in its entirety into memory.

The second approach follows a different strategy, which resembles ideas from *active learning* (34). Correspondingly, a dataset, or a batch of datasets, is created on the fly and then passed through the neural network. This training regime has the advantage that the network never *experiences* the same input data twice. Moreover, training can continue as long as the network keeps improving (i.e., the loss keeps decreasing), since overfitting in the classical sense is nearly impossible. However, if simulations are computationally expensive and researchers need to experiment with different networks or training hyperparameters, it might be beneficial to switch to the first regime, since simulation and training in the active learning regime are tightly intertwined.

Putting It All Together. On an abstract level, our method requires three key ingredients: 1) a mathematical process model $q(\boldsymbol{\theta})$ capable of simulating data $\boldsymbol{x}$; 2) a prior distribution over the model parameters $p(\boldsymbol{\theta})$ encoding our prior beliefs about plausible parameter values; 3) an invertible neural network $f_{\boldsymbol{\phi}}$ capable of approximating a large enough family of probability distributions (see Figure 2). In practice, a chain of up to 10 ACBs should suffice to learn most distributions encountered in the cognitive or life sciences, since they tend to be unimodal and relatively simple (in contrast to the distributions required to represent images or words in a spoken language). From these three ingredients, a universal and reusable sampler can be designed for likelihood-free Bayesian estimation of both tractable and intractable mathematical models. **Algorithm** 1 describes the essential steps of the method using an arbitrary summary network and employing the active learning training regime.

The backpropagation algorithm works by computing the gradients of the loss function with respect to the parameters of the neural networks and then adjusting the parameters, so as to drive the loss function to a local minimum. We experienced no instability or convergence issues during training with the ML loss. Note, that steps $12 - 15$ of **Algorithm** 1 can be executed in parallel with GPU support.

In what follows, we apply the method to a toy Bayesian regression example with conjugate priors, and then use it to estimate the parameters of challenging models from population dynamics, cognitive science, epidemiology, and genetics.

Algorithm 1 Bayesian likelihood-free inference with invertible neural networks

Training (via active learning):
repeat
  Sample a batch of $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{m}$ from prior $p(\boldsymbol{\theta})$
  Simulate a batch of datasets $\{\boldsymbol{x}^{(i)}\}_{i=1}^{m}$ by running $\boldsymbol{x}^{(i)} = q(\boldsymbol{\theta}^{(i)})$ for $i = 1, ..., m$
  Pass $\{\boldsymbol{x}^{(i)}\}_{i=1}^{m}$ through summary network $h(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}_{sum})$ to obtain $\{\tilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{m}$
  Pass $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{m}$ and $\{\tilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{m}$ through cINN $f(\boldsymbol{\theta}^{(i)}; \boldsymbol{x}^{(i)}, \boldsymbol{\phi}_{inv})$ to obtain $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$
  Compute ML loss $\mathcal{L}(\boldsymbol{\phi})$ according to Eq.14
  Update neural network parameters $\boldsymbol{\phi}$ via backpropagation
until convergence to $\boldsymbol{\phi}^*$
Inference (given observed or test data $\boldsymbol{x}_{obs}$):
Summarize the observed data by computing $\tilde{\boldsymbol{x}}_{obs} = h(\boldsymbol{x}_{obs}, \boldsymbol{\phi}^*_{sum})$
for $l = 1, ..., L$ **do**
  Sample $\boldsymbol{z}^{(l)} \sim \mathcal{N}_d(\boldsymbol{0}, \boldsymbol{I})$
  Compute $\boldsymbol{\theta}^{(l)} = f^{-1}(\boldsymbol{z}^{(l)}; \tilde{\boldsymbol{x}}_{obs}, \boldsymbol{\phi}_{inv})$
end for
Use $\{\boldsymbol{\theta}^{(l)}\}_{l=1}^{L}$ to approximate the posterior $p(\boldsymbol{\theta}|\boldsymbol{x}_{obs})$

Results

To assess the performance of our method on the following application examples, we consider a number of different metrics. To assess the precision of point estimates (posterior means), we compute the normalized root mean squared error (NRMSE) and the coefficient of determination (R^2) between estimated and true parameter values. To assess the recovery of the full posterior, we compute the Kullback-Leibler (KL) divergence (35) between the true and the approximate distributions for the toy example, and use simulation-based calibration (SBC, (36)) for the other examples where the analytic posterior is not available in closed-form. Details for computing the NRMSE, R^2, and SCB can be found in the **SI**. Code for reproducing the results on all following examples, as well as for all network implementations, is freely available at: https://github.com/stefanradev93/cINN.

Toy Example – Bayesian Regression. As a proof-of-concept, we demonstrate the utility of our method in recovering the true analytic posteriors of the regression coefficients of a conjugate Bayesian regression model. To set the stage, assume we have observed a dataset $\boldsymbol{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ with $\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$. We stack all $\boldsymbol{x}^{(i)}$ row-wise in a design matrix $\boldsymbol{X}$ and model the vector of outcomes $\boldsymbol{y}$ as being conditionally Gaussian given $\boldsymbol{X}$, that is, $\boldsymbol{y} \sim \mathcal{N}_n(\boldsymbol{X\theta}, a^{-1}\boldsymbol{I})$ where $\boldsymbol{\theta} \in \mathbb{R}^d$ is the vector of regression coefficients and a is the precision (inverse noise variance, $a \equiv 1/\sigma_y^2$). We place a d-dimensional diagonal Gaussian prior on the regression coefficients centered at 0: $\boldsymbol{\theta} \sim \mathcal{N}_d(\boldsymbol{0}, b^{-1}\boldsymbol{I})$ where b is the precision of the prior ($b \equiv 1/\sigma_{\boldsymbol{\theta}}^2$). Thus, the likelihood $p(\boldsymbol{D}|\boldsymbol{\theta})$ admits the following proportionality:

$$p(\boldsymbol{D}|\boldsymbol{\theta}) \propto \exp\left(-\frac{a}{2}(\boldsymbol{y} - \boldsymbol{X\theta})^T(\boldsymbol{y} - \boldsymbol{X\theta})\right) \quad [15]$$

Since the prior of $\boldsymbol{\theta}$ is conjugate to the likelihood (both are Gaussian distributions), the posterior of $\boldsymbol{\theta}$ is also Gaussian, given by:

$$p(\boldsymbol{\theta}|\boldsymbol{D}) \propto \exp\left(-\frac{a}{2}(\boldsymbol{y} - \boldsymbol{X\theta})^T(\boldsymbol{y} - \boldsymbol{X\theta}) - \frac{b}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}\right) \quad [16]$$

Therefore, the posterior has the form $p(\boldsymbol{\theta}|\boldsymbol{D}) = \mathcal{N}_d(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$ where $\boldsymbol{\Lambda}$ denotes the posterior precision matrix (inverse covariance matrix), and $\boldsymbol{\mu}$ denotes the posterior mean vector. The posterior moments are computed as follows:

$$\boldsymbol{\Lambda} = a\boldsymbol{X}^T\boldsymbol{X} + b\boldsymbol{I} \quad [17]$$

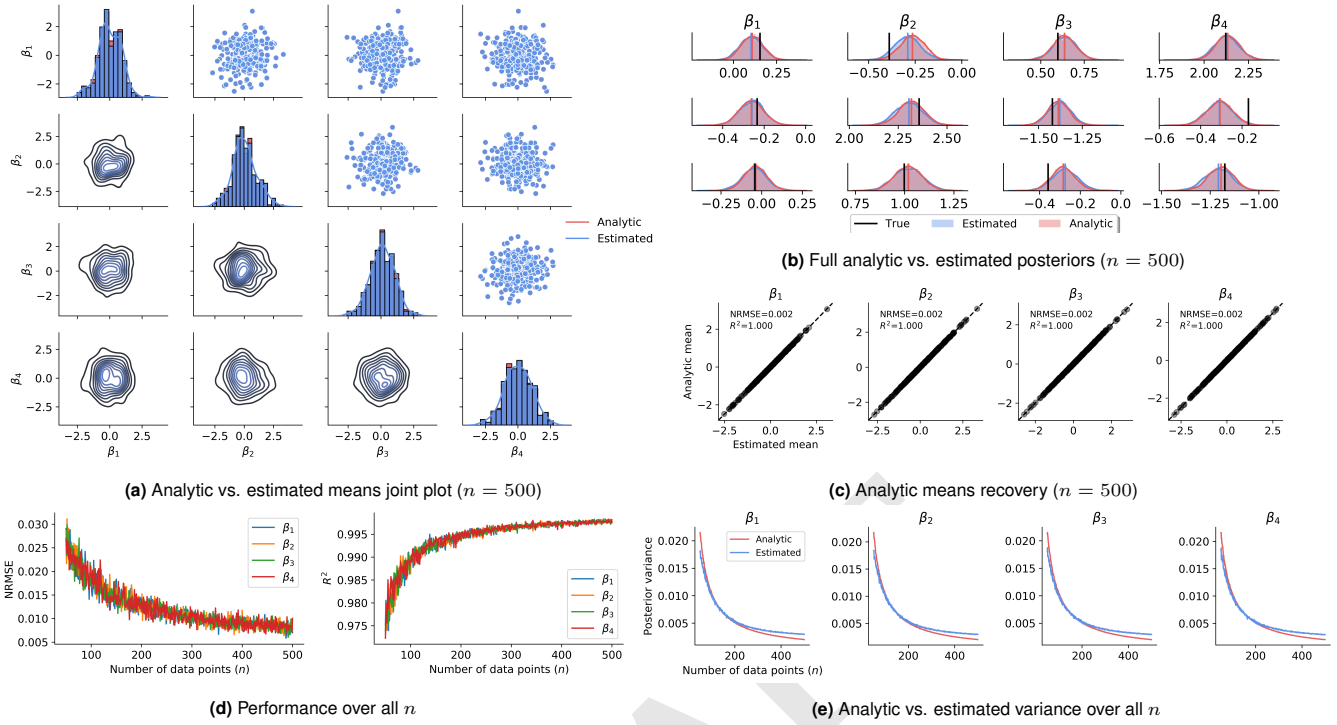
$$\boldsymbol{\mu} = a\boldsymbol{\Lambda}^{-1}\boldsymbol{X}^T\boldsymbol{y} \quad [18]$$

Thus, for known a and b, the posterior of $\boldsymbol{\theta}$ can be easily computed. Even though in real-world applications a is usually not known and a hierarchical model is used instead, the current example is well suited for testing the utility of our method.

For the following application, we set $d = 4$, and $a = b = 1$. The design matrices for each iteration contain a variable number n of *i.i.d.* data points drawn from a unit Gaussian $\boldsymbol{x}^{(i)} \sim \mathcal{N}_4(\boldsymbol{0}, \boldsymbol{I})$ for $i = 1, ..., n$. The number of trials is drawn from a uniform distribution $n \sim \mathcal{U}(50, 500)$ at each training iteration (Lines 2-9 of **Algorithm** 1). Training the networks took approx. half a day with the active learning approach. Inference on 1000 datasets with 2000 posterior samples per parameter took approx. 5.5 seconds.

The results on the toy Bayesian regression are depicted in Figure 3. The approximate posterior means show negligible deviations from the analytic posterior means as quantified by very small NRMSEs (as small as 0.002) and very high R^2 (as high as 1.0) over all n sampled during training. This suggests near-perfect estimation of the true posterior means. Further, the estimates become increasingly more accurate, as the number of data points n increases. An inspection of the posterior variances over the different n reveals that the estimated variance follows closely the decrease in analytic variance with increasing n.

However, the analytic variance is slightly underestimated at smaller n and slightly overestimated at larger n. This pattern is also revealed by the KL divergence plot (see **SI**). This result might be attributable to an underexpressive summary network; another possibility is that the networks need to be trained longer with smaller learning rate decay.



(a) Analytic vs. estimated means joint plot ($n = 500$)

(b) Full analytic vs. estimated posteriors ($n = 500$)

(c) Analytic means recovery ($n = 500$)

(d) Performance over all n

(e) Analytic vs. estimated variance over all n

Fig. 3. Results on the Bayesian toy regression example. **(a)** Pair-plots of the analytic vs. estimated posterior means on the whole validation sample. We observe a near-perfect overlap between analytic and estimated posterior means and no spurious covariances; **(b)** Some example draws from the estimated posteriors. Visual inspections reveals a close match between analytic and estimated posteriors; **(c)** Posterior mean recovery is almost perfect at $n = 500$; **(d)** NRMSE and R^2 over all n. Performance improves with increasing number of observed data points; **(e)** Analytic vs. estimated variance over all n. The analytic variance is slightly underestimated at lower n and slightly overestimated at higher n.

Example 1 - The Ricker Model. Discrete population dynamics models describe how the number of individuals in a population changes over discrete units of time (7). In particular, the Ricker model describes the number of individuals x_{t+1} in generation $t + 1$ as a function of the number of individuals in the previous generation t by the following non-linear equation:

$$x_t \sim Pois(\rho N_t) \quad [19]$$

$$N_{t+1} = rN_t e^{-N_t + \epsilon_t} \quad [20]$$

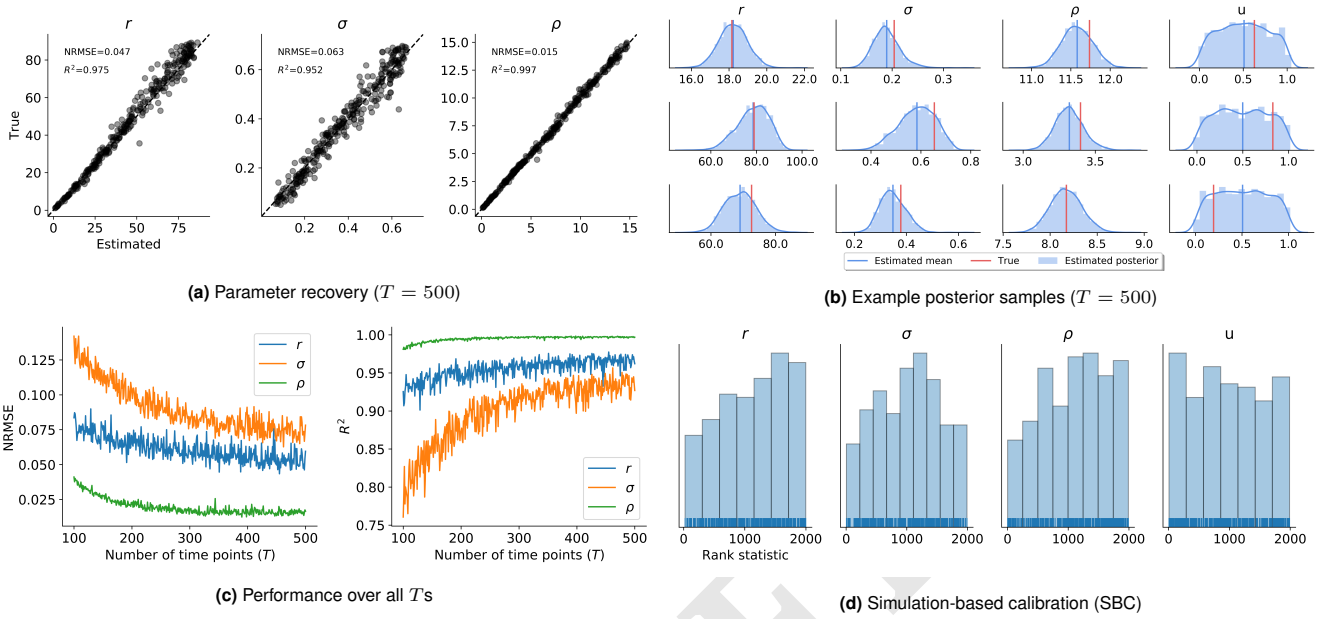
for $t = 1, ..., T$ where N_t is the expected number of individuals at time t, r is the growth rate, ρ is a scaling parameter and $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ is random Gaussian noise (19). The likelihood function for the Ricker model is not available in closed-form, and the model is known to exhibit chaotic behavior. Thus, it is a suitable candidate for likelihood-free inference. The parameter estimation task is thus to recover $\boldsymbol{\theta} = (\rho, r, \sigma)$ from the observed one-dimensional time-series data $\boldsymbol{x} = (x_1, x_2, ..., x_T)$ where each $x_t \in \mathbb{N}$.

During training of the networks, we simulate time-series from the Ricker model with varying lengths. The number of time points T is drawn from a uniform distribution $T \sim \mathcal{U}(100, 500)$ at each training iteration. Training the networks took approx. half a day with the active learning approach. Inference on 1000 datasets with 2000 posterior samples per parameter took approx. 6 seconds.

What if the data does not contain any information about a particular parameter? In this case, any good estimation method should detect this, and return the prior of the particular parameter. To test this, we append a random uniform variable $u \sim \mathcal{U}(0, 1)$ to the parameter vector $\boldsymbol{\theta}$ and train the model with this additional dummy parameter. We expect that the networks learn to ignore this dummy parameter, that is, the estimated posterior of u is (approximately) equal to the uniform prior.

The results on the Ricker model are depicted in Figure 4. As evident from the graphs, parameter recovery becomes better when more time points with data are available (Figure 4c). At $T = 500$, the NRMSEs range between 0.015 and 0.063, and the R^2 metrics between 0.997 and 0.952, indicating very good recovery of the posterior means (Figure 4a). The parameter σ seems to be the hardest to recover. Inspecting the full posteriors, we further see that the posterior distribution of the dummy noise variable u closely resembles the prior. This is expected due to the complete lack of mutual information between the observed data x and u (Figure 4b). Finally, the plots of the rank statistic computed for SCB suggest no systematic distortions of the posterior across all parameters (Figure 4d). Interpreting deviations from uniformity according to (36), the

approximate posteriors of r and ρ slightly underestimate the true posterior means, whereas the approximate posterior of σ tends to overestimate the true posterior variance. These deviations appear to be due to the fact that recovery worsens at extreme values of the parameters. This is unsurprising, as the data generated with these parameters is highly implausible, which in some cases might even render a model unidentifiable.



(a) Parameter recovery ($T = 500$)

(b) Example posterior samples ($T = 500$)

(c) Performance over all Ts

(d) Simulation-based calibration (SBC)

Fig. 4. Results on the Ricker model. **(a)** Parameter recovery for the maximum number of generations used during training ($T = 500$); **(b)** Example posteriors for three test datasets. We observe that the posterior of the uniform noise variable u is equal to the prior, i.e., the method detects that no information is present in data for this variable; **(c)** NRMSE and R^2 performance metrics over all Ts used in training. We observe that the recovery remains good over all Ts, and becomes progressively better as more data is available; **(d)** Plots of the rank statistics indicative of the accuracy of the full posterior. Accordingly, the approximate posteriors of r and ρ slightly underestimate the true posterior means, whereas the approximate posterior of σ tends to overestimate the true posterior variance.

Example 2 - The Lévy-Flight Model. Evidence accumulator models (EAMs) describe human decision making by a set of neurocognitively motivated parameters (37). EAMs are most often applied to choice reaction times (RT) data to obtain an estimate of the neurocognitive processes governing observed RT distributions in human (or animal) participants. Most EAM variants share four underlying assumptions: *i)* information about a stimulus (response option) is accumulated continuously through time; *ii)* stochasticity in the form of noisy accumulation ensures variability; *iii)* empirical response times can be decomposed into a decision time component and a non-decision time component accounting for pre-decisional perceptual (encoding time) and post-decisional motor processes (response execution); and *iv)* a decision is met when the activation of an accumulator reaches a threshold. In its most general formulation, the forward model of EAMs takes the form of a stochastic differential equation given by (4):

$$dx = vdt + cd\xi \quad [21]$$

where dx denotes a change in activation of an accumulator, v denotes the average speed of information accumulation (often termed the drift rate), and $d\xi$ represents a stochastic additive component, usually modeled as following a Gaussian distribution centered around 0: $d\xi \sim \mathcal{N}(0, c^2)$.

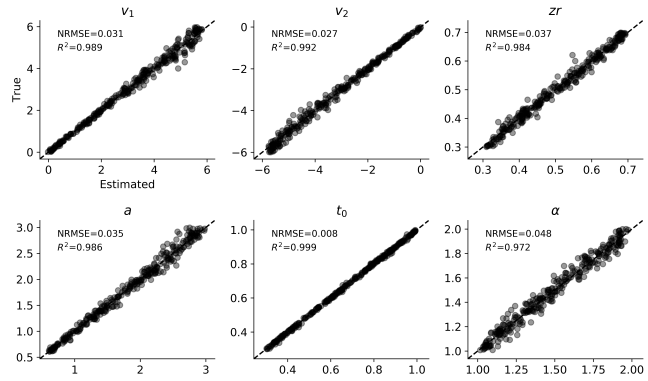
EAMs are particularly amenable for likelihood-free inference, since the likelihood of some members of this model family turn out to be intractable (38). This intractability has precluded many interesting applications and empirically driven model refinements. Here, we apply our method to estimate the parameters of the recently proposed Lévy-Flight Model (LFM, (39)). The LFM assumes an *alpha-stable* noise distribution of the evidence accumulation process in order to model "jumps" in the decision process. However, the inclusion of *alpha-stable* noise leads to a model with intractable likelihood; further, to our knowledge, a fully Bayesian treatment of the model is still missing in the literature. The forward equation of the LFM is given by:

$$dx = vdt + \xi dt^{1/\alpha} \quad [22]$$

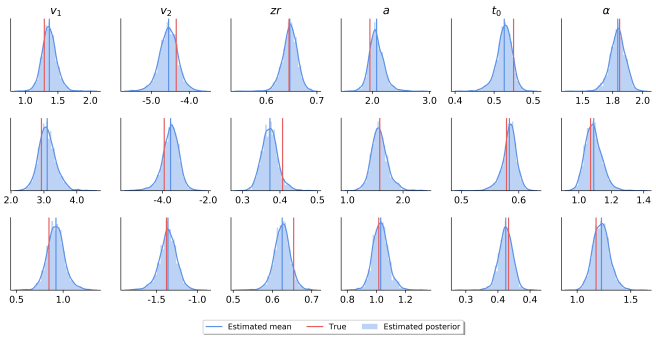
$$\xi \sim AlphaStable(\alpha, 0, 1, 0) \quad [23]$$

The LFM has three additional parameters: the threshold a determining the amount of evidence needed for the termination of a decision process; a relative starting point, zr, determining the amount of starting evidence available to the accumulator before the actual decision alternatives are presented; and an additive non-decision time t_0.

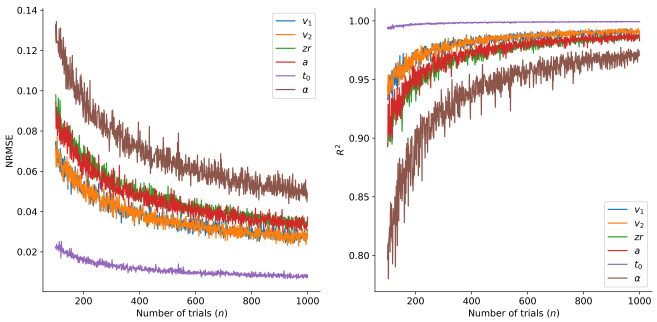
During training of the networks, we simulate response times data from two experimental conditions with two different drift rates, since such a design is often encountered in psychological research. The parameter estimation task is thus to recover the parameters $\boldsymbol{\theta} = (v_0, v_1, a, t_0, zr, \alpha)$ from two-dimensional *i.i.d.* RT data $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ containing variable number or RT trials. The number of trials is drawn from a uniform distribution $n \sim \mathcal{U}(100, 1000)$ at each training iteration. Training the networks took a little less than a day with the active learning approach. Inference on 1000 datasets with 2000 posterior samples per parameter took approx. 7.39 seconds.



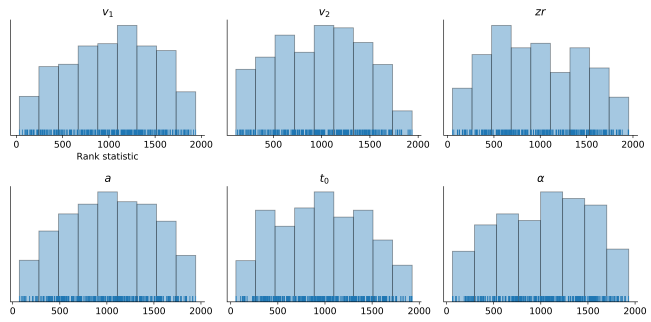
(a) Parameter recovery ($n = 1000$)



(b) Example posterior samples ($n = 1000$)



(c) Performance over all trial numbers



(d) Simulation-based calibration (SBC)

Fig. 5. Results on the LFM model. **(a)** Parameter recovery for the maximum number of trials used during training ($n = 1000$); **(b)** Example posteriors for three test datasets; **(c)** NRMSE and R^2 performance metrics over all n trials used during training. Again, we observe that the recovery remains overall very good, and becomes progressively better as more data is available; **(d)** Plots of the rank statistics indicative of the accuracy of the full posterior. Accordingly, the approximate posteriors tend to overestimate the true posterior variance.

The results on the LFM model are depicted in Figure 5. To our knowledge, this is the first Bayesian treatment of the LFM, as its intractability makes traditional methods prohibitively slow. We observe excellent recovery of all LFM parameters with NRMSEs ranging between 0.008 and 0.048 and R^2 between 0.972 and 0.999. Further, estimation remains very good across all trial sizes, with goodness increasing as more trials become available. The parameter α appears to be most challenging to estimate, requiring more data for good estimation, whereas the non-decision time parameter t_0 is almost perfectly reconstructed for all trial sizes. Last, inspecting the SCB plots, we notice that the histograms tend to be slightly peaked, indicating a slight overestimation of the posterior variance (36).

Example 3 – The Stochastic SIR Model. Compartmental models in epidemiology are used to describe the stochastic dynamics of infectious diseases as they spread over a population of individuals (9, 10, 40). The parameters of compartmental models encode important characteristics of diseases, such as the rates of infection or recovery from the disease. The stochastic SIR model describes the transition dynamics of N individuals between three discrete states: susceptible (S), infected (I), and recovered (R). The transition dynamics are given by the following equations:

$$\triangle S = -\triangle N_{SI} \quad [24]$$

$$\triangle I = \triangle N_{SI} - \triangle N_{IR} \quad [25]$$

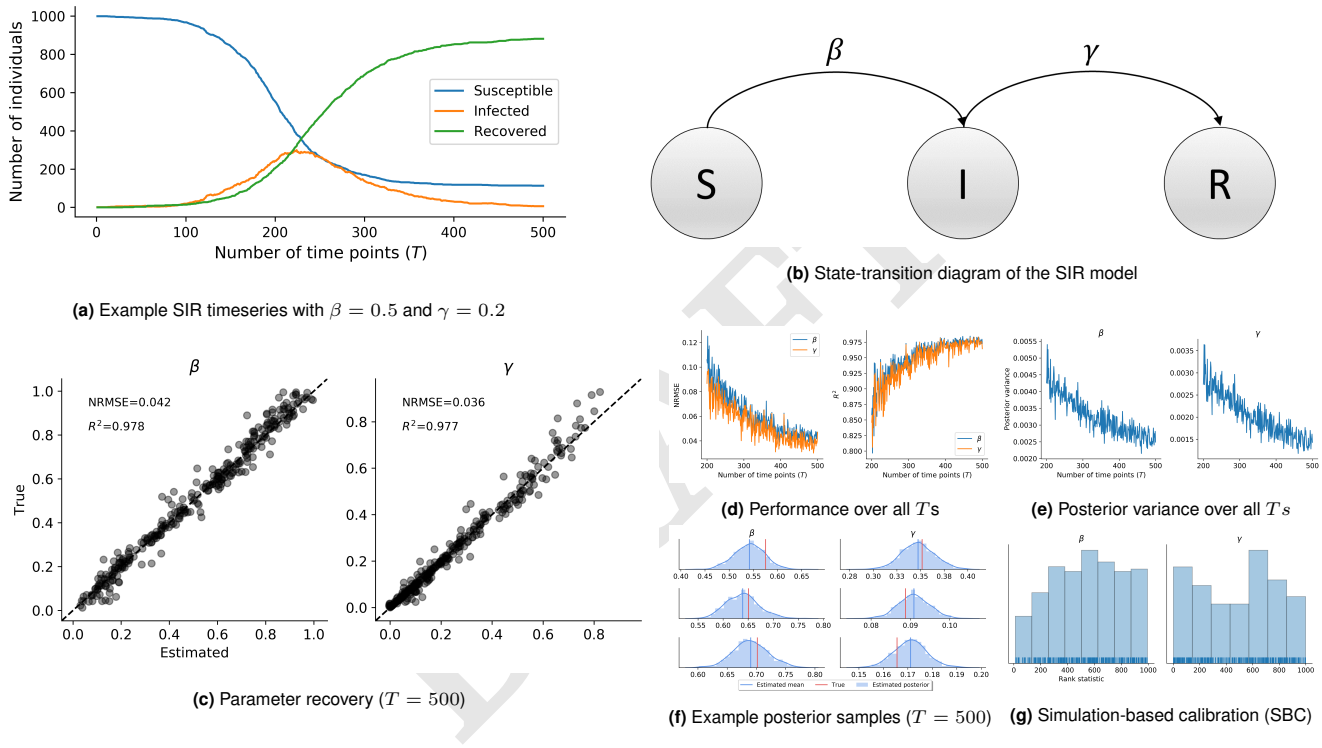
$$\triangle R = \triangle N_{IR} \quad [26]$$

$$\triangle N_{SI} \sim Binomial(S, 1 - \exp\left(-\beta \frac{I}{N} \triangle t\right)) \quad [27]$$

$$\triangle N_{IR} \sim Binomial(I, 1 - \exp\left(-\gamma \triangle t\right)) \quad [28]$$

where $S+I+R=N$ give the number of susceptible, infected, and recovered individuals, respectively. The parameter β controls the transition rate from being susceptible to infected, and γ controls the transition rate from being infected to recovered (see Figure 6). The number of individuals moving from S to I, given by $\triangle N_{SI}$, and the number of people moving from I to R, given by $\triangle N_{IR}$, over a time interval $\triangle t$ are modeled as binomial random variables. The above listed stochastic system has no known analytic solution and thus requires numerical simulation methods for recovering parameter values from data. Cast as a parameter estimation task, the challenge is to recover $\boldsymbol{\theta} = \{\beta, \gamma\}$ from three dimensional time-series data $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T)$ where each $\boldsymbol{x}_t \in \mathbb{N}^3$ is a triple containing the number of susceptible (S), number of infected (I), and recovered (R) individuals at time t.

During training of the networks, we simulate time-series from the stochastic SIR model with varying lengths. The number of time points T is drawn from a uniform distribution $T \sim \mathcal{U}(200, 500)$ at each training iteration. Usually, at lower Ts, the system has not converged to an equilibrium (i.e., not all individuals have transitioned from being I to R). Thus, it is especially interesting to see if our method can recover the rate parameters, even if the process dynamics are still unfolding over time. Training the networks took approx. half a day with the active learning approach. Inference on 1000 datasets with 2000 posterior samples per parameter took approx. 1.1 seconds.



(a) Example SIR timeseries with $\beta = 0.5$ and $\gamma = 0.2$

(b) State-transition diagram of the SIR model

(c) Parameter recovery ($T = 500$)

(d) Performance over all Ts

(e) Posterior variance over all Ts

(f) Example posterior samples ($T = 500$)

(g) Simulation-based calibration (SBC)

Fig. 6. Results on the stochastic SIR model. **(a)** Example SIR timeseries generated with $\beta = 0.5$ and $\gamma = 0.2$; **(b)** State transition diagram of the SIR model; **(c)** Parameter recovery depicting also NRMSE and R^2 metrics; **(d)** NRMSE and R^2 performance over all Ts seen by the networks during training. We observe similar patterns as in the previous examples; **(e)** Posterior variances of the SIR parameters over all Ts; **(f)** Example full posteriors of the parameters recovered from three test datasets; **(g)** Plots of the rank statistic, according to which the posterior of β slightly overestimates the true mean.

The results on the SIR model are depicted in Figure 6. In line with the previous examples, we observe very good recovery of the true parameters, with NRMSE at $T = 500$ around 0.04, and R^2s around 0.98. Further, we observe decent performance even at smaller Ts, with performance increasing as T increases. Conversely, the estimated posterior variance drops as T increases, capturing the fact that more information becomes available to the networks. Finally, the SCB plots indicate a slight underestimation of the true posterior means by the estimated posterior means. This trend is visible when β and γ both assume very similar values, and the generated timeseries are far from equilibrium at the final time point T. Thus, such datasets contain very little information about the parameters given the considered time range of $T_{max} = 500$.

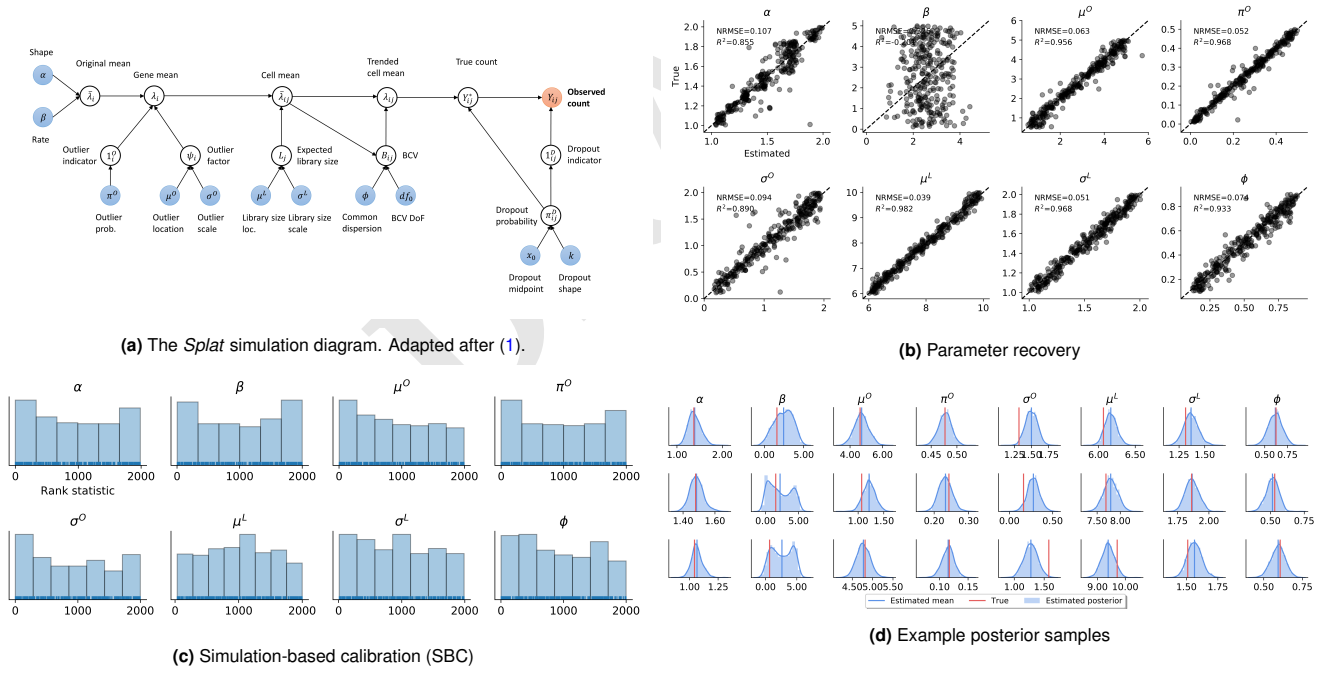
Example 4 - Single-Cell RNA Sequencing. Single-cell RNA sequencing (scRNA-seq) is a method to uncover the dynamics of gene expression within single cells (1, 41). Simulation models for scRNA-seq attempt to mimic the process of sequencing real data samples by combining statistical and algorithmic procedures. The recently developed *Splat* simulation model (1) implements a hierarchical model where the mean expression levels for multiple genes are samples from a Gamma distribution, and the number of times a gene has been sequenced in a cell is then sampled from a Poisson distribution. The output of the *Splat* simulation is thus a matrix $\boldsymbol{X}$ of *Gene x Cell* counts. Figure 7a depicts the *Splat* algorithm for simulating scRNA-seq data (1).

Table 1. Splat target parameters estimated by our method

Parameter	Name	Description
α	Mean shape	Shape of the mean gene expression gamma distribution
β	Mean rate	Rate of the mean gene expression gamma distribution
μ^L	Library size location	Location of the library size log-normal distribution
σ^L	Library size scale	Scale of the library size log-normal distribution
π^O	Outlier probability	Probability that a gene is an expression outlier
μ^O	Outlier location	Location of the expression outlier factor log-normal distribution
σ^O	Outlier scale	Scale of the expression outlier factor log-normal distribution
ϕ	Common BCV	Common BCV dispersion across all genes

As a last example for our method, we use simulations from the *Splat* model to recover the data-generating parameters. This is a challenging task for any Bayesian method, as the data-generating mechanism is a complicated algorithm with no available likelihood for joint estimation of the parameters. Moreover, the observed data exhibits multiple local and non-local dependencies. We simulate 200 000 *Gene x Cell* count matrices with different parameter settings. Each entry in a count matrix contains the number of times a gene has been sequenced in a different cell. We simulate counts for 40 genes and 100 cells, that is, each $\boldsymbol{X} \in \mathbb{R}^{40\times100}$. The parameters considered for estimation, along with a short description, are listed in Table 1.

The parameter estimation task is thus to recover $\boldsymbol{\theta} = (\alpha, \beta, \mu^L, \sigma^L, \pi^O, \mu^O, \sigma^O, \phi)$ from an observed count matrix $\boldsymbol{X}$. We train the networks for 30 epochs through the entire dataset and evaluate the performance on a separate validation set of 300 count matrices. To stabilize training and avoid exploding gradients due to large counts in the input matrices, we apply a log transformation to each $\boldsymbol{X}$. Simulating the data took approx. 8 hours. Training the networks took approx. one day with the reference table approach. Inference on 300 datasets with 2000 posterior samples per parameter took approx. 1.7 seconds.



(a) The *Splat* simulation diagram. Adapted after (1).

(b) Parameter recovery

(c) Simulation-based calibration (SBC)

(d) Example posterior samples

Fig. 7. Results on the *Splat* scRNA simulation model. **(a)** The *Splat* simulation diagram; **(b)** Parameter recovery; **(c)** Simulation-based calibraton. The SCB plots indicate a slight underestimation of the true posterior variance. **(d)** Example full parameter posteriors from three datasets.

The results on the scRNA example are depicted in Figure 7. First, recovery of most parameters is very good, with maximum R^2 of 0.98 and minimum NRMSE of 0.039 for the library location parameter μ^L. Interestingly, the rate parameter β appears to be nearly unrecoverable, as indexed by the scatterplots in Figure 7b. Furthermore, the estimated posterior of β is often bimodal, covering the whole prior range (Figure 7c). This might indicate that the parameter is not identifiable, or that some multivariate trade-offs exist in the parameter space of the *Splat* model. The latter is also confirmed by observing a few marked outliers in the scatterplots of α, σ^O, and π^O. Finally, the SCB histograms indicate that our method tends to slightly underestimate the true posterior variance of the α, β, μ^O π^O, and σ^O parameters. This issue might be due to the fact that the networks were trained with the reference-table approach and might disappear if the active learning approach is employed.

Discussion

In the current work, we proposed and explored a novel method which uses invertible neural networks to perform approximate Bayesian inference on any mathematical process model. The method requires only simulations from a model to learn a probabilistic mapping between data and parameters. We demonstrated the utility of the method by applying it to models and data from various scientific domains. Further, we explored two possible training approaches suitable for different simulation scenarios, namely, an active learning approach, and a reference-table approach. Both training approaches lead to excellent parameter estimation throughout the examples considered in the current work.

Our method combines the universal approximation power of deep learning methods (42) with the crucial uncertainty quantification assets of Bayesian inference (30, 31). Besides being capable of performing full Bayesian inference on intractable mathematical models, our method provides a general framework for designing reusable *parameter estimation machines* for various research domains. Moreover, it can also prove as a viable alternative in modeling contexts where standard inference methods are available, but inference is nevertheless computationally inconvenient.

Inspired by previous machine learning approaches to likelihood-free inference (5, 6, 18–22), our method shares many of the advantages of these methods and, further, overcomes some of their important limitations.

First, the introduction of separate summary and invertible networks renders the method independent of the shape or the size of the observed data. The summary network learns a fixed-size vector representation of the data in an automatic, data-driven manner. Since the summary network is optimized jointly with the invertible network, the learned data representation is encouraged to be maximally informative for parameter estimation. This is particularly useful in settings where appropriate summary statistics are not known and, as a consequence, relevant information is lost through the choice of suboptimal summary functions. However, if highly informative or even *sufficient* statistics are available in a given domain, one might dispose with the summary network altogether and feed these statistics directly to the invertible network.

Second, we showed that the ML loss directly maximizes the posterior over the target model parameters without any assumptions on the shape of the posterior. This is in contrast to ELBO-based methods which optimize a lower-bound on the posterior (22, 33), and usually assume Gaussian posteriors. Therefore, posterior inference is exact when the ML is globally minimized (24, 26). Our results support this theoretical claim. Further, since researchers are often interested in some summary of the posterior (e.g., posterior means or variances), we also showed that our method exhibits excellent recovery of the posterior means throughout all examples. Further, parameter recovery becomes better with increasing number of observations, whereas the posterior variance of the parameters decreases. These are important and highly desirable properties of any Bayesian parameter estimation method, as it mirrors the increase in information following increasing number of observations.

Third, the largest computational cost of our method is paid during training. Once trained, the networks can be used and reused to perform inference on large numbers of datasets within seconds and across a given research domain. Indeed, there are many instances of research domains where a single model is extensively explored and independently fitted by multiple researches to test scientific hypotheses (1, 37, 39, 43). These research domains are expected to benefit the most from learning the *model universe* once and then inverting the model multiple times for fast inference on multiple datasets. In this regard, our method is similar to the recently introduced prepaid method (19) which uses a database of pre-computed summary statistics and nearest-neighbors for inference. Note, however, that our method does not need to store training data, since the *knowledge* about the relationship between data and parameters is compressed into the networks' weights. This not only makes the global sharing of pre-trained parameter estimation networks across researchers extremely easy, but also makes their local storage very efficient. Finally, all computations involved in our method benefit from a high degree of parallelism and can thus utilize the advantages of modern parallel computing paradigms.

These advantages notwithstanding, some limitations of our method deserve mention. Although high-level deep learning libraries, such as *TensorFlow* or *Torch*, allow for rapid and relatively straightforward development of various neural network architectures, the implementational burden associated with the current method is still reasonably high. In order to ease the understanding and independent application of the method, we provide fully functioning code to reproduce and study all of the examples tackled in this paper (https://github.com/stefanradev93/cINN). In addition, we provide implementation of all metrics and visualization tools for performance validation used throughout the paper. Moreover, we are currently developing a general user-friendly software, which should abstract away most of the methodological complexities from the user. Another potential shortcoming of the method is the seemingly overwhelming number of hyperparameters that might require fine-tuning by the user for optimal performance on a given task. However, we observe that many of the default hyperparameter values are sufficient to achieve excellent performance, and starting with a relatively large default network of 10 ACBs does not appear to hurt performance or destabilize training, even if the mathematical model to be learned is relatively simple. Based on our results, we expect that a single architecture should be able to perform well on almost all models from a given domain. Future research should investigate the question of generality by applying the method to challenging parameter estimation tasks across different research domains.

We hope that the new method will enable researchers from a variety of fields to accelerate model-based inference and will further prove its utility beyond the examples considered in this paper.

ACKNOWLEDGMENTS. We thank Jeffrey Rouder, Raphael Hartmann, David Izydorczyk, Hannes Wendler, Satya Almasian, and Karin Prillinger for their invaluable comments and suggestions that greatly improved the manuscript.

1. Zappia L, Phipson B, Oshlack A (2017) Splatter: simulation of single-cell rna sequencing data. *Genome biology* 18(1):174.
2. Beaumont MA, Zhang W, Balding DJ (2002) Approximate bayesian computation in population genetics. *Genetics* 162(4):2025–2035.

3. Palestro JJ, Sederberg PB, Osth AF, Van Zandt T, Turner BM (2018) *Likelihood-free methods for cognitive science*. (Springer).
4. Usher M, McClelland JL (2001) The time course of perceptual choice: the leaky, competing accumulator model. *Psychological review* 108(3):550.
5. Hwang SJ, Tao Z, Kim WH, Singh V (2018) Conditional recurrent flow: Conditional generation of longitudinal samples with applications to neuroimaging. *arXiv preprint arXiv:1811.09897*.
6. Lueckmann JM, et al. (2017) Flexible statistical inference for mechanistic models of neural dynamics in *Advances in Neural Information Processing Systems*. pp. 1289–1299.
7. Wood SN (2010) Statistical inference for noisy nonlinear ecological dynamic systems. *Nature* 466(7310):1102.
8. Geritz SA, Kisdi É (2004) On the mechanistic underpinning of discrete-time population models with complex dynamics. *Journal of Theoretical Biology* 228(2):261–269.
9. Keeling MJ, Rohani P (2011) *Modeling infectious diseases in humans and animals*. (Princeton University Press).
10. Hethcote HW (2000) The mathematics of infectious diseases. *SIAM review* 42(4):599–653.
11. Csilléry K, Blum MG, Gaggiotti OE, François O (2010) Approximate bayesian computation (abc) in practice. *Trends in ecology & evolution* 25(7):410–418.
12. Toni T, Stumpf MP (2009) Simulation-based model selection for dynamical systems in systems and population biology. *Bioinformatics* 26(1):104–110.
13. Turner BM, Sederberg PB (2014) A generalized, likelihood-free method for posterior estimation. *Psychonomic bulletin & review* 21(2):227–250.
14. Sunnåker M, et al. (2013) Approximate bayesian computation. *PLoS computational biology* 9(1):e1002803.
15. Mertens UK, Voss A, Radev S (2018) Abrox—a user-friendly python module for approximate bayesian computation with a focus on model comparison. *PloS one* 13(3):e0193981.
16. Frazier DT, Martin GM, Robert CP, Rousseau J (2018) Asymptotic properties of approximate bayesian computation. *Biometrika* 105(3):593–607.
17. Mertens UK (2019) Ph.D. thesis.
18. Radev ST, Mertens UK, Voss A, Köthe U (2019) Towards end-to-end likelihood-free inference with convolutional neural networks. *British Journal of Mathematical and Statistical Psychology*.
19. Mestdagh M, Verdonck S, Meers K, Loossens T, Tuerlinckx F (2018) Prepaid parameter estimation without likelihoods. *arXiv preprint arXiv:1812.09799*.
20. Raynal L, et al. (2018) Abc random forests for bayesian parameter inference. *Bioinformatics* 35(10):1720–1728.
21. Jiang B, Wu Ty, Zheng C, Wong WH (2017) Learning summary statistic for approximate bayesian computation via deep neural network. *Statistica Sinica* pp. 1595–1618.
22. Papamakarios G, Murray I (2016) Fast ε-free inference of simulation models with bayesian conditional density estimation in *Advances in Neural Information Processing Systems*. pp. 1028–1036.
23. Ardizzone L, et al. (2018) Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730*.
24. Kingma DP, Dhariwal P (2018) Glow: Generative flow with invertible 1x1 convolutions in *Advances in Neural Information Processing Systems*. pp. 10215–10224.
25. Grover A, Dhar M, Ermon S (2018) Flow-gan: Combining maximum likelihood and adversarial learning in generative models in *Thirty-Second AAAI Conference on Artificial Intelligence*.
26. Dinh L, Sohl-Dickstein J, Bengio S (2016) Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.
27. Bloem-Reddy B, Teh YW (2019) Probabilistic symmetry and invariant neural networks. *arXiv preprint arXiv:1901.06082*.
28. Gers FA, Schmidhuber J, Cummins F (1999) Learning to forget: Continual prediction with lstm.
29. Long J, Shelhamer E, Darrell T (2015) Fully convolutional networks for semantic segmentation in *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 3431–3440.
30. Kendall A, Gal Y (2017) What uncertainties do we need in bayesian deep learning for computer vision? in *Advances in neural information processing systems*. pp. 5574–5584.
31. Gelman A, et al. (2013) *Bayesian data analysis*. (Chapman and Hall/CRC).
32. Abadi M, et al. (2016) Tensorflow: A system for large-scale machine learning in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. pp. 265–283.
33. Kingma DP, Welling M (2014) Auto-encoding variational bayes. *stat* 1050:1.
34. Mnih V, et al. (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529.
35. Hershey JR, Olsen PA (2007) Approximating the kullback leibler divergence between gaussian mixture models in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*. (IEEE), Vol. 4, pp. IV–317.
36. Talts S, Betancourt M, Simpson D, Vehtari A, Gelman A (2018) Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788*.
37. Ratcliff R, McKoon G (2008) The diffusion decision model: theory and data for two-choice decision tasks. *Neural computation* 20(4):873–922.
38. Miletić S, Turner BM, Forstmann BU, van Maanen L (2017) Parameter recovery for the leaky competing accumulator model. *Journal of Mathematical Psychology* 76:25–50.
39. Voss A, Lerche V, Mertens U, Voss J (2019) Sequential sampling models with variable boundaries and non-normal noise: A comparison of six models. *Psychonomic bulletin & review* pp. 1–20.
40. Sahneh FD, Vajdi A, Shakeri H, Fan F, Scoglio C (2017) Gemfsim: a stochastic simulator for the generalized epidemic modeling framework. *Journal of computational science* 22:36–44.
41. Ozsolak F, Milos PM (2011) Rna sequencing: advances, challenges and opportunities. *Nature reviews genetics* 12(2):87.
42. Goodfellow I, Bengio Y, Courville A (2016) *Deep learning*. (MIT press).
43. De Valpine P, Hastings A (2002) Fitting population models incorporating process noise and observation error. *Ecological Monographs* 72(1):57–76.