Doctoral thesis submitted to
the Faculty of Behavioural and Cultural Studies
Heidelberg University
in partial fulfillment of the requirements of the degree of
Doctor of Philosophy (Dr. phil.)
in Psychology

Title of the publication-based thesis

Deep Learning methods for likelihood-free inference –
approximating the posterior distribution with convolutional neural networks

presented by
Ulf Kai Mertens, M.Sc.

year of submission
2019

Dean: Prof. Dr. Dirk Hagemann
Advisor: Prof. Dr. Andreas Voss

Acknowledgments

My time as a PhD student can best be described as a long hiking adventure with varying levels of difficulty, various changes of weather and a few involuntary stops along the way. Here, I would like to thank the people I have met during my travels and who accompanied me along the way.

First of all, I would like to thank my supervisor Prof. Dr. Andreas Voss who not only made this journey possible in the first place, but also let me take the trails I found the most beautiful even if the main road was an opportunity. Although my choices turned out to be detours or even deadlocks from time to time, he nevertheless always trusted in me not losing sight of the destination. Fortunately, the exploration of new paths was not only bad as we were able to see a few nice and untouched spots that would otherwise have stayed hidden.

Shortly after I started the hike, I met a new companion, Alica Bucher, who thankfully was on the same journey. We started out being really good friends and Alica was always there for me when I was fearful not to make it to the end. We also helped each other along the way by sharing all of our concerns, fears and pleasant anticipations about the remaining route. I am utterly grateful we decided to continue to walk side by side on journeys yet to come.

One day, I decided to take a rather difficult path over a large mountain that had not been explored a lot but I thought it was worthwhile. After some time though, I was about to give up when I heard about a Bulgarian man called Stefan Radev who was very passionate about hill climbing. I immediately started to search for this guy and luckily found him soon after. Stefan helped me get to the top of the mountain but moreover, it turned out he was as enthusiastic about hill climbing as I am. We talked for countless hours and planned several new trips in the future.

Of course, these were not the only people I met along the way and there were many others I enjoyed spending time with who made the journey as a whole a real pleasure: Veronika Lerche, Andreas Neubauer, Mischa von Krause, Marie Wieschen, Jan Rummel, Lena Steindorf, Anne Bülow, Julia Karl, Marvin Schmitt, and Yannick Roos.

Table of Contents

List of Scientific Publications of the Publication-Based Dissertation

Manuscript 1

Mertens, U. K., 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. https://doi.org/10.1371/journal.pone.0193981

Manuscript 2

Radev, S., Mertens, U. K., Voss, A., & Köthe, U. (2019). Towards end-to-end likelihood-free inference with convolutional neural networks. *British Journal of Mathematical and Statistical Psychology.* Manuscript accepted for publication.

Manuscript 3

Voss, A., Mertens, U. K., & Radev, S. *Learning the Likelihood: Using DeepInference for the Estimation of Diffusion-Model and Lévy Flight Parameters.* Manuscript under review.

1 Introduction

The precise explanation and prediction of processes that underlie our cognition is the ultimate goal in psychological research. Yet, mental characteristics of the human mind are far too complex to be described in full detail. Hence, researchers have to make simplified assumptions about particular aspects and incorporate them within a model that best mimics the dynamics of the real process. Such an endeavor forces researchers to be as precise as possible in formulating the respective assumptions underlying the model. Only if an exact description of process is made, will the model be falsifiable and research can make progress. The ideal approach to ensure that the mentioned requirements are met is a mathematical specification of the inner workings of the model/theory. Such a mathematical model most often consists of several equations and corresponding parameters where each equation/function describes the process of how a cognitive process emerges (or a behavior is generated). Ideally, in psychology, all associated parameters of a mathematical model should be meaningful and interpretable in that they can be mapped to psychological or neurocognitive processes.

Besides the precise description within a mathematical framework, a crucial requirement for gaining insights about complex human behavior, experiences or cognitive processes is the ability to assess how mathematically formulated assumptions are in line with observed data. Measuring the goodness-of-fit of a model to data, expressed as the plausibility of certain characteristics of a statistical model, i.e., its parameters, with respect to some data at hand, plays one of the most important roles throughout the field of statistical inference. The function that maps each possible parameter value to a plausibility score is commonly denoted as the *likelihood*. In frequentist statistics, the likelihood builds the foundation for the method of maximum likelihood estimation, a standard approach for parameter estimation to find the parameters of a model that are the most likely, given some observed data. In the field of Bayesian statistics, the likelihood, represented as a conditional distribution of the observed data given the parameters, is essential to update prior beliefs about parameters in light of new data. Since the likelihood is deeply rooted in some of the most widely used statistical procedures, it is hard to imagine doing inferences if there were no likelihood. However, in the constant strive for a better understanding of psychological phenomena, existing statistical models are adapted and extended. As models become more and more diverse and involved, there is no guarantee that the likelihood function can still be expressed which in turn leads to a demand for more sophisticated statistical analyses.

As a result, researchers from a variety of disciplines are increasingly faced with scenarios where no proper likelihood function is available or it is too complex and inference becomes infeasible due to limited computational resources.

In this thesis, I will introduce the reader to an area of research, so called approximate Bayesian computation (ABC), also known as likelihood-free inference, that attempts to circumvent the problem of a missing likelihood function by approximating it. At the time of writing, likelihood-free inference still plays a rather secondary role in psychological research. However, there has been increased interest in applying ABC techniques in recent years (Palestro, Sederberg, Osth, van Zandt, & Turner, 2018). For instance, Turner, Dennis, and Van Zandt (2013) applied ABC to two influential episodic memory models, namely the Bind Cue Decide Model of Episodic Memory (BCDMEM; Dennis & Humphreys, 2001) and the Retrieving Effectively from Memory model (REM; Shiffrin & Steyvers, 1997). Due to missing neuroscientific knowledge, both models are constrained in that no mathematical statements relating parameters to dependent variables can be made (Turner, Dennis, & Van Zandt; 2013). As a result, both models do not have a likelihood function and it is only possible to simulate data from these models, given a set of parameter values.

Research on likelihood-free inference has come up with a range of clever ideas to approximate the likelihood or to replace it with concepts of similar meaning. However, bypassing the likelihood has not been feasible without paying the price of making several important decisions regarding one's own data. Even worse, there is little if no guarantee those decisions have been the rights ones.

My major contribution to this area of research is the development of a machine learning algorithm, which we named *DeepInference,* that allows to overcome a few of the most important limitations of existing procedures, while at the same time leading to state-of-the-art results in terms of accuracy of parameter estimation. In this thesis, I will furthermore show how *DeepInference* can be applied to one of the most widely known sequential sampling models, the drift diffusion model (DDM; Ratcliff, & McKoon, 2008; Voss, Nagler, & Lerche, 2013). Although the DDM is only one specific model especially targeted at the analysis of response time data for binary decision tasks, it is perfectly suited as an example that reinforces the arguments above for the increasing importance of likelihood-free inference. Whereas researchers interested in estimating parameters of the classic DDM have a likelihood function available and are hence able to use common techniques, no statements can be made about the likelihood for interesting extensions of the DDM, such as the Lévy flight model

(Voss A., Lerche, Mertens, & Voss, J., 2019). The example clarifies how new advances in likelihood-free inference might support or even facilitate new insights that would otherwise stay hidden. The Lévy flight model, an extension of the DDM which stretches some familiar assumptions of the DDM, will be the final application of *DeepInference.*

My thesis will be organized as follows: First, I will briefly explain the main ideas of Bayesian statistics, which then allow me to outline the basic principles of approximate Bayesian computation (throughout this thesis abbreviated as *ABC)*, by explaining one of the first and simplest algorithms, the so-called ABC rejection algorithm (Pritchard, Seielstad, Perze-Lezaun, & Feldman, 1999; Tavaré, 1997). The ideas will be presented in a way to ease the understanding of more advanced techniques and algorithms, presented in later sections of this thesis, and to highlight their commonalities. Next, in order to lead the reader to the research conducted in my first Manuscript (Manuscript 1), I will put aside the area of parameter estimation and focus on ABC techniques for hypothesis testing. The advantages and capabilities of *ABrox*, a graphical user interface I developed that is especially suited for hypothesis testing in an ABC context, will be highlighted. Before diving into the details of *DeepInference* (explained in detail in Manuscript 2), I want to outline what machine learning is and how it already contributed to the field of ABC. Following this section, I will present results from two parameter recovery studies (see Manuscript 3) in which we applied *DeepInference* to the drift diffusion model and the Lévy flight model. The thesis will be concluded by a discussion of some interesting open research questions/applications which I hope will be addressed in future research projects.

2 The basics of approximate Bayesian computation

The general idea of Bayesian statistics (see e.g., Gelman, 2013) is to treat the parameters of a statistical model as random variables and the observed data as fixed. Hence, by placing probability distributions over the parameters, the inherent uncertainty of statistical inference is quantified. Assuming a single parameter, the researcher starts with a *prior* distribution thereof, i.e., a probability distribution representing some prior belief about the values of the parameter. Depending on the current stage of knowledge, one might either start with strong prior beliefs (e.g., the probability of a coin landing heads is ~50%) or no information at all. After having collected some data, these prior beliefs are updated using the *likelihood*, i.e., the plausibility of the parameter values given the data. It is important to mention a slightly different interpretation of the likelihood within the Bayesian framework compared to the frequentist view as being a conditional density of the data given the parameters, often denoted as $p(\boldsymbol{x}|\boldsymbol{\theta})$.

The *posterior* then is the belief about the values of the parameter, after the prior beliefs have been weighted with their corresponding plausibility (likelihood). By looking at the *posterior* distribution, it not only becomes evident which value is the most probable one, but at the same time, the distribution contains information about the uncertainty of the parameter. Only in a Bayesian context is it correct to state that values of a parameter lie within a certain region with a prespecified probability (Hoekstra, Morey, Rouder, & Wagenmakers, 2014). The mathematical formulation of these ideas is incorporated in Bayes' rule:

$$p(\boldsymbol{\theta}|\boldsymbol{x}, M) = \frac{p(\boldsymbol{x}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M)}{\int p(\boldsymbol{x}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M)d\boldsymbol{\theta}} \quad (1)$$

In the above equation, M is a specific statistical model with parameter vector $\boldsymbol{\theta}$. Any collected dataset is represented by $\boldsymbol{x}$. Note that being explicit about the role of a model helps in clarifying Bayesian model comparison as explained in a later section of the thesis. In equation 1, $p(\boldsymbol{x}|\boldsymbol{\theta}, M)$ is the *likelihood*, in other contexts explicitly referred to as a function of the parameters ($f(\boldsymbol{\theta}|\boldsymbol{x})$). $p(\boldsymbol{\theta}|M)$ is the (joint) prior distribution and $p(\boldsymbol{\theta}|\boldsymbol{x}, M)$ the posterior distribution expressed as a conditional distribution of the parameter vector $\boldsymbol{\theta}$, given model M and dataset $\boldsymbol{x}$. The denominator of Bayes' rule, $\int p(\boldsymbol{x}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M)d\boldsymbol{\theta}$, is commonly known as *the normalizing constant* or *marginal likelihood* and is sometimes written in short form as simply $p(\boldsymbol{x}|M)$. The *marginal likelihood* is a scalar that renormalizes the product of *prior* and

likelihood so that the *posterior* is a proper probability distribution integrating to 1. Since the marginal likelihood is hard to compute when the parameter space is high-dimensional, modern algorithms such as Markov Chain Monte Carlo (MCMC), allow to make inferences without the need to calculate it (Gilks, Richardson, & Spiegelhalter, 1996). Thus, the equation above can be simplified by writing:

$$p(\boldsymbol{\theta}|\boldsymbol{x}, M) \propto p(\boldsymbol{x}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M) \quad (2)$$

If the likelihood function is not available as a weighting or updating factor of the prior beliefs about $\boldsymbol{\theta}$, other ways have to be found to assess their plausibility in light of data. Fortunately, most models can be seen as a set of assumptions that try to imitate a stochastic process from which the collected data might have been generated from, i.e., the *data-generating process*. Thus, many existing statistical models can also be used as functions to generate new data $\tilde{\boldsymbol{x}}$, conditioned on prespecified parameters.

Assuming that such a *generative model* $q(\cdot\,|\boldsymbol{\theta})$ is able to capture the true data-generating process, one would assume that there exists a sample of parameter values $\boldsymbol{\theta}$ such that the resulting generated data are identical to the observed data. In this context, the *likelihood* is the probability of the aforementioned event happening ($\Pr(\boldsymbol{x} = \tilde{\boldsymbol{x}})$). This probability is approximated via an iterative process, known as the rejection sampling algorithm (Rubin, 1984):

Algorithm 1. Exact rejection algorithm.

for $i \leftarrow 1$ to N
  repeat
    Sample $\boldsymbol{\theta} \sim p(\cdot)$
    Simulate $\tilde{\boldsymbol{x}} \sim q(\cdot\,|\boldsymbol{\theta})$
  until $\tilde{\boldsymbol{x}} = \boldsymbol{x}$
  $\boldsymbol{\theta}^{(i)} \leftarrow \boldsymbol{\theta}$
end for

For every iteration, one samples $\boldsymbol{\theta}$ from the prior distribution $p(\cdot)$ and, using the sampled $\boldsymbol{\theta}$, simulates data $\tilde{\boldsymbol{x}}$ according to $q(\cdot\,|\boldsymbol{\theta})$. As long as $\tilde{\boldsymbol{x}}$ is not equal to $\boldsymbol{x}$, the two steps (sampling and simulating) are repeated. Only if $\tilde{\boldsymbol{x}}$ is exactly equal to $\boldsymbol{x}$, the process stops and the currently sampled $\boldsymbol{\theta}$ is stored. The whole procedure continues until a total of N samples have

been stored. The N kept samples from the rejection algorithm are samples from the exact posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x}, M)$.

The presented algorithm, in the same manner as a usual likelihood function, 'awards' parameter values generating data resembling the observed data with high plausibility scores, and downgrades others by assigning lower scores, respectively. While **Algorithm 1** makes intuitive sense, it is not practical for high-dimensional continuous $\boldsymbol{x}$, since the probability that simulated data will be exactly equal to the observed data decreases as the dimensionality of $\boldsymbol{x}$ increases.

An ABC version of Algorithm 1 (see **Algorithm 2**) is able to approximate the true posterior by introducing three hyperparameters. First, instead of keeping samples only if $\tilde{\boldsymbol{x}} = \boldsymbol{x}$, a distance function $d(\cdot)$ has to be chosen to measure the similarity of $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x}$. Secondly, a threshold ϵ is necessary for deciding how many samples to keep (which distances are accepted or rejected). Lastly, a function $\eta(\cdot)$ to compute summary statistics of the raw data has to be picked. For instance, if $\boldsymbol{x}$ can be represented as a vector of response times, one might choose common statistical values such as the mean, standard deviation, kurtosis and skewness as summary statistics. The reason for the importance of summary statistics is as follows: If raw, potentially high-dimensional data are directly compared (instead of the comparing summary statistics), the chance of random deviations between the raw observed data and the raw simulated data is higher. To account for this fact, a large threshold ϵ has to be picked for the algorithm to stay reasonably efficient. The necessity to increase the threshold ϵ however causes many accepted datasets to be accepted which are actually different to the observed data and the approximation of the posterior distribution becomes worse. The ABC rejection algorithm reads as follows (see Tavaré, 1997):

Algorithm 2. ABC rejection algorithm.

for $i \leftarrow 1$ to N

  repeat

    Sample $\boldsymbol{\theta} \sim p(\cdot)$

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

  until $\boldsymbol{d}\big(\boldsymbol{\eta}(\tilde{\boldsymbol{x}}), \boldsymbol{\eta}(\boldsymbol{x})\big) \leq \boldsymbol{\epsilon}$

  $\boldsymbol{\theta}^{(i)} \leftarrow \boldsymbol{\theta}$

end for

Algorithm 2 generates samples from an approximate posterior distribution $p_\epsilon(\boldsymbol{\theta}|\boldsymbol{\eta}(\tilde{\boldsymbol{x}}), M)$. Setting ϵ to zero and using as $\eta(\cdot)$ the identity function, **Algorithm 2** reduces to **Algorithm 1**. An increase in the threshold ϵ makes the resulting posterior less accurate but causes a decrease in time required to obtain N samples. Besides the right choice of ϵ, choosing summary statistics that are sufficient, i.e., reduce the dimensionality without loss of information about the parameters, is a crucial aspect with which all ABC algorithms have to deal. In manuscript 2 of this thesis, I will present a way to overcome this serious issue of having to compute summary statistics using an algorithm from machine learning, namely *convolutional neural networks* (LeCun et al., 1989).

The ideas presented so far deal with the problem of *parameter estimation* when no likelihood is available. In psychological research though, often the interest lies in comparing models and quantifying which model, or hypothesis, is more probable given some collected data. From a psychologist's perspective, being able to easily conduct Bayesian hypothesis testing in a likelihood-free setting is at least of equal importance as parameter estimation. The first manuscript of this thesis therefore addressed the topic of ABC model comparison and introduced an easy-to-use interface for conducting such analyses.

3 ABrox – a python module for approximate Bayesian computation (Manuscript 1)

When interested in Bayesian model comparison, the goal is to compute the ratio of posterior probabilities of two competing models (M_1 and M_2), given the data $\boldsymbol{x}$. The corresponding Bayes formula for each of the models is similar to Equation 1:

$$p(M_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|M_k)p(M_k)}{\int p(\boldsymbol{x}|M_k)p(M_k)dM} \tag{3}$$

In this thesis, I explicitly distinguish between the denominator term of Equation 1 and the denominator of Equation 3, which is why I refer to the latter as the *evidence.* The *evidence* is the likelihood averaged over all models under consideration (in contrast to the *marginal likelihood* which is the likelihood of a specific model averaged over its parameters). The ratio of posterior model probabilities is the ratio of the right-hand sides of Equation 3. Since the *evidence* terms cancel out, we are left with:

$$\frac{p(M_1|\boldsymbol{x})}{p(M_2|\boldsymbol{x})} = \frac{p(\boldsymbol{x}|M_1)}{p(\boldsymbol{x}|M_2)} * \frac{p(M_1)}{p(M_2)} \tag{4}$$

In the above expression, $p(M_1|\boldsymbol{x})/p(M_2|\boldsymbol{x})$ is the ratio of posterior model probabilities, $p(M_1)/p(M_2)$ is the ratio of prior model probabilities and $p(\boldsymbol{x}|M_1)/p(\boldsymbol{x}|M_2)$ is the ratio of marginal likelihoods known as the *Bayes factor (BF)* (Jeffreys, 1961; Kass & Raftery, 1995). By taking a look at Equation 1 again, one immediately notices that the computation of the Bayes factor requires the calculation of *marginal likelihoods*. If researchers do not intend to favor any of two models or hypotheses a priori, which is usually the case, the Bayes factor is equal to the ratio of posterior model probabilities, the actual term of interest. The Bayes factor is interpreted as the evidence of one model relative to the other. For instance, a BF_{12} of 3 states that model M_1 is three times more likely than M_2 whereas a BF_{12} of 0.2 indicates that M_2 is 1/0.2, i.e., 5 times as likely as model M_1. For some common statistical procedures, an analytic solution for the Bayes factor can be derived (see e.g., Rouder, Speckman, Sun, Morey, & Iverson, 2009). However, for more complex models, the Bayes factor has to be approximated via Monte Carlo techniques (Diciccio, Kass, Raftery, & Wasserman, 1997; Gronau et al, 2017). The aim of Manuscript 1 was to announce *ABrox* as a graphical user interface for ABC and to investigate how well an approximation of the Bayes factor due to a missing likelihood function was able to approach an exact Bayes factor. For all comparisons within Manuscript 1, I used an adaptation of the above presented ABC rejection algorithm, illustrated below:

Algorithm 3. ABC rejection algorithm for model comparison.

for $i \leftarrow 1$ to N

  repeat

    Sample $M \sim p(M)$

    Sample $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|M)$

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

  until $d\big(\eta(\tilde{\boldsymbol{x}}), \eta(\boldsymbol{x})\big) \leq \epsilon$

  $M^{(i)} \leftarrow M$

end for

The result of **Algorithm 3** is an array storing the model indices that led to a distance smaller or equal to ϵ, e.g., $\left(M_1^{(1)}, M_2^{(2)}, M_2^{(3)}, M_2^{(4)}, \ldots, M_1^{(N)}\right)$. The number of occurrences of M_1 divided by the number of occurrences of M_2 is the approximation of the Bayes factor (throughout this thesis denoted as ABC Bayes factor). As an example, if 75 out of N=100 model indices belong to M_1 and hence 25 to M_2, the resulting Bayes factor is 75/25 = 3.

As a first example, I was interested in the comparison of both BF versions (approximated and exact) for the independent samples *t*-test. Rouder, Speckman, Sun, Morey, & Iverson (2009) offer a 'default' Bayes factor where the parameter of interest is the effect size Cohen's *d*. For one model, the alternative hypothesis ($\mathcal{H}_1$), Cohen's *d* is allowed to differ from zero by using a Cauchy distribution with a scale of 0.707 as the prior distribution. The null hypothesis ($\mathcal{H}_0$), in contrast, assumes Cohen's *d* to be exactly zero (see details in the Manuscript). I implemented a version of **Algorithm 3** using the same setup as Rouder et al. (2009). For the comparison, we simulated observed data which varied in the size of the effect (Cohen's *d*) and the total sample size. For every such constructed dataset, we calculated the default Bayes factor and the ABC Bayes factor. Overall, results indicated that the ABC Bayes factor of the independent samples *t*-test was highly correlated with the default Bayes factor.

In manuscript 1, I wanted to go further and demonstrate the usefulness of ABC Bayes factors on two other tests/models for which no easy way to compute Bayes factors was available at the time of writing. The first of the two examples was the Levene-Test which tests whether the variances of two samples are homogeneous. Here, since no exact Bayes factor could be computed, we compared the ABC Bayes factor with the *p* values extracted

from the frequentist version of the Levene test. Results showed that in general, the ABC Bayes factor and the *p* value led to the same decision (retaining or discarding $\mathcal{H}_0$), yet for small sample sizes, there was also a non-neglectable amount of distinct decisions.

For the last example in the paper, I was interested in a more realistic and hence more complex class of models, multinomial processing tree models (MPT; Riefer & Batchelder, 1988). I picked two prominent MPT models from the area of research on stereotypes. Each model makes different predictions about the roles of automaticity and control in the so-called weapon misidentification task (Conrey, Sherman, Gawronski, Hugenberg, & Groom, 2005; Bishara & Payne, 2009). Whereas the *Process Dissociation Model* (Jacoby, 1991; Payne, 2001) assumes controlled processing being a more important factor than automaticity, the *Stroop Model* (Lindsay, & Jacoby, 1994) supposes the reverse order. Without elaborating any further on the characteristics of both models (see Manuscript 1 for more details), the ABC Bayes factor seemed to be a promising way for comparing MPT models, even computed with the most basic algorithm such as **Algorithm 3**. Assuming a reasonable choice of the prior distributions for all parameters, results indicated that the ABC Bayes factor favored the model for which the observed data had been simulated from.

The goal of manuscript 1 was not only to draw attention to the flexibility and potential of Bayes factors in a likelihood-free setting as mentioned above, but at the same time to introduce *ABrox*, a user-friendly graphical user interface for approximate Bayesian computation. *ABrox* was mainly developed to help researchers apply the presented techniques in their own labs with as few obstacles as possible. Although quite a few other software packages are available (see e.g., Csillery, Francois, & Blum, 2011; Liepe et al., 2010), *ABrox*, to the best of my knowledge, is the first one with a focus on model comparison that aims to be applicable in different research areas. During my research on open-source software for ABC, I was surprised that many tools were specifically designed for only a certain class of models, e.g., models encountered in Systems biology. These tools, while extremely helpful for users working in the field, were barely or not at all applicable in other domains. With *ABrox*, I tried to build a tool that can be used by anyone, regardless of one's research background, by both experienced researchers and people new to the field of ABC or programming in general. While more versed users can apply the syntax mode of *ABrox*, others might prefer the GUI mode. I hope many will profit from ABrox making ABC techniques much more accessible than they would have been before.

After finishing my work on Manuscript 1, I focused my research less on applications of existing algorithms, but on the development of new ways to overcome some longstanding deficits of all ABC algorithms, especially the necessity to choose and compute summary statistics. First, I will briefly introduce the field of machine learning and neural networks. Then I present the characteristics of *DeepInference*, a specific algorithm from the field of deep learning. The following section will also introduce the terminology used throughout this thesis.

4 The basics of machine learning

Quantitative research methods applied in psychological research and machine learning are both forms of applied statistics with many commonalities. Yet, their focus is rather different. Researchers in the field of empirical psychology follow an *explanatory* approach (trying to explain behavior) by leveraging the concept of significance (Yarkoni, & Westfall, 2017). In machine learning, however, the focus lies predominantly on predictive accuracy and generalization of results. In machine learning, the goal is to build machines, i.e., algorithms, that are able to learn from data (Goodfellow, 2016). The following definition of machine learning by Mitchell (1997) will be used to introduce some important concepts used throughout this thesis: '*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*' (p. 2).

There are many types of tasks *T* one might be interested in. The most common tasks in applied statistics are either classification (e.g., using binary logistic regression) or regression (e.g., using linear regression). The performance *P* of an algorithm is quantified by a *loss function*. A loss function quantifies how close the predictions $\hat{\boldsymbol{y}}$ of a model are to the true scores $\boldsymbol{y}$. For instance, in classification tasks, a simple loss function would just be the accuracy, i.e., the number of times the model produces the correct output. Finally, the definition above introduces the notion of experience *E*. *Supervised learning algorithms*, the core of this thesis, experience a dataset $\boldsymbol{X}$ of *n* examples and *n* corresponding targets ($\boldsymbol{y}$) and are instructed to find a function $f(\boldsymbol{X}, \boldsymbol{w})$ to map $\boldsymbol{X}$ on $\boldsymbol{y}$. The experienced dataset together with the targets denotes the *training set*, the available data to find good parameters $\boldsymbol{w}$ (e.g., regression coefficients in linear regression).

Depending on the task, each example $\boldsymbol{X}_i$, is either a vector of predictor variables (called *features* from now on) or a whole matrix. Throughout this thesis, each observation (example) of $\boldsymbol{X}$ will be represented as a matrix $\boldsymbol{\mathcal{X}}$ where a vector is expressed as a matrix with one row. Similarly, each target, corresponding to one example, is represented as a row vector. In multiple regression analysis, for instance, $\boldsymbol{\mathcal{X}}$ is typically represented as a 1-dimensional matrix (row vector) containing the features and $\boldsymbol{y}_i$ is a metric scalar outcome.

Many machine learning algorithms have one or more parameters that cannot be learned from data and are set by the user beforehand. Although these *hyperparameters* are not changed during training, an objective is still to find good values so that the training loss is

low. An example of a hyperparameter in linear regression analysis is the degree of polynomial which alters the ability of the model to fit the data. If different degrees would be evaluated on the training set, the highest degree polynomial would always be picked as the best setting since it leads to the lowest error on the training set (training error). Unfortunately, the model which performs best on the training set might not be the one that best generalizes to new data. To circumvent these problems, the performance of an algorithm is tested on a separate *validation set*. The validation set is usually a small fraction of examples that are removed from the training set and used to validate the performance of the algorithm during the training phase. By using a validation set, different settings of hyperparameters can be evaluated without the aforementioned problem. If the underlying relationship between $\boldsymbol{X}$ and $\boldsymbol{y}$ is linear, a high degree polynomial performs worse on the validation set than a simpler model with no polynomial term.

Finally, in order to test how well the algorithm actually performs, it is asked to make a prediction for new examples $\boldsymbol{X}_{new}$. These new examples with their associated targets are called the *test set*. Hence, besides minimizing the loss on the training set, the goal is to make the gap between training and test error as small as possible (see Goodfellow, 2016).

5 Neural networks

In this section, I will briefly explain what basic 'Neural Networks' are before presenting a specific type of neural network called *convolutional neural network* (LeCun et al., 1989). I will start with the description of one of the simplest neural network architectures, commonly known as binary logistic regression. The binary logistic regression model can be seen as a 1-layer neural network and has the form:

$$\widehat{\boldsymbol{Y}} = \sigma(\boldsymbol{XW}) \tag{5}$$

In the above expression, $\boldsymbol{X}$ is the *input layer* containing *n* observations (rows) where each observation $\boldsymbol{\mathcal{X}}$ is a matrix represented as a *d*-dimensional row vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^T$. Each $\boldsymbol{\mathcal{X}}$ contains as many values (columns) as there are features. As an example, if six variables from 100 participants would be used for prediction of a dichotomous outcome, $\boldsymbol{X}$ would be a dataset with 100 rows and six columns. $\boldsymbol{W}$ is the matrix with *k* columns representing the number of outputs per observation. Each column represents a vector $\mathbf{w} = (w_1, w_2, \ldots, w_d)$ of weights (*regression coefficients*).

In logistic regression analysis, there is only one predicted value ($k = 1$). Hence, $\boldsymbol{W}$ is a matrix with one column. Picking up on the example above, $\boldsymbol{W}$ would be a matrix with one column and six rows containing the regression coefficients. Each entry of $\boldsymbol{A} = \boldsymbol{XW}$ is referred to as an activation value *a* which is further passed through an *activation function*, here the logistic sigmoid function.

$$\sigma(\boldsymbol{A}) = \frac{1}{1 + \exp(-\boldsymbol{A})} \tag{6}$$

The sigmoid function, applied elementwise, returns the final prediction matrix $\widehat{\boldsymbol{Y}}$ where each row is a vector of predictions $\hat{\boldsymbol{y}}_i = (\hat{y}_{i1}, \hat{y}_{i2}, \ldots, \hat{y}_{ik})^T$. Since $k = 1$ in the LR model, $\widehat{\boldsymbol{Y}}$ is a matrix with one column. The output of a Neural Network is also termed as the *output layer*.

Although the logistic regression model can be seen as a neural network with only input and output layer, the actual power of neural networks results from the possibility to add one or more intermediate stages of processing, so called *hidden layers*. Each hidden layer itself has a set of individual weights associated with it and a separate nonlinear activation function. The number of weight matrices $\boldsymbol{W_1}$, $\boldsymbol{W_2}$, …, $\boldsymbol{W_l}$ determines the depth of the network. The LR model from above can be extended to have more than one weight matrix $\boldsymbol{W}$ associated with it by incorporating a hidden layer. In the literature (Goodfellow, 2016; Bishop, 2011), such an

architecture is known as a *multilayer perceptron* or *feedforward network*. Equation 7 shows a 2-layer LR model:

$$\widehat{\boldsymbol{Y}} = \sigma\big(h(\boldsymbol{A})\boldsymbol{W}^{(2)}\big) \tag{7}$$

With

$$A = \boldsymbol{X}\boldsymbol{W}^{(\mathbf{1})} \tag{8}$$

where $\boldsymbol{W}^{(1)}$ is a matrix of size $\{d\ x\ l\}$ with l being the number of intermediate outputs. $\boldsymbol{W}^{(2)}$ is a matrix of size $\{l\ x\ k\}$ with k equal to 1 in the LR model.

In Equation 7, instead of directly passing the activation matrix $\boldsymbol{A}$ to the sigmoid activation as in Equation 6, $\boldsymbol{A}$ is treated as an intermediate result that gets passed through an activation function h to get an intermediate output $\boldsymbol{Z} = h(\boldsymbol{A})$. $\boldsymbol{Z}$, taking the role of $\boldsymbol{X}$ in the input layer, is weighted again with weight matrix $\boldsymbol{W}^{(2)}$. The result is finally forwarded to the sigmoid function to get predictions. One common choice for the activation function h of a hidden layer, besides the sigmoid function, is the *rectified linear unit* (*ReLU*):

$$ReLU(\boldsymbol{A}) = \begin{cases} \boldsymbol{A}, & \text{if } \boldsymbol{A} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

The size of each layer in a neural network is determined by the number of units. For the LR model, the input layer has d units where each unit represents a feature, the output layer has 1 unit. The number of units (intermediate outputs) in hidden layers can vary depending on the application. Figure 1 shows both 1-layer and 2-layer neural network as a network diagram.

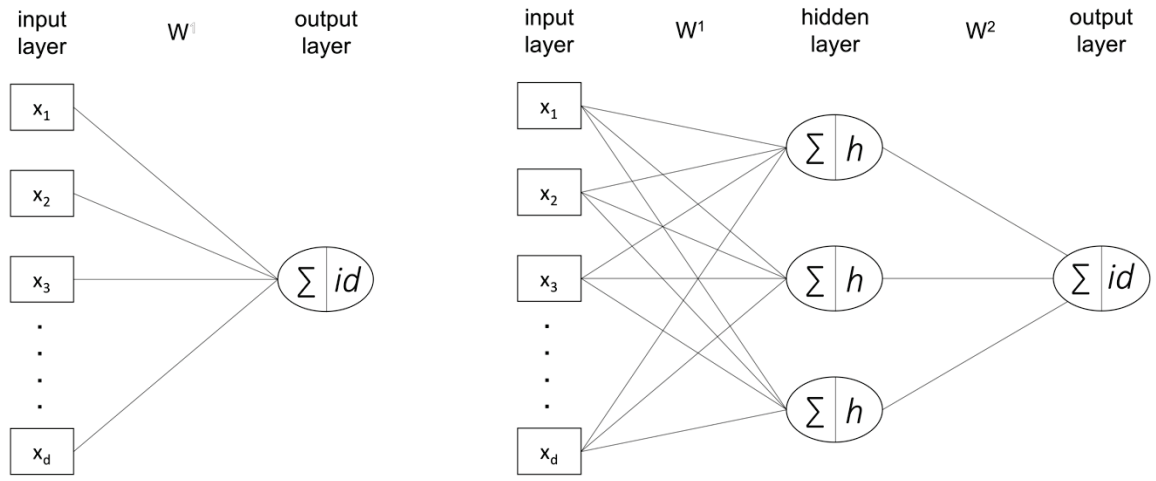

Figure 1. Left: Linear regression model shown as a 1-layer neural network with the identity function (id) as output activation function. Right: 2-layer neural network with 3 units in the hidden layer and nonlinear activation function h.

It has been proven (see e.g., Ripley, 1996), that neural networks with a linear output activation function and a hidden layer (such as the one presented above) are able to approximate continuous functions to any degree provided enough data and a large number of hidden units. Thus, neural networks with 2 or more layers, other than linear or logistic regression, can learn highly nonlinear relationships between $\boldsymbol{X}$ and $\boldsymbol{y}$. By stacking multiple layers on top of each other, this hierarchy of layers enables the network to learn simple concepts in the first layers which are then passed to later layers to learn more complex concepts (Goodfellow, 2016). Research concerned with the development and application of such neural networks with many layers stacked together, is known as *deep learning* (Aggarwal, 2018).

Convolutional neural networks (LeCun et al., 1989) or CNNs are a special type of neural network that exploit the spatial structure in the data. CNNs have first been applied to image data which is why I will use images throughout this section to explain how they work. Starting with images as an example for CNNs will also help to smoothly proceed with *DeepInference* in the next section. Recall from above that usually, when dealing with regression analyses, each observation $\boldsymbol{x}$ is a matrix with one row and d columns. The values $x_{1i}, x_{2i}, \ldots, x_{di}$, only share information about the specific observation i but other than that, there is no connection of any kind among those values. However, one can easily think of naturally structured or correlated features. In a simple time-series analyses, for instance, $\boldsymbol{x}$ contains measurements of the same feature measured at different time intervals. Hence, there is presumably a strong dependency between values measured at nearby time intervals. Besides

time-series, in the field of computer vision, $\boldsymbol{x}$ is an image expressed as a 3-dimensional matrix with rows representing the height (h) of the image in pixels, columns representing the width (w) in pixels and the third dimension depicting the color channels (c) red, green, and blue (rgb). Each value x_{whc} is a single pixel intensity value and is again, as in time-series data, strongly correlated to its neighboring pixel values. In a classic feed-forward neural network, this inherent structure in the image cannot be properly accounted for and each image would have to be flattened out into a single row matrix with d columns with d equal to $h * w * c$. In addition to the fact that CNNs take the grid-like structure of the input into account and can process the data as is, they often use far less weights than their feed-forward counterparts.

CNNs make use of the grid-like structure in the data by applying a mathematical operation called *convolution.* A convolution operation in CNN context can be described as a dot product between two matrices returning a scalar value. In CNNs, a volume (e.g., an image) is separated into small parts (windows) of a certain size, e.g., 5 x 5 x 3 pixels. Starting from the upper left corner of the image, a *filter matrix* of the same size (5 x 5 x 3) is convoluted with the corresponding pixel values, i.e., each element of the filter matrix is multiplied with the respective pixel value and all 5 x 5 x 3 = 75 values are summed up. After the first convolution, the filter matrix is shifted to the right by some number of pixels and is applied again to the pixel values in this window. Broadly speaking, a filter matrix can be seen as a feature extractor or object detector; if applied (via convolution operation) at all regions of the image, the resulting values, also denoted as a *feature map*, indicate the presence or absence of an object in the respective regions of the image. It has to be emphasized that the entries (weights) in the filter matrix, instead of having to manually set them, are learned during the training phase. In other words, the CNN learns to detect objects properties that are relevant for a good performance on the task (e.g., classification).

In every practical application of CNNs, the researcher chooses to apply not only one but many such filter matrices to the image where the number of filter matrices depends on the specific application. The dimension of the volume resulting from all filter matrices applied to all regions in the image is reduced in width and height[1] and has as many channels (depth) as there are filter matrices. The convolution operation is repeated with the new volume with a new window size and new filter matrices.

Similar to feed-forward neural networks, many such convolutional layers are stacked together with nonlinear activation functions in between each layer pair. After each convolutional layer, the spatial dimension decreases depending on the chosen size of the windows and the amount of shifting, until there is only a volume left of size 1 x 1 x d where *d* is the number of filter matrices applied to the previous volume. This 3-dimensional matrix, in a last step, is flattened out into a d-dimensional vector which can be thought of as a *learned feature vector* containing all necessary information about the image in a compressed form. Finally, this vector can be used as input to a classic feed-forward neural network as presented in previous sections.[2]

[1] The output width (w_{new}) and height (h_{new}) depend on the size of the filter matrix (f) and the amount of shifting (s) and can be computed via the following formula:

$$w_{new} = \left(\frac{w - f}{s}\right) + 1$$
$$h_{new} = \left(\frac{h - f}{s}\right) + 1$$

Where a value of $s = 1$ means to shift the window one pixel to the right. If the right border of the image has been reached, the window is then shifted down by one pixel.

[2] Note that the both processes, computing the summary-statistics vector and training a feed-forward network with it can be run in one model in an end-to-end fashion. It is not necessary to train another model with the summary-statistics vector.

6 Towards end-to-end likelihood free inference (Manuscript 2)

For all simulator-based models, i.e., models for which data can easily be simulated from, it is straightforward to frame the problem of approximate Bayesian computation as a supervised learning problem. Within the framework of supervised learning, there are two potential advantages compared to the simple ABC rejection algorithm. If a satisfactory mapping from $\boldsymbol{X}$ to $\boldsymbol{y}$ via function $f(\boldsymbol{X}, \boldsymbol{W})$ can be found, then there is no need to decide upon which distance metric $d(\cdot)$ or threshold value ϵ to choose anymore. As a downside, however, supervised learning algorithms are currently not able to return a full (posterior) distribution (see Discussion for an approach to overcome this problem). Rather, only the first two posterior moments (mean and variance) are estimated.

In order to use neural networks or other supervised learning algorithms, a dataset $\boldsymbol{X}$ of n examples and n corresponding targets ($\boldsymbol{y}$) has to be constructed. Each example (or observation) in $\boldsymbol{X}$ is a simulated dataset $\boldsymbol{x}$ expressed as a 2-dimensional matrix consisting of m rows (trials) and d columns[3] and $\boldsymbol{y}$ is a vector containing the parameters with which the data have been simulated from $(\theta_1, \theta_1, \ldots, \theta_k)$. In the framework of ABC, the training set is called *reference table* and is constructed via **Algorithm 4**:

Algorithm 4. Generation of the reference table.

for $i \leftarrow 1$ to N

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

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

Note that in **Algorithm 4**, the simulated data, prior to storage, are passed through the pre-processing function $\eta(\cdot)$ which could either be a function to compute summary statistics (predefined features) or the identity function if no summary-statistics should be computed.

[3] In the case of summary statistics, each observation in $\boldsymbol{X}$ is a d-dimensional vector where each element of the vector is a specific summary statistic.

One obvious way to represent a simulated dataset is via its summary statistics. Summary statistics are useful in two ways: they simplify the data and make the algorithm independent of the actual trial size *m*. Marin, Raynal, Pudlo, Ribatet, and Robert (2017) applied a *random forest* (Breiman, 2001) to learn the mapping from a *d*-dimensional summary statistics vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$, with *d* equal to the number of summary statistics computed, to estimate the posterior mean and posterior variance of $\boldsymbol{\theta}$ (see details in Manuscript 2). Although their attempt was promising, it nonetheless suffered from two major downsides. First, they had to construct useful and sufficient summary statistics before applying the algorithm. Secondly, a separate random forest for each θ_i in $\boldsymbol{\theta}$ had to be trained. *DeepInference*, introduced shortly, overcomes both problems.

As a second approach, Jiang, Wu, Zheng, and Wong (2015) used a feed-forward neural network on the raw data to predict the posterior mean of $\boldsymbol{\theta}$. After the network was trained, they used the predictions of the network as summary statistics and applied the usual ABC rejection algorithm by comparing observed summary statistics (prediction of the network on the observed data) and simulated summary statistics (predictions for all observations in the reference table). Although Jiang and colleagues did not use any prespecified summary statistics, their approach eventually suffered from having to choose a threshold ϵ. Also, in contrast to Marin et al. (2017), their neural network could not be reused since it had to be trained anew if the trial size of the observed data changed.

DeepInference is a convolutional neural network that takes the raw simulated data as input, learns a condensed and informative representation of the raw data internally (see the notion of *learned feature vector* above), and outputs a prediction of the first two posterior moments of the parameter vector $\boldsymbol{\theta}$. I will demonstrate the features of *DeepInference* that distinguish it from all previous attempts using a multiple regression model as an example case. Given an observed dataset suited for multiple regression analysis with *m* rows and *d* columns (with *d-1* predictor variables and one outcome variable), the goal is to predict the parameter vector $\boldsymbol{\theta}$ of the *d-1* regression coefficients[4]. The reference table is constructed as described in **Algorithm 4** where each simulated dataset contains as many rows as the observed dataset. Taking the feed-forward neural network approach of Jiang et al. (2015) would not be practical in such a scenario since flattening out each simulated dataset would result in a *d x m*-dimensional vector of features. Imagine a multiple regression dataset with

[4] In this example, the intercept term is neglected.

200 participants (rows), 10 predictor variables and one outcome (10 + 1 columns), the neural network had to start with 200 x 11 = 2200 features. Setting the number of units in the first hidden layer to 64, this already would imply a weight matrix of 2200 * 64 = 140800 values that had to be learned during the training phase[5].

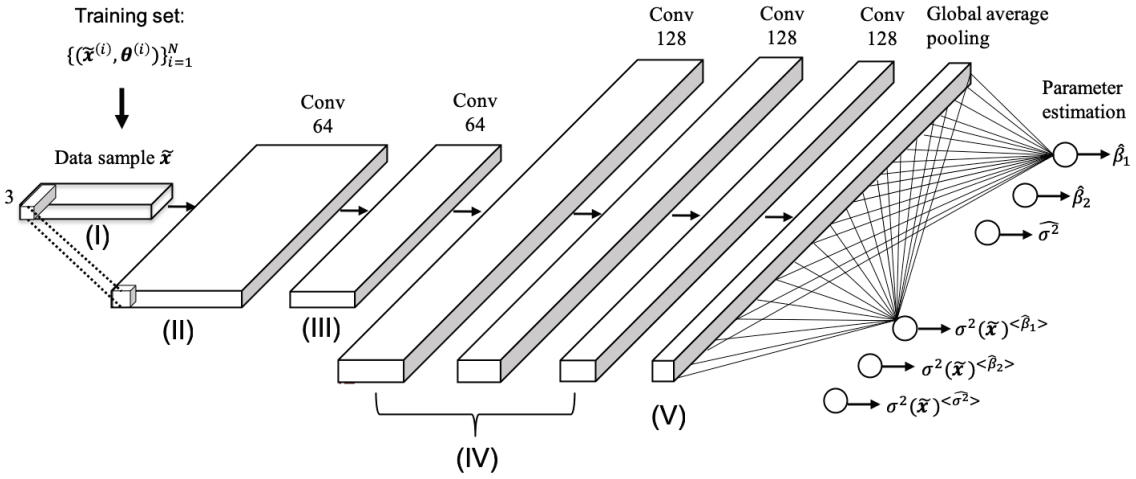

Figure 2. A 6-layer DeepInference neural network architecture illustrated for a multiple regression with two predictor variables. A multivariate dataset with n rows and 3 columns (x1, x2, y) is reshaped into a 3-dimensional matrix of dimensions (1 x n x 3 (I)). 64 filters are then applied with a stride of 1 resulting in a 1 x n x 64 volume (II). The next convolutional layer applied 64 filters of size 3 with a stride of 2 resulting in a 1 x m x 64 volume (III) with m equal to (n-3)/2 + 1. Three layers with 128 filters of size 3 with stride 2 are added consecutively (IV). Finally, for each of the 128 filters, an average is computed (GAP = global average pooling) so that 128 values are left (V). Given these 128 values, the network predicts the two regression coefficients and the residual variance together with their posterior variances.

One of the major insights that allows *DeepInference* to process the raw data is to make use of the inherent structure in the data. In any multivariate dataset without any time-dependencies, each row is independent of all other rows (*independent and identically distributed*). However, the values of each row share some common information that could be leveraged. In the multiple regression scenario, for instance, the *d-1* instances of the predictor variables are somehow linked to the instance of the outcome variable and reveal a tiny bit of information about the relationship between predictor variables and outcome. The convolutional network applies multiple filters of size *1 x d* to each row of the simulated

[5] Of course, taking such an approach is still feasible given a large enough reference table.

datasets. Since each filter has only *d* weights that have to be learned, the number of weights to be learned is reduced tremendously (if 64 filters are used as in the feed-forward neural network above, this results in only *64 x 12 = 768* weights[6]).

Whereas a classic neural network requires input of the same shape, thus making the network dependent on a specific trial size, in *DeepInference*, I used a technique called *global average pooling.* Instead of reducing the spatial dimension in each convolutional layer until only a volume of size 1 x 1 x q is left (see description of CNNs above), the average over each feature map is taken as shown in step (IV) to (V) in Figure 2, irrespective of the number of elements in the feature map[7]. *DeepInference* is, to the best of my knowledge, the first algorithm for approximate Bayesian computation that works with variable input sizes without the need to figure out summary statistics. Hence, *DeepInference*, after training on data from a specific model configuration, can be reused by other researchers.

Another strength of *DeepInference* is the use of a specific loss function. In linear regression, the true scores $\boldsymbol{y}$ are assumed to be conditionally normally distributed with a constant variance term:

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{W}) = \mathcal{N}(f(\boldsymbol{X},\boldsymbol{W}),\sigma^2) \tag{10}$$

where $f(\boldsymbol{X},\boldsymbol{W})$ are the predictions of the neural network[8]. In maximum likelihood estimation, $\boldsymbol{W}$ is chosen to minimize the product of densities of each value y_i in $\boldsymbol{y}$**.** It can be shown (see e.g., Bishop, 2011), that by taking the negative logarithm of the likelihood function, the resulting loss function becomes the common mean squared error loss:

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left\|\hat{y}^{(i)} - y^{(i)}\right\|_2^2 \tag{11}$$

Whereas the mean squared error loss function relies on the assumption of homoscedasticity (constant variance), the heteroscedastic loss function (Kendall & Gall, 2017; Nix & Weigend, 1994) stretches this assumption.

[6] Each filter has an additional bias term, hence *(64 x 11) + 64 = 64 x 12*.
[7] CNNs with global pooling are sometimes also referred to as fully convolutional networks (FCN).
[8] Although $f(\boldsymbol{X},\boldsymbol{W})$ seems to contain only one weight matrix $\boldsymbol{W}$, $f(\boldsymbol{X},\boldsymbol{W})$ can still be a neural network with several layers where the weight matrices have been concatenated together into $\boldsymbol{W}$**.**

As the mean squared error loss is based on homoscedasticity, so does the heteroscedastic follow from the assumption of heteroscedasticity. In heteroscedastic regression, each data point is assumed to be normally distributed with an individual variance term:

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{W}) = \mathcal{N}\big(f(\boldsymbol{X},\boldsymbol{W}), \sigma^2(\boldsymbol{X})\big) \tag{12}$$

Taking the negative logarithm of the product of densities as in Equation 11, the result becomes the heteroscedastic loss:

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \frac{\left\|\widehat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}\right\|_2^2}{2\boldsymbol{\sigma}^2(\boldsymbol{X}^{(i)})} + \frac{1}{2}\log \boldsymbol{\sigma}^2\big(\boldsymbol{X}^{(i)}\big) \tag{13}$$

By using the loss above, the neural network is required to output two vectors $\widehat{\boldsymbol{\theta}}$ and $\widehat{\boldsymbol{\sigma}}(\boldsymbol{\theta})$ each of length *k* where *k* is the number of parameters of interest. Note that $\boldsymbol{\sigma}^2$ is not estimated in a supervised fashion, i.e., the variances of each parameter are not available, but inferred implicitly via the loss above. If the deviation of $\hat{y}_i$ and y_i is large, the variance term σ_i^2, in order for the overall loss to stay low, has to be increased. Notice that the network could learn to set the variance term in the denominator to infinity which would imply an infinitely small loss. This undesirable behavior is counteracted via the term $\frac{1}{2}\log \boldsymbol{\sigma}^2(\boldsymbol{X}^{(i)})$ which prevents the estimated variance term to grow to infinity.

In Manuscript 2, we applied *DeepInference* to two toy examples, namely a bivariate normal model and a multiple regression model. These two examples, although equipped with a likelihood function, were well suited to test how accurately we could approximate both posterior mean and posterior variance of each parameter. For both models, a closed-form posterior distribution of each parameter was available which was used to assess the performance of the predictions. We could show (see details in Manuscript 2) that the neural network, when evaluated on the test set of each of the two models, indeed made accurate predictions for both posterior mean and posterior variance.

As a last and more realistic scenario, we chose to estimate parameters of the leaky competing accumulator model (LCA) for which no likelihood-function is available (Usher & McClelland, 2001). The LCA model belongs to the class of sequential sampling models and can be seen, just as the Lévy flight model, as an extension of the classic DDM. Usher and

McClelland (2001) draw attention to the fact that in the classic DDM, information is accumulated without decay or loss thus leading to perfect discrimination between two differing stimuli given enough time. They address this assumption and argue that there might be scenarios in which further sampling of information does not help in making better or more accurate decisions. In order to account for this fact, Usher and McClelland (2001) introduce a leakage parameter (κ) with which the amount of loss can be captured. Moreover, since the LCA model allows for one accumulator per choice alternative (other than the relative evidence of one alternative with respect to another), it can be applied to experimental designs with more than two choice alternatives (see details in Manuscript 2). In a previous attempt to recover the parameters of the LCA model, Miletić, Turner, Forstmann, and van Maanen (2017) used a technique called Probability Density approximation, a state-of-the-art technique for likelihood-free inference (Turner & Sederberg, 2014). We demonstrated in the manuscript that *DeepInference* outperformed the Probability Density approximation method in recovering the parameters of the LCA model.

Having seen the potential of our approach for the LCA model, we wanted to apply it on two other prominent sequential sampling models, namely the drift diffusion model and the Lévy flight model. Although the DDM is provided with a likelihood function, we were still interested in a comparison of maximum likelihood estimation and our approach as explained shortly.

7 Learning the Likelihood (Manuscript 3)

In complex cognitive models such as the DDM, no analytical solution for the maximum likelihood estimates of the parameters is available. As a consequence, multidimensional search procedures (ML search) have to be used (see e.g., the SIMPLEX-search; Nelder & Mead, 1965). ML search procedures for maximum likelihood estimation are not only time-consuming but also not guaranteed to converge to a global minimum (as they may get stuck in local minima). Moreover, the precision with which the likelihood function is evaluated is sometimes not accurate enough which in turn leads to wrong updates of the parameters. Lastly, every algorithm based on a complex likelihood function is very sensitive to unusual data points (outliers) (see details in Manuscript 3).

From this point of view, *DeepInference* thus becomes an interesting alternative even for models where a likelihood function is available as it completely bypasses the evaluation of the likelihood. For the stated difficulties associated with ML search, we were first interested in a comparison of the DDM and the deep learning approach in terms of parameter recovery. Furthermore, we tested the sensitivity of both ML search and *DeepInference* in the presence *contaminants,* i.e. unusual responses not emerging from a diffusion process (Lerche, Voss, & Nagler, 2017). The same analyses were repeated for the Lévy Flight model. In the next two sections, I will briefly present the two sequential sampling models and the main findings of the simulation studies, starting with the DDM and continuing with the Lévy Flight model.

The drift diffusion model is a model for the analysis of response time data from binary decision tasks. The DDM assumes that information during a binary decision accumulates continuously according to a Wiener diffusion process until one of two thresholds (corresponding to the choice of one of the two alternatives) is reached (e.g., Lerche & Voss, 2017; Voss, Nagler, & Lerche, 2013). Instead of a constant increment of information, the DDM treats the information accumulation process probabilistically by adding Gaussian noise, thus allowing for different response times and decisions across trials. The classical drift diffusion model consists of four parameters, namely drift rate (v), threshold separation (a), starting point (z), and non-decision time (t_0), where each parameter reflects a different cognitive process underlying the decision. Hence, the DDM, other than a simple comparison of mean response times, helps to determine what psychological process leads to a fast or slow response by looking at its respective parameters. The parameter a (threshold separation) for instance can be interpreted as the amount of information necessary to make a decision (e.g., Voss, Nagler, & Lerche, 2013). Accordingly, a person for which a high value of a has been

estimated can be seen as being rather conservative (and vice versa). The *full diffusion model* has three additional intertrial variability parameters for drift rate (s_v), starting point (s_z, e.g., Ratcliff & Rouder, 1998), and non-decision time (s_{t0}, Ratcliff & Tuerlinckx, 2002).

We decided to train *DeepInference* on simulated data from the full diffusion model with two different stimulus types. For each stimulus type, all parameters but the drift rate were held constant. In summary, the final model had a total of eight parameters. Since the goal was to make *DeepInference* as reusable as possible, we decided to train the network only once on a simulated reference table containing both clean and contaminated data and reuse it to compare the performance on three different test sets (0%, 10%, or 100% contaminated datasets[9]. In order to make it learn the most robust mapping from data to parameters, we followed the approach used by Lerche, Voss, and Nagler (2017) and inserted two different types of *contaminants* to the training data. For each simulated dataset, we randomly chose 0% to 10% of the trials to be fast guesses resulting in either correct or false response. In addition, for 0% to 10% of the trials, we simulated distractions of participants (slow *contaminants*) by making the respective response times longer without changing the corresponding response types.

Moreover, as another important and new aspect of the training procedure, we explicitly focused on the ability of *DeepInference* to handle datasets of variable size (different trial sizes). We wanted to examine if the network performs well on a range of different trial-sizes and get the best performance regardless of the respective trial size. Previously, the network was trained on a reference table of simulated datasets with equal trial sizes. In Manuscript 3, however, we changed the training of the network by providing the architecture with different trial sizes, ranging from 100 trials (50 per stimulus type/condition) to 1000 trials (500 per stimulus type/condition) during training[10]. With such an approach, the network was forced to find global patterns in the datasets without the possibility to focus on one specific trial size.

We compared the ML search with *DeepInference* on a test set with 0%, 10% and 100% contaminated datasets. For each predicted/estimated parameter, we looked at the root mean square error (RMSE), the Pearson correlation coefficient of predicted and true

[9] From the perspective of making use of a trained model, training the network on both clean and contaminated data is not advisable since users would have to know whether their data are contaminated or not.
[10]During training, we randomly picked a trial size (*ts)* between 100 and 1000. We then randomly picked *ts/2* trials from the first condition and also from the second condition so that the network processed a dataset with varying number of trials but a balanced number of trials within each condition.

parameter values, and the variance explained by a line from the identity function. For clean data, results revealed that both approaches were equally good on recovering the parameters. For contaminated data though (both 10% and 100%), *DeepInference* showed up to be superior and more robust than multidimensional parameter search.

The Lévy-Flight model (see Voss et al., 2019) forgoes the assumption that the noise in accumulating information is distributed according to a normal distribution. Rather, the normal distribution is replaced with an *alpha-stable* distribution that is more flexible in that it can account for varying decision processes among participants. The additional parameter α controls the width of the noise distribution and has a theoretical range from 0 to 2. The commonly known cauchy distribution ($\alpha = 1$) and the normal distribution ($\alpha = 2$) are special cases of the *alpha-stable* distribution. Due to the possibility to make the error distribution more heavy-tailed (values of α close to 1) than in the usual diffusion model, the Lévy-Flight model allows for random jumps in the decision process. Voss et al. (2019) showed that relaxing the assumption of normal noise and hence allowing for random jumps considerably improved the goodness-of-fit to experimental data. As promising as the Lévy-Flight model has been shown to be, as cumbersome it is to estimate its parameters since there is no likelihood function available. In Manuscript 3, as a comparison with *DeepInference*, we used an approach introduced by Heathcote, Brown, & Mewhort (2002) (see details in Manuscript 3) which still enables parameter estimation but turns out to be enormously computationally expensive and therefore rather inpratical for most purposes.

The procedure to compare both estimation procedures was almost identical to the drift diffusion model comparison, except that we simulated data without the inter-trial variability parameters. Accordingly, the Lévy-Flight model was a 6-parameter model with two drift rates (ν_1, ν_2), threshold separation (a), starting point (z), non-decision time (t_0), and the parameter from the alpha-stable distribution (α). Results indicated two important findings. First, as in the drift diffusion model, the usual parameters were equally well recovered for clean data but much better recovered by *DeepInference* when the data were contaminated. Secondly, the main parameter of interest, α, was estimated with higher precision by *DeepInference* than when using the ML search by Heathcote et al. (2002).

8 Discussion

Whenever researchers adopt existing mathematical models to better account for their experimental data, it might become difficult to find an appropriate likelihood function that accounts for the changes of the respective model. Oftentimes, increased modeling complexity is accompanied with the loss of the likelihood function and sometimes models can only be described as simulator-based models in the first place. Parameter inference for such complex likelihood-free cognitive models is an important area of research and several algorithms have been developed in the past with the goal to accurately estimate the parameters of models without a likelihood function. Although parameter inference is not a prerequisite for the development of new models, it is nonetheless crucial if these new models and their associated theoretical considerations should be tested in a quantitative manner. Equally important is the ability to directly compare novel models with existing ones. If a new modification, no matter how theoretically intriguing it might be, describes existing data worse in most cases, it cannot be seen as a real advancement. Model comparison/hypothesis testing thus is a vital aspect to assess and evaluate innovative new advances.

In this thesis, I was first focused on model comparison and showed how existing algorithms for likelihood-free inference can be applied to hypothesis testing using an approximation of the Bayes factor. While the Bayes factor for common statistical tests can easily be computed with existing software, this is not the case for more complex models. For instance, computing Bayes factors is not as straightforward when users want to change the prespecified prior distributions over the parameters or when one is interested in running non-standard tests. In addition, users are often only allowed to compare nested models (hypotheses) whereas ABC methods can naturally be applied to the comparison to both nested and non-nested models. With *ABrox,* an open-source graphical user-interface I developed, users are equipped with a flexible toolbox to conduct approximate Bayesian hypothesis testing and compute ABC Bayes factors with as less effort as possible.

Previous attempts on likelihood-free inference made more or less use of the fact that most models allow for easy simulation of data given specific parameter values. Assuming such simulator-based models, many algorithms are then based on the idea of comparing simulated datasets to the actual dataset of interest for which the parameters should be estimated. If the parameter values chosen for a specific simulation lead to data similar to the observed data, this information is used as a proxy for a high likelihood of these parameters. To address the question of how to compare simulations with observed data, many algorithms

use summary statistics of both observed and simulated data for the final comparison to reduce the dimensionality of the raw data. Summary statistics are useful and necessary in most cases since easy metrics (such as Euclidean distance) can be used to examine their similarity. Unfortunately, as a downside, relying on summary statistics prevents users to investigate models for which it is not straightforward or even impossible to compute summary statistics containing the relevant information in the data. Moreover, all algorithms for likelihood-free inference we are aware of, are bound to a particular observed dataset. In other words, for every dataset one is interested in, the algorithm of choice has to be run. Since many algorithms are computationally extremely intensive and require a long time to finish, the use of these algorithms drastically constrains their applicability.

The main contribution of this thesis is the improvement of existing techniques to estimate parameters for likelihood-free models using deep learning. We showed that a convolutional neural network with *global average pooling* combined with the heteroscedastic loss function, which we termed *DeepInference*, overcomes the limitations of having to compute summary statistics and being bound to a particular observed dataset. A convolutional neural network with *global average pooling* is able to process datasets of different size (variable number of trials). Thus, only one network needs to be trained for a specific cognitive model design, e.g., a DDM with upper and lower threshold, which can then be reused for other datasets with different trials sizes. Besides the choice of the neural network architecture, the heteroscedastic loss function was used to not only estimate the posterior mean of each parameter but also its posterior variance.

We first tested first *DeepInference* on two toy examples for which the posterior distribution of each parameter can be computed analytically. For several simulated datasets, we then calculated both the true mean and variance of the posterior distribution of each parameter and compared it to the predicted posterior means and variances of *DeepInference.* Results suggested that our method could approximate both mean and variance of the posterior distribution to a high degree. After this first successful application, we applied it to three models from the family of sequential sampling models, namely the Leaky competing accumulator model (Manuscript 2), the drift diffusion model (Manuscript 3), and the Lévy flight model (Manuscript 3). For all three models, parameters could be recovered as good as with existing state-of-the art techniques and under specific conditions (contaminanted data) even better. In summary, *DeepInference* is the first end-to-end ABC algorithm, taking raw

data as input and predicting the first two posterior moments of the parameters' posterior distribution.

In the next sections, I will discuss a few limitations of *DeepInference* as well as some open questions for future research.

8.1 Model comparison

Throughout this thesis (Manuscript 2 and 3), the deep learning model was only applied to parameter estimation. Both in Manuscript 2, where we introduced *DeepInference*, and in Manuscript 3, where we applied the model to estimate the parameters of two prominent sequential sampling models, the aim was to estimate the first two posterior moments of the posterior distribution of parameters. However, the task for the *DeepInference* architecture could easily be framed as a classification task. Whereas previously, the model was supervised by providing it with the true parameter estimates for training, in model classification, the model is, for each $\boldsymbol{X}$, provided with the corresponding model (index) from which the data were simulated from. The neural network then tries to find patterns in the simulated datasets of each model in order to distinguish them. To implement this, the architecture essentially needs to be changed in three ways. First, the linear (identity) activation function would be replaced with a so-called softmax activation function (see Equation 14 below). The softmax activation function is the extension of a sigmoid function for a comparison of more than two models. The network produces an output vector of *k* real numbers with *k* equal to the number of models being compared. This vector is then passed through the softmax function so that the sum of the resulting elements equals to 1. Afterwards, each final element can be treated as a probability that the respective model generated the data. A prediction is made by simply picking the maximum value (probability).

$$\sigma(\boldsymbol{A})_j = \frac{\exp(\boldsymbol{A}_j)}{\sum_{k=1}^{K} \exp(\boldsymbol{A}_k)} \tag{14}$$

Secondly, instead of storing the true parameter values for each $\boldsymbol{X}$ in the reference table, a model index (or one-hot encoded vector of model indices) has to be recorded. Finally, the heteroscedastic loss function should be replaced with another loss function accounting for the fact that one is interested in classification accuracy rather than regression. The simplest choice of a loss function is the cross-entropy loss function (see e.g., Bishop, 2011):

$$\mathcal{L}(\boldsymbol{W}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \ln(\hat{y}_{nk}) \tag{15}$$

where N is the total number of observations in the training set and K is the total number of models being compared. In the cross-entropy loss, the loss is increased if a prediction $\hat{y}_{nk}$ is far off of the true value y_{nk}.

One important thing to notice is that model comparison, conducted in such a way, is potentially applicable for any models, as long as data from these models can be simulated in a reasonable amount of time. However, the final decision of such a neural network (from which of the K models the data come from) does not take into account the parsimony of the models. The only source of information it gets to differentiate among the models is the resulting datasets. Put differently, if a complex model is compared to a parsimonious one, and for a particular dataset the network predicts a probability of the dataset to belong to the latter of 50%, we cannot state whether the dataset is equally likely under both models or rather the network itself is not accurate enough (e.g., has not been trained long enough). Theoretically though, assuming a perfect network, the user could assign the datasets for which the network was indecisive (e.g., predicted probabilities between 0.45 and 0.55), to the simplest model.

8.2 Fully-Bayesian approach

In this thesis, I introduced *DeepInference* as an algorithm that comes a step closer to the goal of real end-to-end likelihood-free inference. Most importantly, I showed how the algorithm achieves state-of-the art results with respect to parameter recovery without the need to compute summary statistics. Nonetheless, *DeepInference* is only 'partly-bayesian' in the sense that it outputs only the first two posterior moments of a parameter's posterior distribution instead of the whole distribution. Fortunately, there is a promising approach that might have the potential to overcome even this shortcoming in the near future, namely the *conditional variational autoencoder* (cVAE) (e.g., Kingma & Welling, 2013; Doersch, 2016; Sohn, Lee, & Yan, 2015). In the next section, I will describe on a high-level what a cVAE is and how it might be adapted to generate whole posterior distributions of parameters.

Before explaining cVAE, I will first describe what an autoencoder is. An autoencoder is a neural network that is trained to reproduce its input. In other words, instead of learning a mapping from $\boldsymbol{X}$ on $\boldsymbol{y}$, it learns a mapping from $\boldsymbol{X}$ to $\boldsymbol{X}$. However, instead of the rather simple task to produce as output its own input, the autoencoder is used as a dimensionality reduction

technique. The autoencoder consists of two neural networks that are trained end-to-end, the *encoder network* and the *decoder network* (see Figure 3). The encoder network takes the input and reduces its dimensionality to a certain degree. It finally 'outputs' a vector containing a dense representation of the raw input. The decoder network then takes this lower-dimensional vector and tries to reconstruct the original raw input using only this condensed version of the input. Moreover, the autoencoder is forced to put as much information in this dense representation (summary vector) as possible so that a minimum of information is lost. If the *bottleneck* layer, as it is sometimes referred to, is too small, it might not be possible to reproduce the input adequately. If, on the other hand, the input is not compressed enough, the resulting summary-vector might not be interpretable in any way. Consider the scenario of an autoencoder applied to images of faces where the summary vector is allowed to contain only 10 elements (attributes). The autoencoder learns to compress an image of a face into 10 attributes that might or might not suffice to reconstruct the original image. Interestingly, after training, each element of the vector might be interpretable as an attribute of a face, e.g., smile, gender, skin color, etc.

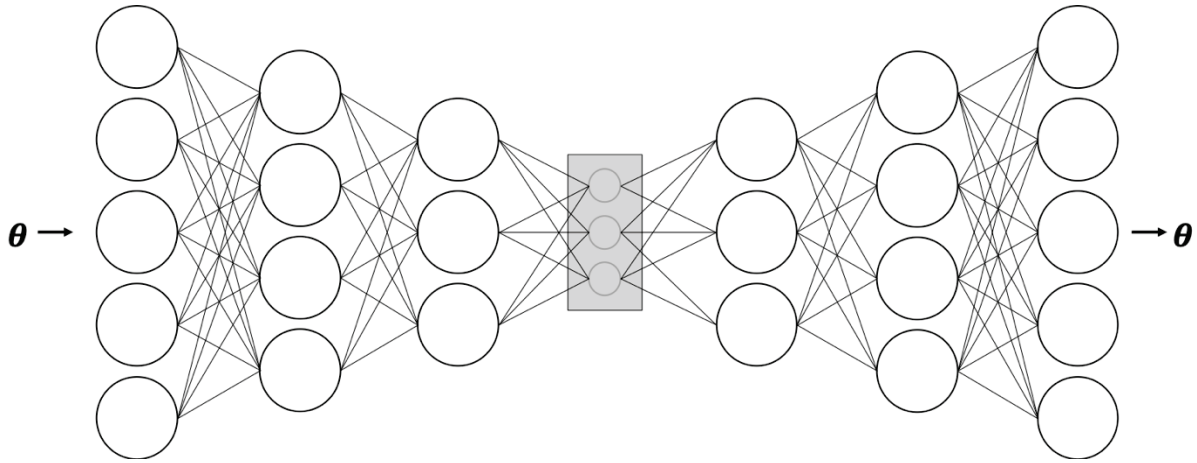

Figure 3. An autoencoder taking as input a parameter vector $\boldsymbol{\theta}$ *consisting of five parameters.* $\boldsymbol{\theta}$ *is passed forward to the encoder network which compresses the size to three values. The latent representation vector (grey box), consisting of three values, is passed over to the decoder network which tries to reconstruct* $\boldsymbol{\theta}$.

With the variational autoencoder (VAE), each possible attribute does not have to be represented using only a single value, but is rather described as a probability distribution. The VAE (see Figure 4) works similarly to the autoencoder but instead of generating the internal

representation (summary vector) directly, the encoder network only outputs means and variances. Each combination of mean and variance is then used to generate each element of the summary vector by sampling from a normal distribution. Thus, every time the same image is passed to the encoder, it is represented slightly different, depending on the size of the variances. The decoder network is asked to reconstruct the same original input using these different representations. The total loss in the VAE consists of two terms. The first term is the reconstruction loss and is decreased the more similar the reconstructed input is to the original input. The second term is the *Kullback-Leibler divergence* which enforces the latent distribution spanned by each mean and variance term to be unit Gaussian (standard normally distributed)[11]. The VAE, other than the simple autoencoder, can be used a *generative model*. In order to generate (sometimes also referred to as *hallucinate*) a new input, one generates the latent representation (summary vector) by drawing values from a standard normal distribution. This constructed vector is then passed through the decoder network to generate an output.

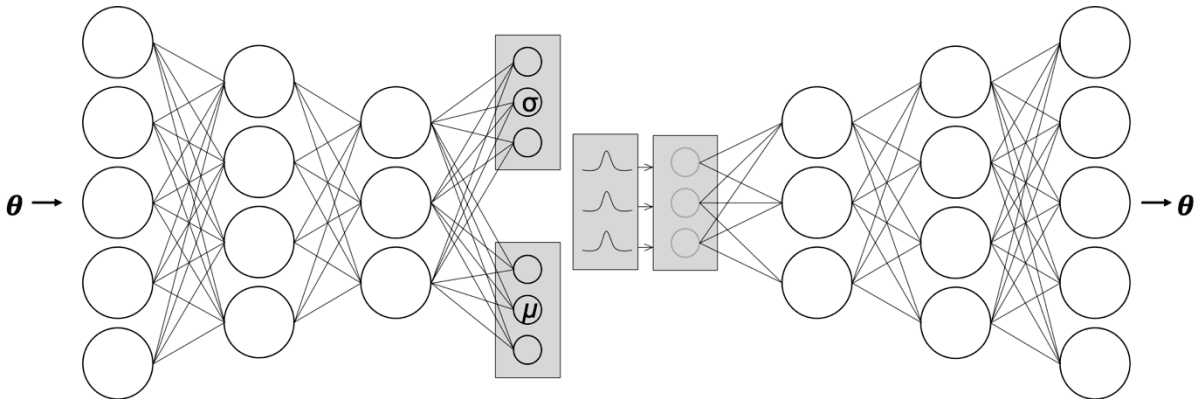

Figure 4. A variational autoencoder taking as input a parameter vector $\boldsymbol{\theta}$ *consisting of five parameters.* $\boldsymbol{\theta}$ *is passed forward to the encoder network which compresses the size to three values. The latent representation vector is generated by sampling each latent attribute from a normal distribution with the respective* μ *and* σ*. The latent representation, consisting of three samples from different normal distributions is passed over to the decoder network which tries to reconstruct* $\boldsymbol{\theta}$*.*

The word *conditional* in CVAE refers to the ability to generate data conditional on some attribute. For instance, one could train a VAE by providing both encoder and decoder

[11] Forcing the latent representation to be standard normally distributed induces each attribute to be uncorrelated (independent) to the others, thus making it more interpretable.

network with an additional label of the image (e.g., 0 = if the face is from a male person, 1 = if the face is from a female person). After training, it becomes possible to generate faces conditional on the given labels (e.g., generate only male faces).

We could apply such a cVAE to the field of likelihood-free inference in the following manner. A reference table is constructed in the same way as for *DeepInference*. However, instead of choosing $\boldsymbol{x}$ to be a simulated dataset, $\boldsymbol{x}$ becomes the parameter vector $\boldsymbol{\theta}$, previously used as the target. Using a simple feed-forward neural network, the task for the variational autoencoder is to compress the parameter vector $\boldsymbol{\theta}$ and reconstruct it. In addition, however, the idea is to condition the process on a particular simulated dataset. One obvious difficulty with such a conditioning is the representation of a dataset. Instead of conditioning on a simple vector with labels (see example above), it is not clear how to condition on a whole matrix. One possible solution would be to apply *DeepInference* as a way to compress the complexity into a *learned feature vector* (see section on Neural Networks) which could then be used for the conditioning. The hope in using a CVAE for likelihood-free inference is to generate multiple hallucinated parameter vectors $\boldsymbol{\theta}$ which can be treated as samples from the approximate posterior distribution of $p(\boldsymbol{\theta}|\boldsymbol{x}, M)$. It has to be emphasized though that at the time of writing, there is no mathematical justification for why such generated samples might approximate the posterior distribution but the described procedure seems nevertheless to be a promising future perspective worth studying.

8.3 Neural networks as 'black-boxes'

Modern machine learning algorithms are often assumed to be 'black boxes' meaning that there is no way to figure out how the algorithm actually learns (see e.g., Alain & Bengio, 2016). Although deep learning models were tremendously successful in recent years and reached human-level performance or even exceeded it in quite a few domains such as image classification, speech understanding or sentiment analysis, they still lack in transparency in that it is tedious and often impossible to gain information about their inner workings (see Samek, Wiegand, & Müller, 2017). This major drawback, the inability to fully interpret and explain how exactly a model transforms its input and arrives at a decision in the end, has gained a lot of attention in recent years and can be seen as one of the current hot-topics in deep learning research. If researchers would be able to enhance their theoretical understanding of the learning processes of deep neural networks, networks would not only be easier to improve but also important insights about how the human brain works might be derived (Lei, Chen, & Zhao, 2018).

CNNs applied to images constitute an exception in that manner since it is easy to visualize what such a CNN learned by looking at the learned filters. Commonly, researchers make a CNN more transparent by looking at activation patterns, i.e., the type of patterns that maximally activate the filters used (lead to the largest dot-product of filter and input). By looking at the resulting patterns of each filter, it has been demonstrated (see e.g. Zeiler & Fergus; 2013) that early filters seem to learn low-level features such as corners and edges. Later filters build up on those low-level features and learn increasingly abstract features. If applied to images of faces, such later filters are activated for instance by high-level features of faces such as eyes, mouths, teeth, etc.

Although *DeepInference* is a CNN, understanding its internal mechanism is complicated. The approach described above is not suitable because activations prototypical for specific filters cannot be represented as images. While we as humans are used to looking at images and making sense of them, inspecting activations inferred from multivariate datasets is a nontrivial task or perhaps even a scientific deadlock. However, even if a thorough review of the current research about explainable machine learning is important it has to be emphasized that we exclusively used *DeepInference* as a way to estimate psychologically meaningful and interpretable parameters from cognitive models.

9 Summary and Conclusions

Machine learning, and more specifically deep learning, is an exciting field of research which has grown rapidly in recent years and has already led to major scientific contributions in various disciplines, including medicine, biology and physics (Jordan, & Mitchell, 2015). However, in Psychology, machine learning tools have not yet gained as much attention as in other disciplines so far (see e.g. Yarkoni, & Westfall, 2017). In this thesis, I combined modern psychological research and machine learning to improve the estimation of parameters from interesting sequential sampling models. Cognitive models with psychologically meaningful parameters are important tools to uncover human behavior. Nonetheless, reliably estimating these parameters for highly complex models is often a non-trivial task.

I think that in the next years, models will become even more sophisticated to account for the many characteristics and subtleties of human behavior. Keeping up with the increasing refinement of theories and mathematical models respectively, i.e., being able to accurately estimate the model's parameters, will be a crucial aspect of future psychological research. While it is commonly straightforward to generate data based on specific parameter values, the reverse problem, namely estimating parameters given some data is hard. Neural networks, an algorithm from the field of deep learning, are known to be universal function approximators and offer a way to learn a function for this reverse problem.

I would like to conclude this thesis by emphasizing the great potential that lies in joining both psychological research and deep learning methods whenever data simulation is relatively easy and cheap. Although I only covered a few specific cognitive models in this thesis, the presented algorithm is generalizable to any multivariate dataset and therefore a large amount of other interesting psychological models. At the time of writing, most popular models in psychological research are provided with a likelihood function, which one might think drastically reduces the areas of application of the presented techniques. Yet, as shown in this thesis, *DeepInference* can be an interesting alternative way to estimate parameters even in domains where a likelihood function exists.

References

Aggarwal, C.C. (2018). Neural Networks and Deep Learning. *Springer International Publishing*.

Bishara A.J, Payne B.K. (2009). Multinomial process tree models of control and automaticity in weapon misidentification. *Journal of Experimental Social Psychology*. 45(3):524–534.

Bishop, C. M. (2011). *Pattern recognition and machine learning*. New York: Springer.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

Conrey FR, Sherman J, Gawronski B, Hugenberg K, Groom CJ. (2005). Separating Multiple Processes in Implicit Social Cognition: The Quad Model of Implicit Task Performance. *Journal of Personality and Social Psychology*. 89:469–87.

Csilléry, K., Francois, O., & Blum, M.G. (2011). abc: an R package for Approximate Bayesian Computation (ABC). *Methods in Ecology and Evolution*.

Dennis, S., & Humphreys, M. S. (2001). A context noise model of episodic word recognition. *Psychological Review, 108*(2), 452-478.

DiCiccio, T., Kass, R., Raftery, A., & Wasserman, L. (1997). Computing Bayes Factors by Combining Simulation and Asymptotic Approximations. *Journal of the American Statistical Association, 92*(439), 903-915.

Doersch, C. (2016). Tutorial on Variational Autoencoders (cite arxiv:1606.05908)

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

Gilks W. R., Richardson S. and Spiegelhalter D. J. (1996). *Markov chain Monte Carlo in Practices*, Chapman and Hall, London.

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

Gronau, Q.F., Sarafoglou, A., Matzke, D., Ly, A.,Boehm, U., Marsman, M.; Leslie, D. S., Forster, J. J., Wagenmakers, E.J., Steingroever, H. (2017). A tutorial on bridge sampling. *Journal of Mathematical Psychology*. Vol. 81. pp. 80-97.

Heathcote, A., Brown, S. D., & Mewhort, D. J. (2002). Quantile maximum likelihood estimation of response time distributions. *Psychonomic Bulletin & Review, 9*(2), 394-401.

Hoekstra, R., Morey, R. D., Rouder, J. N., & Wagenmakers, E.-J. (2014). Robust misinterpretation of confidence intervals. *Psychonomic Bulletin & Review, 21*(5), 1157-1164.

Jacoby L.L. (1991). A Process Dissociation Framework: Separating Automatic from Intentional Uses of Memory. *Journal of Memory and Language*. 30:513–541.

Jeffreys, H. (1961). *Theory of Probability*. Oxford, England: Oxford.

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*.

Kass, R. & Raftery, A. (1995) Bayes Factors. *Journal of the American Statistical Association*, 90, 773-795.

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. & Welling, M. (2013). Auto-Encoding Variational Bayes. *CoRR*, abs/1312.6114.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541-551.

Lerche, V., & Voss, A. (2017). Experimental validation of the diffusion model based on a slow response time paradigm. *Psychological Research.* Advance online publication.

Lerche, V., Voss, A., & Nagler, M. (2017). How many trials are required for parameter estimation in diffusion modeling? A comparison of different optimization criteria. *Behavior Research Methods*, *49*(2), 513-537.

Liepe, J., Barnes, C.P., Cule, E., Erguler, K., Kirk, P.D., Toni, T., & Stumpf, M.P. (2010). ABC-SysBio—approximate Bayesian computation in Python with GPU support. *Bioinformatics*.

Lindsay, D. S., & Jacoby, L. L. (1994). Stroop process dissociations: The relationship between facilitation and interference. *Journal of Experimental Psychology: Human Perception and Performance, 20*(2), 219-234.

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*

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.

Mitchell, T. M. (1997). *Machine Learning*. New York: McGraw-Hill. ISBN: 978-0-07-042807-2

Nelder, J. A., & Mead, R. (1965). A Simplex-Method for Function Minimization. *Computer Journal, 7*(4), 308-313.

Nix, D.A., Weigend, A.S. (1994). Estimating the mean and variance of the target probability Distribution. *Proceedings of IEEE International Conference on Neural Networks, 1,* 55-60.

Palestro, J. J., Sederberg, P. B., Osth, A. F., Van Zandt, T., and Turner, B. M. (2018). Likelihood-free methods for cognitive science. In Criss, A. H., editor, *Computational Approaches to Cognition and Perception*, pages 1–129. Springer International Publishing.

Payne B. (2001). Prejudice and Perception: The role of automatic and controlled processes in misperceiving a weapon. *Journal of Personality and Social psychology*. 81:181–92.

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. *Mol. Biol. Evol*. 16:1791-1798.

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

Ratcliff, R., & Rouder, J. N. (1998). Modeling response times for twochoice decisions. *Psychological Science*, 9(5), 347–356.

Ratcliff, R., & Tuerlinckx, F. (2002). Estimating parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic Bulletin & Review*, 9(3), 438–481.

Riefer D., Batchelder W. (1988). Multinomial modeling and the measurement of cognitive processes. *Psychological Review*. 95:318–339.

Ripley, B. (1996). *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press

Rouder, J. N., Speckman, P. L., Sun, D., Morey, R. D., & Iverson,G. (2009). Bayesian t tests for accepting and rejecting the null hypothesis. *Psychonomic Bulletin & Review*, 16, 225-237.

Rubin, D. B. (1984). Bayesianly justifiable and relevant frequency calculations for the applied statistician. *Ann. Statist.* 12(4), 1151-1172.

Shiffrin, R. M., & Steyvers, M. (1997). A model for recognition memory: REM—retrieving effectively from memory. *Psychonomic Bulletin & Review, 4*(2), 145-166.

Sohn, K., Lee, H., & Yan, X. (2015). Learning Structured Output Representation using Deep Conditional Generative Models. *NIPS.*

Tavaré, S., Balding, D. J., Griffiths, R. C., & Donnelly, P. (1997). Inferring coalescence times from DNA sequence data. *Genetics*, *145*(2), 505-18.

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., Dennis, S., & Van Zandt, T. (2013). Likelihood-free Bayesian analysis of memory models. *Psychological review*, *120*(3), 667-78.

Usher, M., & McClelland, J. L. (2001). The time course of perceptual choice: The leaky, competing accumulator model. *Psychological Review, 108*(3), 550-592.

Voss, A., Lerche, V., Mertens, U. K., & Voss, J. (2019). Sequential Sampling Models with Variable Boundaries and Non-Normal Noise: A Comparison of Six Models. *Psychonomic Bulletin & Review.* Advance online publication.

Voss, A., Nagler, M., & Lerche, V. (2013). Diffusion models in experimental psychology: a practical introduction. *Experimental Psychology, 60*(6), 385-402.

Yarkoni, T., & Westfall, J. (2017). Choosing Prediction Over Explanation in Psychology: Lessons From Machine Learning. *Perspectives on Psychological Science*, *12*(6), 1100–1122.

Zeiler, M. D. & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks (cite arxiv:1311.2901v3)

List of Figures

Appendix A1

Manuscript 1: ABrox – a user-friendly python module for approximate Bayesian computation with a focus on model comparison.

PLOS ONE

RESEARCH ARTICLE

ABrox—A user-friendly Python module for approximate Bayesian computation with a focus on model comparison

Ulf Kai Mertens*, Andreas Voss, Stefan Radev

Department of Psychology, Heidelberg University, Heidelberg, Germany

* ulf.mertens@psychologie.uni-heidelberg.de

OPEN ACCESS

Citation: 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. https://doi.org/10.1371/journal.pone.0193981

Editor: Yong Deng, Southwest University, CHINA

Received: May 3, 2017

Accepted: February 22, 2018

Published: March 8, 2018

Data Availability Statement: All datasets containing the simulation results are available from the Zenodo data repository (https://zenodo.org/record/1010125#.WeB0VYb-ui4).

Funding: We acknowledge financial support by Deutsche Forschungsgemeinschaft and Ruprecht-Karls-Universität Heidelberg within the funding programme Open Access Publishing (UKM).

Competing interests: The authors have declared that no competing interests exist.

Abstract

We give an overview of the basic principles of approximate Bayesian computation (ABC), a class of stochastic methods that enable flexible and likelihood-free model comparison and parameter estimation. Our new open-source software called *ABrox* is used to illustrate ABC for model comparison on two prominent statistical tests, the two-sample t-test and the Levene-Test. We further highlight the flexibility of ABC compared to classical Bayesian hypothesis testing by computing an approximate Bayes factor for two multinomial processing tree models. Last but not least, throughout the paper, we introduce *ABrox* using the accompanied graphical user interface.

Introduction

Approximate Bayesian computation (ABC) is a computational method founded in Bayesian statistics. ABC enables parameter estimation as well as model comparison when the probability of observed data under the model of consideration is unknown, or put differently, when the likelihood function is not available or computationally intractable. The idea of ABC is to bypass the computation of the likelihood by comparing model predictions with observed data, thus taking a simulation-based approach.

In psychological research, the classical null hypothesis significance testing (NHST) based on the p-value falls more and more under disrepute due to its many weaknesses [1–6]. One main criticism among others, especially stressed by Wagenmakers and colleagues [7], is the fact that p-values only depend on what is expected under the null hypothesis ($\mathcal{H}_0$) but not what is expected under the alternative hypothesis ($\mathcal{H}_1$). Hence, a small p-value only states that the observed data is unlikely under $\mathcal{H}_0$ but it might be even more unlikely under $\mathcal{H}_1$. Furthermore, the p-value is often misinterpreted (e.g. as the probability of the null hypthesis being true) [8]. As a solution, many authors suggest the use of Bayes factors as they allow to quantify evidence for both $\mathcal{H}_1$ and $\mathcal{H}_0$ and can be easily interpreted as the relative evidence of one hypothesis with respect to another hypothesis [1, 2, 4, 7, 8].

Despite the advantages of Bayes factors, a shortcoming is that their computation is often hard or even impossible which in turn narrows their applicability [9]. Wagenmakers and colleagues [5] address this aspect by stating:

> "In order to profit from the practical advantages that Bayesian parameter estimation and Bayes factor hypothesis tests have to offer it is vital that the procedures of interest can be executed in accessible, user-friendly software package"
>
> (p. 4).

They provide such a software package, called JASP [10], in order to ease the process of Bayesian hypothesis-testing. With JASP, they continue,

> "[. . .] users are able to conduct classical analyses as well as Bayesian analyses, without having to engage in computer programming or mathematical derivation"
>
> (p. 4).

Yet, users of JASP are restricted when it comes to the specification of user-defined prior distributions or the application of non-standard tests. In addition, users are limited in the application of Bayesian hypothesis-testing in the sense that JASP always assumes a point null hypothesis and thus the models being compared are always nested. Although such a scenario is undoubtedly the most common one in psychological research, it is desirable to extend the methods for Bayesian model comparison to allow for non-nested model comparison. *ABrox* exactly addresses these issues by providing a flexible toolbox to estimate Bayes factors in areas where an exact mathematical solution is not available.

The outline of the paper is as follows. First, we will give an introduction to ABC with a focus on model comparison (technical details on ABC can be found elsewhere, [11, 12] or [13]). We will then introduce our software called *ABrox* and demonstrate its validity by analysing two models. For this purpose, we choose the Bayesian two-sample t-test and the Levene test. Although there exists a bayesian version of the Levene test [14], a default Bayes factor is only available for the two-sample t-test [15]. Lastly, we show the flexibility of *ABrox* by comparing two multinomial processing tree models applied to the weapon-misidentification task [16, 17].

Approximate Bayesian computation for model selection

As previously mentioned, ABC can be used for both parameter estimation and model comparison. If one is interested in the latter, the general idea of ABC methods is to test how well simulated data from different models can mimic the observed data. If a model does not represent the true data-generating process of the observed data reasonably well, then simulated data from this model will not resemble the observed data adequately. Imagine for instance that some data follow a quadratic trend. Trying to simulate such data with a model assuming a linear trend will fail most of the time. After multiple runs of the algorithm, the models are compared by counting the number of times each one was able generate data that resemble the observed data sufficiently well. One common algorithm is the ABC rejection algorithm for model selection, in its basic form introduced by Rubin (1984) [18] (see explanation below):

1. Draw m^* from the prior $P(m)$
2. Sample θ^* from the prior $P(\theta \mid m^*)$
3. Simulate a candidate dataset $D^* \sim f(D \mid \theta^*, m^*)$
4. Compute the distance. If $d(D_0, D^*) \leq \epsilon$, accept (m^*, θ^*), otherwise reject it.
5. Return to 1 until k particles are accepted.

First (step 1), one of the models under consideration (m^*) is picked based on the (prior-) probabilities of the models. In step 2, given the model, parameters (θ^*) from the prior distributions of the parameters from the chosen model m ($P(\theta \mid m^*)$) are drawn. Using model and parameters, data (D^*) is simulated in step 3 and is then compared with the observed data in step 4. For this purpose, a distance measure d has to be defined that quantifies the dissimilarity of simulated and observed data. If the distance d exceeds a predefined threshold (ϵ), the so-called particle containing both model indicator and the sampled parameters (m^*, θ^*) is accepted and stored. The process is repeated N times until k particles have been accepted. The number of accepted particles is then divided by N to get a marginal posterior distribution of a model (m') stating how likely the model is given the data at hand (D_0) [11].

$$P(m' \mid D_0) \approx \frac{\#\ accepted\ \ particles(m', .)}{N} \tag{1}$$

With this metric, we can calculate the Bayes factor (BF) as the ratio of marginal likelihoods ($p(D_0 \mid m_1)/p(D_0 \mid m_2)$). The Bayes factor ($BF_{12}$) describes the amount of change from prior odds to posterior odds after the data has been observed [5].

$$\frac{p(D_0 \mid m_1)}{p(D_0 \mid m_2)} = \frac{p(m_1 \mid D_0)}{p(m_2 \mid D_0)} \times \frac{p(m_2)}{p(m_1)} \tag{2}$$

In Eq 2, $p(m_2)/p(m_1)$ is referred to the model prior odds, indicating how likely one model (m_2) is relative to the other (m_1) before seeing the data. $p(m_1 \mid D_0)/p(m_2 \mid D_0)$ is the ratio of marginal posterior distributions as approximated via ABC.

Summary statistics

In step 4 of the aforementioned algorithm, the distance could be computed by taking every data point in the observed and simulated dataset into account. While computing the distance based on every data point is reasonable theoretically, it is infeasible in most settings. If the datasets are large, then there is always a deviation from simulated data to observed data (the distance would only be zero if every simulated data point was exactly equal to the corresponding observed one). Hence, the acceptance rates would be extremely low if the distance threshold (ϵ) was too strict. However, simply increasing the distance threshold to account for the low acceptance rates would not be a reasonable decision either because exceedingly large thresholds distort the approximation [19]. In order to account for these problems, the distance d between observed and simulated data is often based on a summary statistic $s(D)$ (e.g., mean and (co)variance). While this procedure of computing summary statistics suffers less from the *curse of dimensionality*, it often comes with a loss of information. Didelot et al. [12] showed that insufficient summary statistics, not capturing the whole information of the raw data, can lead to inconsistencies. Hence, the ABC model selection may fail to recover the true model. The challenge is to balance out consistency and information loss [12]. Therefore, the approximation of the true Bayes factor depends on the summary statistic capturing the necessary information in the data.

The threshold ϵ

Besides picking an appropriate summary statistic, the notion of simulated data being "close enough" to observed data has to be defined. Users interested in ABC have to set a specific threshold value ϵ based on a distance metric in order to decide whether a particle (m^*, θ^*) should be accepted or discarded. The smaller the threshold ϵ, the better the approximation of the marginal posterior distribution to the true posterior distribution [11]. If the threshold is

too liberal, too many particles fall below the threshold and the resulting posterior distribution is not a good approximation. If, on the other hand, the threshold is too strict, it might take very long to find the required number of particles that lead to distances smaller than ϵ and the procedure becomes computationally inefficient. The challenge is to find an ϵ that leads to a good approximation while retaining a tolerable running time of the algorithm. To tackle the problem of finding a good threshold, Beaumont, Cornuet, Marin, and Robert [20] proposed to select ϵ as the k^{th} percentile of computed distances prior to the actual start of the algorithm (with k being a small value such as 5 or 1).

Introducing *ABrox*

We developed *ABrox* as an open-source python module which enables approximate Bayesian model comparison and parameter estimation. With *ABrox*, we introduce a graphical user interface (GUI) which is designed to be used as a tool for all-purpose ABC, making the methods much more accessible to researchers interested in applying ABC. The GUI is especially useful for researchers without much prior knowledge of ABC. All necessary steps are separated into small chunks to keep the structure clear and simple. Users can also save and load their projects which allows for good maintenance and easy sharing with collaborators. Moreover, prior distributions are automatically visualized for a better user-experience. *ABrox* facilitates high flexibility regarding the specification of mathematical models and is especially useful if one is interested in Bayesian model comparison where none of the traditional methods are suitable, usual assumptions underlying those methods are not met, models of interest are not nested, or software does not permit to change prior settings. Detailed instructions on how to install *ABrox* can be found online (see https://github.com/mertensu/ABrox).

Comparison to other software packages

The main advantages of *ABrox* compared to other software packages such as the R-package abc [21] or the Python module ABC-SysBio [22] are the domain-independence, meaning that it is not designed for a specific field of research, and its ease of use due to the GUI. The software *DIYABC* [23] is another attempt at simplifying the use of ABC by providing a GUI for experimental biologists. *DIYABC* provides a convenient way to conduct ABC inferences about population history using SNPs (Single Nucleotide Polymorphism), DNA sequence and microsatellite data, the areas of research it was designed for. *DIYABC* does not, as *ABrox*, necessitate prior experience with a programming language. Due to its focus on specialized fields of research, the process of simulating data and computing summary statistics is handled by the program itself which is a helpful feature for users. However, it is limited to a few specific fields of research.

The ABC reference table

The *reference table* in ABC is the starting point for most algorithms. It is a large table containing in each row the the model index (only used for model comparison), summary statistics of a simulated dataset, the parameter(s) used for simulating the data and the distance with respect to the observed summary statistics. The *reference table* contains as many rows as there are simulated datasets. *ABrox* usually generates the *reference table* internally but it also possible to import an external *reference table* stored in a *comma separated value* (csv) file. The header of csv-file needs to satisfy the the following conditions. There has to be a column named *idx* containing the model index. Furthermore, if there are k summary statistics, each value has to be stored in separate columns named *s_* followed by a number (usually 1 to k). The same logic applies to the parameters where each parameter column name should start with *p_*. In addition

to the columns containing information about summary statistics and parameter prior distributions, there has to be a column named *distance* containing the distance to the observed summary statistics.

Features of *ABrox*

Implemented algorithms. For model comparison, either the rejection algorithm or a random forest approach can be chosen. Random forests [24] are a supervised learning algorithm used extensively in machine learning. Given a sample of input-output pairs, the goal of a random forest is to learn a mapping from the input to the output by training multiple decision trees (a standard non-parametric algorithm for classification and regression) and aggregating their decision function outputs. It incorporates the techniques bootstrap aggregation and random feature selection to reduce over-fitting and thus increases the predictive generalization of the model. In the context of ABC, random forests learns to map summary statistics to either model indices or parameters of a model. Readers interested in more details should consult the work by Pudlo et al. (2015) [25]. Besides model comparison, parameter estimation can be conducted either via the basic rejection algorithm or a Markov chain Monte Carlo (MCMC) based algorithm by Wegmann [26].

Cross-validation. *ABrox* allows for cross-validating the results from both model comparison and parameter estimation using leave-one-out cross-validation as used in [21]. In leave-one-out cross-validation, a randomly chosen simulated summary statistic is treated as pseudo-observed summary statistic and an ABC algorithm is run in order to estimate parameters or a Bayes factor (posterior model probability). This procedure is repeated N times. In the case of parameter estimation, *ABrox* then provides a prediction error (see Eq 3) stating how different the predicted parameters ($\tilde{\theta}$) are from the true parameters (θ).

$$E_{pred} = \frac{\sum_i (\widetilde{\theta}_i - \theta_i)^2}{Var(\theta_i)} \tag{3}$$

Because the information provided by Eq 3 is limited, an additional pdf-file is automatically saved. This file contains one plot for each parameter in which the estimated parameter is plotted against the true parameter.

When performing model comparison, *ABrox* presents a confusion matrix stating how often the pseudo-observed summary statistics could be correctly classified to the model they were generated from. In this matrix, the diagonal elements should be large and the values in off-diagonal elements should be small. The confusion matrix is also stored in a heatmap-style as a pdf-file to ease the interpretation of the results.

The building blocks of *ABrox*

To calculate a Bayes factor, the user needs to provide several pieces of information.

1. the data to be analyzed (optional)
2. the prior distributions for all parameters of all models.
3. functions to simulate data from the models.
4. a function to compute summary statistics from data.
5. a function to compute the distance between summary statistics of observed and simulated data (optional).

The functions to compute the summary-statistic and the distance are the same for all compared models. Prior distributions are internally stored as python dictionaries whereas simulation-, summary-, and distance-function are user-defined functions.

Data import in *ABrox*. The data tab in *ABrox* shown in S1 Fig is used to import an external data-file. The file should be stored as a comma-separated value file (*.csv*). The imported data can be inspected and modified in the data tab. Note that users are able to access the data in the embedded Python Console at the bottom by typing *data*.

The prior distribution in *ABrox*. Prior distributions for the parameters of each model have to be specified. S2 Fig shows the prior settings tab of *ABrox* with a parameter called *x* and its corresponding prior distribution showing a standard normal distribution. In the current version of *ABrox*, a total of nine different prior distributions are available.

The simulate-function in *ABrox*. In a next step, the user has to write a Python function for each model that simulates data. The simulated dataset has to be in the form of a NumPy array [27]. The simulate-function can be written in the *simulate* tab (S3 Fig). Note that the names of the functions should not be changed and default to *simulate*. The function has only one argument (*params*) which is a dictionary with keys corresponding to the names of the parameters specified and the values containing one sampled value from the respective prior distribution of the parameter. Thus, each parameter value can easily be accessed by the parameter's name.

The user has to take care that the format of the imported (observed) data is identical to the format of the simulated data. Put differently, the return type of the simulate-function has to be the same as the type of the imported data. If, for instance, the observed data is a multidimensional array with 100 rows and two columns, the simulate function has to return a multidimensional array of the same shape. If this is not the case, *ABrox* will throw an error message.

The summary and distance-function in *ABrox*. After the specification of each model (priors and *simulate*-function), a new Python function to calculate the summary statistics from the data has to be written. The function takes as input the output of the simulate function specified in the previous step. The function should return the summary statistics. The summary statistics can be a scalar or a vector.

In *ABrox*, the user can optionally write a user-defined distance function for computing the distance between observed summary statistics and simulated summary statistics. If the function is left empty, *ABrox* automatically uses the default distance metric which is the Euclidean distance scaled on the median absolute deviation (MAD).

The settings tab in *ABrox*. In the settings tab (S4 Fig), the working directory has to be set. Next, the type of analysis, that is model comparison or parameter estimation, has to be selected. Note that *ABrox* allows to compare more than two models by simply adding more models to the *Project Tree*. Each model needs to be specified with a unique *simulate*-function and the corresponding prior distribution(s) for the parameter(s). If the rejection algorithm is selected, *ABrox* calculates a matrix containing approximate Bayes factors. The random forest approach, on the other hand, calculates posterior model probabilities.

In the case of parameter estimation, a summary of the posterior distributions of the parameters is returned. In addition, a data frame with all samples from the posterior of each parameter is saved in the working directory.

The *Console Panel* at the bottom of S4 Fig shows the progress of the computation and informs the user as soon as the computation is finished. The results are stored inside a python variable which can be accessed from the integrated *Python Console*.

The user interface of *ABrox* generates a python script in the working directory with all the configurations specified in the GUI. This feature of saving a python-file with all the configurations is especially useful if users want to use *ABrox* within the Python framework.

Besides the obligatory settings, we added the option to generate pseudo-observed data from a model instead of having to import data (*Model Test Settings*). By choosing this option, one can simply check if the approximate Bayes factor favors the model the data have been simulated from or if the parameters chosen for the pseudo-observed data can be estimated accurately.

Application example 1: The independent-samples t-test

In the following, we will first show how to compute an approximate Bayes factor for an independent samples t-test assuming normally distributed data with equal variances. The corresponding *ABrox* project file can be downloaded at https://github.com/mertensu/ABrox/tree/master/project_files. We chose the t-test as the first example since it is one of the most common tests in statistics. Furthermore, the t-test qualifies for a comparison of the approximate Bayes factor and the default Bayes factor as there is already software available to compute a default Bayes factor for the t-test [15]. For all of the following examples, we refer to *Bayes factor* as the Bayes factor expressing support for the alternative hypothesis ($\mathcal{H}_1$) over the null hypothesis ($\mathcal{H}_0$), which is usually indicated as BF_{10}.

Model specification

Simulation. In order to adapt the default Bayesian t-test for *ABrox*, we have to consider how data are simulated from both $\mathcal{H}_0$ and $\mathcal{H}_1$. In a first step, we know that data from both samples are drawn from a normal distribution with a specific standard deviation (σ) and mean (μ). $\mathcal{H}_0$ assumes that there is no difference in means ($\mu_1 = \mu_2$). As a consequence, we restrict $\mathcal{H}_0$ to generate two normally distributed samples with the same mean. The model representing $\mathcal{H}_1$, in comparison, is allowed to simulate data such that the two means differ.

Summary statistic. After that, a useful summary statistic has to be chosen. In the t-test scenario, we choose the empirical effect size Cohen's *d* as the summary statistic on which the comparison between observed data and simulated data is based [28]. Note that by comparing the data only by the effect size *d*, we do not get any information about other parameters that might be of interest in a parameter estimation scenario (such as the overall standard deviation) but focus only on model comparison.

Parameters and priors. Following Rouder et al. [15], $\mathcal{H}_1$ gets a prior for the population effect size Cohen's *d*. Whereas $\mathcal{H}_0$ does not have a parameter and therefore no prior, we put a Cauchy prior with scale $\gamma = 0.707$ on the effect size *d* for $\mathcal{H}_1$ (see Fig 1).

Distance function. In order to assess the similarity between the observed and simulated Cohen's *d*, the squared difference is chosen as the distance metric.

Simulation results

In the following section, we provide simulation results focusing on the similarity between the default Bayes factor and the approximated Bayes factor. We simulated N = 1000 data sets with an Cohen's *d* varying uniformly between no effect ($d = 0$) and a moderate effect of $d = 0.5$. For each generated data set, we computed both the default Bayes factor using the R package BayesFactor (version 0.9.12.2) [29], as well as the approximated Bayes factor using *ABrox*. Furthermore, we varied the sample size between $N_{total} = 50$ (25 per group) and $N_{total} = 200$ (100 per group).

Support for $\mathcal{H}_1$. Results in Fig 2 indicate that, in cases where there is support for $\mathcal{H}_1$, the approximation to the default Bayes factor is extremely good. There is a strong correlation when the default Bayes factor lies somewhere between anecdotal and very strong support for the $\mathcal{H}_1$. In areas where there is extreme support for $\mathcal{H}_1$ (to the right of very strong support in

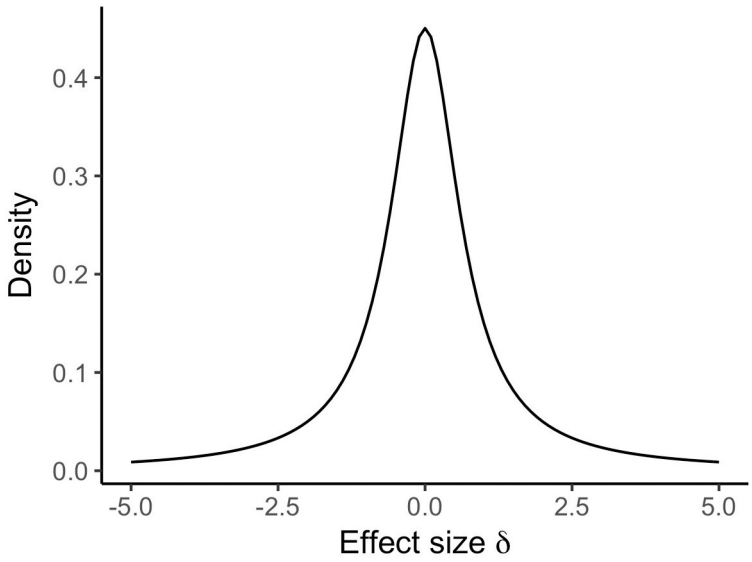

Fig 1. Cauchy distribution with a scale of $\gamma = 0.707$ used as the prior distribution.

https://doi.org/10.1371/journal.pone.0193981.g001

Fig 2, the accuracy of the approximation decreases somewhat). Even though this pattern of increasing differences is not optimal, it is not a serious issue since it does not matter that much how extreme the support for $\mathcal{H}_1$ actually is. Note that if the null hypothesis ($\mathcal{H}_0$) did not cross the threshold at all (no data could be simulated that were close enough to the observed data), this indicates extreme evidence for $\mathcal{H}_1$. In this case, no ratio could be computed (division by zero). In such cases, we set the approximated Bayes factor to a value of 10000 (resulting in ln (10000) = 9.21) which is an arbitrary choice simply reflecting a very large number.

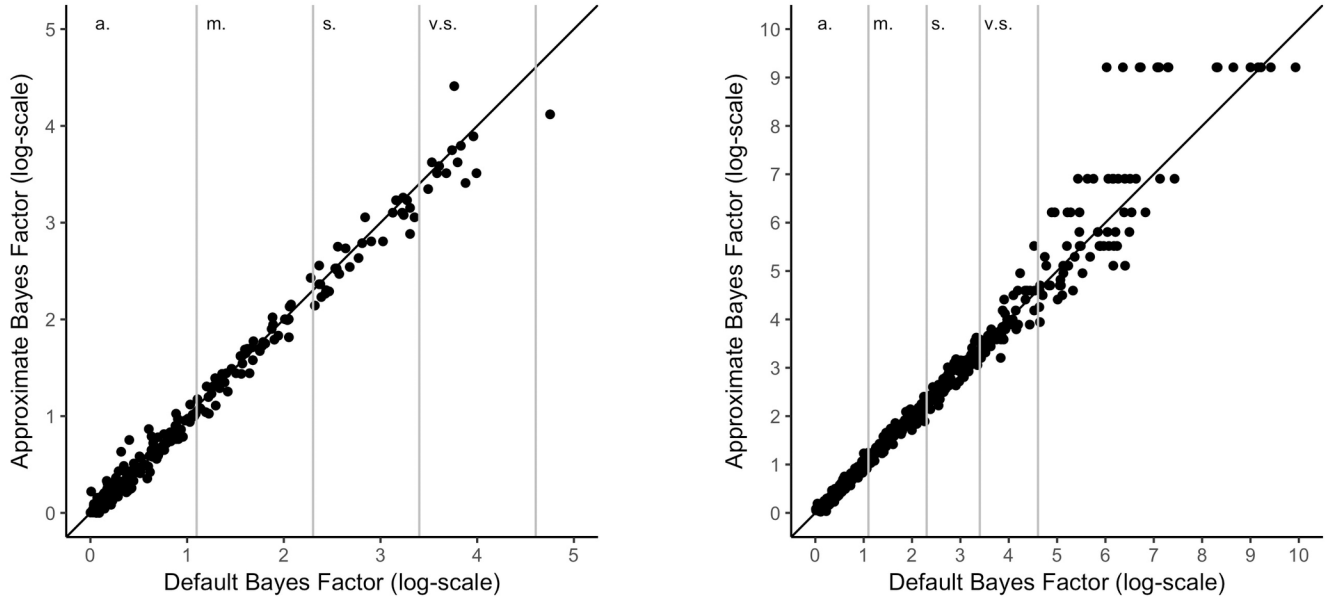

Fig 2. Relationship of default Bayes factor and approximated Bayes factor for a two-sample t-test with support for the alternative hypothesis; left: N = 50, right: N = 200. *Note.* a. = anecdotal. m. = moderate. s. = strong. v.s. = very strong evidence.

https://doi.org/10.1371/journal.pone.0193981.g002

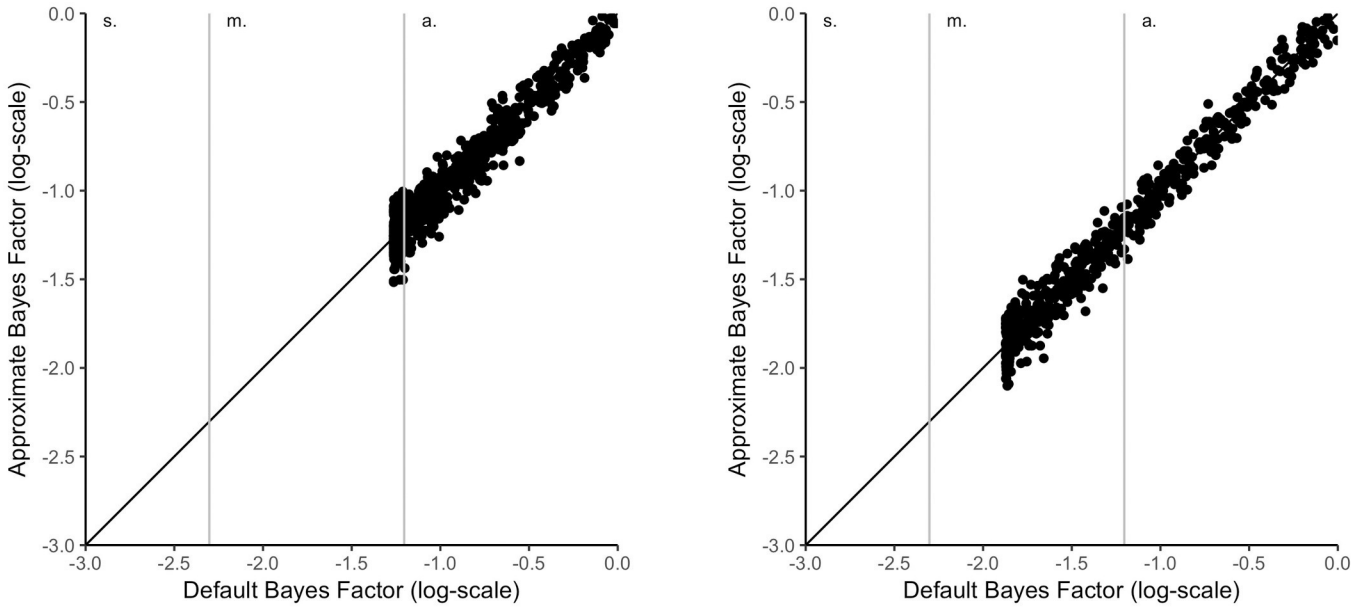

Fig 3. Relationship of default Bayes factor and approximated Bayes factors for a two-sample t-test with support for the null hypothesis; left: N = 50, right: N = 200. *Note*. a. = anecdotal. m. = moderate. s. = strong. v.s. = very strong evidence.

https://doi.org/10.1371/journal.pone.0193981.g003

Support for $\mathcal{H}_0$. Fig 3 shows the relationship between the Bayes factors when there is support for $\mathcal{H}_0$. There are two things to notice. First, there is again a very high correlation between the default and the approximated Bayes factor. Secondly, a lower limit of the default Bayes factor can be seen if the difference in means is exactly zero. One might very well ask why a mean difference of zero does not result in a Bayes factor with strong evidence for $\mathcal{H}_0$. The reason for this is that the prior for $\mathcal{H}_1$ also puts mass on an effect of zero so that it is still possible for the model to account for the findings. However, the more vague the prior (increasing the scale) or the larger the sample sizes, the lower the limit of the Bayes factor becomes. The lower limit of the Bayes factor in the left plot of Fig 3 is ($log(BF_{10})$ = -1.26) and ($log(BF_{10})$ = -1.87) for the right plot.

Application example 2: A Bayesian Levene test

In this section, we show the flexibility of ABC model comparison by implementing a Bayesian Levene test for two samples. The corresponding *ABrox* project file can be downloaded at https://github.com/mertensu/ABrox/tree/master/project_files. The Levene test checks whether variances among different samples are equal (homogeneous). Assuming two samples, the null hypothesis ($\mathcal{H}_0$) is specified as:

$$H_0 : \sigma_1^2 = \sigma_2^2 \tag{4}$$

Using the `ABrox` framework, we construct two models. $\mathcal{H}_0$ is restricted in the sense that both population variances are homogeneous. For the alternative hypothesis ($\mathcal{H}_1$) however, the variances between both groups are allowed to differ. Note that the $\mathcal{H}_0$ has one parameter (the identical variance within groups) whereas $\mathcal{H}_1$ has two parameters (one variance for each sample). Although this parameterization is valid, we reparameterized both models such that $\mathcal{H}_0$ has no parameter (we fixed the variance in both groups to 1).

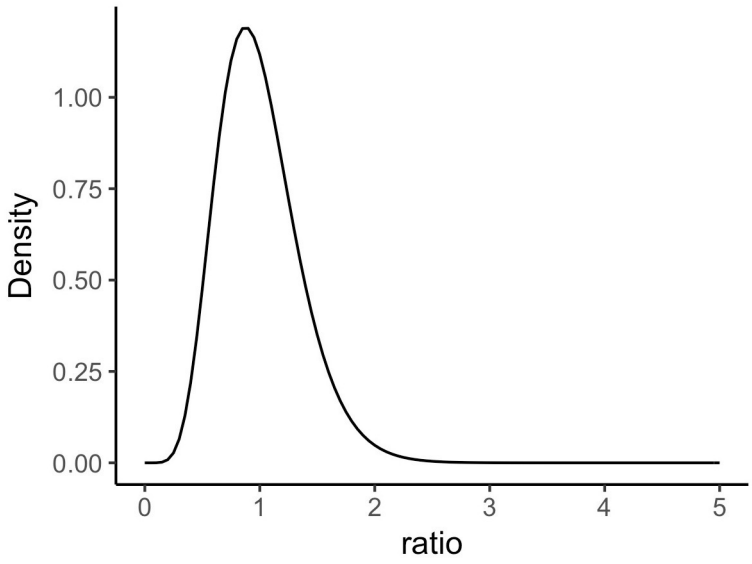

Fig 4. Gamma distribution used as the prior for the ratio of standard deviations. Parameters are k = 8 and θ = 0.125.

https://doi.org/10.1371/journal.pone.0193981.g004

Following the approach for the two-sample t-test, we chose for $\mathcal{H}_1$ a prior specifying the ratio between the two variances. A prior for a ratio should have a mean of 1 and has to be strictly positive. In order to meet both conditions, we chose a gamma distribution with shape k = 8 and scale θ = 0.125. The distribution has the form shown in Fig 4.

Simulation results

For the Levene test, we simulated datasets with a ratio of variances varying uniformly between 1 (homogeneity) and 2. For each of two sample sizes (N = 50 and N = 200), we simulated 1000 datasets.

Fig 5 shows the relationship of *p*-values from the Levene test and approximate Bayes factors. The vertical lines in Fig 5 represent the critical value of $\alpha = 0.05$. All *p*-values left to this line would lead to the decision of accepting $\mathcal{H}_1$ (the variances are heterogeneous). The horizontal lines corresponds to a BF_{10} of 1. Therefore, all values above this line would indicate support for $\mathcal{H}_1$, whereas all Bayes factors below this line would indicate support for the $\mathcal{H}_0$ (homogeneous variances). Although both *p*-value and Bayes factor often lead to the same decision about the homogeneity of variances, according to the left plot of Fig 5, there are many datasets where decisions differ, if sample sizes are small. The area above the horizontal line and to the right of the vertical line shows datasets where there is support for heterogeneity of variances based on the Bayes factor. However, the result of the Levene test is not significant ($p > .05$).

Application example 3: Multinomial processing trees

In this last example, we demonstrate the usefulness of ABC model comparison by computing approximate Bayes factors for multinomial processing tree models (MPT) [30]. MPT models are models for categorical data that are often used in the cognitive and social sciences. These models assume that information processing follows one of several possible paths in a so-called processing tree. For each fork in the path, probabilities are estimated. Note that recently an alternate approach to compute Bayes factors for MPT models was published [31].

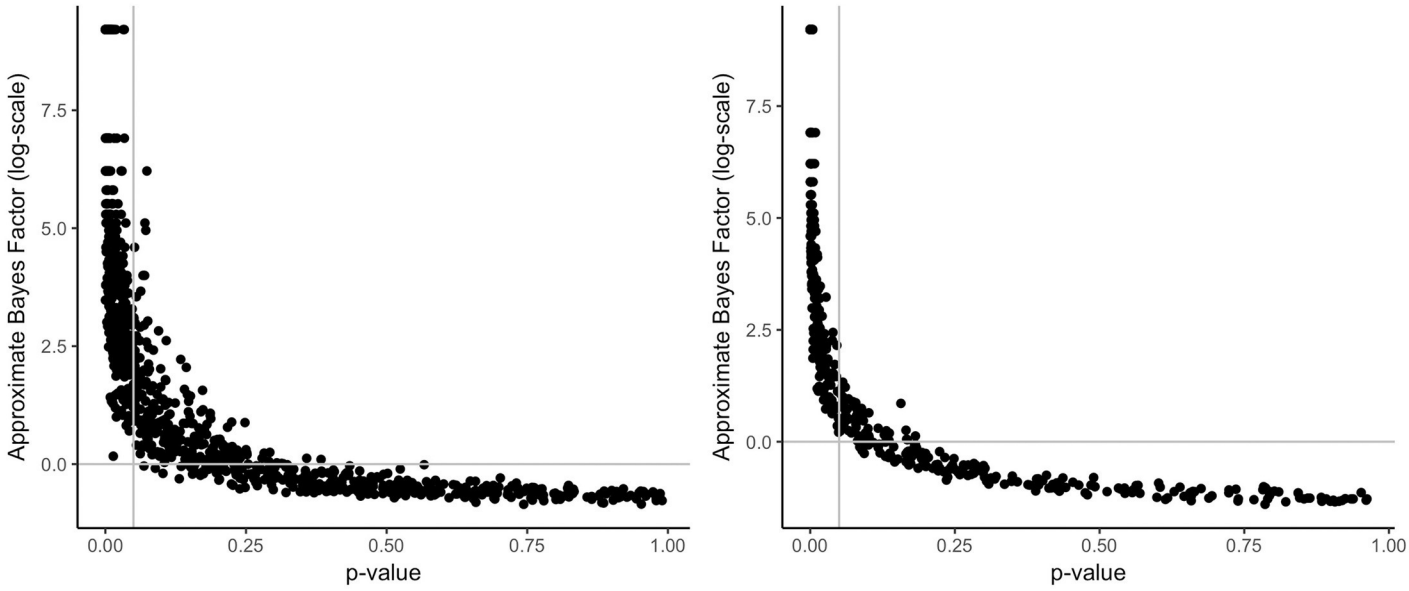

Fig 5. Relationship of *p*-values and approximated Bayes factors for a Levene test; left: N = 50, right: N = 200.

https://doi.org/10.1371/journal.pone.0193981.g005

For this example, we choose two prominent models that aim to determine the role of control and automaticity in the so-called weapon misidentification task [16, 17]. In this task, participants have to identify either guns or tools under high time pressure after being primed with either white or black faces. If a stereotype effect is apparent, the probability of erroneous "gun" responses is increased after black primes and vice versa. In this paradigm, it is an important question what role automatic and controlled processes play for the response selection. The Process Dissociation Model [32, 33], depicted in Fig 6, assumes that controlled processing dominates automaticity. A correct response is given whenever controlled processes prevail (probability *C*). Automatic processing, on the other hand, determines the response only if controlled processing fails *(1-C)*. The increased probability of an stereotype-consistent answer is

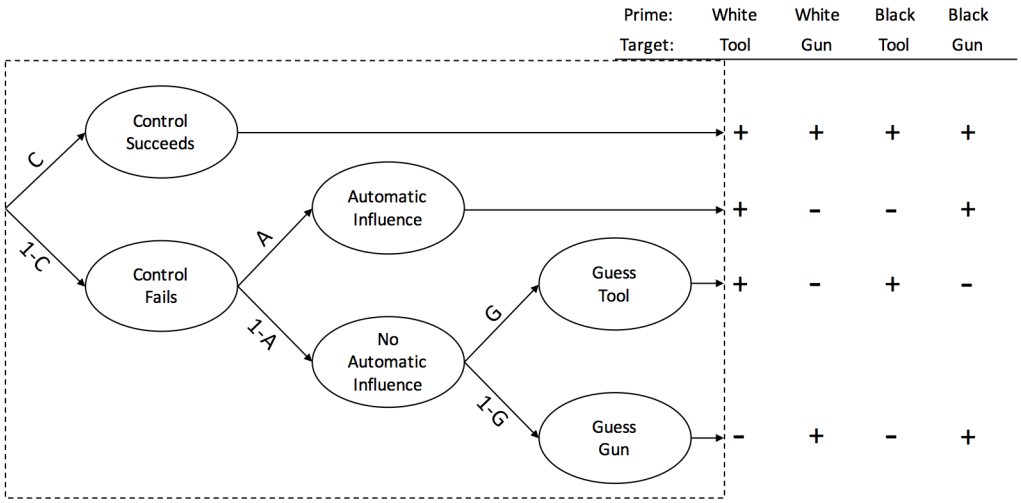

Fig 6. The Process Dissociation Model with guessing. Branches lead to correct (+) and incorrect (-) responses.

https://doi.org/10.1371/journal.pone.0193981.g006

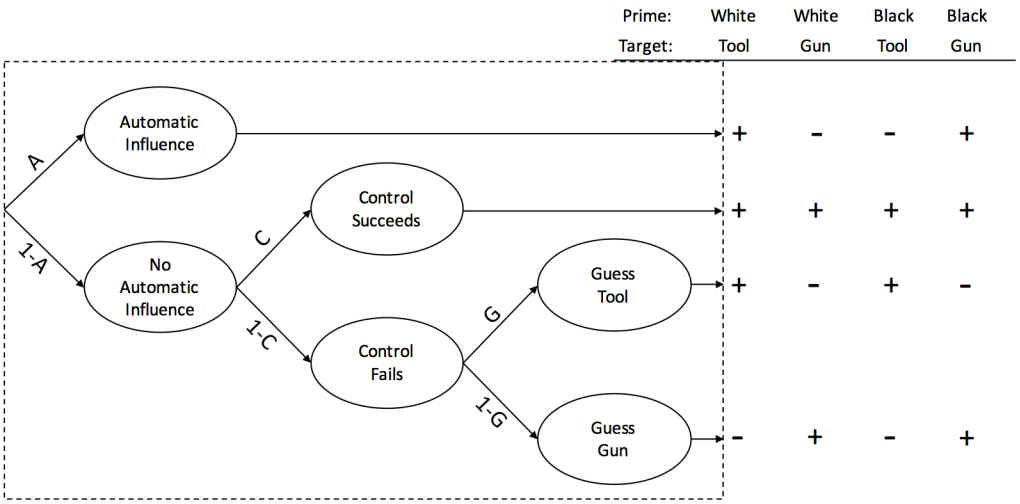

Fig 7. The Stroop Model with guessing. Branches lead to correct (+) and incorrect (-) responses.

https://doi.org/10.1371/journal.pone.0193981.g007

captured in parameter *A*. A sterotype-consistent response is only correct for conditions where a gun is presented after the prime of a black face and a tool is presented after the prime of a white face. The Stroop Model [34], however, claims a reversed order of automatic and controlled processes. (see Fig 7). First, the automatic influence of a prime determines whether a stereotype-consistent response is given. With probability *(1-A)*, there is no automatic influence and controlled processing becomes relevant [16, 17].

Since MPT models can be quite complex but the data (or the simulated output of the models) only consists of counts, it is often necessary to run experiments with one or more experimental conditions in order to estimate parameters adequately. Therefore, we compare both models by implementing the design of Lambert et al. (2003) [35]. The authors of this study were interested in whether stereotypical responses are more pronounced in private settings or anticipated public settings. It was hypothesized, based on previous findings, that the anticipated public condition, in which participants were told that they had to talk to others about their results after the experiment, leads to more stereotype-consistent behavior [35]. Following Bishara and Payne (2009) [17], both models have 11 free parameters in total. Whereas the probability of controlled processing (*C*) is assumed to vary only between both conditions (private and anticipated public), the probability that the automatic response is stereotype-consistent(*A*) is allowed to vary by condition (public vs. private), prime (white vs. black) and target gender (female and male). Finally, there is one additional guessing parameter *G* that does not vary by condition.

Prior distributions

For each of the parameters (i.e., the path probabilities), we have to set a prior distribution. Based on the parameter estimates presented in Bishara and Payne (2009) [17], we use mildly informed prior distributions. According to those results, parameter *A*, representing the probability of an stereotype-consistent answer, seems to be quite small. Therefore, a beta distribution placing more weight on small values is chosen ($\alpha = 2, \beta = 10$). Controlled processing (parameter *C*), occurs in about half of the trials which is why we chose a symmetric beta distribution capturing this information ($\alpha = 3, \beta = 3$). Note, that by choosing rather small values for both α and β for the prior of parameter *C*, we indicate the uncertainty regarding the parameter. In

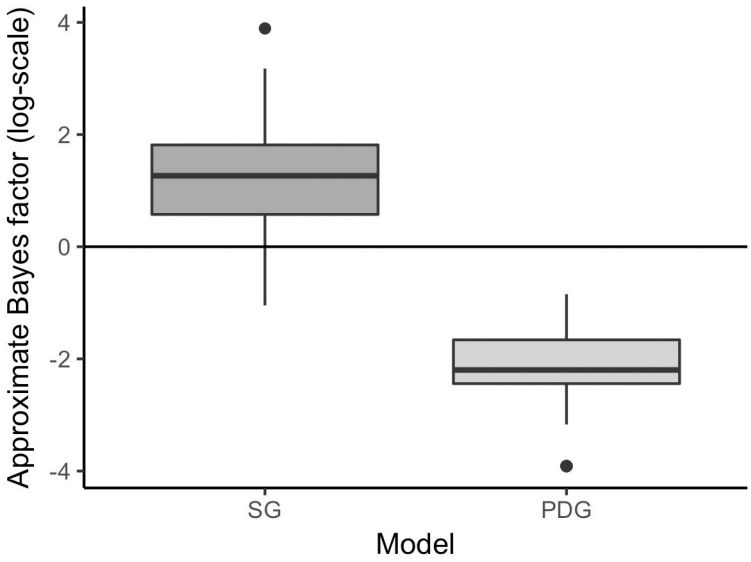

Fig 8. Approximate Bayes factors (log-scale) expressing support for the Stroop Model. Data are simulated from the Stroop Model (left) or the Process Dissociation Model (right).

https://doi.org/10.1371/journal.pone.0193981.g008

contrast, when dealing with the guessing parameter G, we restrict it to more narrow range around 0.5. This assumption is expressed in a beta distribution with parameters $\alpha = 10$ and $\beta = 10$.

Simulation results

In the previous simulations, we compared the Approximate Bayes factor with the default Bayes factor. For this last example, we generate 100 pseudo-observed datasets for each of the two models and compute the approximate Bayes factor. For each simulated dataset, we draw the parameters from beta distributions. The three distributions are chosen to mimic common parameter estimates (e.g. [17]).

- $A \sim \text{Beta}(\alpha = 2, \beta = 10)$
- $C \sim \text{Beta}(\alpha = 60, \beta = 40)$
- $G \sim \text{Beta}(\alpha = 50, \beta = 50)$

The results shown in Fig 8. For the log-Bayes factors, positive values express the support for the Stroop Model. The left boxplot in Fig 8 shows the approximate Bayes factor when the data are simulated from the Stroop Model. As expected, most of the approximate Bayes factors are supporting the model the data have been generated from (in this case the Stroop Model). In a few cases, the approximate Bayes factor favored the PDG Model. These false decisions might be due to the fact that the threshold was set rather loose (0.05). The boxplot on the right side of Fig 8 shows the approximate Bayes factors expressing support for the Stroop Model when the data are simulated from the Process Dissociation Model. Thus, values below zero indicate support for the Process Dissociation Model. The pattern of results is similar to the one on the left-hand side. All calculated Bayes factors favor the Process Dissociation Model.

Concluding remarks

This article introduces the reader to *ABrox*, a python package for approximate Bayesian computation with a user-friendly interface. We demonstrated the similarity of approximate and default Bayes Factor on two prominent statistical tests and demonstrated the flexibility of ABC on a comparison of multinomial processing tree models. *ABrox* defaults to the rejection algorithm for both parameter inference and model choice. However, more sophisticated algorithms are implemented. For parameter inference, a Markov chain Monte Carlo (MCMC) based algorithm by Wegmann [26] is available. Concerning model comparison, there is also the option to use random forests to calculate the posterior model probabilities [25]. Moreover, we plan to extend the set of available machine-learning algorithms for ABC by adding neural networks to the toolbox. With *ABrox*, we hope to reduce the burden for researchers to actually apply ABC methods in their labs.

Supporting information

S1 Fig. The data tab.
(PNG)

S2 Fig. Prior specifications in *ABrox*; example of a standard normal distribution.
(PNG)

S3 Fig. Function to simulate data for a model.
(PNG)

S4 Fig. The settings tab in *ABrox*.
(PNG)

Author Contributions

Conceptualization: Ulf Kai Mertens.

Methodology: Ulf Kai Mertens, Stefan Radev.

Software: Ulf Kai Mertens, Stefan Radev.

Supervision: Andreas Voss.

Visualization: Ulf Kai Mertens.

Writing – original draft: Ulf Kai Mertens.

Writing – review & editing: Ulf Kai Mertens.

References

1. Berger JO, Wolpert RL. In: Gupta SS, editor. Chapter 3: The Likelihood Principle and Generalizations. vol. Volume 6 of Lecture Notes–Monograph Series. Hayward, CA: Institute of Mathematical Statistics; 1988. p. 19–64.
2. Nickerson RS. Null hypothesis significance testing: A review of an old and continuing controversy. Psychological Methods. 2000; 5(2):241–301. https://doi.org/10.1037/1082-989X.5.2.241 PMID: 10937333
3. Wagenmakers EJ. A practical solution to the pervasive problems of p values. Psychonomic bulletin & review. 2007; 14(5):779–804. https://doi.org/10.3758/BF03194105
4. Dienes Z. Bayesian Versus Orthodox Statistics: Which Side Are You On? Perspectives on Psychological Science. 2011; 6(3):274–290. https://doi.org/10.1177/1745691611406920 PMID: 26168518
5. Wagenmakers EJ, Marsman M, Jamil T, Ly A, Verhagen AJ, Love J, et al. Bayesian Statistical Inference for Psychological Science. Part I: Theoretical Advantages and Practical Ramifications. Psychonomic Bulletin & Review. in press;.

6. Wagenmakers EJ, Love J, Marsman M, Jamil T, Ly A, Verhagen AJ, et al. Bayesian Statistical Inference for Psychological Science. Part II: Example Applications with JASP. Manuscript submitted for publication. 2016;.
7. Wagenmakers EJ, Verhagen J, Ly A, Matzke D, Steingroever H, Rouder JN, et al. In: The Need for Bayesian Hypothesis Testing in Psychological Science. Psychological Science Under Scrutiny: Recent Challenges and Proposed Solutions; In Press.
8. Wasserstein RL, Lazar NA. The ASA's Statement on p-Values: Context, Process, and Purpose. The American Statistician. 2016; 70(2):129–133.
9. Vandekerckhove J, Matzke D, Wagenmakers EJ, Busemeyer J, Wang Z, Townsend J, et al. Model Comparison and the Principle of Parsimony. In: Oxford Handbook of Computational and Mathematical Psychology; 2014.
10. JASP Team T. JASP (Version 0.8.0.1)[Computer software]; 2017. Available from: https://jasp-stats.org/ .
11. Toni T, Welch D, Strelkowa N, Ipsen A, Stumpf MPH. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. Journal of The Royal Society Interface. 2009; 6(31):187–202. https://doi.org/10.1098/rsif.2008.0172
12. Didelot X, Everitt RG, Johansen AM, Lawson DJ. Likelihood-free estimation of model evidence. Bayesian Analysis. 2011; 6(1):49–76. https://doi.org/10.1214/11-BA602
13. Grelaud A, Robert CP, Marin JM, Rodolphe F, Taly JF. ABC likelihood-free methods for model choice in Gibbs random fields. Bayesian Anal. 2009; 4(2):317–335. https://doi.org/10.1214/09-BA412
14. Statisticat, LLC. LaplacesDemon Tutorial; 2016. Available from: https://web.archive.org/web/20150206004624/http://www.bayesian-inference.com/software.
15. Rouder JN, Speckman PL, Sun D, Morey RD, Iverson G. Bayesian t tests for accepting and rejecting the null hypothesis. Psychonomic Bulletin & Review. 2009; 16(2):225–237. https://doi.org/10.3758/PBR.16.2.225
16. Conrey FR, Sherman J, Gawronski B, Hugenberg K, Groom CJ. Separating Multiple Processes in Implicit Social Cognition: The Quad Model of Implicit Task Performance. Journal of Personality and Social Psychology. 2005; 89:469–87. https://doi.org/10.1037/0022-3514.89.4.469
17. Bishara AJ, Payne BK. Multinomial process tree models of control and automaticity in weapon misidentification. Journal of Experimental Social Psychology. 2009; 45(3):524–534. https://doi.org/10.1016/j.jesp.2008.11.002
18. Rubin DB. Bayesianly Justifiable and Relevant Frequency Calculations for the Applied Statistician. Ann Statist. 1984; 12(4):1151–1172. https://doi.org/10.1214/aos/1176346785
19. Beaumont MA, Zhang W, Balding DJ. Approximate Bayesian computation in population genetics. Genetics. 2002; 162:2025–2035. PMID: 12524368
20. Beaumont MA, Cornuet JM, Marin JM, Robert CP. Adaptive approximate Bayesian computation. Biometrika. 2009; 96(4):983–990. https://doi.org/10.1093/biomet/asp052
21. Csillery K, Francois O, Blum MGB. abc: an R package for approximate Bayesian computation (ABC). Methods in Ecology and Evolution. 2012;http://dx.doi.org/10.1111/j.2041-210X.2011.00179.x.
22. Liepe J, Barnes C, Cule E, Erguler K, Kirk P, Toni T, et al. ABC-SysBio—approximate Bayesian computation in Python with GPU support. Bioinformatics. 2010; 26(14):1797–1799. https://doi.org/10.1093/bioinformatics/btq278 PMID: 20591907
23. Cornuet JM, Pudlo P, Veyssier J, Dehne-Garcia A, Gautier M, Leblois R, et al. DIYABC v2.0: a software to make approximate Bayesian computation inferences about population history using single nucleotide polymorphism, DNA sequence and microsatellite data. Bioinformatics. 2014; 30(8):1187–1189. https://doi.org/10.1093/bioinformatics/btt763 PMID: 24389659
24. Breiman L. Random Forests. Machine Learning. 2001; 45(1):5–32. https://doi.org/10.1023/A:1010933404324
25. Pudlo P, Marin JM, Estoup A, Cornuet JM, Gautier M, Robert C. Reliable ABC model choice via random forests. Bioinformatics. 2015; 32. PMID: 26589278
26. Wegmann D, Leuenberger C, Excoffier L. Efficient Approximate Bayesian Computation Coupled With Markov Chain Monte Carlo Without Likelihood. Genetics. 2009; 182(4):1207–1218. https://doi.org/10.1534/genetics.109.102509 PMID: 19506307
27. van der Walt S, Colbert SC, Varoquaux G. The NumPy array: a structure for efficient numerical computation. CoRR. 2011;abs/1102.1523.
28. Cohen J. Statistical Power Analysis for the Behavioral Sciences. Lawrence Erlbaum Associates; 1988.
29. Morey RD, Rouder JN. BayesFactor: Computation of Bayes Factors for Common Designs; 2015. Available from: https://CRAN.R-project.org/package=BayesFactor.

30. Riefer D, Batchelder W. Multinomial Modeling and the Measurement of Cognitive Processes. Psychological Review. 1988; 95:318–339. https://doi.org/10.1037/0033-295X.95.3.318

31. Gronau QF, Wagenmakers EJ, Heck DW, Matzke D. A Simple Method for Comparing Complex Models: Bayesian Model Comparison for Hierarchical Multinomial Processing Tree Models Using Warp-III Bridge Sampling. Manuscript submitted for publication and uploaded to PsyArXiv. 2017;.

32. Jacoby LL. A Process Dissociation Framework: Separating Automatic from Intentional Uses of Memory. Journal of Memory and Language. 1991; 30:513–541. https://doi.org/10.1016/0749-596X(91)90025-F

33. Payne B. Prejudice and Perception: The Role of Automatic and Controlled Processes in Misperceiving a Weapon. Journal of Personality and Social psychology. 2001; 81:181–92. https://doi.org/10.1037/0022-3514.81.2.181 PMID: 11519925

34. Stephen Lindsay D, Jacoby LL. Stroop Process Dissociations: The Relationship Between Facilitation and Interference. Journal of Experimental Psychology Human Perception and Performance. 1994; 20:219–34. https://doi.org/10.1037/0096-1523.20.2.219

35. Lambert A, Payne B, Jacoby LL, Shaffer LM, Chasteen A, Khan S. Stereotypes as Dominant Responses: On the "Social Facilitation" of Prejudice in Anticipated Public Contexts. Journal of Personality and Social Psychology. 2003; 84:277–95. https://doi.org/10.1037/0022-3514.84.2.277 PMID: 12585804

Appendix A2

Manuscript 2: Towards end-to-end likelihood-free inference with convolutional neural networks.

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

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

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.

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 learn the posterior mean and variance of multi-dimensional posterior distributions solely from raw simulated data. Once trained on the 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 is reusable by multiple researchers across a given domain. 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 (Goodellow 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 $\boldsymbol{\theta}^{(i)} \sim \pi(\theta)$

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

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 $p_\epsilon\big(\boldsymbol{\theta}\big|\eta(\tilde{\boldsymbol{x}})\big)$. 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 $p_\epsilon\big(\boldsymbol{\theta}\big|\eta(\tilde{\boldsymbol{x}})\big)$ 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 by incurred by computing it. Last but not least, the choice of a particular distance metric $d(\cdot,\cdot)$ is often arbitrary (i.e. *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 is asked 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 completely eliminated.

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 specificity of the ABC-RF methodology is the fact that one needs to train 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 mean and variance estimation:
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:

for $t \leftarrow 1$ to M

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

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

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 $Var[\theta|\eta(X), \mathbf{\Theta}]$ 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 DNN 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) model used widely in cognitive science and psychology (Miletić et al., 2017) and compare our results with the latest state-of-the-art method. 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:

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

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

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:

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

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

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

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 implicitly 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 $\left(\hat{\theta},\ \sigma^2(\tilde{\boldsymbol{x}})\right) = f_{\boldsymbol{W}}(\tilde{\boldsymbol{x}})$. 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 loss, 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 over the first spatial dimension, 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 1 in the first layer of the network. We allow as many (or almost as many) kernels as there are data points in order to avoid excessive loss of information from the input layer to the first convolutional layer. Since these sliding convolutional kernels have the

property to extract *equivariant representations* (see the **Discussion** section for more details on why CNN properties are useful for extracting information from raw data samples), shuffling the distribution does not dramatically change the output of a particular kernel, which has hopefully learned to respond to a given range of values, regardless of the position of the value in the input sample. In later layers, we allow for kernel sizes > 1 in an attempt to implicitly learn 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 which is 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.

We now proceed to evaluate our proposed method 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 marginal posterior distributions (see Gelman et al., 2013, pp. 72–73). 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 $\{\boldsymbol{\Lambda}_m, v_m, \boldsymbol{\mu}_m, k_m\}$ are the posterior model parameters.

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 validation. Finally, to evaluate the performance of the model, we generated 100 independent datasets $\boldsymbol{X}_{test}$, on which we estimated 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 obtaining 10 000 samples from the true joint posterior distribution of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, as dictated by the sampling procedure 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 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 asked 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 empirical 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):

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

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.

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 the particular parameter.

Table 2 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

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). Even regarding 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).

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.

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

Methods comparison on the multivariate normal toy example.

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: Root mean squared error (RMSE) scores given in bold highlight the best approximation in a given column.

Table 2

Method comparison on the Bayesian regression toy example.

Method	$\mathbb{E}[\beta_1 \mid \boldsymbol{x}]$	$\mathbb{E}[\beta_2 \mid \boldsymbol{x}]$	$\mathbb{E}[\sigma^2 \mid \boldsymbol{x}]$	$Var[\beta_1 \mid \boldsymbol{x}]$	$Var[\beta_2 \mid \boldsymbol{x}]$	$Var[\sigma^2 \mid \boldsymbol{x}]$
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 in a given column.

Table 3

Methods comparison on the LCA example.

	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

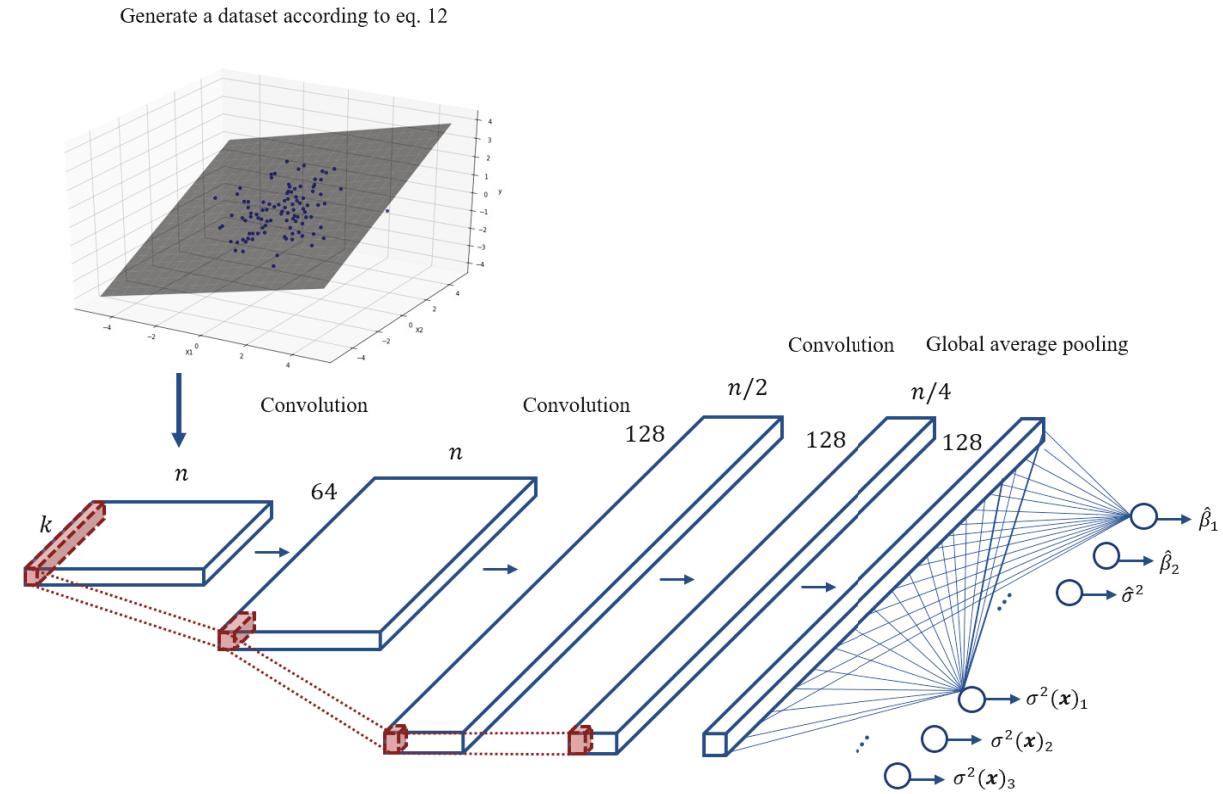

Figure 1. Graphical illustration of the CNN architecture used throughout this paper. The input represents data sets generated from the Bayesian regression model (toy example 2), organized as an n x k grid, with k equal to the number of predictors (in this case 2) + the outcome, and n equal to the number 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

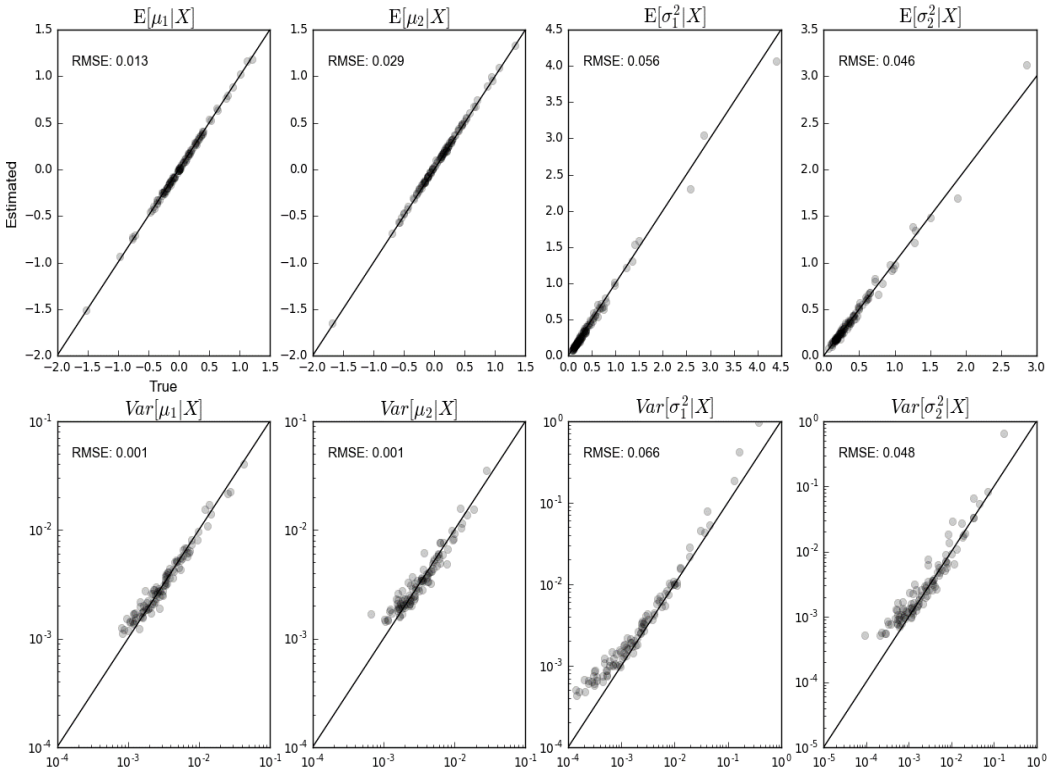

Figure 2. Comparison of the estimates obtained by the CNN model with the true parameter values on the multivariate normal toy example. True parameter values are plotted on the x-axis, and corresponding estimates on the y-axis. The first row depicts the posterior expectations of the model parameters.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

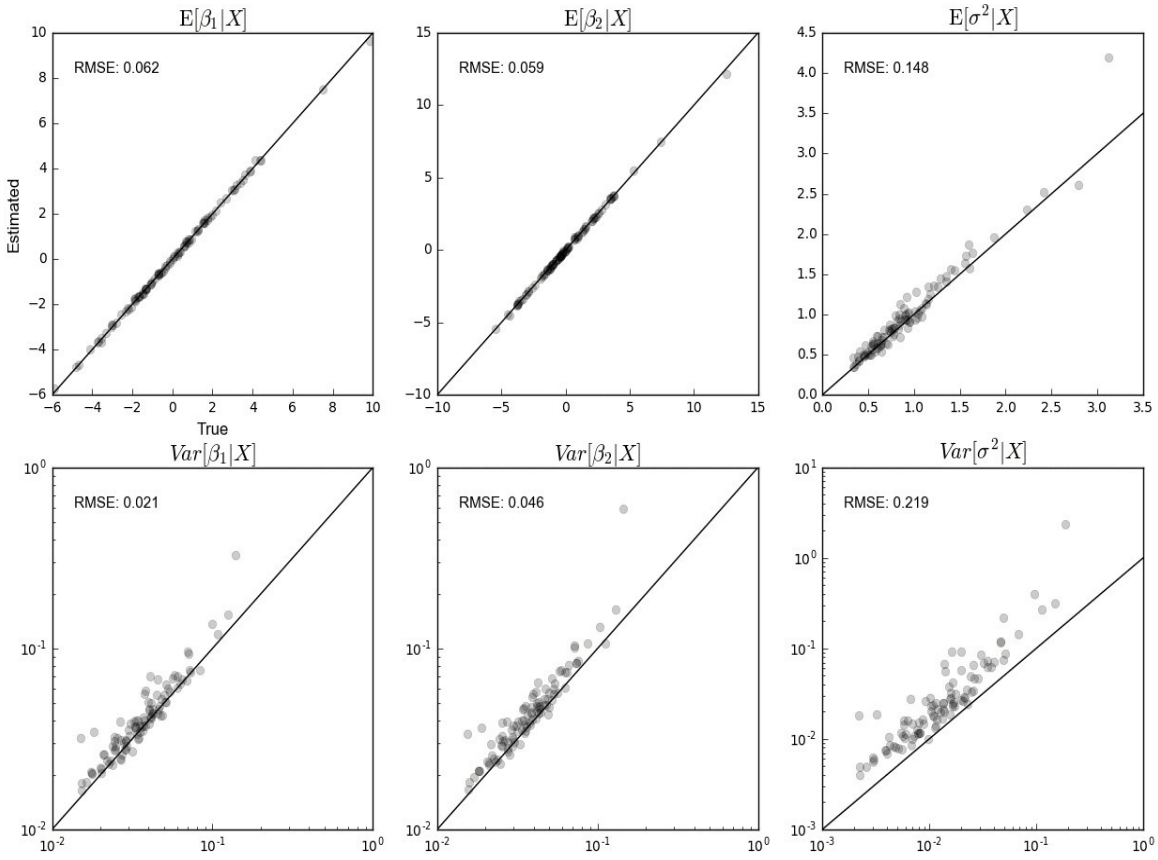

Figure 3. Comparison of the estimates obtained by the CNN model with the true posterior parameter values on the Bayesian linear regression example. True parameter values are plotted on the x-axis, and corresponding estimates on the y-axis. The first row depicts the posterior expectations of the model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

LCA example results

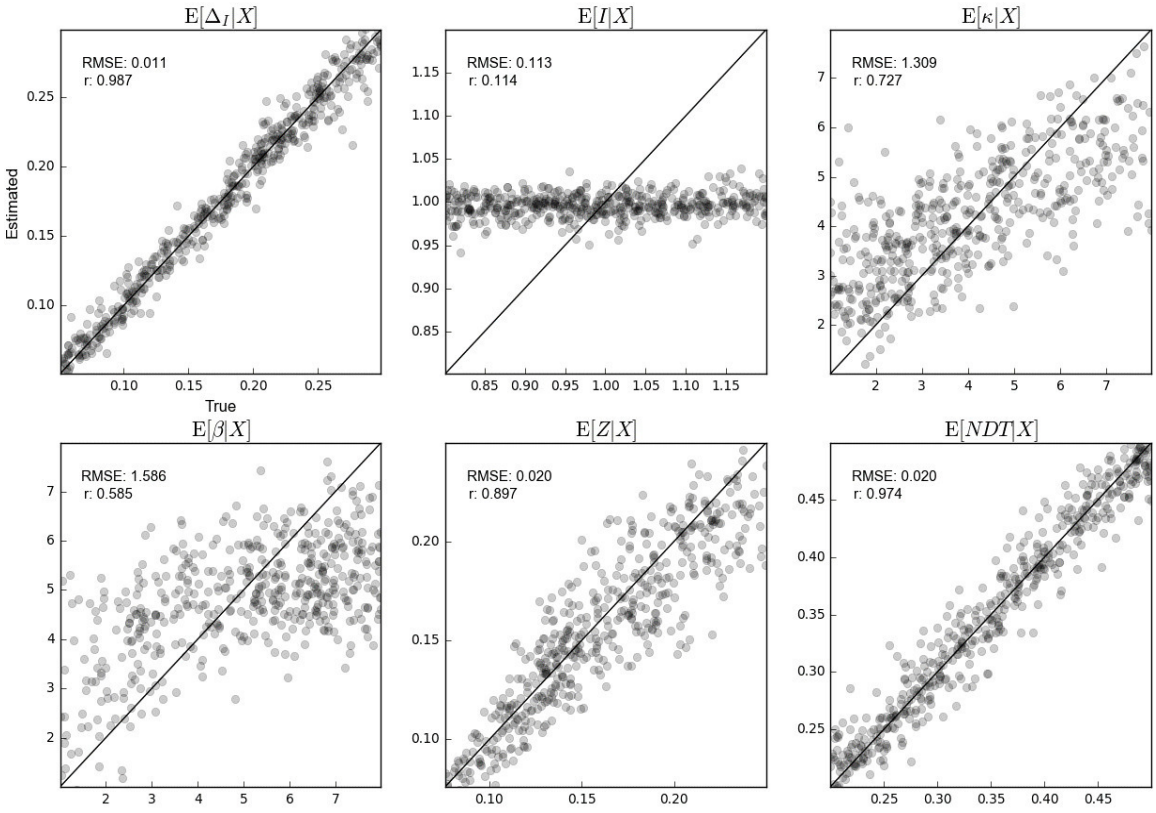

Figure 4. Depiction of parameter recovery of the LCA model. Values of the data-generating parameters are plotted on the x-axis, and corresponding CNN estimates on the y-axis. Both root mean squared error (RMSE) and Pearson's correlation coefficient (*r*) are also reported.

Appendix A3

Manuscript 3: Learning the Likelihood: Using DeepInference for the Estimation of Diffusion-Model and Lévy Flight Parameters.

Note: This is the pre-print version of the manuscript that is currently under review in the *Journal of Mathematical Psychology.*

Learning the Likelihood: Using DeepInference for the Estimation of Diffusion-Model and Lévy Flight Parameters

Andreas Voss, Ulf K. Mertens, and Stefan T. Radev

Heidelberg University

Authors' Note:
Andreas Voss, Institute of Psychology, Heidelberg University, Hauptstr. 47-51, 69117 Heidelberg, Germany. Ulf Mertens, Institute of Psychology, Heidelberg University, Hauptstr. 47-51, 69117 Heidelberg, Germany. Stefan Radev, Institute of Psychology, Heidelberg University, Hauptstr. 47-51, 69117 Heidelberg, Germany.

The present research was supported by Grant Vo-1288-2 to Andreas Voss

Correspondence concerning this article should be addressed to Andreas Voss, e-mail andreas.voss@psychologie.uni-heidelberg.de

Abstract

In the present paper, we use the recently develop DeepInference architecture as a general likelihood-free method to estimate parameters of cognitive models. DeepInference is a machine-learning algorithm based on the training of convolutional neural networks. In a first step, the network has to be trained with simulated data to learn the relation of parameters and data. Then, the trained network can be used to re-estimate parameters for real data. The efficiency and robustness of this approach was tested for two decision models based on continuous evidence accumulation. Study 1 investigated the recovery of parameters of the diffusion model, and Study 2 addressed the same question for a Lévy-Flight model. Results demonstrate that the machine-learning approach is superior to traditional multidimensional search algorithms that maximize the likelihood, both in terms of correlations of estimated parameters with true parameters and with regard to absolute deviations. The new approach also excels the maximum likelihood based search pertaining the robustness in the presence of contaminated data.

Keywords: Machine Learning, Parameter Estimation; Cognitive Model; Diffusion Modell; Lévy-Flight

Learning the Likelihood: Using Deep Inference for the Estimation of Diffusion-Model and Lévy Flight Parameters

In recent years, a great increase in interest for cognitive modelling became evident in many areas of psychology (Heathcote, Brown, & Wagenmakers, 2015; Rodgers, 2010). The development of formal mathematical models for cognitive processes has many advantages: Firstly, the translation of a theoretical model into a mathematical formulation enforces a detailed definition of the model that avoids – or at least uncovers – remaining ambiguities. Secondly, the definition of a theoretical model allows for the derivation of well-founded hypotheses that in turn make possible stringent tests of the theoretical model. Thirdly – and most important for the present paper – cognitive models can be applied as measurement tools, that is, model parameters can be used as precise and process-specific measures for ongoing cognitive processes. For example, in the domain of diffusion models (Voss, Nagler, & Lerche, 2013), separate parameters map processing speed (drift rate), speed-accuracy settings (threshold separation), decision bias (starting point), or duration of non-decision processes like motoric response execution (non-decision time). Whereas traditional measures of performance (i.e., response times or accuracy rates) are the result of complex interactions of different processes, the drift rate from a diffusion model analysis is a purer measure for the speed with which the presented stimuli are processed, for example, because the drift is estimated independently from individual speed-accuracy settings and the speed of motor processes.

However, the use of parameters from cognitive models as measures for inter-individual differences requires a precise and reliable estimation of these parameters(Lerche & Voss, 2017; Lerche, Voss, & Nagler, 2016). For complex models, this can be a difficult endeavour. Even if the likelihood function for a specific model is known, its calculation can be time demanding, and analytical solutions for the maximum likelihood (ML) solution are often not available. In this case, multidimensional search algorithms to maximize the likelihood like the SIMPLEX-search (Nelder & Mead, 1965) have to be used (see Voss, Voss, & Lerche, 2015, for an example from diffusion modelling). For stochastic models with a high-dimensional parameter space, however, such searches are not only slow but often inaccurate as well, for example, when there are local minima, or when the likelihood is not calculated with the necessary precision[1]. One solution for this problem are Markov-Chain-Monte-Carlo (MCMC) algorithms that tend to be more efficient for high-dimensional problems (van

[1] For example, the calculation of the likelihood for the diffusion model contains an infinite sum, and the precision of calculation depends on the number of evaluations of this sum (Navarro & Fuss, 2009).

Ravenzwaaij, Cassey, & Brown, 2018). These MCMC approaches bring the additional advantage that they make posterior distributions for parameters available, which provide information about the precision of parameter estimates. Although Bayesian modelling sometimes helps to overcome problems of parameter estimation, some problems remain: Firstly, the application of this approach is limited to models for which the likelihood is known. Secondly, as the likelihood is very sensitive to outliers, results can be strongly biased in case of contaminated data. Thirdly, the implementation can be quite demanding. We will discuss these limitations in more detail below, again drawing upon the example of diffusion modelling.

Problems of likelihood-based parameter search

While generally likelihood-based parameter search algorithms (traditional ML-methods or Bayesian approaches) are mathematically most efficient, their usefulness is limited in certain cases. We will illustrate these limitations within the diffusion model framework (for general introductions, see Ratcliff & McKoon, 2008; Voss et al., 2013). However, the presented arguments are not specific to diffusion models but apply also to other model classes.

The first limitation of likelihood-based approaches regards the availability of the likelihood of a model. While the likelihood can be calculated easily for most statistical models, it often becomes intractable for more complex cognitive models. For example, in diffusion modelling, the density for first passage times of a Wiener diffusion process with two absorbing boundaries (i.e., for the decision time) is well known (e.g., Navarro & Fuss, 2009). However, there are many plausible ways to extend or change the diffusion model that make the calculation of the density (and thus of the likelihood) intractable. Such changes comprise, for example, collapsing boundary models that assume that decision thresholds are decreased over time (e.g., Hawkins, Forstmann, Wagenmakers, Ratcliff, & Brown, 2015), models with changing drift rates (e.g., Ulrich, Schroter, Leuthold, & Birngruber, 2015), or so-called Lévy-flight models assuming non-Gaussian noise distributions in evidence accumulation (Voss, Lerche, Mertens, & Voss, 2018). In these examples, there is no known analytical function to calculate the likelihood of a response from the model parameters[2]. A restriction of research to models providing an analytic form of the likelihood would be a severe obstacle to progress in the research of cognitive processes.

A second limitation of likelihood-based approaches regards their sensitivity for outliers. Any data outside the scope of predictions of a model will result in a likelihood of

[2] Of course, the likelihood can be approximated by a simulation approach (Heathcote, Brown, & Mewhort, 2002). That is, responses are simulated many times, and the likelihood is estimated for response-time bins. However, this is a very slow method that might also lack of the required precision.

zero, thus excluding some parameter values completely. On a first glance, it makes sense that if a model cannot account for observed data, the model needs to be invalid. However, assume for a moment that in a 500-trial experiment, a continuous accumulation process (as assumed by the diffusion model) guides the participant's response in 499 trials, but that in the one remaining trial the participant uses a fast-guessing strategy, and responds before actually perceiving the stimulus. Performance in this one trial might have a likelihood of zero for the true parameter values from the 499 valid trials (because the observed RT is faster the minimum predicted non-decision time), thus rendering the total likelihood to be zero as well. Therefore, the parameter search is forced to adopt values that can encompass all data, that is, also the one fast response. Specifically, this would result in a lowered non-decision time, which in turn will result in a bias of other parameters as well. In this example, a careful pre-processing of data might suffice to remove the outlier before the modelling. However, Lerche et al. (2016) showed that fast contaminants strongly bias ML-based parameter estimation for diffusion models, even if the contaminants are no outliers in a statistical sense (i.e., the proportion of fast responses is increased at a point in time where the true model already predicts some responses). One way to respond to this problem is to extend the model so that it encompasses more than one response strategy. For example, Ratcliff and Tuerlinckx (2002) included a "guessing parameter" in the diffusion model to account for contaminations of data. However, this makes models more complex and relies on additional assumptions (i.e., the nature of contamination needs to be known). Another possibility is to substitute the ML search by methods that are more robust, like, for example, the Kolmogorov-Smirnov-Distance between observed and predicted cumulative response time distributions (Lerche & Voss, 2017; Lerche et al., 2016; Voss, Rothermund, & Voss, 2004). However, this comes at the cost of a reduced efficiency of the parameter search, and the precision of results will be reduced in the case of sparse data (i.e., low trial numbers).

The last problem we want to discuss here regards implementational issues. In the case of complex models, results of a multi-dimensional parameter search are often not very robust and can depend on details of the implementation (e.g., regarding the search algorithm and the precision of calculations). For Bayesian modelling, the implementation became much easier in recent years due to the development of efficient and flexible MCMC samplers like JAGS (http://mcmc-jags.sourceforge.net/) or Stan (http://mc-stan.org/). However, despite the great flexibility of these packages, they do not allow to implement all cognitive (process) models. For example, Stan includes the density distributions for a basic version of the diffusion model, but not for the full Ratcliff model containing inter-trial variabilities of starting point, drift, and

non-decision time (Ratcliff & Rouder, 1998; Ratcliff & Tuerlinckx, 2002).[3] Thus, developing a parameter estimation procedure still often requires a skilled programmer, and implementing a complete MCMC solution might be too demanding for many researchers.

Typically, simulating data from a model is much easier, and this allows for a completely different approach for the parameter estimation in cognitive models. We propose a recently developed method based on Deep Learning to estimate estimate model parameters (Radev, Mertens, Voss, & Köthe, 2018). This approach is very flexible; it requires no knowledge of the likelihood, and no specification of an optimization criterion. The algorithm learns the relation of parameter values and observed responses. In the following section, we present a short introduction to the employed deep-learning algorithm that is based on convolutional neural networks (CNN).

DeepInference: Parameter recovery with convolutional neural networks

Recently, we proposed to train a fully convolutional neural network to approximate the first two moments of the posterior distributions of relevant model parameters (Radev et al., 2018). In particular, we cast the problem of inference as a supervised learning task, where simulated datasets represent the inputs to the algorithm, and the corresponding data-generating parameters represent its outputs. The rationale of the method stems from the fact that minimizing the mean squared error (MSE) between actual parameters and estimated parameters via an optimization procedure (e.g., stochastic gradient descent) results in an approximation of the conditional expectation of the parameters given the data (Kendall & Gal, 2017). In order to quantify the uncertainty of the obtained parameter estimates, we optimize the heteroscedastic loss given by

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

where N denotes the number of simulated data sets, $\boldsymbol{W}$ denotes the weights of the neural network, $\hat{\theta}$ gives the estimated parameter value, and θ the actual parameter value, and $\sigma^2(\tilde{\boldsymbol{x}})$ represents an uncertainty parameter learned directly from the data $\tilde{\boldsymbol{x}}$.

Minimizing the heteroscedastic loss, in contrast to the MSE loss, allows the neural network model to capture the inherent uncertainty of estimates caused by training with finite data samples. In the case of a Gaussian model, this uncertainty has a nice interpretation as the

[3] Vandekerckhove, Tuerlinckx, and Lee (2011) suggested implementing these inter-trial-variability parameters hierarchically via prior distributions for drift, starting point, and non-decision time. However, this is conceptually not the same as an implementation of the full diffusion model where inter-trial-variability parameters are used directly to calculate the densities, because in the hierarchical approach these "priors" will eventually be overridden by data, that is, the impact of the variability parameters will be reduced as a function of trial numbers (Andrew Heathcote, personal communication).

posterior variance of the target parameter distribution (Radev et al., 2018). On an intuitive level, if the estimated parameter value is too far off from the actual parameter value, the squared $L2$ loss term $\left\|\hat{\theta}^{(i)} - \theta^{(i)}\right\|_2^2$ is going to penalize the model by increasing the total loss. In order to correct for this, the model will need to increase the $\sigma^2(\tilde{\boldsymbol{x}}^{(i)})$ uncertainty term to keep the total loss low. At the same time, the $\log \sigma^2(\tilde{\boldsymbol{x}}^{(i)})$ term acts as a regularizer, which prevents the uncertainty from growing to infinity.

Once the network is trained on a set of simulated data-parameters pairs, all parameters of new (real) data sets, as well as corresponding uncertainties, can be obtained simultaneously through a single forward pass through the network. Moreover, the use of fully convolutional neural networks makes it possible to fully exploit the grid-like structure of most empirical data and also work with data sets of variable size (i.e., different trial numbers).

Simulation Study 1: Recovering diffusion model parameters

Aim of Study 1 was to test whether the deep inference algorithm explained above can recover diffusion model parameters with a similar accuracy as a multidimensional search maximizing the likelihood (Lerche et al., 2016). To test the robustness of results, we additionally compared precision of recovery of parameters form the pure data with performance when data were contaminated by fast guessing in some of the trials.

Simulation of data

Data were simulated following the rationale of the diffusion model by a C program employing the gnu scientific library (gsl; www.gnu.org/software/gsl) for random number generation. For each trial of the simulation, a path of the diffusion process was simulated using the stochastic Euler method (Voss et al., 2018). Specifically, the decision process started at time $t = 0$ at point $P(t = 0) = zr \cdot a$. Then, in each step of time the process was changed following Equation 2:

$$P(t + \Delta t) = P(t) + \Delta t \cdot v + \sqrt{\Delta t} \cdot \epsilon;\ \epsilon \sim N(0,1), \tag{2}$$

where Δt is a small increment in time (1 ms in our simulations), v is the drift rate of the information accumulation, and ϵ is Gaussian noise. This process was run until it reached an upper threshold at a or a lower threshold at 0. To the process duration t the non-decision time t_0 was added. For the training of the CNN we coded the response by multiplying simulated response times with -1, when the process terminated at the lower threshold.

Because our model incorporated inter-trial variability of starting point, drift, and non-decision time (Ratcliff & Rouder, 1998; Ratcliff & Tuerlinckx, 2002), specific values for z_r,

v, and t_0 for each trial were drawn from uniform or normal distribution given by Equations (3) to (5):

$$t_0' \sim U(t_0 - 0.5 \cdot s_{t0}; t_0 + 0.5 \cdot s_{t0}) \quad (3)$$

$$z_r' \sim U(z_r - 0.5 \cdot s_{zr};\ z_r + 0.5 \cdot s_{zr}) \quad (4)$$

$$v' \sim N(v; s_v) \quad (5)$$

Training data

For the training of the CNN, 500.000 datasets[4], each comprising a total of 500 trials from two conditions were simulated. For the two conditions, all parameters but the drift were held constant; different drift rates were used for the first vs. second half of trials. For each data set, parameters were randomly chosen from uniform distributions (Table 1).

In a second step, contaminated data sets were generated by including a small proportion (4%) of fast-guessing trials in each dataset. That is, in each dataset response latencies form 4% of all trials were replaced by a fast response time, drawn from a uniform distribution ranging from the lower edge of the predicted RT distribution (t0-st0/2) minus 100ms to the lower edge plus 100ms (Lerche et al., 2016). Responses (i.e., the sign of the simulated RTs) for these fast guessing trials were determined randomly.

To test whether including contaminated data into the training set makes estimation more robust, we compare performance when the training data contains 0%, 10%, or 100% contaminated data sets (note that each contaminated data set comprised only 4%, or 20 trials, of simulated guessing).

Test data

To assess the precision of parameter estimation, additional test data were simulated. Test data were generated following the same procedure as the training data. One-thousand "pure" datasets and 1,000 dataset containing 4% contaminants each were simulated. Again, each dataset comprised 500 trials from two conditions.[5]

Estimation procedure

Parameters from pure and contaminated test data were re-estimated in four ways: The machine-learning algorithm was employed with training based on 500.000 datasets from which 0%, 10% or 100% contained contaminants, and a ML-based parameter search using *fast-dm-30* (Voss & Voss, 2007; Voss et al., 2015) served as a reference method.

[4] In pilot studies, we compared performance using different sizes of training data. We found a slight but notable increase in accuracy of results when the number of training samples was increased from 100.000 to 200.000. The additional increase from 200.000 to 500.000 training samples was much smaller, suggesting that a ceiling for accuracy was approached.

[5] Test data from both studies are available under https://doi.org/10.11588/data/HY4OBJ.

Results and Discussion

As criteria for the quality of recovery, we use (1) the correlation of true parameter values from the simulation with the recovered parameter values and (2) the normalized mean absolute error (NMAE) that was calculated as

$$NMAE = \frac{\frac{1}{n}\cdot\sum|\hat{\theta}-\theta|}{\max(\theta)-\min(\theta)} \quad (6)$$

where n is the number of analysed datasets (i.e., $n = 1000$), θ is the true parameter value, and $\hat{\theta}$ is the recovered parameter value.

Correlations of true parameter values with estimates and mean absolute deviations (MNAE) are presented in Tables 2 and 3, respectively. Mean correlations averaged across all parameters using Fishers Z transformation and mean NMAE values are displayed in Figure 1. Correlations are generally high, which is little surprising given the large datasets. Replicating previous results, recovery of inter-trial-variability of starting point and drift is much less precise than recovery of other parameters (Lerche & Voss, 2017; Lerche et al., 2016; van Ravenzwaaij & Oberauer, 2009). For uncontaminated data, all methods perform accurately, with a slight advantage for the deep learning algorithm trained with no (0%) or little (10%) contaminated data. The advantage of the new machine-learning method over the multidimensional search for the ML solution performed by *fast-dm* becomes more pronounced in the case of contaminated data. Here, DeepInference proves to be generally more robust, but especially so if the training data incorporates contaminated samples. Results regarding the NMAE values show the same pattern as do correlational results. To sum up, machine learning proves to be superior to the multidimensional parameter search, both for pure and for uncontaminated data. Especially, including 10% of contaminated datasets into the training data notably increases the robustness of the procedure without doing much harm in the case of pure test data.

DeepInference also provides the variance of estimation errors for all parameters (Bayesian speaking, this is an estimate for the variance of the posterior parameter distributions). The corresponding standard deviations are presented in Table 4, along with the percentage of 95% confidence-intervals that contain the original (i.e., true) parameter values. As can be seen from the table, variance is slightly overestimated, resulting in a minimal exaggeration of the confidence intervals that often contain true scores for more than 95% of estimated parameters (Radev et al., 2018). If the training data does not match the test data (i.e., training with pure data and recovery with contaminated data or vice versa), the size of confidence intervals for some parameters (especially s_{t0}) is dramatically underestimated.

However, training the model with a moderate amount of contaminated data makes results again much more robust.

Simulation Study 2: Recovering Lévy-Flight model parameters

Recently, we observed that model fit of the diffusion model to real data can be notably improved, when the typical assumption of normal noise in the accumulation process is relinquished (Voss et al., 2018). Specifically, in the new model, accumulation of information is no longer mapped by a diffusion process but by a Lévy-Flight that is characterized by a constant drift and heavy-tailed (alpha-stable distributed) noise. The heavy-tailed noise distribution increases the probability of extreme events, that is, of jumps in the information accumulation process. The new model contains the additional parameter α ($0 \leq \alpha \leq 2$), which indicates the stability of the process. The diffusion model with normal noise is a special case of the more general model proposed here, because for $\alpha = 2$ the alpha-stable distribution becomes a normal distribution. Unfortunately, we cannot compute the likelihood for the Lévy Flight model, which makes fitting the model quite cumbersome. However, simulating data is still easy, so that the CNN approach is an ideal candidate for parameter estimation.

Simulation of data

Simulation of data followed exactly the same procedures as in Study 1 with the one exception that the diffusion process was replaced by a Lévy-Flight using the additional stability parameter α (see Table 1). The simulation of a Lévy Flight differs slightly from that of a diffusion process, because the relation of the error term with step size in time depends on α. The change in accumulated evidence over time Δt (1ms in our simulation) is given by

$$P(t + \Delta t) = P(t) + \Delta t \cdot v + \Delta t^{\frac{1}{\alpha}} \cdot \epsilon;\ \epsilon \sim aS(\alpha) \quad (7)$$

where v is the drift, and ϵ follows a symmetric alpha-stable distribution with stability α. For α a range of 1.0 to 2.0 was assumed (Table 1), which corresponds roughly to the values observed by Voss et al. (2018).

Estimation procedure

Parameter estimation of the DeepInference algorithm was identical to Study 1, with the exception that an additional parameter (α) was estimated. For comparison, also a multi-dimensional parameter search was implemented in a C program. For this purpose, the likelihood was estimated by a simulation-based approach (Heathcote et al., 2002): In each step of the search, 50,000 Lévy-Flight processes were simulated for each of the two stimulus types from the actual parameter values. Then, the likelihood for each response was estimated by the portion of simulated processes terminating in the same RT "bin" (spaced 20ms) as the

observed response. The total likelihood was minimized using a variant of the simplex search[6] (Nelder & Mead, 1965). This search procedure took about 22 minutes per dataset (about 4 weeks for the 2 x 1000 test samples) of processor time on an Intel i7 CPU with 2.93 GHz.

Results and Discussion

Correlations of true parameter values and estimates and absolute deviations (MNAE) are presented in Tables 5 and 6, respectively. Table 7 presents the estimated uncertainty, that is, the mean standard deviations of estimation errors (approximating the standard deviations of the posterior parameter distributions) for all parameters. Mean correlations averaged across all parameters using Fishers Z transformation and mean NMAE values are displayed in Figure 2.

Results closely resemble those of Study 1: Again, results from the DeepInference approach are more precise than those of the ML based approach, both in terms of correlations with true parameter values and in terms of absolute deviations. One main source of the disadvantage of the ML method compared to DeepInference approach is obviously located in problems of recovering the stability parameter α, which is estimated with much higher precision by the latter method. Also mimicking results from the previous study, DeepInference is much less influenced by outliers, and this robustness is even increased when contaminated datasets are included in the reference table for the training of the CNN.

General Discussion

In this paper, we propose DeepInference as a general, likelihood-free method to estimate parameters from cognitive models. The method can be applied to all types of models that allow simulating data from a given set of parameter values. Typically, generation of data is easy and fast for well-defined models. The fact that the likelihood no longer needs to be computed can boost research in the area of cognitive modelling because now researchers can easily apply new and more complicated models to their data.

Although the procedure has its greatest advantages when the likelihood is not available and thus traditional methods of parameter estimation cease to apply, results from Study 1 demonstrate that deep learning is also a valuable tool for models for which the likelihood is known. Especially in cases where the ML solution cannot be directly computed, and thus a multidimensional search maximizing the likelihood is necessary, ML based searches have their own problems. For example, the likelihood function may be noisy when the precision of computation is limited. In the case of diffusion modelling, for instance, an exact calculation of the likelihood is not possible because the likelihood function contains an infinite sum. The introduced noise in the likelihood function can – in regions of the multidimensional space

[6] The simplex2 algorithm of the gsl library (www.gnu.org/software/gsl) was used for this purpose.

where the slope of the likelihood function is quite flat – produce artificial local minima, in which the search procedure gets stuck. The DeepInference algorithm is more robust and – as the present results show – often better at finding the optimal solution.

Parameter recovery from pure data

In the present paper, the DeepInference algorithm has been validated in two simulation studies for two different accumulator models. Firstly, we re-estimated parameters from the standard diffusion model (Voss et al., 2013), for which the likelihood can be computed. Thus, it was possible to use a ML-based search for comparison. Secondly, we tested the DeepInference algorithm for a Lévy-Flight model (Voss et al., 2018) that assumes heavy-tailed noise in the accumulation of evidence. For this model, the likelihood function for the process durations in a model with two absorbing boundaries is unknown (at least to the authors of this paper), which makes the parameter estimation difficult. To get some standard for comparison here as well, we used a simulation-based approach to estimate the likelihood for bins on the time axis.

Both simulation studies demonstrate that the DeepInference algorithm is very efficient. For both models, estimates generated by the DeepInference approach are related closely to the true parameters values with mean correlations well above $r = .90$, and, most important, above the correlations from the traditional ML-based search algorithms. An inspection of the correlations for the separate parameters shows similar pattern for DeepInference and the multidimensional search: Replicating previous results (Lerche & Voss, 2017; Lerche et al., 2016; van Ravenzwaaij & Oberauer, 2009), inter-trial variability of starting point and drift can be estimated with much lower precision compared to all other parameters. To test whether the methods differ in a potential bias (that would not be detected by correlations), the normalized mean absolute errors (NMAE) were inspected as a second evaluation criterion. The analyses of NMAE-values corroborate the correlational findings, again demonstrating that the DeepInference algorithm provides even more accurate estimates than the multidimensional search.

Parameter recovery from contaminated data

In the application of cognitive modelling, contamination of data is often a problem, and ML results can be strongly biased by even very small percentages of contamination (Lerche et al., 2016). Theoretically, there are two ways to deal with contamination of data. Firstly, when contaminants are outliers in a statistical sense, they should be removed from data prior to the modelling. Secondly, sometimes it is possible to incorporate processes of contamination explicitly in the model (e.g., Ratcliff & Tuerlinckx, 2002). However, this not only makes the

model less parsimonious, but also requires knowledge about the distribution of contamination, which might not be available. Because it is not always possible to capture all kinds of contamination with the two methods sketched above, parameter estimation procedures need not only to be efficient, but also to possess some robustness. To test for robustness, we investigated parameter recovery in the presence of one type of contamination: fast outliers. These were simulated by including 4% of trials within each data set with very fast response times (that were no statistical outliers, however) and random responses.

As expected, the ML-based procedures were corrupted strongly by the presence of contaminants, as evident from notably reduced correlations and increased NMAE values. In comparison, the DeepInference algorithm was much less affected by outliers, and robustness was even stronger, when the training data contained some contaminants as well. A very good performance for both pure and contaminated data was achieved, when the CNN was trained with 90% pure data sets and 10% contaminated data sets. Increased robustness due to *data augmentation* techniques are often encountered in other domains of Deep Learning. For instance, in computer vision, much better classification accuracies are obtained when the training set not only contains high resolution images but also noisy or blurry images (Jindal, Nokleby, & Chen, 2016)

Uncertainty of estimates

DeepInference is not only an efficient and robust method to obtain point estimates for all parameters. Simultaneously, for each parameter a variance is estimated that is a measure for the uncertainty of the parameter recovery and can be seen as an approximation of the true posterior variance. Although our approach does not provide a full posterior distribution, providing a measure of uncertainty is an important feature of this algorithm. For example, confidence intervals allow assessing whether parameters in a model differ from specific values (e.g., whether there is a decision bias with the relative starting point differing from 0.5).

Mean standard deviations show notable differences between parameters: While non-decision time and starting point can be assessed very accurately with the 500 trials used here, uncertainty in estimates is generally large for inter-trial variability of starting point or for drift rates. Whenever these parameters need to be estimated with high precision, even larger trial numbers are necessary.

Limitations of the deep inference approach

Although we believe that deep learning provides a fairly general framework that can be applied to a wide variety of cognitive models there are of course a number of limitations that need to be considered.

Firstly, machine learning rests on a large data basis, so usually many datasets need to be simulated. For the present application, we trained the models with 500.000 datasets consisting of 500 trials each. Although simulating these data is no real problem for modern computers, simulation of datasets as well as training the algorithms still needs a lot of processor time, working memory and space on the computer's hard disc. The more complicated a model is the more training data will be required to allow for a precise parameter recovery. On the other hand, DeepInference is an architecture that is designed to be reused. Once it has been trained on a large number of simulated datasets, parameter estimation on new datasets is extremely fast.

Secondly, it can be seen as a disadvantage that machine learning operates to a certain degree as a black box, that is, it is difficult to understand how the machine learns to recover parameters. However, in a way the algorithm just translates the relation of parameter values to the generated data as specified by the data-generating model. Therefore, the algorithm implicitly learns the likelihood of data given certain parameter constellations. Thus, the algorithm does not add anything to the theoretical model.

Thirdly, there is the more practical limitation that – since no likelihood is calculated – the information criteria (AIC, BIC) cannot be used to compare model fit. There are several possibilities to solve this problem: For example, it is possible to approximate the likelihood after parameter estimation by simulating again many responses from the final parameter values. Then, the likelihood of each response – given the estimated parameters – can be approximated by the portion of responses that are close to the response (Heathcote et al., 2002). This approach has been used – with a limited accuracy – for the comparison method in Study 2. For parameter estimation, this simulation approach can be used only with a reduced accuracy, because the likelihood has to be approximated in each of many steps of the multidimensional search. However, if this approach is employed to assess the likelihood of the final model, a much larger number of simulated trials can be used to ensure a high precision (cf. Hawkins et al., 2015). Furthermore, DeepInference itself could easily be adjusted to predict the best-fitting model from observed data.

Outlook: Future developments

In the present paper, the usefulness of a deep learning algorithm was verified on two variants of accumulator models. In future research, the universality of the advantages of this approach

should be investigated both within this class of decision models and for other types of models as well. Within the class of accumulator models, for example, the robustness of more parsimonious models (e.g., excluding the intertrial variability parameters, Lerche & Voss, 2016), or the precision of results for smaller data sets should be investigated.

Although the application of machine learning algorithms in psychology is rising, it might still be challenging for many cognitive researchers to implement a deep learning algorithm for their own model. To make this method available for a broader community we recommend two strategies: On the one hand, general software (like ABrox, Mertens, Voss, & Radev, 2018) should be developed that perform likelihood-free parameter estimation from simulated data which needs to be provided by the user in form of a reference table. The advantage of this general method is that it can be used for all kinds of models as long as a reference table can be generated. On the other hand, specific models can be trained first and then be published, for example in form of an *R* package. This latter approach is of course applicable only to specific implementations of specific models. However, these packages can then be applied to data without any further training; that is, no simulation of data is required for the user of the model.

Another useful future development of the principle discussed in the present paper goes one step further: Principally, a trained convolutional neural network can be reversed, that is, translated to a generative model. So, a trained model might be used to generate samples of parameter-sets from given data. In future, this principle could allow generating a full posterior distribution for the models parameters, without requiring a likelihood.

Conclusion

DeepInference via convolutional networks provides a highly efficient, precise, fast, robust, and likelihood-free method to estimate parameters of cognitive models. From the standard deviations of estimation errors, confidence intervals can be calculated for the model parameters. The only requirement for the application of this approach is the possibility to simulate (huge amounts of) data from model parameters. We believe that this approach has great potential to foster the application of cognitive models, and especially to use these models to assess inter-individual differences in latent cognitive processes.

References

Hawkins, G. E., Forstmann, B. U., Wagenmakers, E. J., Ratcliff, R., & Brown, S. D. (2015). Revisiting the evidence for collapsing boundaries and urgency signals in perceptual decision-making. *J Neurosci, 35*(6), 2476-2484. doi:10.1523/JNEUROSCI.2410-14.2015

Heathcote, A., Brown, S. D., & Mewhort, D. J. (2002). Quantile maximum likelihood estimation of response time distributions. *Psychon Bull Rev, 9*(2), 394-401. doi:10.3758/BF03196299

Heathcote, A., Brown, S. D., & Wagenmakers, E.-J. (2015). An introduction to good practices in cognitive modeling. In B. U. Forstmann, E.-J. Wagenmakers, B. U. Forstmann, & E.-J. Wagenmakers (Eds.), *An introduction to model-based cognitive neuroscience.* (pp. 25-48). New York, NY, US: Springer Science + Business Media.

Jindal, I., Nokleby, M., & Chen, X. (2016, 12-15 Dec. 2016). *Learning Deep Networks from Noisy Labels with Dropout Regularization.* Paper presented at the 2016 IEEE 16th International Conference on Data Mining (ICDM).

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.

Lerche, V., & Voss, A. (2016). Model Complexity in Diffusion Modeling: Benefits of Making the Model More Parsimonious. *Front Psychol, 7*, 1324. doi:10.3389/fpsyg.2016.01324

Lerche, V., & Voss, A. (2017). Retest reliability of the parameters of the Ratcliff diffusion model. *Psychol Res, 81*(3), 629-652. doi:10.1007/s00426-016-0770-5

Lerche, V., Voss, A., & Nagler, M. (2016). How many trials are required for parameter estimation in diffusion modeling? A comparison of different optimization criteria. *Behavior Research Methods, 49*(2), 513-537. doi:10.3758/s13428-016-0740-2

Mertens, U., 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. doi:10.1371/journal.pone.0193981

Navarro, D. J., & Fuss, I. G. (2009). Fast and accurate calculations for first-passage times in Wiener diffusion models. *Journal of Mathematical Psychology, 53*(4), 222-230. doi:10.1016/j.jmp.2009.02.003

Nelder, J. A., & Mead, R. (1965). A Simplex-Method for Function Minimization. *Computer Journal, 7*(4), 308-313. doi:DOI 10.1093/comjnl/7.4.308

Radev, S., Mertens, U., Voss, A., & Köthe, U. (2018). Towards end-to-end likelihood-free inference with convolutional neural networks. *Paper submitted for publication*.

Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: theory and data for two-choice decision tasks. *Neural Comput, 20*(4), 873-922. doi:10.1162/neco.2008.12-06-420

Ratcliff, R., & Rouder, J. N. (1998). Modeling response times for two-choice decisions. *Psychological Science, 9*(5), 347-356. doi:Doi 10.1111/1467-9280.00067

Ratcliff, R., & Tuerlinckx, F. (2002). Estimating parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic Bulletin & Review, 9*(3), 438-481. doi:Doi 10.3758/Bf03196302

Rodgers, J. L. (2010). The epistemology of mathematical and statistical modeling: a quiet methodological revolution. *Am Psychol, 65*(1), 1-12. doi:10.1037/a0018326

Ulrich, R., Schroter, H., Leuthold, H., & Birngruber, T. (2015). Automatic and controlled stimulus processing in conflict tasks: Superimposed diffusion processes and delta functions. *Cogn Psychol, 78*, 148-174. doi:10.1016/j.cogpsych.2015.02.005

van Ravenzwaaij, D., Cassey, P., & Brown, S. D. (2018). A simple introduction to Markov Chain Monte-Carlo sampling. *Psychon Bull Rev, 25*(1), 143-154. doi:10.3758/s13423-016-1015-8

van Ravenzwaaij, D., & Oberauer, K. (2009). How to use the diffusion model: Parameter recovery of three methods: EZ, fast-dm, and DMAT. *Journal of Mathematical Psychology, 53*(6), 463-473. doi:10.1016/j.jmp.2009.09.004

Vandekerckhove, J., Tuerlinckx, F., & Lee, M. D. (2011). Hierarchical diffusion models for two-choice response times. *Psychol Methods, 16*(1), 44-62. doi:10.1037/a0021765

Voss, A., Lerche, V., Mertens, U., & Voss, J. (2018). Sequential Sampling Models with Variable Boundaries and Non-Normal Noise: A Comparison of Six Models. *Paper submitted for publication*.

Voss, A., Nagler, M., & Lerche, V. (2013). Diffusion models in experimental psychology: a practical introduction. *Exp Psychol, 60*(6), 385-402. doi:10.1027/1618-3169/a000218

Voss, A., Rothermund, K., & Voss, J. (2004). Interpreting the parameters of the diffusion model: an empirical validation. *Mem Cognit, 32*(7), 1206-1220.

Voss, A., & Voss, J. (2007). Fast-dm: a free program for efficient diffusion model analysis. *Behav Res Methods, 39*(4), 767-775.

Voss, A., Voss, J., & Lerche, V. (2015). Assessing cognitive processes with diffusion model analyses: a tutorial based on fast-dm-30. *Front Psychol, 6*, 336. doi:10.3389/fpsyg.2015.00336

Table 1: Ranges for the parameter values used for the simulations

	a	z_r	v_{mean}	v_{diff}	t_0	s_{zr}	s_v	s_{t0}	α
min	0.5	0.3	-2.0	1.0	0.2	0.0	0.0	0.0	1.0
max	2.0	0.7	2.0	6.0	0.5	0.6	2.0	0.2	2.0

Notes: a: threshold separation; z_r: relative starting point; v_{mean}: drift criterion (mean drift across conditions); v_{diff}: difference of drift rates between conditions; t_0: non-decision time; s_{zr}: inter-trial-variability of starting point; s_v: inter-trial-variability of drift; s_{t0}: inter-trial-variability of non-decision time; α: Stability parameter of the noise distribution (only used in Study 2). Drift rates were re-parameterized as $v_0 = v_{mean} - 0.5 \cdot v_{diff}$ and $v_1 = v_{mean} + 0.5 \cdot v_{diff}$.

Table 2: Correlations of true and recovered values for the diffusion model parameters (Study 1)

Algorithm	a	z_r	v_0	v_1	t_0	s_{zr}	s_v	s_{t0}
				Uncontaminated Test Data				
DL 0%	0.98	0.97	0.97	0.97	0.99	0.38	0.68	0.85
DL 10%	0.98	0.97	0.97	0.97	0.99	0.36	0.69	0.82
DL 100%	0.98	0.95	0.97	0.97	0.98	0.32	0.68	0.76
fast-dm	0.98	0.98	0.96	0.96	0.98	0.38	0.60	0.89
				Contaminated Test Data				
DL 0%	0.98	0.88	0.96	0.97	0.97	0.26	0.66	0.71
DL 10%	0.98	0.94	0.97	0.97	0.98	0.36	0.67	0.71
DL 100%	0.98	0.96	0.97	0.97	0.99	0.45	0.68	0.83
fast-dm	0.96	0.69	0.92	0.92	0.96	0.04	0.51	0.63

Notes: DL 0%, DL 10%, and DL 100%: Deep Learning algorithm trained with 0%, 10%, or 100% contaminated data.

Table 3: Normalized Mean Absolute Error (NMAE) of recovered values from true diffusion model parameters (Study 1)

Algorithm	a	z_r	v_0	v_1	t_0	s_{zr}	s_v	s_{t0}
				Uncontaminated Test Data				
DL 0%	0.04	0.04	0.13	0.12	0.02	0.38	0.27	0.19
DL 10%	0.04	0.04	0.13	0.12	0.02	0.38	0.27	0.22
DL 100%	0.04	0.05	0.14	0.14	0.03	0.37	0.27	0.65
fast-dm	0.05	0.04	0.05	0.05	0.04	0.34	0.24	0.10
				Contaminated Test Data				
DL 0%	0.04	0.07	0.15	0.14	0.04	0.37	0.30	1.05
DL 10%	0.04	0.05	0.14	0.13	0.03	0.38	0.28	0.30
DL 100%	0.04	0.04	0.13	0.14	0.02	0.36	0.28	0.21
fast-dm	0.07	0.16	0.07	0.07	0.11	0.38	0.25	0.81

Notes: DL 0%, DL 10%, and DL 100%: Deep Learning algorithm trained with 0%, 10%, or 100%, respectively, contaminated data.

Table 4: Mean standard deviations (SD) of estimation errors and percentage of true parameter scores that fall within corresponding 95% confidence intervals (Study 1).

Algorithm		a	z_r	v_0	v_1	t_0	s_{zr}	s_v	s_{t0}
					Uncontaminated Test Data				
DL 0%	SD	0.091	0.030	0.353	0.348	0.017	0.389	0.158	0.030
	% in CI	0.985	0.975	0.976	0.966	0.987	0.956	0.968	0.964
DL 10%	SD	0.092	0.032	0.348	0.348	0.018	0.389	0.160	0.033
	% in CI	0.987	0.979	0.971	0.967	0.988	0.951	0.972	0.968
DL 100%	SD	0.087	0.032	0.358	0.355	0.016	0.386	0.157	0.033
	% in CI	0.979	0.897	0.957	0.957	0.896	0.939	0.948	0.555
					Contaminated Test Data				
DL 0%	SD	0.093	0.031	0.355	0.347	0.018	0.388	0.156	0.023
	% in CI	0.975	0.718	0.930	0.907	0.921	0.936	0.908	0.121
DL 10%	SD	0.095	0.039	0.360	0.364	0.019	0.397	0.159	0.038
	% in CI	0.974	0.934	0.964	0.960	0.973	0.950	0.965	0.883
DL 100%	SD	0.089	0.033	0.359	0.355	0.017	0.393	0.156	0.031
	% in CI	0.973	0.967	0.973	0.973	0.988	0.953	0.976	0.949

Notes: DL 0%, DL 10%, and DL 100%: Deep Learning algorithm trained with 0%, 10%, or 100% contaminated data.

Table 5: Correlations of true and recovered values for the Lévy-Flight model parameters (Study 2)

Algorithm	a	z_r	v_0	v_1	t_0	s_{zr}	s_v	s_{t0}	α
Uncontaminated Test Data									
DL 0%	0.95	0.95	0.97	0.97	0.98	0.30	0.84	0.95	0.95
DL 10%	0.95	0.95	0.97	0.96	0.98	0.30	0.84	0.95	0.95
DL 100%	0.95	0.92	0.97	0.96	0.97	0.26	0.83	0.95	0.92
ML	0.86	0.91	0.92	0.89	0.96	0.06	0.59	0.49	0.58
Contaminated Test Data									
DL 0%	0.94	0.87	0.96	0.96	0.95	0.28	0.83	0.94	0.87
DL 10%	0.95	0.91	0.96	0.96	0.96	0.29	0.83	0.95	0.91
DL 100%	0.95	0.94	0.97	0.96	0.98	0.31	0.83	0.95	0.94
ML	0.84	0.88	0.94	0.84	0.96	-0.01	0.58	0.50	0.45

Notes: DL 0%, DL 10%, and DL 100%: Deep Learning algorithm trained with 0%, 10%, or 100% contaminated data. ML: Maximum-likelihood based parameter search.

Table 6: Normalized Mean Absolute Error (NMAE) of recovered values for the Lévy-Flight model parameters (Study 2)

Algorithm	a	z_r	v_0	v_1	t_0	s_{zr}	s_v	s_{t0}
Uncontaminated Test Data								
DL 0%	0.07	0.07	0.04	0.04	0.04	0.23	0.13	0.17
DL 10%	0.07	0.07	0.04	0.04	0.04	0.23	0.13	0.18
DL 100%	0.07	0.09	0.04	0.05	0.09	0.24	0.13	0.29
fast-dm	0.13	0.10	0.06	0.08	0.07	0.32	0.21	0.31
Contaminated Test Data								
DL 0%	0.08	0.11	0.05	0.05	0.15	0.25	0.14	0.24
DL 10%	0.08	0.10	0.05	0.05	0.07	0.23	0.13	0.19
DL 100%	0.07	0.08	0.04	0.04	0.05	0.23	0.13	0.16
fast-dm	0.14	0.12	0.06	0.09	0.09	0.31	0.20	0.64

Notes: DL 0%, DL 10%, and DL 100%: Deep Learning algorithm trained with 0%, 10%, or 100%, respectively, contaminated data.

Table 7: Mean standard deviations (SD) of estimation errors and percentage of true parameter scores that fall within corresponding 95% confidence intervals (Study 2).

Algorithm		a	z_r	v_0	v_1	t_0	s_{zr}	s_v	s_{t0}
					Uncontaminated Test Data				
DL 0%	SD	0.135	0.038	0.37	0.352	0.019	0.310	0.162	0.041
	% in CI	0.950	0.953	0.973	0.953	0.986	0.953	0.978	0.947
DL 10%	SD	0.133	0.039	0.368	0.368	0.019	0.314	0.163	0.041
	% in CI	0.956	0.965	0.97	0.958	0.983	0.946	0.982	0.955
DL 100%	SD	0.129	0.036	0.372	0.369	0.018	0.328	0.163	0.038
	% in CI	0.925	0.87	0.949	0.932	0.739	0.965	0.929	0.673
					Contaminated Test Data				
DL 0%	SD	0.162	0.042	0.357	0.342	0.019	0.271	0.151	0.044
	% in CI	0.963	0.856	0.912	0.879	0.45	0.85	0.901	0.808
DL 10%	SD	0.146	0.045	0.378	0.375	0.025	0.314	0.16	0.044
	% in CI	0.961	0.939	0.959	0.945	0.932	0.94	0.974	0.929
DL 100%	SD	0.138	0.039	0.368	0.365	0.02	0.314	0.159	0.037
	% in CI	0.955	0.955	0.958	0.958	0.98	0.941	0.973	0.95

Notes: DL 0%, DL 10%, and DL 100%: Deep Learning algorithm trained with 0%, 10%, or 100% contaminated data.

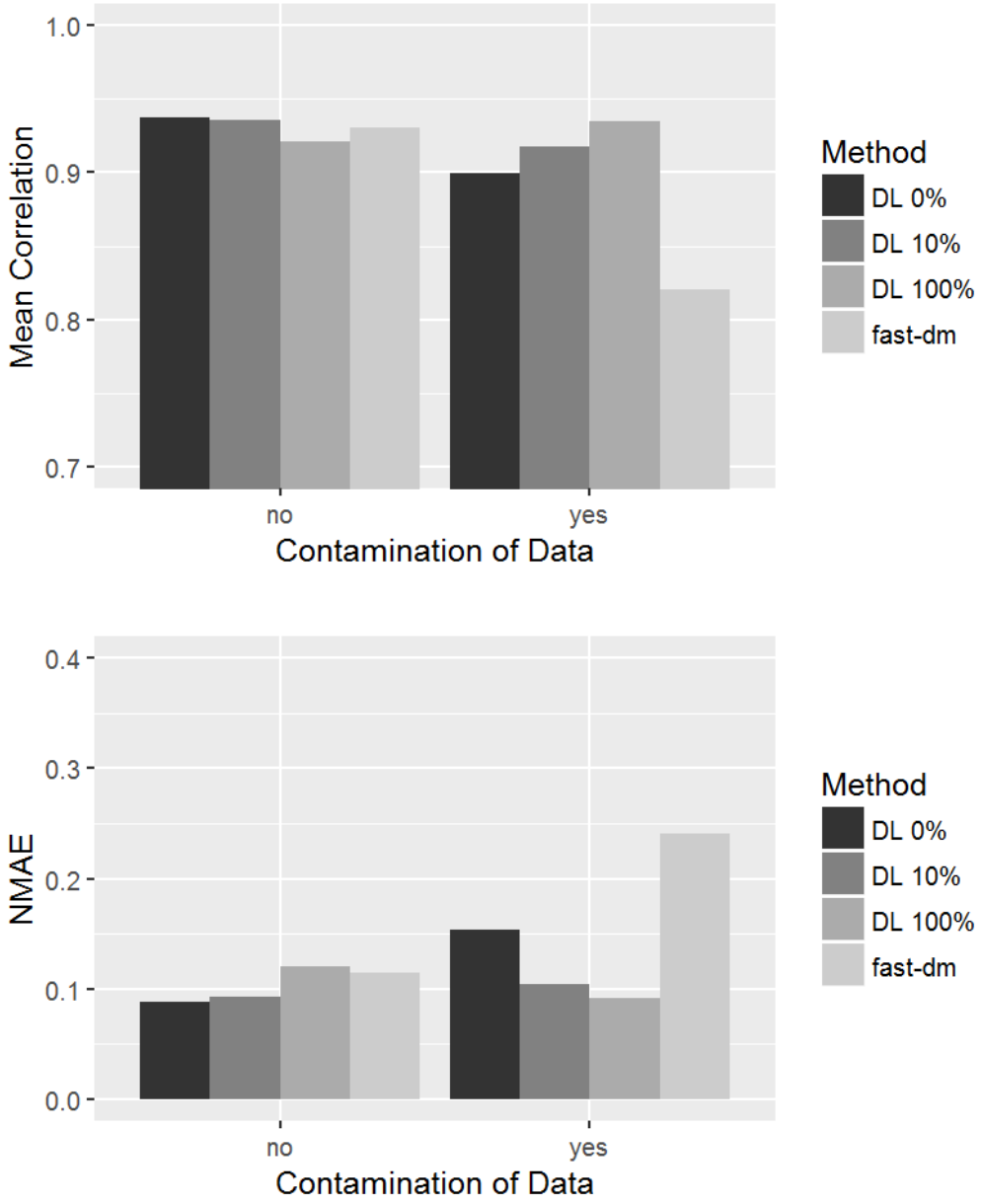

Figure 1. Recovery of Diffusion model Parameters. Mean Correlations (upper panel) and Normalized Mean Absolute Errors (NMAE, lower panel) are displayed as a function of the estimation algorithm and the contamination of data with 4% fast errors. DL 0%, DL 10%, and DL 100% indicates the Deep Learning algorithm trained with 0%, 10%, or 100%, respectively, contaminated data.

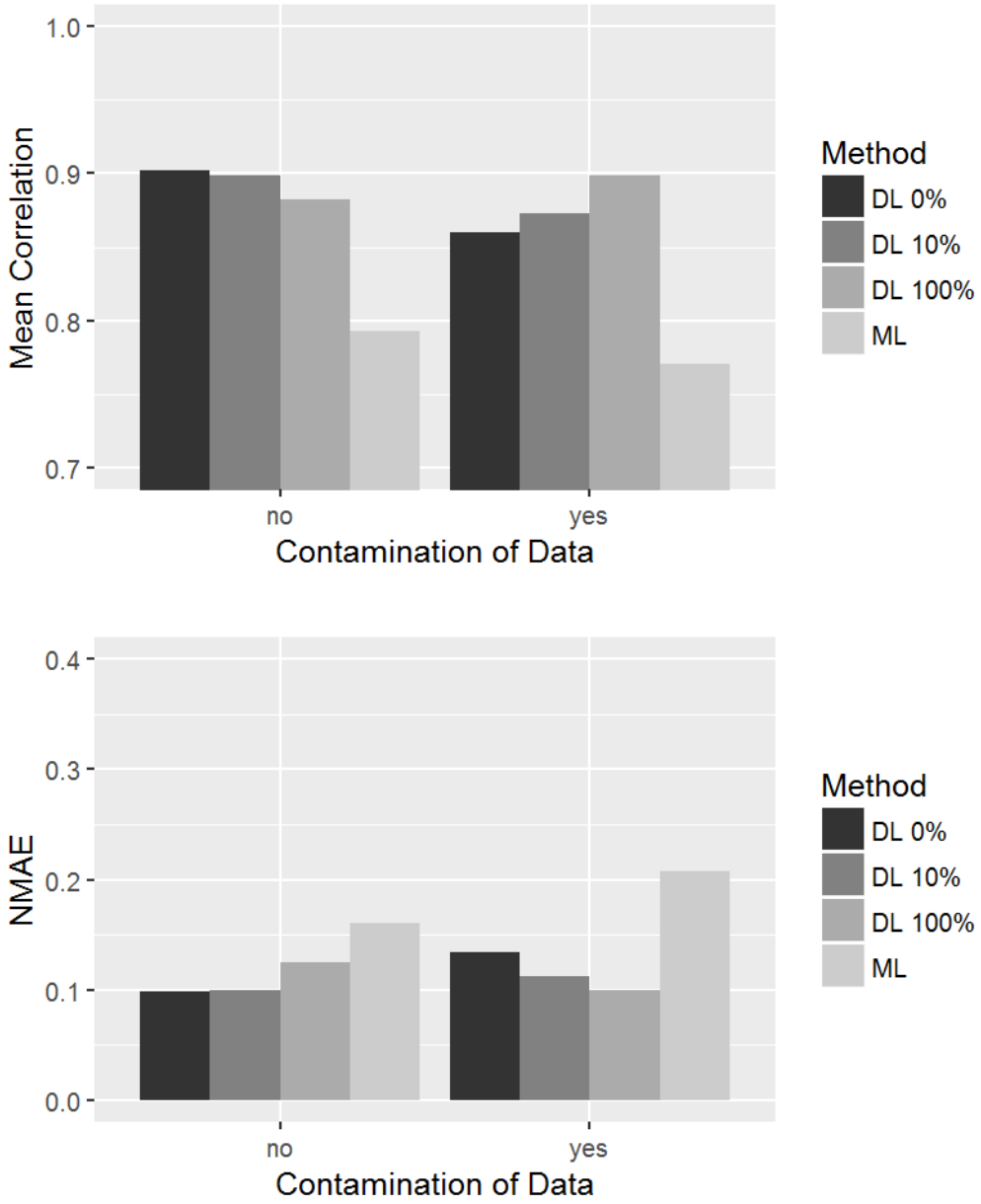

Figure 2. Recovery of Lévy-Flight model Parameters. Mean Correlations (upper panel) and Normalized Mean Absolute Errors (NMAE, lower panel) are displayed as a function of the estimation algorithm and the contamination of data with 4% fast errors. DL 0%, DL 10%, and DL 100% indicates the Deep Learning algorithm trained with 0%, 10%, or 100%, respectively, contaminated data.

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

**FAKULTÄT FÜR VERHALTENS-
UND EMPIRISCHE
KULTURWISSENSCHAFTEN**

Promotionsausschuss der Fakultät für Verhaltens- und Empirische Kulturwissenschaften der Ruprecht-Karls-Universität Heidelberg
Doctoral Committee of the Faculty of Behavioural and Cultural Studies of Heidelberg University

Erklärung gemäß § 8 (1) c) der Promotionsordnung der Universität Heidelberg für die Fakultät für Verhaltens- und Empirische Kulturwissenschaften
Declaration in accordance to § 8 (1) c) of the doctoral degree regulation of Heidelberg University, Faculty of Behavioural and Cultural Studies

Ich erkläre, dass ich die vorgelegte Dissertation selbstständig angefertigt, nur die angegebenen Hilfsmittel benutzt und die Zitate gekennzeichnet habe.
I declare that I have made the submitted dissertation independently, using only the specified tools and have correctly marked all quotations.

Erklärung gemäß § 8 (1) d) der Promotionsordnung der Universität Heidelberg für die Fakultät für Verhaltens- und Empirische Kulturwissenschaften
Declaration in accordance to § 8 (1) d) of the doctoral degree regulation of Heidelberg University, Faculty of Behavioural and Cultural Studies

Ich erkläre, dass ich die vorgelegte Dissertation in dieser oder einer anderen Form nicht anderweitig als Prüfungsarbeit verwendet oder einer anderen Fakultät als Dissertation vorgelegt habe.
I declare that I did not use the submitted dissertation in this or any other form as an examination paper until now and that I did not submit it in another faculty.

Vorname Nachname
First name Family name _____

Datum, Unterschrift
Date, Signature _____