TOWARDS END-TO-END LIKELIHOOD-FREE INFERENCE WITH CONVOLUTIONAL NEURAL NETWORKS

Stefan T. Radev*, Ulf K. Mertens*, Andreas Voss, and Ullrich Köthe

Heidelberg University, Germany

Authors' note

Stefan Radev, Institute of Psychology, Heidelberg University, Hauptstr. 47-51, 69117 Heidelberg, Germany; Ulf Mertens, Institute of Psychology, Heidelberg University, Hauptstr. 47-51, 69117 Heidelberg, Germany; Andreas Voss, Institute of Psychology, Heidelberg University, Hauptstr. 47-51, 69117 Heidelberg, Germany; Ullrich Köthe, Heidelberg Collaboratory for Image Processing (HCI), Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany.

*S.T. Radev and U.K. Mertens contributed equally to this work

Correspondence concerning this article should be addressed to Stefan Radev, e-mail: stefan.radev93@gmail.com; Ulf Mertens, e-mail: ulf.mertens@psychologie.uni-heidelberg.de.

Acknowledgments

We thank Steven Miletić and Leendert van Maanen, as well as Louis Raynal and Jean-Michel Marin for providing us with their R code for simulating data from the LCA and regression models, respectively. We also thank Alica Bucher for helping us with the visualization of our model architecture.

Abstract

Complex simulator-based models with non-standard sampling distributions require sophisticated design choices for reliable approximate parameter inference. We introduce a fast, end-to-end approach for Approximate Bayesian Computation (ABC) based on fully convolutional neural networks (CNNs). The method enables users of ABC to simultaneously derive the posterior mean and variance of multi-dimensional posterior distributions directly from raw simulated data. Once trained on simulated data, the CNN is able to map real data samples of variable size to the first two posterior moments of the relevant parameter's distributions. Thus, in contrast to other machine learning approaches to ABC, our approach allows to generate reusable models that can be applied by different researchers employing the same model. We verify the utility of our method on two common statistical models, namely a multivariate normal distribution and a multiple regression scenario, for which the posterior parameter distributions can be derived analytically. We then apply our method to recover the parameters of the Leaky Competing Accumulator (LCA) model and compare our results to the current state-of-the art technique, the Probability Density Estimation (PDA). Results show that our method exhibits lower approximation error compared to other machine learning approaches to ABC. It also outperforms PDA in recovering the parameters of the LCA model.

Keywords: approximate bayesian computation, likelihood-free inference, convolutional network, machine-learning, leaky competing accumulator

TOWARDS END-TO-END LIKELIHOOD-FREE INFERENCE WITH CONVOLUTIONAL NEURAL NETWORKS

A major goal in statistical modeling is to describe data generation processes by a finite parameter vector $\boldsymbol{\theta}$. Due to different sources of uncertainty (Goodfellow et al., 2016, p. 52), such descriptions are inherently stochastic. Naturally, researches are interested in quantifying the uncertainty in their parametric estimates (Kendall & Gal, 2017; Schoot et al., 2014; Lee, 2008). By drawing on the mathematical framework of probability theory, one way to represent uncertainty is to place probability distributions over parameters, and update the parameters of these distributions as the current state of knowledge changes. In fact, this is the quintessential idea incorporated in Bayesian statistics (Gelman et al., 2013).

Suppose we have collected a dataset $\boldsymbol{x} = \{x^{(i)}\}_{i=1}^{N}$ in an observational or an experimental setting. At the core of all Bayesian data analysis lies Bayes' rule:

$$p(\boldsymbol{\theta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{\int p(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}} \tag{1}$$

In the above expression, $p(\boldsymbol{\theta}|\boldsymbol{x})$ denotes the posterior distribution of the model parameters, $p(\boldsymbol{x}|\boldsymbol{\theta})$ denotes the *likelihood function*, $\pi(\boldsymbol{\theta})$ the *prior probability distribution,* and the denominator $p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}$ the *marginal probability* of the data. Considered as a normalization constant, the computation of the marginal probability can often be bypassed, so the posterior distribution can be expressed as being proportional to the prior times the likelihood:

$$p(\boldsymbol{\theta}|\boldsymbol{x}) \propto p(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}) \tag{2}$$

If the likelihood belongs to a known family of probability distributions, and if the prior is conjugate to the likelihood, then the posterior belongs to the same distribution family as the

prior. Therefore, by choosing mathematically convenient priors, one can obtain a closed-form expression for the posterior which has the same algebraic form as the prior, merely with modified parameter values. In the case of non-conjugate priors, one needs to resort to Monte Carlo sampling methods, such as Markov Chain Monte Carlo (MCMC) algorithms.

A different problem arises when the likelihood $p(\boldsymbol{x}|\boldsymbol{\theta})$ is computationally intractable, or cannot be specified algebraically. This situation occurs when models become increasingly complex, or rely on a simulation-based mechanism for generating the data (Turner, 2012). In this case, approximation methods are required in order to perform any type of Bayesian inference.

ABC methods

Approximate Bayesian Computation (ABC) methods are part of a larger class of techniques aimed at bypassing the *intractability problem* arising when parametric models require a non-standard solution based on a likelihood function.

ABC was first successfully implemented in genetics research (Tavare et al., 1997) and the utility of the method has been steadily increasing across research domains ever since (Pritchard et al., 1999). The main idea of ABC methods is to approximate the likelihood function by repeatedly sampling parameters from prior distributions and simulating artificial datasets conditioned on the sampled parameters. Even though exact numerical evaluation of the likelihood $p(\boldsymbol{x}|\boldsymbol{\theta})$ might be intractable, a generative model of the form $q(\cdot\,|\boldsymbol{\theta})$, usually specified as a function code in a programming language, is necessary for performing the simulations.

The starting point for all ABC algorithms is the creation of the so-called *reference table* (Raynal et al., 2017; Mertens et al., 2018). The reference table is a special data structure used to store a large number of simulated datasets or summary-statistics of such produced from an approximation of the posterior distribution given by $p(\boldsymbol{\theta}|\boldsymbol{x}) \propto p(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}) \approx q(\cdot\,|\boldsymbol{\theta})\pi(\boldsymbol{\theta})$ as

well as the corresponding samples from the prior distributions(s) used to generate these datasets. In the following, the details of creating the reference table are discussed.

In line with the notation embraced by the literature on ABC, let $\boldsymbol{\theta}$ denote a random vector of parameters and $\pi(\boldsymbol{\theta})$ denote the joint prior distribution over the parameter vector. Let $\{\tilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{N}$ represent a collection of artificial datasets simulated by a generative model $q(\cdot\,|\boldsymbol{\theta})\pi(\boldsymbol{\theta})$, $\boldsymbol{x}$ – the observed dataset, and $\eta(\tilde{\boldsymbol{x}}^{(i)})$ – a (vector) summary statistic obtained by some form of dimensionality reduction applied to each $\tilde{\boldsymbol{x}}^{(i)}$. Consequently, **Algorithm 1** describes the standard procedure for generating the ABC reference table.

Algorithm 1. Generation of the reference table.

for $i \leftarrow 1$ to N

Sample $\boldsymbol{\theta}^{(i)} \sim \pi(\cdot)$

Simulate $\tilde{\boldsymbol{x}}^{(i)} \sim q\left(\cdot\,\middle|\boldsymbol{\theta}^{(i)}\right)$

Compute $\eta\left(\tilde{\boldsymbol{x}}^{(i)}\right)$

Store the pair $\left(\eta\left(\tilde{\boldsymbol{x}}^{(i)}\right), \boldsymbol{\theta}^{(i)}\right)$ in row i of the reference table

end for

There are multiple ways to utilize the reference table for the purpose of parameter inference. The rejection algorithm, initially proposed by Rubin (1984), and refined by Tavaré et al. (1997) and Pritchard et al. (1999), requires specifying a distance function $d(\eta\left(\tilde{\boldsymbol{x}}^{(i)}\right), \eta(\boldsymbol{x}))$ quantifying the *deviation* of a given simulated sample from the observed data as well as a threshold (also termed the tolerance level) $0 < \epsilon \leq 1$, according to which a fraction of $1 - \epsilon$ parameters is rejected, and the remaining are considered as samples from an approximate posterior distribution $p_{\epsilon}\left(\boldsymbol{\theta}\middle|\eta(\tilde{\boldsymbol{x}})\right)$. Even though, in theory, the rejection method is doubly asymptotically consistent (Frezier et al., 2017), its practical limitations have been repeatedly

noted by researchers (Raynal et al., 2017; Martin et al., 2012; Blum, 2010), necessitating the use of more sophisticated methods. First, the rejection method suffers from the *curse of dimensionality*, meaning that as the dimensions of the parameter space increase, the number of simulations required to obtain reasonable convergence grows exponentially large. Second, in order for $p_\epsilon\big(\boldsymbol{\theta}|\eta(\tilde{\boldsymbol{x}})\big)$ to approximate the true posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x})$ within a reasonable amount of computational time, the parameter ϵ needs to be carefully tuned. Third, the summary statistic $\eta(\cdot)$ has to be *sufficient*, meaning that no loss of information from the sample should be incurred by computing it. Finally, the choice of a particular distance metric $d(\cdot,\cdot)$ is often arbitrary (e.g., *Euclidian distance*), and thus presents another nontrivial hyperparameter of the method requiring further attention.

Machine learning approaches to ABC

Recently, it has been proposed to "borrow" techniques from the machine learning literature in order to confront the above mentioned shortcomings of traditional ABC methods. In the next two sections, we briefly review two recent applications of the supervised machine learning approach to ABC. We address both the advantages and limitations of these approaches, and proceed to introduce our method.

Random forest methodology

The random forest (RF) algorithm, originally developed by Breiman (2001), appears to provide an especially promising approach to estimating the first two moments (see Raynal et al., 2017 for details) and quantiles of a posterior parameter distribution. In particular, random forest regression is a supervised learning algorithm, which, given a sample of input-output pairs, learns a highly nonlinear mapping from the input to the output by training an ensemble of decision trees

(a standard non-parametric algorithm for classification and regression) and aggregating their regression function outputs.

In the context of ABC, the inputs to the RF algorithm are the simulated and summarized datasets $\{\eta(\tilde{\boldsymbol{x}}^{(i)})\}_{i=1}^{N}$ from the reference table, while the outputs are the corresponding $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{N}$ parameter values. In other words, the algorithm learns to recover the unknown parameters used to generate a given dataset. **Algorithm 2** describes the RF-ABC procedure.

There are at least three main justifications to apply RF regression to ABC parameter inference. First, the *representational capacity* of the algorithm enables it to learn complex non-linear input-output mappings and thus provides reliable and theoretically sound approximations of the posterior mean and variance: $\mathbb{E}[\boldsymbol{\theta}|\eta(\tilde{\boldsymbol{x}}), \boldsymbol{\Theta}]$, $Var[\boldsymbol{\theta}|\eta(\tilde{\boldsymbol{x}}), \boldsymbol{\Theta}]$, where $\boldsymbol{\Theta}$ represents the random forest parameters. Second, the robustness of RF regression to the presence of irrelevant predictors and to different hyperparameter settings has been repeatedly emphasized and demonstrated (Pudlo et al., 2015; Roy & Larocque, 2012). Finally, since the method does not aim at performing full Bayesian inference, but instead relies on approximating the first two moments and quantiles of the posterior distribution, the need for selecting a tolerance level ϵ and a distance metric $d(\cdot,\cdot)$ is no longer required.

The ABC-RF methodology inevitably reduces the burden of meticulously hand-crafting a small number of sufficient summary statistics, since one can potentially define a (very) large number of summaries. However, it does not bypass the problem completely, since the user still needs to manually decide on and compute a (potentially large) collection of summary statistics. A further feature of the ABC-RF methodology is the fact that existing implementations involve the training a separate random forest ensemble for each individual model parameter θ_k of the parameter vector $\boldsymbol{\theta}$. Even though this approach could easily be parallelized across parameters,

sequential solutions might experience computational bottleneck, especially when dealing with high-dimensional parameter spaces, combined with a large reference table.

Algorithm 2. ABC-RF methodology (Raynal et al., 2017).

Construct a reference table $\{\left(\eta\left(\tilde{\boldsymbol{x}}^{(i)}\right),\ \boldsymbol{\theta}^{(i)}\right)\}_{i=1}^{N}$ using **Algorithm 1**.

$K \leftarrow$ # of model parameters

for $k \leftarrow 1$ to K

Train a random forest regression $\hat{f}_{\boldsymbol{\Theta}_k}(\eta(\tilde{\boldsymbol{x}}))$ to predict parameter θ_k

end for

Output a group of random forests $\{\hat{f}_{\boldsymbol{\Theta}_k}\}_{k=1}^{K}$

Posterior uncertainty estimation using OOB samples:

for $k \leftarrow 1$ to K

Estimate $\widehat{\mathbb{E}}[\theta_k|\eta(\tilde{\boldsymbol{x}}), \boldsymbol{\Theta}_k]$, $\widehat{Var}[\theta_k|\eta(\tilde{\boldsymbol{x}}), \boldsymbol{\Theta}_k]$ and posterior quantiles $\hat{Q}_p(\theta_k|\eta(\tilde{\boldsymbol{x}}), \boldsymbol{\Theta}_k)$

end for

Deep neural network methodology

Another promising approach to ABC inspired by machine learning utilizes deep neural networks (DNNs) for automatically learning summary statistics from the reference table (Jiang et al., 2015). At a general level, a neural network defines a mapping $\boldsymbol{y} = f_{\boldsymbol{W}}(\boldsymbol{x})$ from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ and learns the parameters $\boldsymbol{W}$ that lead to the best function approximation (Goodfellow et al., 2016). As in the case of ABC-RF, the challenge of ABC is cast as a supervised learning task, with artificial datasets $\{\tilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{N}$ as inputs, and dataset-generating parameters $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{N}$ as outputs. It is important to note that raw simulated data is used as input to the neural network, since the goal is to learn the functional form of $\eta(\cdot)$ implicitly from the data, treating the predicted parameters as the summary statistic. **Algorithm 3** describes the ABC-DNN procedure.

Algorithm 3. ABC-DNN methodology (Jiang et al., 2015).

Construct a reference table $\{(\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{\theta}^{(i)})\}_{i=1}^{N}$ using **Algorithm 1**.

Train a DNN $f_{\boldsymbol{W}}$ with $\{\tilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{N}$ as input and $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{N}$ as output

Rejection ABC algorithm with DNN outputs as summary statistics

for $t \leftarrow 1$ to M

Sample $\boldsymbol{\theta}^{(t)} \sim \pi(\cdot)$

Simulate $\tilde{\boldsymbol{x}}^{(t)} \sim q(\cdot \,|\boldsymbol{\theta}^{(t)})$

Compute $\widehat{\boldsymbol{\theta}}^{(t)} = f_{\boldsymbol{W}}(\tilde{\boldsymbol{x}}^{(t)})$ as a summary statistic

end for

Compute summary statistic of observed data $\boldsymbol{\theta} = f_{\boldsymbol{W}}(\boldsymbol{x})$

Select $\widehat{\boldsymbol{\theta}}^{(t)}$ such that $d(\boldsymbol{\theta}, \widehat{\boldsymbol{\theta}}^{(t)}) < \epsilon$ where $d(\cdot,\cdot)$ is a distance function and ϵ is a pre-defined tolerance level

Use selected $\widehat{\boldsymbol{\theta}}^{(t)}$ to estimate features of $p(\boldsymbol{\theta}|\boldsymbol{x})$

The most important advantage of using DNNs as supervised approximators is their huge representational power (Goodfellow et al., 2016; Jiang et al., 2015; Hornik, 1991). According to the *universal approximation theorem (*Cybenko, 1989*;* Hornik, 1991), neural networks are able to, in principle, approximate any continuous function to an arbitrary degree of accuracy, given the appropriate conditions and parameters. Furthermore, neural network architectures can easily be designed to incorporate multi-output regression, as is the case when $\boldsymbol{\theta}$ is multidimensional, thus learning each mapping from $\{\tilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{N}$ to individual parameters simultaneously.

One disadvantage of the ABC-DNN method compared to the ABC-RF approach is that it does not offer a straightforward way to quantify the uncertainty inherent in the DNN estimates, which is captured by the estimate of the posterior variance $Var[\boldsymbol{\theta}|\eta(\tilde{\boldsymbol{x}}), \boldsymbol{\Theta}]$ within the ABC-RF framework. As a consequence, Jiang et al. (2015) used the DNN estimates of the posterior expectation as summary statistics in the classical ABC rejection algorithm, thus inheriting the

difficulties in specifying a convenient distance metric $d(\cdot,\cdot)$ and guessing an appropriate tolerance level ϵ. Another, more subtle, yet serious shortcoming of both ABC-DNN and ABC-RF, is the inability to deal with variable input sizes (e.g., the observed sample size). This inevitably confines the dimensionality of the simulated data to the dimensionality of the observed data sample. In addition, reusing the same model in a context where the observed data sample has a different size is not possible. Hence, researchers interested in applying current machine learning models need to generate a completely new reference table and train the models from scratch.

DeepInference

The approach we investigate in this paper is an attempt to fuse the advantages of both the ABC-RF and the ABC-DNN methodologies discussed so far (Raynal et al., 2017; Jiang et al., 2015). It is also inspired by recent advances in modelling uncertainty in deep learning predictions (Kendall & Gall, 2017).

We propose to train a fully convolutional neural network (CNN) on simulated data distributions to approximate the posterior mean of the relevant model parameters, and, at the same time, estimate the uncertainty in the posterior approximations directly from the available data by minimizing the *heteroscedastic loss* (Kendall & Gall, 2017). We argue that this approach overcomes most of the shortcomings of previous machine learning approaches to ABC.

The outline of the rest of this paper is as follows. We first introduce the building blocks of our approach. Then, we confirm on two toy examples that the predictions and uncertainty estimates obtained by the optimization procedure employed in the CNN closely approximate the analytically computed posterior expectations and posterior variances of the data-generating parameters. After that, we demonstrate the usefulness of the method in recovering the parameters

of the Leaky Competitive Accumulator (LCA; Usher & McClelland, 2001) model used widely in cognitive science and psychology and compare our results with the latest state-of-the-art method (Miletić et al., 2017). Finally, we discuss the advantages of the current approach.

Methods

Inference as optimization

In general, a regression model is trained by minimizing the mean squared error (MSE) loss across N training examples (simulations) between actual parameters $\theta^{(i)}$ and parameters predicted by the model $\hat{\theta}^{(i)} = f_{\boldsymbol{W}}(\tilde{\boldsymbol{x}}^{(i)})$. The loss function for a single-parameter model is thus given by:

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N}\left\|\hat{\theta}^{(i)} - \theta^{(i)}\right\|_2^2 \tag{3}$$

and optimized via stochastic gradient descent using the backpropagation algorithm. Since the MSE loss is proportional to the cross-entropy between the empirical distribution of the training set and a Gaussian model, minimizing the MSE loss is equivalent to minimizing the negative log-likelihood of a conditional Gaussian model[1] defined as:

$$p(\theta|\tilde{\boldsymbol{x}}\,;\boldsymbol{W}) = \mathcal{N}(\theta|f_{\boldsymbol{W}}(\tilde{\boldsymbol{x}}), \sigma^2) \tag{4}$$

where the prediction of the neural network $\hat{\theta} = f_{\boldsymbol{W}}(\tilde{\boldsymbol{x}})$ corresponds to the mean of the Gaussian distribution. The negative log-likelihood thus becomes:

[1] Another interpretation of maximum likelihood is that it minimizes the cross-entropy between the data distribution and the model distribution (see Goodfellow et al., 2016, p. 129).

$$-\log p(\theta|\tilde{\boldsymbol{x}};\boldsymbol{W}) = -\sum_{i=1}^{N} \log p(\theta^{(i)}|\tilde{\boldsymbol{x}}^{(i)};\boldsymbol{W}) \tag{5}$$

$$= -\sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\left(\hat{\theta}^{(i)} - \theta^{(i)}\right)^2}\right) \tag{6}$$

$$= \sum_{i=1}^{N} \frac{\left\|\hat{\theta}^{(i)} - \theta^{(i)}\right\|_2^2}{2\sigma^2} + \frac{N}{2}\log(2\pi\sigma^2) \tag{7}$$

and we see that the minimizing the MSE criterion in (3) w.r.t to the DNN weights $\boldsymbol{W}$ is equivalent to minimizing (7) where σ^2 is considered to be a fixed constant. Since setting $\hat{\theta}$ to be equal to the posterior expectation $\mathbb{E}[\theta|\tilde{\boldsymbol{x}}]$ leads to the best possible prediction of θ given simulated data $\tilde{\boldsymbol{x}}$ and hence minimizes the expected MSE, the optimization procedure of the supervised learning approach effectively approximates the posterior mean of a given parameter θ (and similarly for a parameter vector $\boldsymbol{\theta}$ in a multi-dimensional context). This observation also implies that any supervised learning approach which optimizes the MSE criterion can potentially be trained on an ABC reference table to approximate $\mathbb{E}[\boldsymbol{\theta}|\boldsymbol{x}]$, thus re-framing the problem of inference as a problem of optimization.

Heteroscedastic loss

As in the previous machine learning approaches to ABC, we frame the problem of parameter estimation as a supervised learning task, provided that a reference table is available. We propose to train a 1D convolutional neural network (CNN) on the raw simulated datasets $\{\tilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{N}$, optimizing the heteroscedastic loss criterion (Kendall & Gal, 2017; Nix & Weigend, 1994):

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2\sigma^2(\tilde{\boldsymbol{x}}^{(i)})} \left\|\hat{\theta}^{(i)} - \theta^{(i)}\right\|_2^2 + \frac{1}{2}\log\sigma^2(\tilde{\boldsymbol{x}}^{(i)}) \tag{8}$$

It is immediately obvious, that the heteroscedastic loss is proportional to the negative log-likelihood of a Gaussian model with constant variance as defined by eq. (7). However, we now also train the network to explicitly predict the σ^2 term, corresponding to the variance of the target parameter distribution. The target distribution from eq. (4) then becomes:

$$p(\theta|\tilde{\boldsymbol{x}}; \boldsymbol{W}) = \mathcal{N}(\theta|f_{\boldsymbol{W}}(\tilde{\boldsymbol{x}}), \sigma^2(\tilde{\boldsymbol{x}})) \tag{9}$$

where we explicitly denote the dependence of the σ^2 variance term on the simulated input dataset $\tilde{\boldsymbol{x}}$. Thus, the network is trained to predict $\sigma^2(\tilde{\boldsymbol{x}}^{(i)})$ along with $\hat{\theta}^{(i)}$. The network output then becomes $\left(\widehat{\theta},\ \sigma^2(\tilde{\boldsymbol{x}})\right) = f_{\boldsymbol{W}}(\tilde{\boldsymbol{x}})$. Assuming a Gaussian regression model, another interpretation of the heteroscedastic loss suggests that the network's predictions approximate both the posterior mean $\mathbb{E}[\theta|\boldsymbol{x}]$ and the posterior variance $Var[\theta|\boldsymbol{x}]$ of the parameter distribution.

Heteroscedastic regression comes with two important advantages: 1) it does not assume a fixed variance parameter across all inputs $\tilde{\boldsymbol{x}}^{(i)}$; 2) it enables us to learn the variance term $\sigma^2(\tilde{\boldsymbol{x}}^{(i)})$ during training as a function of the data. Intuitively, if the model makes a particularly bad prediction, the squared $L2$ loss term $\left\|\hat{\theta}^{(i)} - \theta^{(i)}\right\|_2^2$ is going to be large, thereby inducing an increase in the $\sigma^2(\tilde{\boldsymbol{x}}^{(i)})$ uncertainty term to keep the overall error low. The $\log \sigma^2(\tilde{\boldsymbol{x}}^{(i)})$ term, on the other hand, prevents the model from "cheating" by not allowing the uncertainty to grow to infinity, which would inevitably reduce the squared error term, but would not result in the model learning any meaningful information from the data.

Neural network architecture

For both toy examples, we used a fully convolutional neural network with three hidden layers (see O'Shea & Nash, 2015 for an in-depth discussion on convolutional networks) where

the last hidden layer computes the average value of each kernel's output, also known as global average pooling (Lin et al., 2013). If m is the number of parameters of the model from which samples are generated, the output layer of the CNN consists of m linear activation units for predicting each component of $\boldsymbol{\theta}$, and further m units predicting each corresponding $\sigma^2(\tilde{\boldsymbol{x}})$. The latter units apply the *softplus* activation function, defined by:

$$\xi(z) = \log(1 + e^z) \tag{10}$$

The softplus activation is introduced in order to ensure that estimates of $\sigma^2(\tilde{\boldsymbol{x}})$ are always non-negative. Thus, for each application, the number of output units is double the number of parameters to be estimated. For all examples, we trained the CNN for 5 epochs using the *Adam* optimization algorithm (Kingma et al., 2014) with learning rate set to 0.001. We did not perform extensive hyperparameter search over the CNN parameter space, and strived to keep the model as small as possible.

The core idea of our convolutional approach is to fully exploit the grid-like structure inherent in most datasets. In the simplest case of 1D $i.i.d.$ datasets, each data point is statistically independent of all others, so it makes sense to use convolutional kernels (also called filters) of size 1 and stride (step size) 1 in the first layer of the network. Importantly, due to the properties of convolutional layers (see the **Discussion** section for more details on why CNN properties are useful for extracting information from raw data samples), shuffling the data sample does not dramatically change the output of a particular kernel, which has hopefully learned to encode a given range of values, regardless of the position of the values in the input sample. In later layers, we allow for kernel sizes > 1 in order to extract further informative characteristics of the data, thus implicitly learning a hierarchy of summary statistics.

In the more complicated case where each simulated dataset forms a multidimensional array, and data points along a given axis (e.g., rows) are connected in some way, the first hidden layer is built from kernels of size appropriate for encoding information contained in combinations or sequences of data points.

To make the latter description more concrete, consider a dataset with n rows and k columns containing $k-1$ predictor variables and an outcome variable in a multiple regression scenario (see also *Figure 1*). Each row is composed of one single value per predictor variable and the corresponding outcome. We exploit this structure by using 1d convolutional filters with a kernel size corresponding to the number of columns and stride of 1 as the first layers. These settings (kernel size = k, stride = 1) ensure that the network is fed all the relevant information inherent in this particular data structure. **Algorithm 4** lays out the essential steps of our method.

Algorithm 4. DeepInference algorithm.

Construct a reference table $\{(\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{\theta}^{(i)})\}_{i=1}^{N}$ using **Algorithm 1**.

Train a convolutional neural network $f_{\boldsymbol{W}}$ with $\{\tilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{N}$ as input and $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{N}$ as output by optimizing the heteroscedastic loss (eq. 8)

Uncertainty estimation:

Perform a forward pass on the observed data $(\boldsymbol{\theta}, \sigma^2(\boldsymbol{x}))) = f_{\boldsymbol{W}}(\boldsymbol{x})$ to obtain estimates of $\mathbb{E}[\boldsymbol{\theta}|\boldsymbol{x}]$ and $Var[\boldsymbol{\theta}|\boldsymbol{x}]$

All CNN models were implemented via the high-level neural networks API keras (https://keras.io/; see also Chollet, 2017) written in the Python programming language. Reference tables used for training the models and model implementation code are readily available on demand.

In the following section, we evaluate the method proposed above on two artificial statistical models with known posterior distributions. Then, we apply the method to the LCA model from cognitive psychology.

Results

Toy example 1: Multivariate normal with unknown mean and variance

We first demonstrate the utility of our method on the simple conjugate multivariate normal (MVN) model introduced in Gelman et al. (2013, pp. 72–73). The conjugate prior distributions for the mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ of the MVN model are given by:

$$\begin{aligned} \boldsymbol{\Sigma} &\sim InvWishart(\boldsymbol{\Lambda}_0, v_0) \\ \boldsymbol{\mu}|\boldsymbol{\Sigma} &\sim \mathcal{N}_d(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}/k_0) \\ \boldsymbol{x}_i|\,\boldsymbol{\mu}, \boldsymbol{\Sigma} &\sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \qquad i = 1, \ldots, m \end{aligned} \tag{11}$$

In this model, $InvWishart(\boldsymbol{\Lambda}_0, v_0)$ denotes an inverse Wishart distribution parameterized by a d-by-d scale matrix $\boldsymbol{\Lambda}_0$ and degrees of freedom v_0, and $\mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes a multivariate normal distribution with a d-dimensional mean vector $\boldsymbol{\mu}$ and a d-by-d covariance matrix $\boldsymbol{\Sigma}$. Since these priors are conjugate to the normal likelihood, it is possible to derive closed-form solutions for the joint posterior distributions (Gelman et al., 2013). Thus, to obtain samples from the true joint posterior distribution of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, one uses the following procedure: first, draw $\boldsymbol{\Sigma}|\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \sim InvWishart(\boldsymbol{\Lambda}_m, v_m)$, then draw $\boldsymbol{\mu}|\boldsymbol{\Sigma}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \sim \mathcal{N}_d(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}/k_m)$ where $(\boldsymbol{\Lambda}_m, v_m, \boldsymbol{\mu}_m, k_m)$ represent the posterior model parameters, computed as functions of the data and the prior parameters (see Gelman et al., 2013, pp. 72–73).

For the current example, we set $d = 2,\ m = 100, \boldsymbol{\Lambda}_0 = \boldsymbol{I}_2, v_0 = 5, k_0 = 3, \boldsymbol{\mu}_0 = (0, 0)^T$ and simulated 100 000 datasets from the model defined by (11). Thus, each row i of the

reference table contains the raw sample $\{\tilde{\boldsymbol{x}}_k^{(i)}\}_{k=1}^{100}$, each element $\tilde{\boldsymbol{x}}_k^{(i)}$ being a realization of a two-dimensional random vector. Given the referenced table as input, the CNN model was trained to predict the corresponding values of $\{\boldsymbol{\mu}^{(i)}, diag(\boldsymbol{\Sigma})^{(i)}\}$. Note that we are only interested in the diagonal elements of the covariance matrix, since we are only estimating the variance of the MVN model. We trained the model on a training set of 99 000 simulated datasets, holding out the remaining 1000 datasets for online validation during training. Finally, to evaluate the performance of the model, we generated 100 independent datasets $\boldsymbol{X}_{test}$ on which we computed the true values of $\mathbb{E}[\boldsymbol{\mu}|\boldsymbol{x}]$, $\mathbb{E}[diag(\boldsymbol{\Sigma})|\boldsymbol{x}]$, $Var[\boldsymbol{\mu}|\boldsymbol{x}]$ and $Var[diag(\boldsymbol{\Sigma})|\boldsymbol{x}]$ by sampling from the true joint posterior as described above. *Figure 2* compares the estimates obtained from the true joint distribution with the estimates obtained from our CNN model.

Table 1 compares the performance of our *DeepInference* approach to the performance of the ABC-RF and ABC-DNN methods as specified by **Algorithm 2** and **Algorithm 3** in terms of root mean squared error (RMSE). We used the same training and test sets for all three methods. For the ABC-RF approach, we computed the sample means, sample variances, the sample covariance and sample median absolute deviations, as well as all possible sums and products with these four metrics as summary statistics. We also used the hyperparameter settings for the random forest regression algorithm recommended by Raynal et al. (2017, p. 16). For the ABC-DNN approach, we created a fully connected neural network with 3 hidden layers, each hidden layer consisting of 100 units, and trained it to minimize the MSE loss between true and predicted parameters (Jiang et al., 2015). For the subsequent rejection sampling scheme, we used the estimates of the trained neural network as summary statistics $\eta(\tilde{\boldsymbol{x}})$ with tolerance level $\epsilon = 0.01$, and Euclidian distance as the distance function $d(\theta, \hat{\theta})$. Posterior moments were computed by considering accepted samples from the approximate posterior distribution $p_\epsilon\big(\theta\big|\eta(\tilde{\boldsymbol{x}})\big)$.

Both graphical and numerical evaluation of the performance of our CNN model suggest a decent approximation of the first two posterior moments of the multivariate normal model. We observe a slight overestimation of the posterior variance $Var[diag(\boldsymbol{\Sigma})|\boldsymbol{x}]$ in the lower and upper regions of the parameter space. Moreover, our method clearly outperforms the ABC-DNN approach, and yields approximations comparable to the ABC-RF approach. However, it is important to note that in this example, it is easy to come up with summary-statistics that capture all the relevant information, thus giving the ABC-RF approach a clear advantage.

Toy example 2: Bayesian regression

We now turn to a slightly more complex example based on Zellner's hierarchical regression model (Raynal et al., 2017; Marin & Robert, 2014, chapter 3). Given a simulated $m\ x\ d$ design matrix of covariates $\boldsymbol{X} = (x_1, x_2, \dots, x_d)$, the hierarchical conjugate model is defined by:

$$\begin{aligned} \sigma^2 &\sim IG(a_0, b_0) \\ \boldsymbol{\beta}|\sigma^2 &\sim \mathcal{N}_d(\mathbf{0}, n\sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}) \\ \boldsymbol{y}|\boldsymbol{\beta}, \sigma^2 &\sim \mathcal{N}_m(\boldsymbol{X}\boldsymbol{\beta}, \sigma^2\boldsymbol{I}) \end{aligned} \tag{12}$$

where $IG(a_0, b_0)$ denotes an inverse gamma distribution with shape parameter a_0 and scale parameter b_0, the regression weights vector $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_d)^T$ follows a d-dimensional normal distribution, and the likelihood follows an m-dimesional normal distribution with an isotropic covariance matrix. As in the previous example, the conjugacy property of the model allows for an analytic computation of the marginal posterior distributions of $\boldsymbol{\beta}$ and σ^2 (see Marin & Robert, 2014, chapter 3).

For this example, we set $d = 2, a_0 = 4, b_0 = 3, m = 100$ and generate a reference table containing 100 000 rows of raw $i.i.d.$ samples $\{\tilde{\boldsymbol{x}}_k^{(i)}\}_{k=1}^{100}$ from the model defined by (12), where each element $\tilde{\boldsymbol{x}}_k^{(i)}$ is an ordered triple containing the two covariates and the outcome generated using the covariates $(x_1^{(i)}, x_2^{(i)}, y^{(i)})_k$. The CNN is then used to predict the corresponding data generating distribution parameters $\{\beta_1^{(i)}, \beta_2^{(i)}, {\sigma^2}^{(i)}\}$. In contrast to the previous example, each kernel in the input layer should learn to encode a particular combination of covariates and outcome (tripe) leading to a set of parameters instead of a single data point.

As in the previous toy example, we trained the model on a training set of 99 000 simulated datasets, holding out the remaining 1000 datasets for validation. Again, for the purpose of evaluation, we generated 100 independent datasets $\boldsymbol{X}_{test}$, on which we computed the true values of $\mathbb{E}[\boldsymbol{\beta}|\boldsymbol{x}]$, $\mathbb{E}[\sigma^2|\boldsymbol{x}]$, $Var[\boldsymbol{\beta}|\boldsymbol{x}]$, and $Var[\sigma^2|\boldsymbol{x}]$. *Figure 3* compares the analytically computed posterior moments with the estimates obtained from our CNN model.

Once again, we observe a very low approximation error (RMSE) of the posterior expectations of the regression weights $\boldsymbol{\beta}$**.** The approximation error of the posterior expectation of the residual variance σ^2 is slightly higher. The approximation of the posterior variance of $\boldsymbol{\beta}$ is also good, whereas we make the observation that the posterior variance of σ^2 is overestimated.

Table 2 compares the performance of our approach to the performance of the ABC-RF and ABC-DNN methods as specified by **Algorithm 2** and **Algorithm 3** in terms of root mean squared error (RMSE). As in the previous example, we used the same training and test sets for all three methods. Hyperparameter settings of the algorithms were identical to those used in the previous example. The summary statistics we computed for the input to the ABC-RF were the same as employed by Raynal et al. (2017): the maximum likelihood estimates of $\boldsymbol{\beta}$, the residual sum of squares, the empirical covariance and correlation between $\boldsymbol{y}$ and $\boldsymbol{X}$, the sample mean,

variance, and median of $\boldsymbol{y}$, resulting in a total of 10 metrics. Network architecture, tolerance level and distance function for the ABC-DNN did not differ from the previous example.

Compared to the ABC-RF approach, our method achieves a slightly better approximation of the posterior expectation and variance of $\boldsymbol{\beta}$, whereas the posterior expectation and variance of σ^2 is better approximated by the ABC-RF approach. Compared to the ABC-DNN approach, our models yields better approximations for all parameters except the variance of σ^2.

Example 3: Leaky Competing Accumulator (LCA) model

As a final example, we apply our approach to a realistic model instance from the cognitive modeling literature – namely, the LCA model (Miletić et al., 2017; Usher & McClelland, 2001). LCA is rooted in the theoretical framework of *sequential sampling models* (SSMs), which attempts to describe perceptual decision-making via the mechanism of noisy evidence accumulation. SSMs assume that, for each decision path, there is a latent process at work, termed an *accumulator*, which samples evidence from sensory input through time. A decision process terminates when an accumulator exceeds a certain threshold of activation, meaning evidence for a given option has culminated in a decision in favor of the given option. The inherent stochasticity of evidence accumulation ensures variation in predicted response times given identical stimuli, and also allows for explaining the occurrence of erroneous responses.

Despite the fact that different researchers have proposed different variants of SSMs (Ratcliff & McKoon, 2008; Voss et al., 2013; Usher & McClelland, 2001), most flavors of SSMs are specified in terms of a finite set of parameters, each interpretable in light of some assumed neurocognitive process or property of such a process. Importantly, all SSMs provide a way to decompose empirical response times into a decision time component and a non-decision time

component, the latter accounting for pre-decisional perceptual (encoding time) and post-decisional motor processes (response execution). Parameters of SSMs are typically estimated from empirical response time distributions via an assumption- and application-dependent estimation procedure (i.e., ML estimation).

In particular, the LCA defines evidence accumulation via the following stochastic differential equation (Miletić et al., 2017):

$$dx_i = \left[I_i - kx_i - \beta \sum_{i' \neq i} x_{i'} \right] \frac{dt}{\tau} + \xi_i \sqrt{\frac{dt}{\tau}}$$
$$x_i = \max(0, x_i) \tag{13}$$

In this model equation, dx_i denotes the change in activation of accumulator i, I_i is the input accumulator i recieves, k is leakage, β is inhibition, $\frac{dt}{\tau}$ denotes the time-step size and ξ represents Gaussian noise.

The LCA model assumes one accumulator process for each option in a decision-making task. Each accumulator competes with all others for reaching a common threshold Z. Observed response times are obtained by an addition of constant non-decision time NDT prior to evidence accumulation and following the reaching of a threshold by an accumulator. The leakage and inhibition parameters are unique to LCA, and they appear to be motivated by neuroscientific observations demonstrating parallels to evidence accumulation on a neural level. Leakage describes the amount of information lost at each time point by an accumulator. Inhibition describes the proportion of an accumulator's activation that suppresses the activation of competing accumulators.

We chose to apply our approach to the LCA model, since the model parameters do not have a known likelihood function. Consequently, most parameter estimation procedures resort to

a simulation-based approach (Miletić et al., 2017). For the following example, we simulate response times from the LCA model and attempt to recover the data-generating parameters. We place the same prior distributions over the model parameters as described in Miletić et al. (2017):

$$\begin{aligned} \Delta I &\sim \mathcal{U}(0.05, 0.3) \\ I &\sim \mathcal{U}(0.8, 1.2) \\ k &\sim \mathcal{U}(1, 8) \\ \beta &\sim \mathcal{U}(1, 8) \\ Z &\sim \mathcal{U}(0.05, 0.25) \\ NDT &\sim \mathcal{U}(0.200, 0.500) \end{aligned} \tag{14}$$

Note, that by choosing uniform priors, the Bayesian approximation of the posterior mean and variance would coincide with the maximum likelihood estimation of the mean and variance, since $p(\boldsymbol{\theta})$ acts solely as a constant multiplicative factor on the likelihood.

For the current example, we fixed the noise variance to $\xi = 0.1$. We generated a reference table with 500 000 simulated datasets, sampling parameters from (14) and generating 1000 response times according to (13). We held out 1000 datasets for validation and generated 500 further independent datasets $\boldsymbol{X}_{test}$ for evaluation. We implemented a deeper CNN architecture consisting of five FC convolutional layers with the following increasing # of kernels in each layer: $(64, 64, 128, 128, 128)$.

With these settings, the model receives as input each simulated response time distribution $\{\tilde{x}_k^{(i)}\}_{k=1}^{1000}$ as well as the information which accumulator "won" the threshold competition, encoded as a one-hot vector. It is then required to predict each of the following LCA parameters: $(\Delta I^{(i)}, I^{(i)}, k^{(i)}, \beta^{(i)}, Z^{(i)}, NDT^{(i)})$. *Figure 4* compares the data-generating parameters with the approximations obtained by the CNN model.

We observe that our model is able to recover the parameters ΔI, NDT, and Z, reasonably well, indexed by low RMSE and high correlation between CNN estimates and actual data-generating parameters. The recovery of the leakage and inhibition parameters, k and β, respectively, is less accurate, but seems to be improvable with the addition of more simulated samples or a deeper architecture with more layers. The CNN is unable to recover the input parameter I, as it appears as if the data did not contain any useful information about this particular parameter.

Table 3 provides a comparison between our approach and two PDA-based methods (maximum likelihood estimation and Bayesian estimation performed via differential evolution Markov-chain Monte Carlo). It is noteworthy, that the relative estimation quality of our approach is similar to that of the PDA-based fitting procedures. However, our approach clearly outperforms PDA with respect to estimating leakage and inhibition, as well as the threshold parameters.

Discussion

Developing effective procedures for estimating parameters of complex models is a vital endeavor of any principled science. In this paper, we proposed a novel method for ABC parameter estimation using CNNs to simultaneously approximate the means and variances of different posterior parameter distributions. We verified the usefulness of our approach on two toy examples and a "real" example from the cognitive modelling literature.

Compared to previous ABC methods rooted in the supervised machine learning approach, our method is truly *end-to-end,* since it requires no computation or selection of summary statistics and no other feature-engineering activities. The possibility of representing raw simulated samples in a grid-like data structure allows us to exploit the advantages of modern

CNNs over traditional DNNs, namely *sparse interactions*, *parameter sharing*, and *equivariant representations* (Goodfellow et al., 2016).

Sparsity of interactions refers to the fact that in place of matrix multiplication, CNNs utilize convolution with a small kernel size, which make CNNs much more efficient in terms of memory usage and computational efficiency. *Parameter sharing* is directly related to the property of sparse interactions. It refers to the fact that parameters of CNN layers are not tied to particular locations of the input, but rather used at every position of the input. Parameter sharing allows the CNN to learn a single set of parameters that capture this particular data point or series of data points. The third property, *equivariance*, is a consequence of the previous two. It implies that shifting the distribution (in the case of time-series), or shuffling the distribution (in the case of $i.i.d.$ sequences), do not dramatically change the representation of the input learned by a particular kernel.

Taken together, these properties make CNNs an excellent choice for implicitly learning summary statistics from simulated samples. In addition, they enable the network to work with *inputs of variable size*, making the size of the simulated samples $\tilde{\boldsymbol{x}}$ independent of the size of the observed data $\boldsymbol{x}$, thus overcoming a further huge limitation of all previous supervised machine learning approaches.

The use of CNNs in the way described in this paper allows for *transfer learning* to enter the field of likelihood-free inference. Transfer learning refers to the process of re-using a pre-trained network for the same or similar task. In other words, once trained, a CNN can be shared across and used as a fast, likelihood-free parameter estimator in a given cognitive domain (i.e. to estimate diffusion model parameters from reaction times in a single forward pass). Moreover, our approach is not confined to instances of models with intractable likelihoods. Even for models

with known likelihood, our approach might offer a good alternative to computationally costly sampling procedures. Moreover, its application as a general *black-box* estimator of hard-to-estimate parameters of various *white box* cognitive models is only limited by the capability of the white-box model to generate representative samples (i.e., a reference table). Needless to say, this transferability is a unique advantage of our deep learning model compared to different architectures.

Throughout our experiments, the approximation of the posterior means was shown to be very accurate. The posterior variance tends to be slightly overestimated – a fact that requires further attention, but, at the very least, implies that our model does not suffer from overconfidence. One possibility for the overestimation of uncertainty is that, due to the finite sample, the estimated $\sigma^2(\tilde{\boldsymbol{x}})$ term fails to capture the true posterior variance. Another possibility is that, since the heteroscedastic loss criterion implies an underlying Gaussian model, deviations of the true posterior from the Gaussian model will cause $\sigma^2(\tilde{\boldsymbol{x}})$ to deviate from the true posterior variance. Nevertheless, the deviations we observed are not dramatic, and further experiments on models with known posteriors should reveal the true extent of the approximation error.

As an attempt to circumvent the shortcomings of ABC rejection algorithms (curse of dimensionality, tolerance level tuning and selection of distance function), our approach does not provide a way to sample from the full joint posterior distribution, but rather summarizes the distribution with its first two moments. However, if one wishes, one can still use the estimates obtained by the CNN as summary statistics in a rejection procedure (Jiang et al., 2015), with the added benefit of having the approximation of the second posterior moment beside the first moment. Furthermore, a full approximate distribution can be computed either by simply sampling from a normal distribution parameterized by the predicted means and variances or by

using this distribution as a more informative prior distribution in a usual rejection sampling approach to drastically improve the efficiency.

It is also straightforward to extend our approach to the problem of ABC model selection. In machine-learning terms, instead of performing regression, we simply need to frame the problem as a classification task, in which the network is trained to classify the label of the model used to generate a particular data sample (see Pudlo et al., 2015 for a random-forest based approach). In a future work, we intend to extend our open-source ABC software ABrox (see Mertens et al., 2018) with *DeepInference* capabilities.

Conclusion

In this paper, we proposed a method for reliable likelihood-free inference using fully convolutional networks to automatically learn optimal summary statistics from raw samples. We obtained accurate estimates of the first two moments of the posterior parameter distribution on two toy examples and a cognitive model of choice reaction times. Furthermore, our method exhibits the following unique features as compared to previous supervised learning approaches to ABC:

1. An end-to-end architecture eliminating the need to manually hand-craft summary statistics of the data distribution;
2. Fast training due to simultaneous parameter estimation;
3. Re-usable models due to the possibility of working with variable-sized inputs;

Further research should test the utility of our approach on various real-world examples from the cognitive modelling literature, which require non-standard solutions or are too expensive to compute with sampling based procedures. Another future avenue for our approach is likelihood-

free model choice, which would require a different optimization procedure for “labelling” the data with the correct model and quantifying uncertainty at the same time.

References

Blum, M. G. (2010). Approximate Bayesian computation: a nonparametric perspective. *Journal of the American Statistical Association, 105*(491), 1178-1187.

Chollet, F. (2017). *Deep learning with python*: Manning Publications Co.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems, 2*(4), 303-314.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian data analysis*: CRC press.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1): MIT press Cambridge.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks, 4*(2), 251-257.

Jiang, B., Wu, T.-y., Zheng, C., & Wong, W. H. (2015). Learning summary statistic for approximate Bayesian computation via deep neural network. *arXiv preprint arXiv:1510.02175*.

Kendall, A., & Gal, Y. (2017). *What uncertainties do we need in bayesian deep learning for computer vision?* Paper presented at the Advances in Neural Information Processing Systems.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lee, M. D. (2008). Three case studies in the Bayesian analysis of cognitive models. *Psychonomic Bulletin & Review, 15*(1), 1-15.

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

Marin, J.-M., Raynal, L., Pudlo, P., Ribatet, M., & Robert, C. P. (2016). ABC random forests for Bayesian parameter inference. *arXiv preprint arXiv:1605.05537*.

Marin, J.-M., & Robert, C. P. (2014). *Bayesian essentials with R* (Vol. 48): Springer.

Miletić, S., Turner, B. M., Forstmann, B. U., & van Maanen, L. (2017). Parameter recovery for the leaky competing accumulator model. *Journal of Mathematical Psychology, 76*, 25-50.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., & Feldman, M. W. (1999). Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular biology and evolution, 16*(12), 1791-1798.

Pudlo, P., Marin, J.-M., Estoup, A., Cornuet, J.-M., Gautier, M., & Robert, C. P. (2015). Reliable ABC model choice via random forests. *Bioinformatics, 32*(6), 859-866.

Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: theory and data for two-choice decision tasks. *Neural computation, 20*(4), 873-922.

Roy, M.-H., & Larocque, D. (2012). Robustness of random forests for regression. *Journal of Nonparametric Statistics, 24*(4), 993-1006.

Schoot, R., Kaplan, D., Denissen, J., Asendorpf, J. B., Neyer, F. J., & Aken, M. A. (2014). A gentle introduction to Bayesian analysis: Applications to developmental research. *Child development, 85*(3), 842-860.

Sunnåker, M., Busetto, A. G., Numminen, E., Corander, J., Foll, M., & Dessimoz, C. (2013). Approximate bayesian computation. *PLoS computational biology, 9*(1), e1002803.

Tavaré, S., Balding, D. J., Griffiths, R. C., & Donnelly, P. (1997). Inferring coalescence times from DNA sequence data. *Genetics, 145*(2), 505-518.

Toni, T., Welch, D., Strelkowa, N., Ipsen, A., & Stumpf, M. P. (2009). Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface, 6*(31), 187-202.

Turner, B. M., & Sederberg, P. B. (2014). A generalized, likelihood-free method for posterior estimation. *Psychonomic bulletin & review, 21*(2), 227-250.

Turner, B. M., & Van Zandt, T. (2012). A tutorial on approximate Bayesian computation. *Journal of Mathematical Psychology, 56*(2), 69-85.

Usher, M., & McClelland, J. L. (2001). The time course of perceptual choice: the leaky, competing accumulator model. *Psychological review, 108*(3), 550.

Voss, A., Nagler, M., & Lerche, V. (2013). Diffusion models in experimental psychology: A practical introduction. *Experimental psychology, 60*(6), 385.

Tables

Table 1

Comparison of parameter estimation methods on the multivariate normal toy example with the Root mean squared error (RMSE) metric.

Method	$\mathbb{E}[\mu_1\|\boldsymbol{x}]$	$\mathbb{E}[\mu_2\|\boldsymbol{x}]$	$\mathbb{E}[\sigma_1^2\|\boldsymbol{x}]$	$\mathbb{E}[\sigma_2^2\|\boldsymbol{x}]$	$Var[\mu_1\|\boldsymbol{x}]$	$Var[\mu_2\|\boldsymbol{x}]$	$Var[\sigma_1^2\|\boldsymbol{x}]$	$Var[\sigma_2^2\|\boldsymbol{x}]$	$\mathbb{E}[\mu_1\|\boldsymbol{x}]$
DeepInference	0.013	0.029	0.056	0.046	**0.001**	**0.001**	0.066	0.048	0.013
ABC-DNN	0.046	0.047	0.361	0.334	0.009	0.1	0.857	0.368	0.046
ABC-RF	0.013	**0.014**	**0.028**	**0.015**	0.002	0.002	**0.012**	**0.013**	0.013

Note: RMSE scores given in bold highlight the best approximation for each parameter (i.e., each column).

Table 2

Comparison of parameter estimation methods on the Bayesian regression toy example with the Root mean squared error (RMSE) metric.

Method	$\mathbb{E}[\beta_1\|\boldsymbol{x}]$	$\mathbb{E}[\beta_2\|\boldsymbol{x}]$	$\mathbb{E}[\sigma^2\|\boldsymbol{x}]$	$Var[\beta_1\|\boldsymbol{x}]$	$Var[\beta_2\|\boldsymbol{x}]$	$Var[\sigma^2\|\boldsymbol{x}]$
DeepInference	**0.062**	**0.059**	0.148	**0.021**	**0.046**	0.219
ABC-DNN	0.456	0.674	0.369	0.232	0.203	0.076
ABC-RF	0.093	0.087	**0.062**	0.055	0.099	**0.037**

Note: RMSE scores given in bold highlight the best approximation for each parameter (i.e., each column).

Table 3

Comparison of parameter estimation methods on the LCA model with the Root mean squared error (RMSE) metric and Pearson's correlation coefficient *r*.

	RMSE						Correlation *r*					
Method	ΔI	I	k	β	Z	NDT	ΔI	I	k	β	Z	NDT
DeepInference	**0.011**	**0.113**	**1.309**	**1.586**	**0.02**	**0.02**	**0.99**	**0.11**	**0.73**	**0.59**	**0.9**	**0.97**
ML-PDA	0.024	0.508	2.487	3.216	0.045	0.044	0.96	0.07	0.4	0.27	0.71	0.91
DE-MCMC	0.021	1.073	2.916	4.742	0.03	**0.02**	N/A	N/A	N/A	N/A	N/A	N/A

Note: RMSEs and correlations for the ML-PDA and DE-MCMC methods with sample of size $N = 1000$ are reproduced from Miletić et al. (2017).

Convolutional neural network architecture

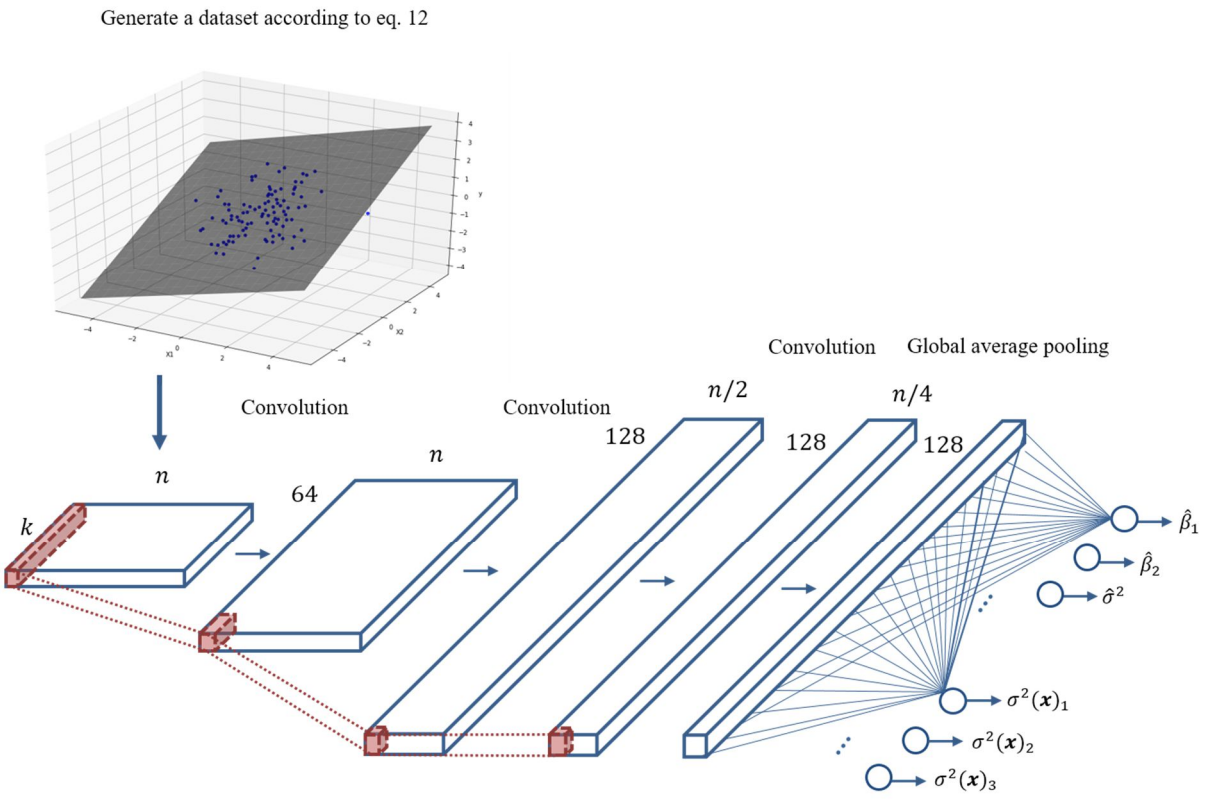

Figure 1. Graphical illustration of the CNN architecture used throughout this paper. The input represents data sets generated from the Bayesian regression model (toy example 2), organized as an $n \; x \; k$ grid, with k equal to the number of predictors (in this case 2) + the outcome, and n equal to the number of simulated samples. The CNN then performs a series of non-linear transformations, each layer applying convolution as a linear transformation followed by a rectified linear (ReLU) non-linearity, which is typical of modern convolutional architectures. The last layer then outputs the estimates of the posteriors mean and variance (one tuple of the form $\left(\widehat{\theta},\, \sigma^2(\widetilde{\boldsymbol{x}})\right)$ for each model parameter). The loss is then evaluated via eq. 8.

Multivariate normal results

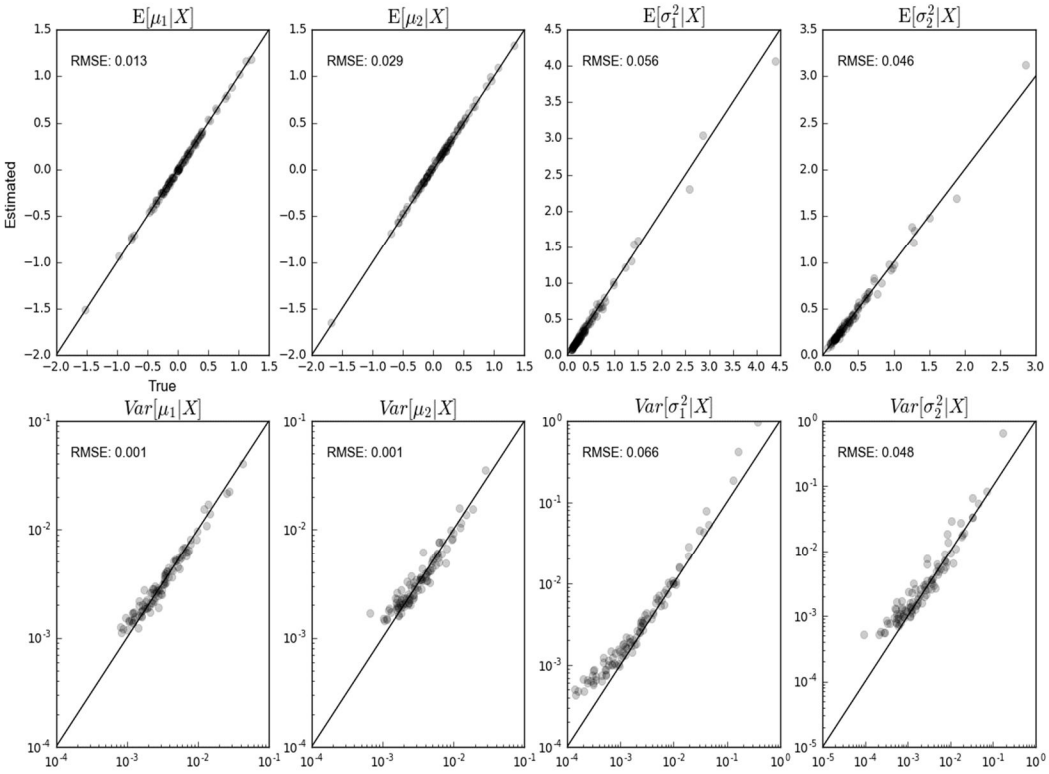

Figure 2. Comparison of the estimates obtained by the CNN model with the true parameter values on the multivariate normal toy example. True parameter values are plotted on the x-axis, and corresponding estimates on the y-axis. The first row depicts the posterior expectations of the model parameters. Posterior variances are depicted in the second row. Root mean squared error (RMSE) is also given for each estimate.

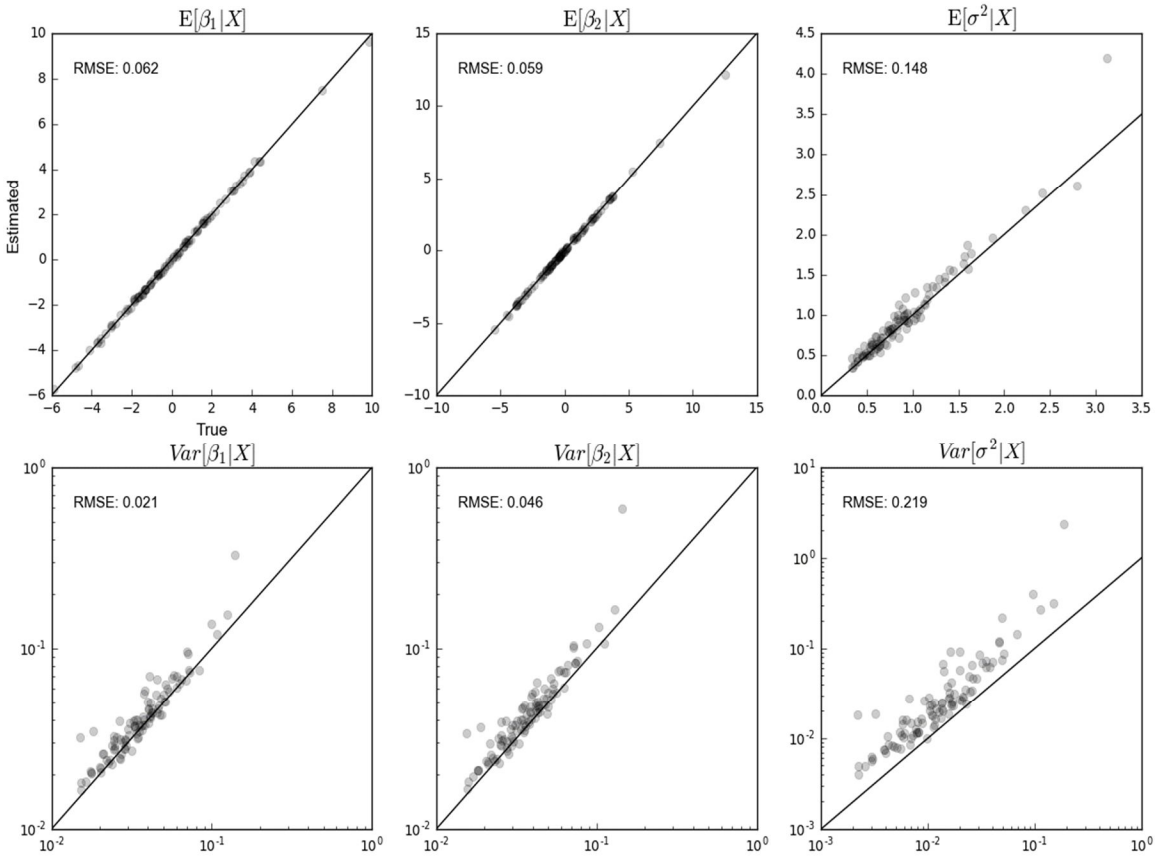

Figure 3. Comparison of the estimates obtained by the CNN model with the true posterior parameter values on the Bayesian linear regression example. True parameter values are plotted on the x-axis, and corresponding estimates on the y-axis. The first row depicts the posterior expectations of the model parameters. Posterior variances are depicted in the second row. Root mean squared error (RMSE) is also given for each estimate.

LCA example results

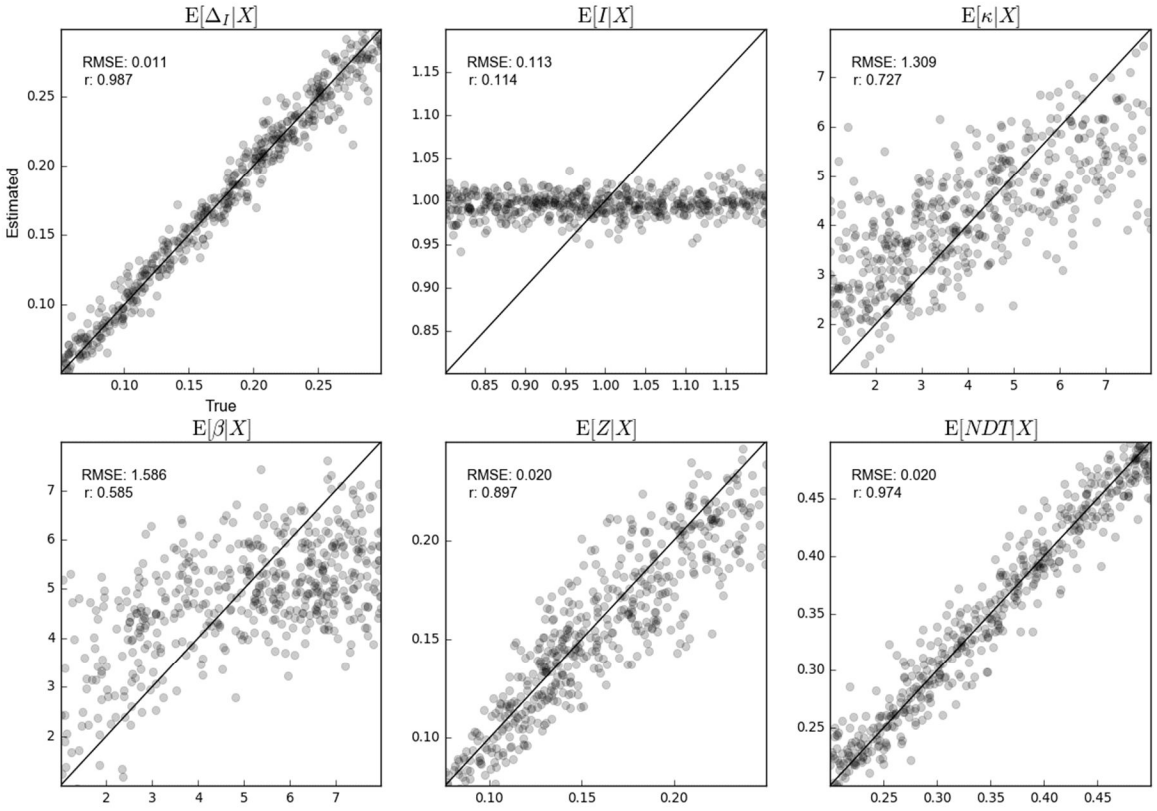

Figure 4. Depiction of parameter recovery of the LCA model. Values of the data-generating parameters are plotted on the x-axis, and corresponding CNN estimates on the y-axis. Both root mean squared error (RMSE) and Pearson's correlation coefficient (*r*) are also reported.