Likelihood Approximations for Bayesian Analysis of Sequential Sampling Models

by
Alexander Fengler
MSc. Cognitive Psychology and Neuroscience, Maastricht University, 2014

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Cognitive, Linguistic & Psychological Sciences at Brown University

Providence, Rhode Island
February 2023

This dissertation by Alexander Fengler is accepted in its present form by the Department of Cognitive, Linguistic & Psychological Sciences as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date _____ _____

Michael J. Frank, Director

Recommended to the Graduate Council

Date _____ _____

Thomas Serre, Reader

Date _____ _____

Jason Ritt, Reader

Approved by the Graduate Council

Date _____ _____

Thomas A. Lewis
Dean of the Graduate School

Vita

MSc. Alexander Fengler

Updated December 28, 2022

Email: alexander_fengler@brown.edu **GitHub**: /AlexanderFengler **Citizenship**: German
Web: alexanderfengler.github.io **LinkedIn**: /alexander-fengler-89560b74

Education

University Maastricht Maastricht, Netherlands
BSc. **International Business** Sept. 2009 – Jul. 2012
Specialization: Finance
Thesis: Market Risk of Defined Contribution Systems in the Netherlands
Exchange Semester: National University Singapore

University Maastricht Maastricht, Netherlands
MSc. **Cognitive Psychology and Neuroscience** Sept. 2012 – Jul. 2014
Specialization: Neuroeconomics
Thesis: An application of the Drift Diffusion Model to Medium Size Choice Sets
Mentors: Prof. Arno Riedl, Prof. Alexander Vostroknutov, Prof. Antonio Rangel (California Institute of Technology)
Research Internship: California Institute of Technology Sept. 2013 - Jul. 2014
Representative Coursework:
- Mathematical Methods for Economics
- Microeconomic Theory
- Neuroanatomy

Brown University Providence, United States
Enrolled in PhD. Program in **Cognitive Science** Aug. 2015 – Jul. 2016 Year
Mentor: Prof. Joseph Austerweil (left department during first year)

Bocconi University Milan, Italy
MPhil. **Statistics** Sept. 2016 – Aug. 2017
Representative Coursework:
- Real Analysis
- Advanced Probability
- Advanced Data Analysis

Brown University Providence,
PhD. in **Cognitive Science** Jan. 2018 – Dec. 2022
Thesis Topic: Likelihood Approximations for Bayesian Analysis of Sequential Sampling Models
Mentors: Prof. Michael J. Frank, Prof. Thomas Serre, Prof. Jason Ritt
Representative Coursework:
- Computational Statistics
- Information Theory

Bayesian Computation

Research and work experience

Civil Service: Sep. 2008 – Apr. 2009
University Cologne, Biochemical Faculty
Responsibilities: Laboratory work (DNA extraction) and full responsibility over the fish-stock held for experimental purposes

Teaching Assistantship: Sept. 2010 – May 2011
University Maastricht, SBE Business School
Classes: Quantitative Methods I and II
Responsibility: Lead weekly sections for 10-15 students in a flipped classroom environment.

Research Assistantship: Sep. 2014 - May 2015
California Institute of Technology,
Rangel Neuroeconomics Laboratory
Continuation of MSc. Thesis project and leading experimental design and data collection process for a project in collaboration with a big software company.

Teaching Assistantships: Sep. 2018 - Sep. 2021
Brown University,
Department of Cognitive, Psychological and Linguistic Sciences
Classes: Quantitative Methods for Psychologists, Introduction to Psychology
Responsibility: Lead student sections, weekly open office hours and contribute to course organization.

Data Science Consulting: Sept. 2021 - now
PyMC Labs
Statistical Consulting with focus on Bayesian approaches. Lead two projects as principal Data Scientist for a client seeking data analysis and code infrastructure solutions concerning Bayesian inference for cognitive process models.

Honors and scholarships

Tuition Refund for top 3% GPA (University Maastricht)	2011
Cum Laude BSc., less than 5% of students (University Maastricht)	2012
Selected for research based Bachelor Thesis (University Maastricht)	2012
PhD. Scholarship (Brown University)	2015
Tuition Waiver for MPhil. (University Bocconi)	2016

Publications

Beyond Drift Diffusion Models: Fitting a Broad Class of Decision and Reinforcement Learning Models with HDDM
Alexander Fengler, Krishn Bera, Mads L. Pedersen, Michael J. Frank.
Journal of Cognitive Neuroscience, 2022.

Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience
Alexander Fengler, Lakshmi Govindarajan, Tony Chen, Michael J. Frank.
eLife, 2021.

Encoder-Decoder Neural Architectures for Fast Amortized Inference of CognitiveProcess Models
Alexander Fengler, Lakshmi Govindarajan, Michael J. Frank.
Proceedings of the annual meeting of the Cognitive Science Society, 2020.

Conference Posters

An application of the Drift Diffusion Model to Medium Size Choice Sets 2014
Alexander Fengler, Antonio Rangel
Society For Neuroeconomics Conference, Miami, United States

Neural Networks for Likelihood Estimation in Approximate Bayesian Computation: Application to Cognitive Process Models 2019
Alexander Fengler, Michael J. Frank
RLDM, Montreal, Canada

Encoder-Decoder Neural Architectures for Fast Amortized Inference of CognitiveProcess Models 2020
Alexander Fengler, Lakshmi Govindarajan, Michael J. Frank
Annual meeting of the Cognitive Science Society, Madison, United States

Likelihood Approximation Networks (LANs) for fast, tractable inference in cognitive process models 2022
Krishn Bera, Alexander Fengler, Michael J. Frank
RLDM, Providence, United States

Talks and Tutorials

Introduction to Approximate Bayesian Methods 2019
Guest Lecture in Computational Modeling Workshop, Brown University, Instructor: Andra Gana

Approximate Bayesian Computation with Neural Networks 2020
Guest Lecture in Carney Computational Modeling Workshop, Brown University, Instructor: Andra Gana

Likelihood Approximation Networks and Approximate Bayesian Computation 2021
Guest Lecture at the Toronto Decision Neuroscience Lab, Toronto University, Principal Investigator: Prof. Cendri Hutcherson

Tutorial on HDDM-LAN extension 2022
Guest Lecture in Carney Computational Modeling Workshop, Brown University, Instructor: Andra Gana

Basic Introduction to HDDM 2022
Workshop, Royal Holloway University, UK

Introduction to Recurrent Neural Networks 2022
Carney Brainstorm EEG Challenge: Workshop & Hackathon, Brown University

Hierarchical Bayesian Inference with SSMs 2022
Tutorial, Mental Effort Workshop, Brown University

Preface

Acknowledgements

Reflecting upon a journey of any significant duration, leads to a clash of prospective and retrospective. A clash that is not only between two states of information, but also between two people. My ideas about how my PhD would unfold were characterized by the expectation of a level of linearity, which the irregularities of my final path through the program make a mockery of. The winding road was ordained by a combination of random events and autonomous interventions, decisions that I made as a result of the growth induced by my experiences in the program itself. These experiences in turn were shaped by the people I intersected with and I would like to extend my gratitude to a few of these individuals in the following few lines. First, I am grateful for the initial chance to attempt the PhD in Cognitive Science at Brown University, granted by my first advisor Joseph Austerweil. Joe unfortunately left Brown after my first year, but he set the tone for a supportive environment fueling excitement for intellectual exploration and putting his students needs before his own. After Joe left I was extremely fortunate to find a second habitat, equally conducive to free spirited inquiry and supportive of students, under the supervisor of this thesis, Michael J. Frank. I can not thank Michael enough for having allowed me to find shelter in his lab after a difficult (unexpectedly nonlinear) moment in my life. Michael guides his students with an invisible hand, that provides the simultaneous feelings of free inquiry and trust that the ship will sail home. A kind of unique magic that leaves one puzzled. But the PhD is finally not defined only by the student/supervisor relationship, it is also shaped by the friends, life partners and unexpected auxiliary advisors we find along the way. I am especially grateful to my friends Babak Hemmatian, Lakshmi Govindarajan, Ivan Felipe Rodriguez, Rannie Xu, Cris Buc Calderon, Ivan Grahek, Tyler Barnes Diana, Sofia Verba and Jatan Buch. I learned a lot from all of you, but most importantly you made my time in Providence eventful and enjoyable. Moreover I thank Nicholas Garcia Trillos, a mentor and friend, whose support at the initial stages of grad school gave me the confidence to continue my pursuits in applied mathematics, a passion that I cultivated since the beginning of the program. I also want to thank my wife Tiantian Li for molding the final three years of the PhD into an even greater adventure and I am looking forward to playing this role in her own future educational exploits. I will never forget your unwavering affection in moments when grad school stresses in combination with Covid-19 had temporarily expanded my physical shape into proportions that you could not have predicted. Lastly, I thank my family for their steady support on this journey which we all had foreseen to be a little shorter.

Contents

List of Tables

List of Figures

Abstract of "Likelihood Approximations for Bayesian Analysis of Sequential Sampling Models" by Alexander Fengler, Ph.D., Brown University, December 2022.

Computational Modeling is a significant part of cognitive neuroscience. It allows arbitration between suggested models by virtue of quantitative fits to behavioral, brain and other data. Such fits are often derived via the machinery of Bayesian statistics. The resulting focus on posterior distributions over model parameters, instead of single point estimates, allows thorough investigation of model properties, including parameter trade-offs, as well as the intelligent infusion of prior knowledge into estimation problems by virtue of specification of prior distributions over said parameters. However an important limitation of the standard Bayesian approach is the reliance on easy to compute likelihood functions to render inference computationally tractable. Utilizing the power of Bayesian statistical inference is complicated by an important aspect of the computational scientist's day to day operations. Newly developed computational models, exactly when they present themselves at the cutting edge frontier of the research landscape, are often available only as data simulators, without the convenience of separately derived analytical likelihood functions, therefore limiting the application of sophisticated statistical machinery for parameter inference. This issue has spurred a host of research activity in the last one and a half decades aiming at tractable Bayesian inference directly from data simulators. The main work in this thesis deals with a particular approach to this problem, which we dubbed likelihood approximation networks (LANs), one manifestation of what has become a paradigm of using deep learning for the pre-computation (amortization) of important derived functions from expensive simulations. Log-likelihoods are learned from model simulations via deep neural networks. These learned likelihoods (and other quantities) are then flexibly re-usable across arbitrary inference scenarios, distinguishing LANs from competing methods which tend to be more specialized towards single inference problems. I define the method, and use as a test-bed the vital class of cognitive processing models, known as sequential sampling models (SSMs), showing how LANs can dramatically expand the class of SSMs accessible for fast Bayesian inference. Moreover, I developed an extension to the widely used HDDM toolbox for hierarchical Bayesian modeling with, a simple but canonical SSM, the drift diffusion model (DDMs). This extension uses LANs to generalize HDDM to a much larger class of SSMs. Lastly, I test the feasibility of using LANs with modern approaches to inference such as Hamiltonian Monte Carlo, which makes use of gradients of likelihood functions for improved sampling performance, as well as Variational Inference, which turns Bayesian inference into an optimization problem and often yields, approximate, posterior distributions much more rapidly than standard Markov Chain Monte Carlo techniques would. The overarching intend of this work is to showcase the power of likelihood approximations for simulation based inference with focus on applications in computational cognitive science and hopefully. As a result, we hope to impact the day to day workflow of computational cognitive scientists in an effort towards broad gains in efficiency and consequently modeling flexibility.

Abstract of "Likelihood Approximations for Bayesian Analysis of Sequential Sampling Models" by Alexander Fengler, Ph.D., Brown University, December 2022.

Computational Modeling is a significant part of cognitive neuroscience. It allows arbitration between suggested models by virtue of quantitative fits to behavioral, brain and other data. Such fits are often derived via the machinery of Bayesian statistics. The resulting focus on posterior distributions over model parameters, instead of single point estimates, allows thorough investigation of model properties, including parameter trade-offs, as well as the intelligent infusion of prior knowledge into estimation problems by virtue of specification of prior distributions over said parameters. However an important limitation of the standard Bayesian approach is the reliance on easy to compute likelihood functions to render inference computationally tractable. Utilizing the power of Bayesian statistical inference is complicated by an important aspect of the computational scientist's day to day operations. Newly developed computational models, exactly when they present themselves at the cutting edge frontier of the research landscape, are often available only as data simulators, without the convenience of separately derived analytical likelihood functions, therefore limiting the application of sophisticated statistical machinery for parameter inference. This issue has spurred a host of research activity in the last one and a half decades aiming at tractable Bayesian inference directly from data simulators. The main work in this thesis deals with a particular approach to this problem, which we dubbed likelihood approximation networks (LANs), one manifestation of what has become a paradigm of using deep learning for the pre-computation (amortization) of important derived functions from expensive simulations. Log-likelihoods are learned from model simulations via deep neural networks. These learned likelihoods (and other quantities) are then flexibly re-usable across arbitrary inference scenarios, distinguishing LANs from competing methods which tend to be more specialized towards single inference problems. I define the method, and use as a test-bed the vital class of cognitive processing models, known as sequential sampling models (SSMs), showing how LANs can dramatically expand the class of SSMs accessible for fast Bayesian inference. Moreover, I developed an extension to the widely used HDDM toolbox for hierarchical Bayesian modeling with, a simple but canonical SSM, the drift diffusion model (DDMs). This extension uses LANs to generalize HDDM to a much larger class of SSMs. Lastly, I test the feasibility of using LANs with modern approaches to inference such as Hamiltonian Monte Carlo, which makes use of gradients of likelihood functions for improved sampling performance, as well as Variational Inference, which turns Bayesian inference into an optimization problem and often yields, approximate, posterior distributions much more rapidly than standard Markov Chain Monte Carlo techniques would. The overarching intend of this work is to showcase the power of likelihood approximations for simulation based inference with focus on applications in computational cognitive science and hopefully. As a result, we hope to impact the day to day workflow of computational cognitive scientists in an effort towards broad gains in efficiency and consequently modeling flexibility.

Chapter 1

General Introduction

Statistical methods serve as the backbone of the inductive process which progresses the body of scientific knowledge in the empirical sciences. They do so by defining the rules of a game in which wrong hypothesis and theories shall be replaced by less wrong hypotheses and theories, in the hope of constructing a stochastic process that, collectively embodied by the work of researchers active in a given discipline, slowly converges closer and closer to some notion of scientific truth. The rules of the game, as to how to adjudicate between such theories and hypotheses, are themselves of course subject to change, driven by fierce debates in the community of statistical methodologists, famously the development of two schools of statistics: Frequentism and Bayesianism. For the empirical scientists this implies a need to have an eye on the paradigmatic developments of statisticians in order to maintain the highest agreed upon standard of rigor at a given time. Demarcation lines may emerge, defining points or periods in time where paradigms have shifted.

In cognitive neuroscience we can define three such shifts. A shift from Frequentist to Bayesian statistics. A shift from a focus on group means to parameter estimation with stochastic generative models of phenomena. Lastly a shift from likelihood based to Likelihood Free Inference (LFI). The last shift, unfolding fervently in the last few decades, served as main motivation for the work in this thesis. Refraining from precise commitments to a strict historical timeline, we characterize these three shifts in some more detail below.

First is the shift from the once dominant paradigm of frequentist statistics, characterized by concepts such as p-values, confidence intervals and null-hypothesis testing, towards the Bayesian statistical paradigm, which defines as its object of interest a posterior distribution over parameters of a given probability model (Gelman, Carlin, et al., 1995). The Frequentist / Bayesian fault line and corresponding debates reach far into the past and deep into the foundations of statistical science (Fisher, 1925; Neyman and Pearson, 1933; Leonard Jimmie Savage, 1959; Leonard J Savage, 1972; Lindley, 1957; Lindley, 2000; De Finetti, 1979). Frequentist approaches, epitomized by the ubiquitous usage of p-values (and confidence intervals) to report and dignify (disgrace) scientific results have largely dominated the landscape across a broad range of scientific disciplines (Ioannidis, 2005; Eric-Jan Wagenmakers, Wetzels, et al., 2011), including psychology, cognitive science and neuroscience. Most recently in the wake of the so-called *replication crisis* in psychology and other diciplines (C. F. Camerer et al., 2016), voices extolling the benefits of a Bayesian approach, were again heard more prominently (Eric-Jan Wagenmakers, Marsman, et al., 2018; Eric-Jan Wagenmakers, Love, et al., 2018) . Inflated false discovery rates (Benjamini and Hochberg, 1995; Collaboration, 2015; Simmons, Nelson, and Simonsohn, 2016), the rate at which claimed discoveries end up being false positives, were linked to the misuse of p-values as a reporting tool for scientific results (Greenland et al., 2016; McShane et al., 2019; Gibson, 2021), resulting in a slew of recommendations concerning the improvement of research methodology (McShane et al., 2019; Ioannidis, 2015; Daniel Lakens et al., 2018; Daniël Lakens, 2021; Benjamin et al., 2018). The Bayesian approach has a few noteworthy virtues, including the ability to directly incorporate knowledge about the a priori probability of hypothesis to be true (via prior specification), by which false discovery rates can be regulated in an intuitive manner (Eric-Jan Wagenmakers, Marsman, et al., 2018; Eric-Jan Wagenmakers, Wetzels,

et al., 2011).

Much has been written in this regard, however important for the work presented here that we can observe an overall shift towards the Bayesian approach. This manifests in the increasing chance to encounter researchers with at least a passing familiarity of Bayesian inference as well es in the proliferation of related teaching materials and supporting software packages (Plummer et al., 2003; Gelman, Carlin, et al., 1995; Gelman and Hill, 2006; Lee and Eric-Jan Wagenmakers, 2014; Kruschke, 2014; McElreath, 2020; Love et al., 2019; Eric-Jan Wagenmakers, Love, et al., 2018; Carpenter et al., 2017; Phan, Pradhan, and Jankowiak, 2019; Bingham et al., 2019; Salvatier, Wiecki, and Fonnesbeck, 2016; Ntzoufras, 2011; D. J. Lunn et al., 2000; D. Lunn et al., 2009; Murphy, 2007).

Second, we can observe a rise in ambition as to what constitutes a model and correspondingly what defines the object that serves as the basis for inference algorithms. The canonical statistical curriculum for experimental scientists was formed by frequentist techniques for the systematic analysis of the behavior of group means across experimental treatments. Orthodox examples are such (still quite ubiquitously applied) methods as the analysis of variance (ANOVA) or the analysis of covariances (ANCOVA). Essentially, the focus of analysis is whether the difference between some or all group means (a group can be defined by a cohort of subjects and / or by a variation in experimental condition) is *statistically significantly* different from the difference expected under a so-called null-model, which one wishes to reject. This paradigm is slowly giving way to another mode of thinking about what constitutes a *statistical model*. Increasingly, computationally minded cognitive scientists and neuroscientists are not satisfied with the definition of a theory or model as a simple collection of expected group means or the expected violation of a specified null pattern of group means. Instead recent decades have seen an increased use of generative computational models, shifting the statistical analysis focus towards parameter estimation and away from the simple discrimination of group means. This in turn reinforces the shift towards Bayesian approaches, since the Bayesian analysis toolbox deals with important aspects of parameter inference, such as the investigation of parameter trade-offs, more naturally than other commonly used inference approaches like maximum likelihood estimation (Gelman, Carlin, et al., 1995; McElreath, 2020). These models can give detailed computational accounts of, e.g., the choice processes underlying perceptual discrimination tasks and allow rigorous quantitative fits to multiple aspects of behavior and brain data.

A prime example of this shift in mindset is exemplified by the drift diffusion model (Ratcliff, 1978a; Ratcliff and McKoon, 2008; Ratcliff, Smith, et al., 2016), originally proposed as a joint account of reaction times and choices in memory tasks (Ratcliff, 1978a). This model has a decades-old history of applications (Ratcliff, Smith, et al., 2016) and variations of this model are still being proposed (Wieschen, A. Voss, and S. Radev, 2020; Cisek, Puskas, and El-Murr, 2009a; Holmes, Trueblood, and Heathcote, 2016; Trueblood et al., 2021; Tillman, Van Zandt, and Logan, 2020). Perhaps the most influential software package to aid researchers in fitting these models to experimental data, is the HDDM package for Python (Wiecki, Sofer, and Frank, 2013). Unlike its competitors (A. Voss and J. Voss, 2007; Vandekerckhove and Tuerlinckx, 2008; Drugowitsch, 2016; Shinn, Lam, and J. D. Murray, 2020; Ahn, Haines, and Zhang, 2017; Heathcote et al., 2019), it fully embraces the

Bayesian approach and has been applied to a large variety of experimental settings, offering great flexibility in accommodating a variety of experimental designs.

More recently, we observe the third shift from likelihood-based methods towards a broader adoption of likelihood-free methods for inference (discussed in detail in chapter 2). This can be viewed as an inevitable consequence of the shift towards computational mechanistic models mentioned above and its resulting world of possible generative models which become candidates for investigation. A major selling point, and crucial factor in the adoption of Bayesian methods, is the promise that the Bayesian paradigm allows researchers to flexibly express their ideas and theories as stochastic generative models followed by a simple invocation of the Bayesian statistical toolbox (typically one or another Markov Chain Monte Carlo method) to perform inference over respective parameters. The *Markov Chain Monte Carlo Revolution* (Diaconis, 2009; C. Robert and Casella, 2011) (simultaneous advances in computing hardware and techniques for posterior sampling) notwithstanding, a crucial detail usually remains implicit only to rear its head again further down the road: standard Bayesian methods rely on the repeated evaluation of the right hand side in Bayes rule,

$$p(\theta|\mathbf{x}) \propto p(\mathbf{x}|\theta)p(\theta)$$

Here θ is the parameter vector of interest, $\mathbf{x}$ is the observed data, $p(\theta)$ is called the prior distribution over said parameter vector, $p(\mathbf{x}|\theta)$ is called the likelihood (which for a given parameter vector determines a probability distribution over possible outcomes of our main generative model) and $p(\theta|\mathbf{x})$ is the posterior distribution over parameters given the observed data, our main object of interest. We notice that the evaluation of the right hand side crucially depends on the ability to evaluate $p(\mathbf{x}|\theta)$. Now, it is often much easier to write a simulator (that is, a function from which we can generate data given some input parameters), than it is to derive an analytical likelihood for said simulator. This puts a dent in the Bayesian promise concerning expressive modeling, which is in fact anchored to the ability to provide such analytical likelihoods a priori. Without easy to evaluate likelihoods, standard inference algorithms cannot be applied.

Recognizing that many such likelihood functions may be difficult to derive analytically, if not completely intractable, instigated a growing research program focused on statistical methods for Bayesian inference in the *likelihood-free* setting (Sisson, Fan, and M. Beaumont, 2018). In this setting, access to a simulator is granted, but no corresponding analytical likelihood functions. Any inference pipeline has to proceed from this starting point. A likelihood function can be approximated only empirically here, using, e.g., Monte Carlo simulation (C. P. Robert, Casella, and Casella, 1999). A prime motivating and canonical example is the Lotka-Volterra predator-prey model (Lotka, 1925; Volterra, 1926), utilized routinely in the field of computational biology. This model is integral to the discipline, but Bayesian parameter estimation is notoriously hard due to a lack of closed-form likelihoods (M. A. Beaumont, 2010).

An essential example in the cognitive and neurosciences can be found in the realm of cognitive process models for the joint analysis of reaction time and choice data., The so-called Ratcliff Diffusion Model (Ratcliff, 1978b; Ratcliff, Smith, et al., 2016) forms the seed of a whole class of applicable computational models, dubbed Sequential Sampling Models (SSMs). Conceptually the idea behind

these models is to treat a decision process as a *diffusion to bound* mechanism, where momentary evidence is integrated over time to drive a decision particle across a prespecified bound. The time-point and location of these boundary crossings jointly represent reaction times and choices (these models will be explained in more detail in 2 and 3). Importantly, the class of SSMs subsumes a host of theoretically significant variations on the drift diffusion model (DDM) (Ratcliff, 1978a; Ratcliff, Smith, et al., 2016), which may incorporate more complicated boundary regions (Cisek, Puskas, and El-Murr, 2009b; Cisek, Puskas, and El-Murr, 2009a; Malhotra et al., 2018; Palestro et al., 2018; Evans, Trueblood, and Holmes, 2020), excitation/inhibition (Usher and McClelland, 2001) or attention (Krajbich, Lu, et al., 2012a) mechanisms , as well as other additions (Wieschen, A. Voss, and S. Radev, 2020; Pedersen, Frank, and Biele, 2017) deemed necessary to capture empirical data collected across a growing range of experimental paradigms (Krajbich and Rangel, 2011; Krajbich, Lu, et al., 2012b; Cavanagh and Frank, 2014; Pedersen, Frank, and Biele, 2017; Herz et al., 2016). These models have strong theoretical support and have garnered wide interest. However, publications that combine computational modeling with empirical data analysis have nearly exclusively focused on the most basic variants of SSMs or close and equally-simple competitors (S. D. Brown and Heathcote, 2008; Holmes, Trueblood, and Heathcote, 2016; Tillman, Van Zandt, and Logan, 2020). The reason is, again, a lack of analytic likelihoods. Few examples to the contrary existed until recently a surge of new likelihood-free methods received attention (Turner and Sederberg, 2014; Holmes, 2015; S. T. Radev et al., 2020; Greenberg, Nonnenmacher, and Macke, 2019; Fengler et al., 2021; Boelts et al., 2022). These methods go by various names, Simulation Based Inference (SBI), Approximate Bayesian Computation (ABC) and Likelihood Free Inference (LFI) being common. I will use them interchangeably, favoring SBI (Cranmer, Brehmer, and Louppe, 2020).

The work in this thesis is motivated by the potential I see in the methods emerging from this shift towards likelihood-free inference and chapter 2 will discuss a large variety of these approaches in detail. I believe that progress on the front of SBI algorithms will help to fully realize the Bayesian promise towards expressive modeling, spurred by scientists' creativity in designing mechanistic models, instead of expertise in mathematical statistics. While the methods I develop in the following are much more general, the discussion will mostly focus on the application to the aforementioned class of SSMs, which has proved to be of great interest to the cognitive science and neuroscience communities. By targeting this relevant and rich class of models, and hence concentrating efforts, I hope that the work exhibited in the following chapters will serve the community of cognitive modelers by ultimately allowing for the discovery of new scientific insights. On the other hand I hope to have contributed a layer to the foundation of methods which I believe will shake the computational sciences more generally in the decades to come (Papamakarios and I. Murray, 2016; Papamakarios, Pavlakou, and I. Murray, 2017; Papamakarios, Sterratt, and I. Murray, 2019; Durkan et al., 2019; Greenberg, Nonnenmacher, and Macke, 2019; Lueckmann et al., 2019; S. T. Radev et al., 2020; Fengler et al., 2021; Boelts et al., 2022).

This thesis is organized as follows. chapter 2 provides an in-depth review of SBI methods, focused in particular on the emerging intersection between Neural Networks and Simulation Based Inference

based inference. Pointers to applications in the cognitive neurosciences help contextualize this review with an eye toward these disciplines as target for application. chapter 3 presents our main contribution to the staple of SBI algorithms: Likelihood approximation networks. I specifically show how this algorithm can be applied to the broad class of cognitive process models under the category of sequential sampling models, and how it distinguishes itself from other competing algorithms in the space of SBI. The following two chapters present natural progressions of the work exhibited in chapter 3. First, chapter 4 presents a tutorial introduction to an extension of the HDDM Python software package, which provides, amongst other innovations, access to likelihood approximation networks, and shows how they can be combined with reinforcement learning models. We discuss how this and other such software can serve as a catalyst for the adoption of likelihood-free inference methods by the wider research community, eventually leading to advances regarding the core scientific questions in the cognitive and neurosciences. Second, chapter 5 presents a proof of concept application of likelihood approximation networks in the context of modern inference algorithms. The work in chapter 5 is motivated by the desire to further improve the speed of inference, which in turn allows experimental cognitive- and neuroscientists to test a wider range of candidate models against their empirical data. I close in chapter 6 with a general discussion of the research presented in the preceding chapters.

References

Ahn, Woo-Young, Nathaniel Haines, and Lei Zhang (2017). "Revealing Neurocomputational Mechanisms of Reinforcement Learning and Decision-Making With the hBayesDM Package". In: *Computational Psychiatry* 1, pp. 24–57. URL: doi:10.1162/CPSY_a_00002.

Beaumont, Mark A (2010). "Approximate Bayesian computation in evolution and ecology". In: *Annual review of ecology, evolution, and systematics* 41, pp. 379–406.

Benjamin, Daniel J et al. (2018). "Redefine statistical significance". In: *Nature human behaviour* 2.1, pp. 6–10.

Benjamini, Yoav and Yosef Hochberg (1995). "Controlling the false discovery rate: a practical and powerful approach to multiple testing". In: *Journal of the Royal statistical society: series B (Methodological)* 57.1, pp. 289–300.

Bingham, Eli et al. (2019). "Pyro: Deep universal probabilistic programming". In: *The Journal of Machine Learning Research* 20.1, pp. 973–978.

Boelts, Jan et al. (2022). "Flexible and efficient simulation-based inference for models of decision-making". In: *Elife* 11, e77220.

Brown, Scott D and Andrew Heathcote (2008). "The simplest complete model of choice response time: Linear ballistic accumulation". In: *Cognitive psychology* 57.3, pp. 153–178.

Camerer, Colin F et al. (2016). "Evaluating replicability of laboratory experiments in economics". In: *Science* 351.6280, pp. 1433–1436.

Carpenter, Bob et al. (2017). "Stan: a probabilistic programming language." In: *Grantee Submission* 76.1, pp. 1–32.

Cavanagh, James F and Michael J Frank (2014). "Frontal theta as a mechanism for cognitive control". In: *Trends in cognitive sciences* 18.8, pp. 414–421.

Cisek, Paul, Geneviève Aude Puskas, and Stephany El-Murr (2009a). "Decisions in changing conditions: the urgency-gating model". In: *Journal of Neuroscience* 29.37, pp. 11560–11571.

— (2009b). "Decisions in changing conditions: the urgency-gating model". In: *Journal of Neuroscience* 29.37, pp. 11560–11571.

Collaboration, Open Science (2015). "Estimating the reproducibility of psychological science". In: *Science* 349.6251, aac4716.

Cranmer, Kyle, Johann Brehmer, and Gilles Louppe (2020). "The frontier of simulation-based inference". In: *Proceedings of the National Academy of Sciences* 117.48, pp. 30055–30062.

De Finetti, Bruno (1979). "Probability and exchangeability from a subjective point of view". In: *International Statistical Review/Revue Internationale de Statistique*, pp. 129–135.

Diaconis, Persi (2009). "The markov chain monte carlo revolution". In: *Bulletin of the American Mathematical Society* 46.2, pp. 179–205.

Drugowitsch, Jan (2016). "Fast and accurate Monte Carlo sampling of first-passage times from Wiener diffusion models". In: *Scientific reports* 6, p. 20490.

Durkan, Conor et al. (2019). "Neural spline flows". In: *Advances in Neural Information Processing Systems*, pp. 7511–7522.

Evans, Nathan J, Jennifer S Trueblood, and William R Holmes (2020). "A parameter recovery assessment of time-variant models of decision-making". In: *Behavior research methods* 52.1, pp. 193–206.

Fengler, Alexander et al. (2021). "Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience". In: *Elife* 10, e65074.

Fisher, Ronald Aylmer (1925). "Theory of statistical estimation". In: *Mathematical proceedings of the Cambridge philosophical society*. Vol. 22. 5. Cambridge University Press, pp. 700–725.

Gelman, Andrew, John B Carlin, et al. (1995). *Bayesian data analysis*. Chapman and Hall/CRC.

Gelman, Andrew and Jennifer Hill (2006). *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press.

Gibson, Eric W (2021). "The role of p-values in judging the strength of evidence and realistic replication expectations". In: *Statistics in Biopharmaceutical Research* 13.1, pp. 6–18.

Greenberg, David, Marcel Nonnenmacher, and Jakob Macke (2019). "Automatic posterior transformation for likelihood-free inference". In: *International Conference on Machine Learning*. PMLR, pp. 2404–2414.

Greenland, Sander et al. (2016). "Statistical tests, P values, confidence intervals, and power: a guide to misinterpretations". In: *European journal of epidemiology* 31.4, pp. 337–350.

Heathcote, Andrew et al. (2019). "Dynamic models of choice". In: *Behavior research methods* 51.2, pp. 961–985.

Herz, Damian M et al. (2016). "Neural correlates of decision thresholds in the human subthalamic nucleus". In: *Current Biology* 26.7, pp. 916–920.

Holmes, William R (2015). "A practical guide to the Probability Density Approximation (PDA) with improved implementation and error characterization". In: *Journal of Mathematical Psychology* 68, pp. 13–24.

Holmes, William R, Jennifer S Trueblood, and Andrew Heathcote (2016). "A new framework for modeling decisions about changing information: The Piecewise Linear Ballistic Accumulator model". In: *Cognitive psychology* 85, pp. 1–29.

Ioannidis, John PA (2005). "Why most published research findings are false". In: *PLoS medicine* 2.8, e124.

— (2015). "How to make more published research true". In: *Revista Cubana de Información en Ciencias de la Salud (ACIMED)* 26.2, pp. 187–200.

Krajbich, Ian, Dingchao Lu, et al. (2012a). "The attentional drift-diffusion model extends to simple purchasing decisions". In: *Frontiers in psychology* 3, p. 193.

— (2012b). "The attentional drift-diffusion model extends to simple purchasing decisions". In: *Frontiers in psychology* 3, p. 193.

Krajbich, Ian and Antonio Rangel (2011). "Multialternative drift-diffusion model predicts the relationship between visual fixations and choice in value-based decisions". In: *Proceedings of the National Academy of Sciences* 108.33, pp. 13852–13857.

Kruschke, John (2014). "Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan". In.

Lakens, Daniël (2021). "The practical alternative to the p value is the correctly used p value". In: *Perspectives on psychological science* 16.3, pp. 639–648.

Lakens, Daniel et al. (2018). "Justify your alpha". In: *Nature human behaviour* 2.3, pp. 168–171.

Lee, Michael D and Eric-Jan Wagenmakers (2014). *Bayesian cognitive modeling: A practical course*. Cambridge university press.

Lindley, Dennis V (1957). "A statistical paradox". In: *Biometrika* 44.1/2, pp. 187–192.

— (2000). "The philosophy of statistics". In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 49.3, pp. 293–337.

Lotka, Alfred James (1925). *Elements of physical biology*. Williams & Wilkins.

Love, Jonathon et al. (2019). "JASP: Graphical statistical software for common statistical designs". In: *Journal of Statistical Software* 88, pp. 1–17.

Lueckmann, Jan-Matthis et al. (2019). "Likelihood-free inference with emulator networks". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 32–53.

Lunn, David et al. (2009). "The BUGS project: Evolution, critique and future directions". In: *Statistics in medicine* 28.25, pp. 3049–3067.

Lunn, David J et al. (2000). "WinBUGS-a Bayesian modelling framework: concepts, structure, and extensibility". In: *Statistics and computing* 10.4, pp. 325–337.

Malhotra, Gaurav et al. (2018). "Time-varying decision boundaries: insights from optimality analysis". In: *Psychonomic bulletin & review* 25.3, pp. 971–996.

McElreath, Richard (2020). *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC.

McShane, Blakeley B et al. (2019). "Abandon statistical significance". In: *The American Statistician* 73.sup1, pp. 235–245.

Murphy, Kevin (2007). "Software for graphical models: A review". In: *International Society for Bayesian Analysis Bulletin* 14.4, pp. 13–15.

Neyman, Jerzy and Egon Sharpe Pearson (1933). "IX. On the problem of the most efficient tests of statistical hypotheses". In: *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 231.694-706, pp. 289–337.

Ntzoufras, Ioannis (2011). *Bayesian modeling using WinBUGS*. John Wiley & Sons.

Palestro, James J et al. (2018). "Some task demands induce collapsing bounds: Evidence from a behavioral analysis". In: *Psychonomic bulletin & review* 25.4, pp. 1225–1248.

Papamakarios, George and Iain Murray (2016). "Fast ε-free Inference of Simulation Models with Bayesian Conditional Density Estimation". In: *Advances in Neural Information Processing Systems* 29, pp. 1028–1036.

Papamakarios, George, Theo Pavlakou, and Iain Murray (2017). "Masked autoregressive flow for density estimation". In: *Advances in Neural Information Processing Systems*, pp. 2338–2347.

Papamakarios, George, David Sterratt, and Iain Murray (2019). "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 837–848.

Pedersen, Mads Lund, Michael J Frank, and Guido Biele (2017). "The drift diffusion model as the choice rule in reinforcement learning". In: *Psychonomic bulletin & review* 24.4, pp. 1234–1251.

Phan, Du, Neeraj Pradhan, and Martin Jankowiak (2019). "Composable effects for flexible and accelerated probabilistic programming in NumPyro". In: *arXiv preprint arXiv:1912.11554*.

Plummer, Martyn et al. (2003). "JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling". In: *Proceedings of the 3rd international workshop on distributed statistical computing*. Vol. 124. 125.10. Vienna, Austria., pp. 1–10.

Radev, Stefan T et al. (2020). "BayesFlow: Learning complex stochastic models with invertible neural networks". In: *arXiv preprint arXiv:2003.06281*.

Ratcliff, Roger (1978a). "A theory of memory retrieval." In: *Psychological review* 85.2, p. 59.

— (1978b). "A theory of memory retrieval." In: *Psychological review* 85.2, p. 59.

Ratcliff, Roger and Gail McKoon (2008). "The diffusion decision model: theory and data for two-choice decision tasks". In: *Neural computation* 20.4, pp. 873–922.

Ratcliff, Roger, Philip L Smith, et al. (2016). "Diffusion decision model: Current issues and history". In: *Trends in cognitive sciences* 20.4, pp. 260–281.

Robert, Christian and George Casella (2011). "A short history of Markov chain Monte Carlo: Subjective recollections from incomplete data". In: *Statistical Science* 26.1, pp. 102–115.

Robert, Christian P, George Casella, and George Casella (1999). *Monte Carlo statistical methods*. Vol. 2. Springer.

Salvatier, John, Thomas V Wiecki, and Christopher Fonnesbeck (2016). "Probabilistic programming in Python using PyMC3". In: *PeerJ Computer Science* 2, e55.

Savage, Leonard J (1972). *The foundations of statistics*. Courier Corporation.

Savage, Leonard Jimmie (1959). *Subjective probability and statistical practice*. Mathematical Sciences Directorate, Office of Scientific Research, US Air Force.

Shinn, Maxwell, Norman H Lam, and John D Murray (2020). "A flexible framework for simulating and fitting generalized drift-diffusion models". In: *Elife* 9, e56938.

Simmons, Joseph P, Leif D Nelson, and Uri Simonsohn (2016). "False-positive psychology: undisclosed flexibility in data collection and analysis allows presenting anything as significant." In.

Sisson, Scott A, Yanan Fan, and Mark Beaumont (2018). *Handbook of approximate Bayesian computation*. CRC Press.

Tillman, Gabriel, Trish Van Zandt, and Gordon D Logan (2020). "Sequential sampling models without random between-trial variability: The racing diffusion model of speeded decision making". In: *Psychonomic Bulletin & Review* 27.5, pp. 911–936.

Trueblood, Jennifer S et al. (2021). "Urgency, leakage, and the relative nature of information processing in decision-making." In: *Psychological Review* 128.1, p. 160. DOI: 10.1037/rev0000255.

Turner, Brandon M and Per B Sederberg (2014). "A generalized, likelihood-free method for posterior estimation". In: *Psychonomic bulletin & review* 21.2, pp. 227–250.

Usher, Marius and James L McClelland (2001). "The time course of perceptual choice: the leaky, competing accumulator model." In: *Psychological review* 108.3, p. 550.

Vandekerckhove, Joachim and Francis Tuerlinckx (2008). “Diffusion model analysis with MATLAB: A DMAT primer”. In: *Behavior research methods* 40.1, pp. 61–72.

Volterra, Vito (1926). *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. Società anonima tipografica" Leonardo da Vinci".

Voss, Andreas and Jochen Voss (2007). “Fast-dm: A free program for efficient diffusion model analysis”. In: *Behavior research methods* 39.4, pp. 767–775.

Wagenmakers, Eric-Jan, Jonathon Love, et al. (2018). “Bayesian inference for psychology. Part II: Example applications with JASP”. In: *Psychonomic bulletin & review* 25.1, pp. 58–76.

Wagenmakers, Eric-Jan, Maarten Marsman, et al. (2018). “Bayesian inference for psychology. Part I: Theoretical advantages and practical ramifications”. In: *Psychonomic bulletin & review* 25.1, pp. 35–57.

Wagenmakers, Eric-Jan, Ruud Wetzels, et al. (2011). “Why psychologists must change the way they analyze their data: the case of psi: comment on Bem (2011).” In.

Wiecki, Thomas V, Imri Sofer, and Michael J Frank (2013). “HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python”. In: *Frontiers in neuroinformatics* 7, p. 14.

Wieschen, Eva Marie, Andreas Voss, and Stefan Radev (2020). “Jumping to conclusion? a lévy flight model of decision making”. In: *TQMP* 16.2, pp. 120–132.

Chapter 2

An overview of deep learning based approximate Bayesian inference

This paper reviews the history of Approximate Bayesian Computation with focus on recent methodological advances stemming from a fruitful cross-pollination with probabilistic deep learning approaches. I pay particular attention to the benefits of these methods for experimental and theoretical cognitive neuroscientists. The promise of these emerging methods is a substantial increase in the class of computational models for which Bayesian inference is feasible, which may result in disruptive changes in the discipline as a whole.

2.1 Introduction

Computational modeling has carved out a substantial niche in cognitive neuroscience today. Computational models can guide the principled interpretations of functional demands of cognitive systems, while at best also maintaining a level of analytical tractability in producing quantitative fits between brain and behavior data. A prominent example is the application of reinforcement-learning (Sutton and Barto, 2018) models to interpret brain and behavioral data (Collins and Frank, 2014; Maia and Frank, 2011; Dayan and Niv, 2008). These have been used to interpret cognitive markers of Parkinson's disease via asymmetric learning behavior (Frank, Seeberger, and O'reilly, 2004), link genetic, neural and phamracological modulators of exploration exploitation behavior (Frank, Doll, et al., 2009; Badre et al., 2012; Kayser et al., 2015; Zajkowski, Kossut, and Wilson, 2017), and moreover link reward prediction errors and neural signals that are predictive of learning (Gläscher et al., 2010; Pessiglione et al., 2006; McCoy et al., 2019). Generally, reinforcement learning models aim to provide an account of decision sequences, for which decision making is progressively adapted (learned) as a result of trial-wise error signals.

Other prominent examples include models of dynamic decision making processes (Busemeyer and Townsend, 1993; Ratcliff and McKoon, 2008; Ratcliff, 1978; Brown and Heathcote, 2008; Holmes, Trueblood, and Heathcote, 2016; Usher and McClelland, 2001; Cisek, Puskas, and El-Murr, 2009; Krajbich, Lu, et al., 2012), which aim to jointly account for reaction time and choice data in a variety of forced alternative choice scenarios. These models generally treat decision making as a result of a threshold-interrupted process of evidence accumulation (although variations abound at this point) and have been applied to a wide variety of choice modalities, (Ratcliff, Huang-Pollock, and McKoon, 2018; Ratcliff and Frank, 2012; Krajbich, Lu, et al., 2012; Milosavljevic et al., 2010; Sullivan et al., 2015; Fisher, 2017; Bakkour et al., 2018). The parameters of such models allow for a principled account of the impacts of, for example, reward magnitude, probabilistic reward distribution and attention (Balci et al., 2011; Krajbich, Lu, et al., 2012; Mads Lund Pedersen, Frank, and Biele, 2017) on decision making behavior.

Modeling frameworks may be linked, allowing one model to serve as a latent process which affects the parameters of a second model across trials. Such a link is established between reinforcement learning and choice/reaction time modeling (Mads Lund Pedersen, Frank, and Biele, 2017; Miletić, Boag, and Forstmann, 2020; Fontanesi, Gluth, et al., 2019; Fontanesi, Palminteri, and Lebreton, 2019).

Similarly, EEG, fMRI (Frank, Gagne, et al., 2015; Cavanagh and Frank, 2014) and Eyetracking (Krajbich, Armel, and Rangel, 2010; Krajbich, Lu, et al., 2012; Cavanagh, Wiecki, et al., 2014) data may generally be used for the same purpose as latent covariate processes. Conceptual links between theories concerning optimal allocation of cognitive resources (Lieder and Griffiths, 2020), fixation patterns and choice / reaction time behavior, can be developed by interlocking component-wise computational models. Figure 2.1 illustrates those connections and the interplay with (adaptive) experimental designs pictorially.

Further motivation for the use of computational models is their potential for turning parameter

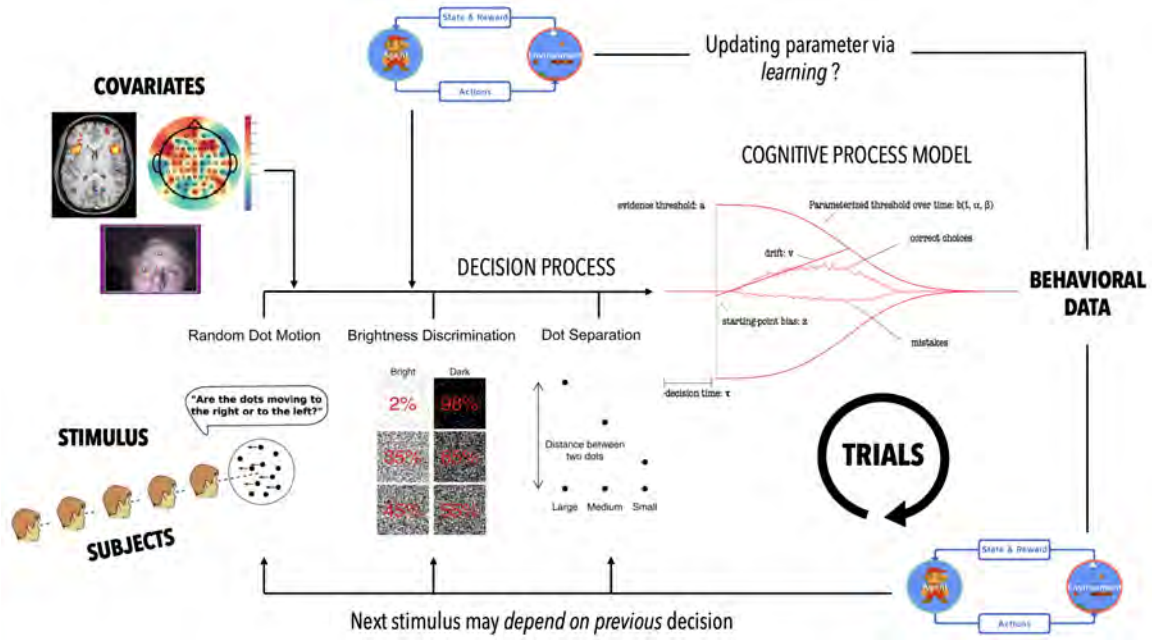

Figure 2.1. Graphical summary of the general connections between computational models discussed in this section. In general our *subjects* perform one of many tasks (three examples given, with random dot motion, brightness discrimination and dot separation). Our *cognitive process model*, is the computational generative model under investigation: I assume subject data derives abstractly from the computational process instantiated in this model. The resulting (observed) behavioral data (*right*) may impact the next stimulus and / or the parameter configuration of the model (e.g. via a process that facilitates learning from feedback). The parameters of the model may also be affected by any type of *covariate* (*upper right*) such as fMRI BOLD / EEG signals or e.g. pupil dilation.

estimates into theoretically driven dimensionality reduction of brain / behavioral data to be used for the prediction of e.g. clinical status in computational psychiatry (Huys, Maia, and Frank, 2016). This contrasts purely data-driven approaches which would rely solely on machine learning techniques. These tend to lack interpretability and therefore hamper the taxonomizing effects which cognitive process models offer for human behavior, healthy and defective. Moreover ad-hoc data-driven approaches tend to lack the affordances in sample complexity (Poggio, Smale, et al., 2003), which a solid quantitative theory with few degrees of freedom provides. The parameters of such theory driven models may therefore prove more successfull in predicting behavior or brain-states than aforementioned data-driven approaches (Geana et al., 2022; Mads L Pedersen et al., 2021; Wiecki, Poland, and Frank, 2015; Wiecki, Antoniades, et al., 2016). Figure 2.2, illustrates these ideas.

Despite the advantages that computational modeling undoubtedly confers, there are significant roadblocks, which hamper its effective deployment and broader adoption.

One principal problem seems to be the necessary time investment to familiarize experimental cognitive neuroscientists with all building blocks of the computational and statistical machinery involved. Mastery of the mathematical / technical toolkits necessary to effectively make use many of the cutting edge computational models and inference procedures can not be expected from the broad

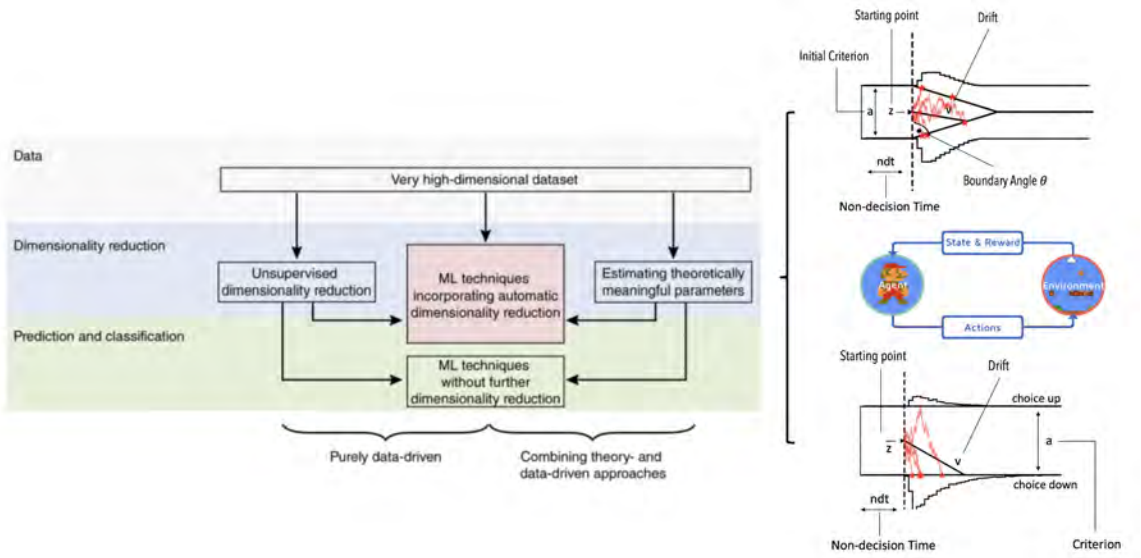

Figure 2.2. This figure, as adapted from (Huys, Maia, and Frank, 2016), conceptualizes the distinction between machine learning based dimensionality reduction of high dimensional behavioral (and other) datasets and an approach that utilizes computational cognitive models for the same purpose. The discipline of computational psychiatry (while not a priori dismissive of data-driven analysis) intends to utilize the later techniques for the purpose of basing diagnostic criteria for mental health on the mechanistic distinction in behavior as seen through the eyes of such computational cognitive models. This contrasts with current practices which form the criteria for classification of mental health problems (First, 2013).

research community. The bridge between the largely separate pools of expertise (experimental and computational) is the supply of easy to use, properly validated software which abstracts away the details of computational and statistical machinery. Examples of such software are plentiful (Shinn, Lam, and J. D. Murray, 2020; Wiecki, Sofer, and Frank, 2013; Vandekerckhove and Tuerlinckx, 2008; A. Voss and J. Voss, 2007) and citations suggest them to be of great use to the community, aiding consistent application of computational (mechanistic) models in the analysis of behavioral and / or brain data. As an example, over 400 papers made use of the HDDM Python package (Wiecki, Sofer, and Frank, 2013) which allows hierarchical Bayesian estimation of Ratcliff Diffusion Models (DDMs) and moreover linkage of model parameters to latent covariates such as EEG or fMRI BOLD signals (Ratcliff, 1978; Ratcliff and McKoon, 2008).

A second fundamental roadblock to the application of computational models in the analysis of experimental data is related to the often overlooked chasm between model construction and effective inference procedures for parameter fitting. The conceptual construction of computational models, the ability to construct data simulators for a given model (forward problem), and the ability to perform inference over the parameters of such a model given observed data (inverse problem) are in fact distinct enough to warrant extra attention. It will be the main focus of this review.

It is usually straightforward to proceed from even highly complicated model formulations to the implementation of a respective stochastic parametric forward simulator, not least because such

models tend to be *conceived* as parameterized stochastic simulators in the first place. Much harder however tends to be the derivation of an *analytical likelihood function* $\ell_{\mathcal{M}}(\theta|\mathbf{x})$, the probability (density) of a data-point $\mathbf{x}$ (dataset) given the parameters θ of such a model M.

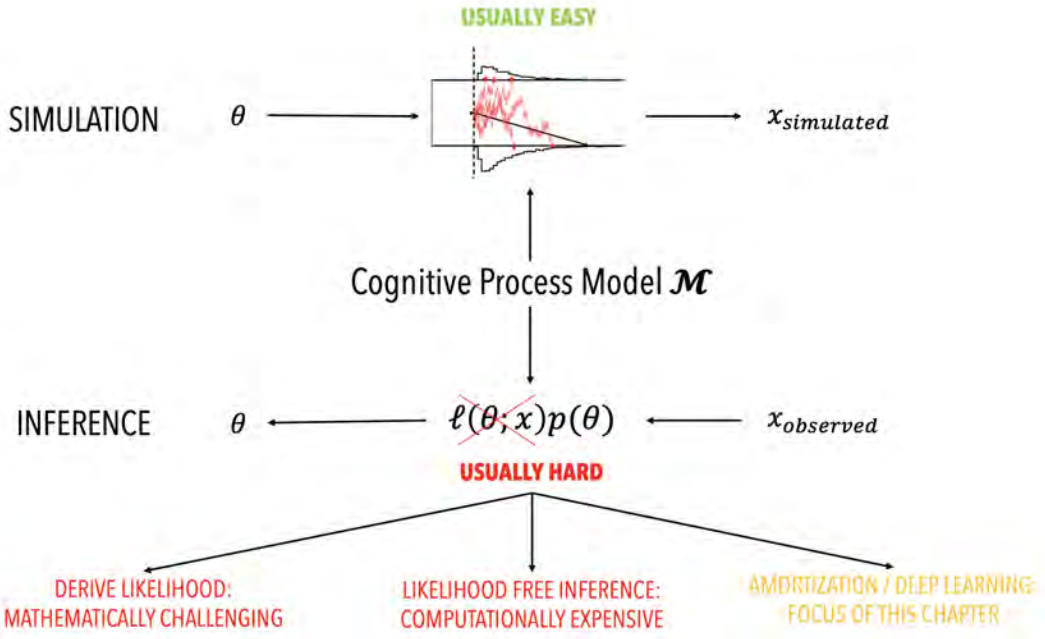

Figure 2.3. The Figure illustrates the high level problem presented when working with stochastic process models without access to a likelihood function. The *forward problem*, simulating data from the model, is usually easy. The *inverse problem*, parameter inference given data, is usually the difficult part. If a-priori one has access only to a simulator, performing Bayesian inference needs to follow one of three routes. Either one *derives a likelihood function* for the given model $\mathcal{M}$. Even for seemingly mundane models such as the *drift diffusion model*, such a derivation takes enormous mathematical expertise. Next, one can rely on *traditional Likelihood Free Inference (LFI)*, which often ends up computationally prohibitively expensive. Lastly, one can make use of the arsenal of *deep learning based approaches* the comparative benefits of which form a major part of the content of this review.

As it turns out, standard parameter inference procedures for probabilistic models such as *maximum likelihood estimation* (MLE) which rely on a point estimate,

$$\underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} \ell(\theta|\mathbf{x_i}) \tag{2.1}$$

or Bayesian inference, which relies on the *posterior distribution*,

$$p(\theta|\{\mathbf{x}_i\}_{i=1}^N) \propto \prod_{i=1}^{N} \ell(\theta|\mathbf{x_i})\pi(\theta) \tag{2.2}$$

all rest on the *likelihood function* in their computations. Here, $\{\mathbf{x}_i\}_{i=1}^N$ is a set of data-points assumed to be independent samples and $\pi(\theta)$ refers to some *prior* distribution over the parameters θ. It is common to *not* have access to the likelihood function directly but have easy access to a data

simulator, motivating *Likelihood Free Inference* (LFI) methods (Tavaré et al., 1997; Sisson, Fan, and M. Beaumont, 2018). The Bayesian variant of LFI is traditionally encountered under the acronym ABC for *Approximate Bayesian Computation* (Sisson, Fan, and M. Beaumont, 2018), alternatively also as *Simulation Based Inference* (SBI) (Cranmer, Brehmer, and Louppe, 2020). These methods all assume access only to a data simulator and devise parameter inference algorithms for this context. ABC algorithms have been successfully applied in a variety of scientific disciplines, including biology (Wood, 2010; M. A. Beaumont, 2010), cognitive science (Turner and Van Zandt, 2012; Turner and Van Zandt, 2014; Turner and Sederberg, 2014; Turner and Van Zandt, 2018) and astronomy (Schafer and Freeman, 2012; Ishida et al., 2015). The emergence of accessible software packages for the *R* and *Python* programming languages (Michael U Gutmann and Corander, 2016; Lintusaari et al., 2018; Ishida et al., 2015; Alsing et al., 2019; Wiecki, Sofer, and Frank, 2013; Salvatier, Wiecki, and Fonnesbeck, 2016) makes an increasingly large subset of cutting edge algorithms available to non-experts.

The aim of this chapter is to provide a comprehensive overview of approaches towards LFI, with an eye towards applications in computational cognitive and neuroscience, specifically to experimental data from arbitrarily complex experimental designs. I focus primarily on methods that facilitate Bayesian inference, however some of the presentation is applicable for frequentist inference as well. Moreover, I emphasize a modern machine learning and especially deep learning point of view towards LFI which is currently gaining traction and will likely provide state of the art methods for some time to come.

In section 2.2 I begin with a set of motivating examples from the computational cognitive science literature. I refer back to these examples when convenient for illustration in later sections. Section 2.3 provides a short discussion of general approaches to approximate inference procedures, to characterize LFI in this territory. Section 2.4 provides a short history of traditional ABC approaches, highlighting the different components, and strengths and weaknesses of respective algorithms. This sets the stage for a discussion of modern deep learning based approaches to ABC in section 2.5. This section gives a thorough overview of the state of the art. Section 3.6 uses the preceding developments of techniques, for a full discussion of respective strengths and weaknesses. I moreover develop guidelines for the application of algorithms, depending on the model / problem structure. Section 2.7 gives a short survey on the existing software landscape dedicated to LFI. Section 2.8 closes with a short conclusion and some remarks on limitations of this review.

2.2 LFI: Motivating example from Computational Cognitive Science

As a motivating example from computational cognitive science I will look at the class of sequential sampling models. These serve as a good example, for two reasons. First, the widely applied simpler versions of these models such as the basic drift diffusion models (e.g. Ratcliff and McKoon, 2008) in fact do provide an analytical likelihood function (Navarro and Fuss, 2009), which drives their wide

adoption as a function of convenience. However simultaneously well motivated but likelihood free alternatives (Cisek, Puskas, and El-Murr, 2009; Palestro, Weichart, et al., 2018; Krajbich, Lu, et al., 2012) (although see Srivastava et al., 2017 for attempts at deriving analytic solutions for a class of such models) tend to be ignored by the experimental community. Second, extensions of basic diffusion models serve as an abstraction link towards neural level models (O'Connell et al., 2018). Such models tend to not be easily estimable due to their complexity, however produce higher level computational summaries which in turn may be modeled with the help of extended drift diffusion models.

The basic drift diffusion model (DDM) (Ratcliff, 1978) is prevalently used since the 1970s to jointly account for reaction time and choice data in a variety of 2 alternative forced choice (2AFC) tasks (see Figure 2.4. According to this model, decisions are the result of an abstract particle crossing an *evidence threshold*, after a process of stochastic *evidence accumulation*, during which perceptual information is integrated. The original formulation of this model considers the evidence threshold fixed over time, and the evidence accumulation process is instantiated as a Gaussian random walk which terminates once it surpasses said threshold. The sign of the random walk position at the point of its magnitude reaching the threshold, signifies the respective choice taken. The reaction time is considered the time-point of threshold crossing. Of theoretical appeal this model instantiates what is known as the sequential probability ratio test (SPRT) (Wald and Wolfowitz, 1948). Interpreting the level of the evidence threshold as the requested power of a statistical test and the choice as the acceptance / rejection of a hypothesis for which discretely sampled observations provide stochastic evidence, this test minimizes the amount of samples needed for a decision. Another way to describe it is that the SPRT tracks the likelihood ratio between two hypothesis via sequential sampling. For a given likelihood ratio threshold, the test minimizes the average number of samples needed to take a decision. While not discussed in further detail here, I note that this paradigm has generalizations to multi-alternative choice problems with correspondingly named Multihypothesis Sequential Probability Ratio Tests (MSPRTs) (Draglia, Tartakovsky, and Veeravalli, 1999; Dragalin, Tartakovsky, and Veeravalli, 2000).

Considering the perceptual data deriving from an experiment stimulus (canonically a trial in a dot-motion task paradigm) as if providing evidence continuously in time, the evidence accumulation process assumed under the DDM is then usually defined as the following stochastic differential equation (SDE),

$$d\mathbf{X}_{\tau+t} = v\,dt + d\mathbf{W}_t,\ \mathbf{X}_\tau = w \tag{2.3}$$

where $\mathbf{X}$ represents the particle in our discussion. The basic DDM then includes as parameters a starting point w (capturing potential response biases or priors), an evidence drift v (capturing the perceptual or other evidence accumulated) and a non-decision time τ (capturing the time for perceptual encoding and motor output). Lastly completing the model we have a parameter a representing the evidence threshold. A choice is then made at a timepoint t if $\mathbf{X}$ either crosses a, or falls below 0. Hence, the parameter vector for the DDM is then $\theta = (v, a, w, \tau)$.

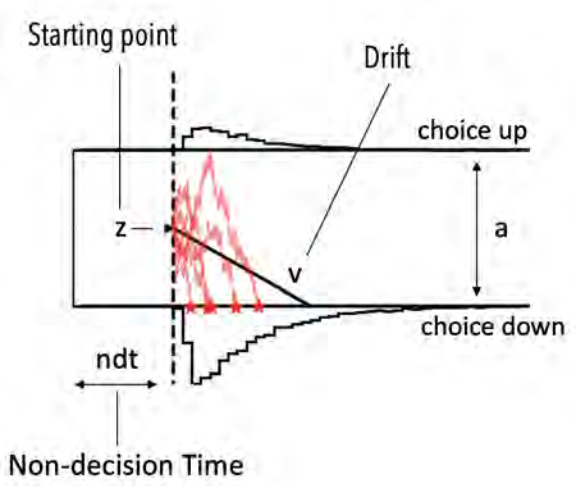

Figure 2.4. Graphical illustration of the basic DDM. Boundary crossings of the red particles (illustrated via the red stars)are treated as choice / reaction time pairs. The black histograms provide example distributions for both choices (up and down). The model's four basic parameters are the starting point (z), which determines some a priori bias in the decision process, a non-decision time (ndt) which captures an initial delay in the decision process, the drift (v) which determines a constant underlying rate of evidence over time and the boundary separation or criterion (a) which determines the amount of evidence necessary for a decision.

The *likelihood* $\ell_c(v, a, z, \tau|t)$ of observing a choice / reaction-time pair $\{t, c\}$, assuming $t \in \mathbb{R}^+$, and $c \in \{-1, 1\}$ is traditionally represented as an infinite series (W. and V., 1968),

$$\ell_c(v, a, z, \tau|t) = \ell(\tilde{v}_c, \tilde{a}, \tilde{z}_c|\tilde{t}) = \begin{cases} \frac{\pi}{\tilde{a}^2} \exp\left(-\tilde{v}_c\tilde{a}\tilde{z}_c - \frac{\tilde{v}_c^2\tilde{t}}{2}\right) \sum_{i=1}^{\infty} k \exp\left(-\frac{k^2\pi^2\tilde{t}}{2\tilde{a}^2}\right) \sin(k\pi\tilde{z}_c) & \text{if } \tilde{t} \geq 0 \\ 0 & \text{if } \tilde{t} < 0 \end{cases} \tag{2.4}$$

where,

$$\begin{aligned} \tilde{t} &= t - \tau \\ \tilde{v}_c &= cv \\ \tilde{a} &= 2a \\ \tilde{z}_c &= \begin{cases} \frac{w}{2a} & \text{if } c = 1 \\ 1 - \frac{w}{2a} & \text{if } c = -1 \end{cases} \end{aligned}$$

Fast algorithms to evaluate this likelihood, based on intelligent truncation, exist (Navarro and Fuss, 2009; Foster and Singmann, 2021). The model is moreover easy to simulate via the Euler-Maruyama method, which suggests discretizing in time into timesteps Δt, setting $\mathbf{X}_\tau = w$, and then following,

$$\mathbf{X}_{\tau+i\Delta t} = \mathbf{X}_{\tau+(i-1)\Delta t} + v\Delta t + \sqrt{\Delta t} * \mathcal{N}(0, 1) \tag{2.5}$$

The DDM allows relatively convenient standard likelihood based statistical analysis such as hierarchical bayesian modeling, including the incorporation of complex experiment designs and trial-by-trial regressions on parameters(Vandekerckhove, Tuerlinckx, and M. D. Lee, 2011; Wiecki, Sofer, and Frank, 2013).

Over time however more elaborate versions of the DDM were proposed, which increased the flexibility of the model. The elaborations were motivated on theoretical grounds (Cisek, Puskas, and El-Murr, 2009; Usher and McClelland, 2001; Wieschen, A. Voss, and S. Radev, 2020; Krajbich, Lu, et al., 2012), but also by more practical considerations such as creating variations which may account for more flexible experimental paradigms (prominent examples include the increase of the number of choice options, or varying the magnitude of the evidence signal within a given trial).

Prominent examples of such models include the full-DDM, which treats τ, w and v as random variables (Ratcliff and McKoon, 2008), the attentional DDM (aDDM) (Krajbich, Lu, et al., 2012) which includes attention as a modulator of the evidence accumulation process, the Leaky Competing Accumulator (LCA) model, which includes particle position dependent inhibition and excitation (Usher and McClelland, 2001), the urgency gating model (Cisek, Puskas, and El-Murr, 2009) as well as other models that allow for variations in the decision threshold which considers accuracy trade-offs over time (most commonly considering the decision bounds to collapse over time to eventually force a decision), or models that allow the drift to change over time due to delayed onset of alternative sources of evidence. Recently (Wieschen, A. Voss, and S. Radev, 2020) suggested evidence accumulation models which can be characterized as Levy Flights. All of the proposed models can be extended to model choice processes over $N > 2$ choice options. The umbrella term *sequential sampling models* (SSMs) is widely used to refer to the resulting class of models.

We can distinguish these models according to the difficulty of obtaining (approximate) likelihoods. Figure 2.5 (Fengler et al., 2020) provides an overview of this difficulty landscape with a couple of examples. As a result of what is illustrated in Figure 2.5, the simple DDM and full-DDM models do possess a supporting infrastructure of software libraries in the programming languages R, Python and Matlab, which aids their broad application to data, while lack of such support can be observed for more complicated models, especially concerning Bayesian inference. This inevitably has hindered their application to experimental data even if they are theoretically just as, if not more, suitable for a given experimental paradigm / dataset. While the support system grows, a notable example being the recent appearance of the Python package pyDDM (Shinn, Lam, and J. D. Murray, 2020), it does not yet cover many of the models listed above.

A striking example of how small changes in the model formulation, which demand only trivial changes to the simulator code, result in vastly different demands from the inference machine, is the contrast between the DDM and the Levy Flight models illustrated in Figure 2.6.

There moreover exist other models which attempt to simplify the DDM for the purpose of easing the statistical analysis, while maintaining what are considered crucial characteristics of the model. Notable examples include the linear ballistic accumulator model (LBA), out of which naturally grew the attentional LBA (aLBA) (Evans, Trueblood, and Holmes, 2020) and the EZ diffusion model

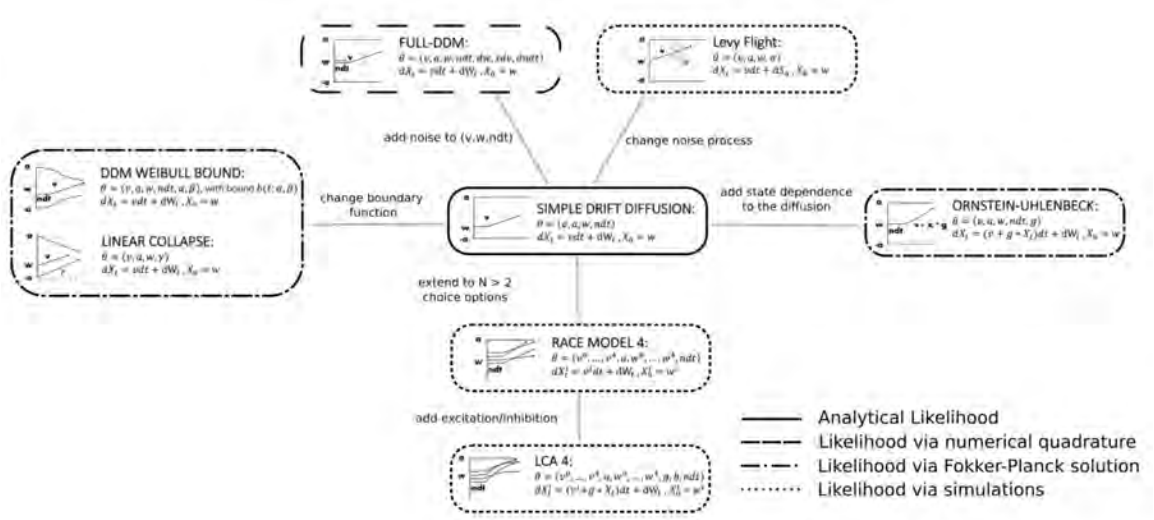

Figure 2.5. Systematically varying aspects of the SSMs gives us a range of relevant model formulations. The resulting models can be divided into four classes according to the difficulty they present with likelihood evaluations. I consider the simple DDM in the *analytical likelihood* (**solid line**) category, although strictly speaking, as discussed in the main text, the likelihood still demands an approximation algorithm. This algorithm is however sufficiently fast to not consider it a major computational bottleneck. The Full-DDM model demands *numerical quadrature* (**dashed line**) for likelihood evaluation, which integrates over variability parameters. This can easily inflated the cost of likelihood evaluation by two orders of magnitude. For some models, likelihood approximations have been derived via the *Fokker-Planck equations* (**dotted-dashed lined**) which similarly incurs non-trivial evalutation cost. Last, for some models no approximations based on analytical methods exist and one needs to resort to computationally expensive *simulations* to get likelihood estimates (**dotted line**).

(Wagenmakers, Van der Maas, and Grasman, 2007). The existence of these models is in large part motivated by the computational burden that derives even from the estimation of the standard DDM model, which is orders of magnitude less than what is demanded for Bayesian inference performed on the models that lack a likelihood function to begin with. The need for such reduced form models serves as one demonstration of the importance of smart methods to minimize the computational burden of the more complex original models.

We are concerned with the following properties of the resulting SSMs.

1. Simulating any of these models using the Euler-Maruyama method is neither conceptually not implementation-wise significantly harder than simulating the simple DDM. It may however be costlier.

2. The derivation of analytical likelihood functions is either unresolved or requires highly advanced mathematical expertise to attempt, or if resolved may lead to computations which are as expensive as simply executing simulations from the model.

3. The separate models can be characterized as varying building blocks of an underlying basic SDE, further combination of which may lead to other meaningful SSMs, again with potentially

very hard to crack problems regarding likelihood derivations. This notwithstanding, theoretical motivation for such models may ask for their testing against experimental data.

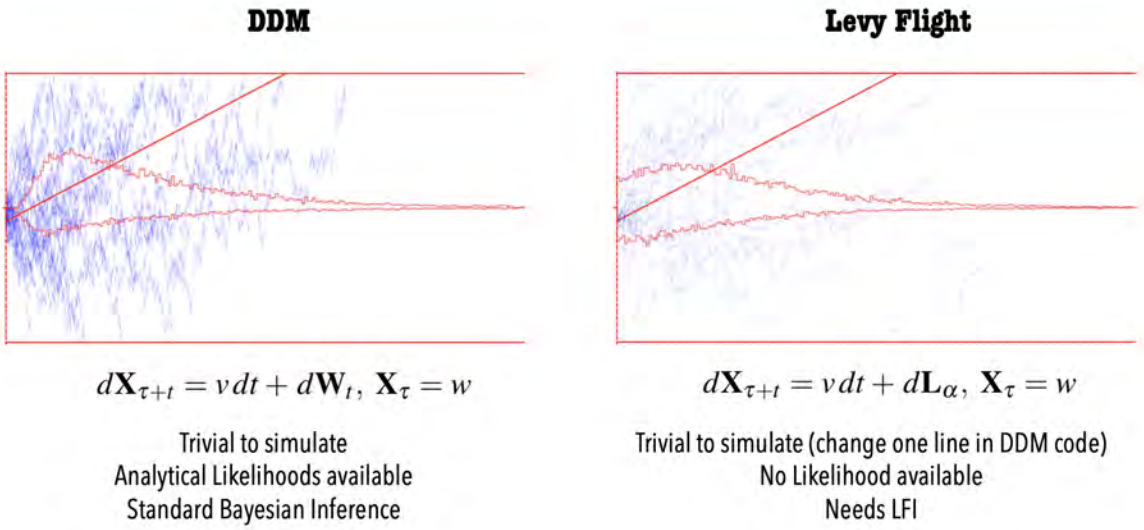

Figure 2.6. Graphical displays illustrating the differences in trajectories between the DDM and Levy Flighy models. The only difference between the illustrated models is the noise process. On the *left*, the DDM model is driven by Gaussian noise, whereas on the *right* the Levy Flight model is driven by an *alpha-stable* distribution (in this case with parameter $\alpha = 1$, which implies a Cauchy distributed noise process). The difference in the observed trajectories shown in blue is stark. The Cauchy noise process leads to much larger instantaneous shifts in the particle location. Simulating either process is equally trivial using the *Euler-Maruyama* method, however Bayesian inference is rendered much harder for the Levy Flight model, because of the missing likelihood function.

The rest of this chapter deals with methods designed to help us with posterior inference of models such as the Levy Flight Model. Using SSMs as an example model class, this section hopefully motivated why such methods generally form an important contribution by enabling experimenters to test their data against a larger variety of theoretically important generative models. The implications of the following however are in no way strictly bound to SSMs.

2.3 Approximate Inference at large

2.3.1 Broad Classes of Likelihood Free Inference Algorithms

In an attempt to clearly delineate the scope of this review, I draw a distinction between,

1. Traditional optimization based approximate inference with access to analytic likelihood functions (or analytic approximations to likelihood functions).
2. Traditional approaches to Likelihood Free Inference (LFI-ABC)
3. Intermediate approaches with larger emphasis on machine learning tools (LFI-ML)

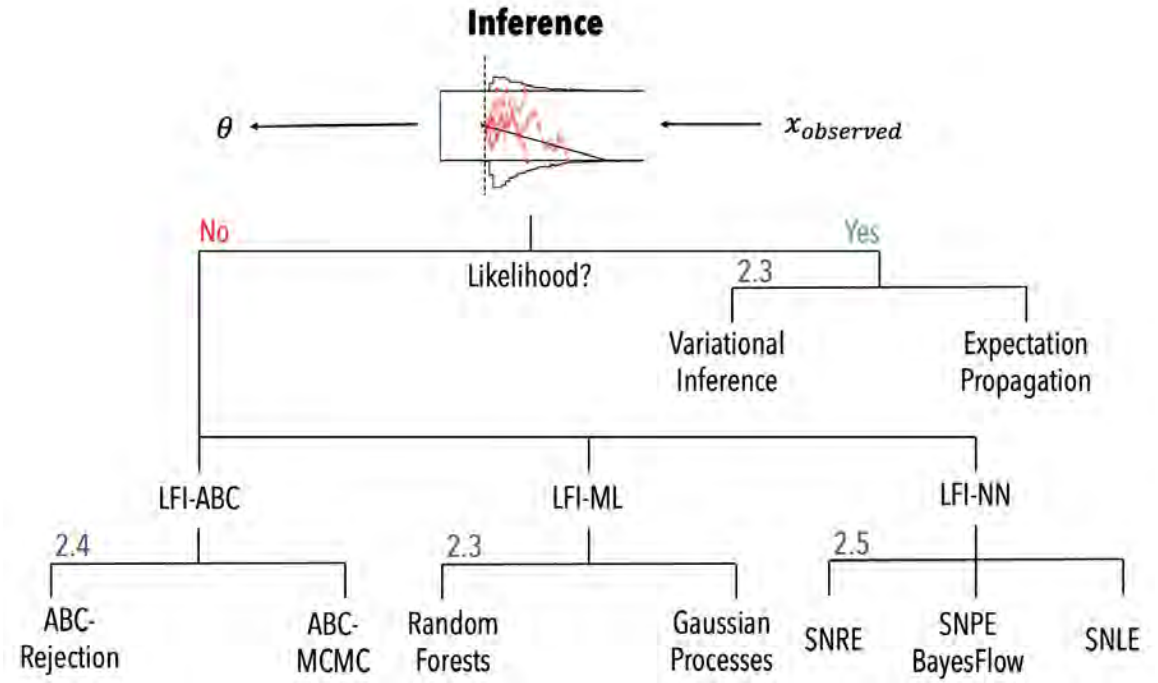

Figure 2.7. Overview of methods under discussion in this review. The discussion will focus mostly on likelihood free methods (left side of the tree), however approximate inference in general includes powerful optimization based methods to perform posterior inference in models where posteriors are complex but access to likelihood functions is granted (right side of the tree).

4. Modern approaches to Likelihood Free Inference, which apply deep learning methods to base LFI algorithms on flexible classifiers and density estimators (LFI-NN)

The main focus of this chapter will be on methods falling under point 4, I however thoroughly motivate these via a proper introduction of methods falling under point 2. While not a main focus, it needs to be acknowledged that there exist a multitude of algorithms which fall somewhere in between LFI-ABC and LFI-NN, and discuss these under as LFI-ML methods below as well. These algorithms may sometimes be preferred over the LFI-NN methods I discuss more deeply, however tend not to offer additional capabilities. These moreover generally tend not to be suitable for *global amortization strategies*, which I detail and emphasize as important for applications in the following. I quickly discuss the most widespread methods under point 1 and point 3 here, so that discernment of the differences and similarities of these broad classes of methods can be aided.

The legacy of traditional optimization based approximate inference is largely represented by two classes of commonly applied algorithms today. Variational Inference (VI) algorithms (Jordan et al., 1999; D. M. Blei, Kucukelbir, and McAuliffe, 2017) , and Expectation Propagation (EP) algorithms (T. P. Minka, 2013). Strictly speaking both methods are two different kinds of *variational* algorithms, as in they both attempt to solve the problem of distribution inference via optimization, however naming convention has allocated the labeling as presented. These methods are distinct from the ones I stress in the following sections in that they do generally make use of analytic likelihoods in

their computations. Their origin does not stem from the problem of Likelihood Free Inference, but rather from a motivation to solve complex posterior estimation problems by approximation to reduce otherwise prohibitive computational demands.

Both methods are designed to approximate a target distribution, commonly denoted as $\pi(\theta)$, by finding the closest matching distribution $q^*(.) \in \mathcal{Q}$, where $\mathcal{Q}$ is a tractable class of distributions. VI defines,

$$q^*_{VI} = \underset{q \in \mathcal{Q}}{\operatorname{argmax}} \; KL(q||\pi) \tag{2.6}$$

whereas EP defines,

$$q^*_{EP} = \underset{q \in \mathcal{Q}}{\operatorname{argmax}} \; KL(\pi||q) \tag{2.7}$$

This subtle seeming difference induces different algorithms for finding q^* and differences in the respective properties of q^*_{EP}, q^*_{VI} relative to the true target distribution $\pi(\theta)$ (B. Wang and Titterington, 2005; T. Minka et al., 2005; Barthelmé and Chopin, 2011).

For purposes of the this review, I focus on the following conceptual principle implicit in both traditional VI and EP algorithms. Inference is turned tractable by a priori simplifying the posterior to belong to a class of distributions that can be handled. This imposes concerns with respect to the calibration of such methods for scenarios where rigorous calibration of statistical methods is crucial (Talts et al., 2018). Related to the discussion of LFI-ABC more subtleties will emerge, however, put simply, this as an additional mode of failure for inference calibration beyond those discussed in the following sections. Consequently proposed applications have to assure that posterior quality is not unduly compromised because of the simplifying assumptions embedded in VI and EP.

Extensions to VI algorithms have been developed (Ranganath, Tran, and D. Blei, 2016; Tran, Ranganath, and D. M. Blei, 2017; Sobolev and Vetrov, 2019), to incorporate increasingly complex model structures, to make scalable the optimization alrgorithms to large datasets, and notably to extend into the realm of Likelihood Free Inference. Extensions of both approaches can be considered as active areas of research. The connection between Variational Inference and Deep Neural Networks (DNNs), made operational through the popularization of what is now commonly known as the reparameterization trick (Diederik P Kingma, Salimans, and Welling, 2015; Fu and Hu, 1995; Fu, 2008) contributes to the continued development of approaches that advance the capacities of VI.

As we will see, modern approaches to ABC implicitly rely on optimization of KL-divergences, which we make precise in section 2.5. At this point I emphasize that the lack of further discussion of VI and EP methods in the following is not motivated by an adversarial attitude towards the future of these approaches, but instead by a desire to keep the scope of this review focused towards currently dominant, and at least equally promising alternatives. I stress that implied synergies abound and in chapter 5 I will in fact investigate these synergies for the method proposed in chapter 3.

For completeness I shortly discuss LFI-ML approaches. Broadly these apply various machine learning techniques to attack both likelihoood estimation and posterior inference. A prominent

method bases posterior inference on random forests (Raynal et al., 2019), however focuses on single parameter estimation. Another widely used machine learning technique is the Gaussian Process (GP) (Rasmussen, 2003), which can be powerful in the LFI setting serving as an efficient guide for the sequential choice of simulation parameters (Wilkinson, 2014; Meeds and Welling, 2014; Acerbi, 2020; Järvenpää, Michael U Gutmann, Vehtari, et al., 2021). This can be made dependent on uncertainty estimates available directly from the GP via active learning (Acerbi, 2019; Järvenpää, Michael U Gutmann, Pleska, et al., 2019). The repeated estimation of Gaussian Process regressions to guide simulations however, can become costly when the number of simulated parameters reaches $N_{sim} > 2000$. This makes such methods hard to apply when global amortization is the goal (to be explained later). Nevertheless, the guiding principles underlying LFI-ML methods are mirrored in LFI-NN approaches, including the desire for sequentially efficient choices of simulation parameters. To the best of my knowledge, hardly any studies that directly compare the performance of LFI-ML methods with LFI-NN methods exist, which generally makes it hard to strictly advocate for one or the other class of approaches in applications spheres were both are suitable. A detailed study of the computational costs incurred in such situations may be desirable, however I am not aware of such potentially conclusive work having been undertaken yet.

However, as mentioned at the start of this section, for the purposes of this review I consider these techniques predecessors to the generally more powerful Neural Network based approaches which will be discussed in depth. The characterization as predecessors is inaccurate from the perspective of strict chronology since research on these approaches is still underway, but I hope to make clear what motivates my classification. It is moreover strictly speaking inaccurate to characterize LFI-ABC categorically as less powerful than LFI-NN approaches since I can not completely rule out application scenarios which may make LFI-ABC preferable.

It is worth mentioning one additional emerging branch of LFI methods, which rests on a slight difference in viewpoint. The starting point here is to realize that many simulation models can be characterized as transformations of random variables to begin with, specifically for model $\mathcal{M}$ we may have $f_{\mathcal{M}}(\theta, \epsilon),\ \epsilon \sim \mathcal{D}$, where $\mathcal{D}$ represents some noise distribution. This viewpoint can then be exploited for LFI. It may lead to a representation of the simulator which is *differentiable* (Tran, Ranganath, and D. M. Blei, 2017), or it may help with making certain a prior intractable likelihood computations tractable by accessing simulator internals (Brehmer et al., 2020). These approaches are potentially very powerful, just as LFI-ML methods, if applied suitably. I will forgo further discussion for the sake of keeping this review focused on methods that are generally applicable, especially for amortization purposes.

2.3.2 Computational Strategies

An additional distinction in kind between LFI approaches is worth mentioning at this point. Apart from specific sampling algorithms and modeling tools, one may define three separate *computational strategies* (see 2.8) for LFI methods, of which most algorithms I discuss admit more than one.

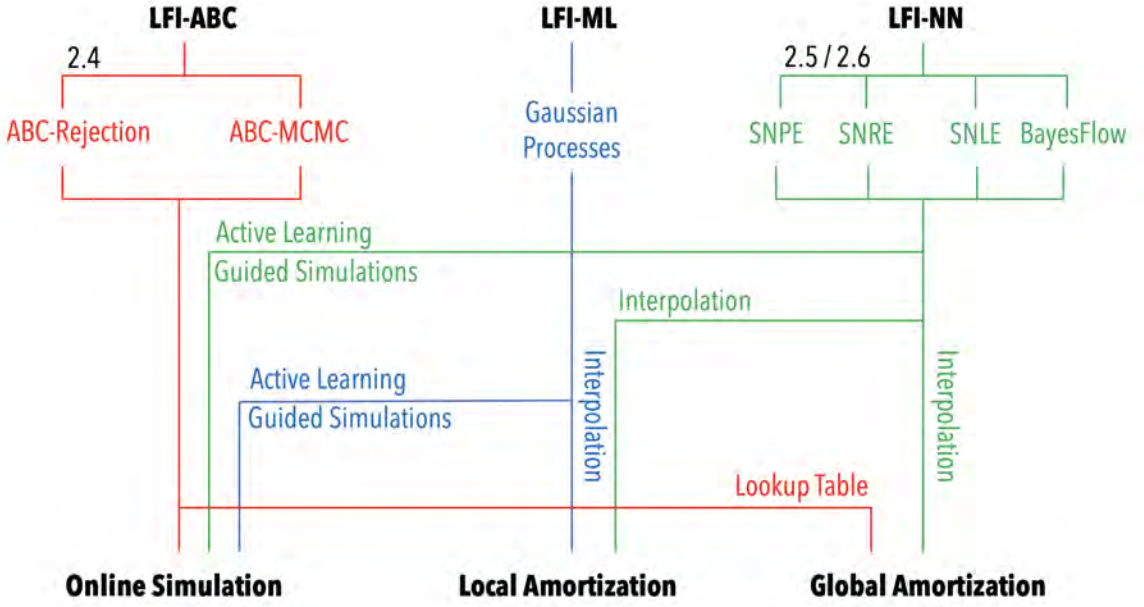

Figure 2.8. Overview of computation strategies and their connection to the three broad categories of approximate LFI methods under discussion in this chapter. The main focus in this chapter are LFI-ABC methods to contextualize the history of ideas, and LFI-NN methods representing the modern approaches to LFI.

1. Online Simulation
2. Local Amortization
3. Global Amortization

I refer to a strategy that relies on model simulations at the point of inference as a *online simulation* strategy. *Local amortization* I consider a strategy that incurs some upfront simulation cost to make available cheap inference (simulation free) for a *particular observed dataset*. Lastly a strategy is referred to as *global amortization* if it incurs such upfront simulation cost with the ambition to make cheap inference available across a large variety of datasets.

The discussion will emphasize the importance of global amortization as a valuable strategy for enabling cheap inference on complex likelihood free models by the broader research community.

2.4 LFI-ABC

As discussed, traditional ABC methods (Sisson, Fan, and M. Beaumont, 2018) assume access to a parametrized simulator model $\mathcal{M}(\theta)$ where $\theta \in \Theta$ is a vector of parameters, without explicit access to a likelihood function. For any given dataset $\mathbf{x}_{obs}$ we are generally interested in approximating $p(\theta|\mathbf{x}_{obs})$. In this section I attempt to trace the genealogy of ABC approaches which we, in contrast to the ideas introduced in 2.5, call *traditional*.

2.4.1 The ABC rejection sampler

The most basic ABC algorithm is known as the *ABC rejection sampler* (Tavaré et al., 1997; Pritchard et al., 1999). It forms the nucleus of what has since grown into a rather large body of literature and still a good starting point from which to bootstrap this small conceptual history of ABC methods.

One of the very standard computational probability algorithms to sample from a arbitrary probability distribution $\pi(\theta)$, is the *rejection sampling algorithm*.

Algorithm 1: Standard Rejection Sampler

Input: Target distribution: $\pi(\theta)$;
Proposal Distribution: $g(\theta)$;
Desired Number of samples: N;

initialization: $n = 1$, $C \geq \max_\theta \frac{\pi(\theta)}{g(\theta)}$;

while $n \leq N$ **do**
1. Propose $\theta_i \sim g(\theta)$;
2. Set $\theta_n = \theta_i$ with probability $\frac{\pi(\theta_i)}{Cg(\theta_i)}$, and increment n;
Else go to 1.

end

Result: Samples: $\theta_1, ..., \theta_N \sim \pi(\theta)$

For purposes of our discussion we specialize the rejection sampler a little further to capture directly the spirit of posterior inference. Assume that $\pi(\theta)$ is our posterior $p(\theta|\mathbf{x}_{obs}) \propto p(\mathbf{x}_{obs}|\theta)\pi_{prior}(\theta)$ and $g(\theta) = \pi_{prior}$. This leaves us with,

Algorithm 2: Prior / Posterior Rejection Sampler

Input: Target distribution: $p(\mathbf{x}_{obs}|\theta)\pi_{prior}(\theta)$;
Proposal Distribution: $\pi_{prior}(\theta)$;
Desired Number of samples: N;

initialization: $n = 1$;

while $n \leq N$ **do**
1. Propose $\theta_i \sim \pi(\theta)$;
2. Set $\theta_n = \theta_i$ with probability $p(\mathbf{x}_{obs}|\theta)$, and increment n;
Else go to 1.

end

Result: Samples: $\theta_1, ..., \theta_N \sim \pi(\theta)$

The major realization is that, while we cannot compute $p(\mathbf{x}_{obs}|\theta)$ directly, we can in fact treat it as a *bernoulli event* and therefore implicitly access this probability. Since we have access to the simulator $\mathcal{M}(\theta)$, we can simulate $\mathbf{x} \sim \mathcal{M}(\theta)$ and accept samples only when for a given sample $\mathbf{x} = \mathbf{x}_{obs}$. This happens exactly with probability $p(\mathbf{x}_{obs}|\theta)$. This insight leads to the *exact rejection ABC sampler* algorithm, our first mechanism to sample from posterior distributions over parameters of a simulator model $\mathcal{M}(\theta)$ in the category LFI-ABC, and the origin of an immense surge of research towards more general and efficient such samplers.

Algorithm 3: Exact ABC rejection sampler

Input: Dataset: $\mathbf{x}_{obs}$;
Simulator: $\mathcal{M}(\theta)$;
Proposal Distribution: $\pi_{prior}(\theta)$;
Desired Number of samples: N;

initialization: $n = 1$;

while $n \leq N$ **do**
1. Propose $\theta_i \sim \pi(\theta)$;
2. Sample $\mathbf{x}_i \sim \mathcal{M}(\theta_i)$, and increment n;
if $\mathbf{x}_i = \mathbf{x}_{obs}$ **then**
$\theta_n = \theta_i$;
end

end

Result: Samples: $\theta_1, ..., \theta_N \sim \pi(\theta)$

While this sampler is exact and works in principle, constraints prevent it from being broadly useful for practical applications. In fact the probability of generating $\mathbf{x}_{obs}$ *exactly* may be arbitrarily low even under the *true* model with the *correct* parameter vector.

Two elementary examples illustrate this point and suggest the two main levels of approximation usually involved in ABC computation. *First*, if the output variables $\mathbf{x} \sim \mathcal{M}(\theta)$ are continuous the probability of generating any particular observation $\mathbf{x}_{obs}$ is by definition 0, even under the true model. To account for this, instead of asking for $\mathbf{x} = \mathbf{x}_{obs}$ we can instead ask for $||\mathbf{x} - \mathbf{x}_{obs}|| \leq \epsilon$, where $||.||$ defines some *norm* with which we can measure the distance between vectors $\mathbf{x}$ and $\mathbf{x}_{obs}$ and ϵ serves as a *tolerance hyperparameter*.

Naively applying this approximation by exchanging the $\mathbf{x} = \mathbf{x}_{obs}$ condition in Algorithm 3 for $||\mathbf{x} - \mathbf{x}_{obs}|| \leq \epsilon$, let's us sample not from the *exact*, but from an approximate posterior,

$$p^{\epsilon}_{ABC}(\theta|\mathbf{x}_{obs}) \propto \int \frac{1}{Vol(\mathcal{B}_{\epsilon}(\mathbf{x}))} \mathbb{I}(||\mathbf{x} - \mathbf{x}_{obs}|| \leq \epsilon) p(\mathbf{x}|\theta)\pi(\theta)\, d\mathbf{x} \tag{2.8}$$

where ϵ serves to adjust the degree of approximation. Now further refining the approximation by using a smoothing kernel $K_h(||.||)$ instead of the hard binary cut-off imposed by $\mathbb{I}(||\mathbf{x} - \mathbf{x}_{obs}||$, leaves us with the the ABC approximation to the posterior $p(\theta|\mathbf{x}_{obs})$,

$$p^{h}_{ABC}(\theta|\mathbf{x}_{obs}) \propto \int K_h(||\mathbf{x} - \mathbf{x}_{obs}||) p(\mathbf{x}|\theta)\pi(\theta)\, d\mathbf{x} \tag{2.9}$$

where now the so-called *bandwidth parameter* h of the smoothing Kernel adjusts the degree of approximation.

By the basic *law of large numbers*, we can rewrite,

$$p^h_{ABC}(\theta|\mathbf{x}_{obs}) \propto \int K_h(||\mathbf{x}-\mathbf{x}_{obs}||)p(\mathbf{x}|\theta)\pi(\theta)\,d\mathbf{x} \tag{2.10}$$

$$\propto \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} K_h(||\mathbf{x}_n-\mathbf{x}_{obs}||)\pi(\theta), \quad \mathbf{x}_i \sim \mathcal{M}(\theta) \tag{2.11}$$

We can now translate this discussion into the the pivotal *ABC rejection sampler*.

Algorithm 4: ABC Rejection Sampler

Input: Dataset: $\mathbf{x}_{obs}$;
Simulator to generate: $\mathbf{x} \sim p(\mathbf{x}|\theta)$;
Proposal Distribution: $\pi_{prior}(\theta)$;
Smoothing Kernel: $K_h(||.||)$;
Desired Number of samples: N;

initialization; $n = 1$, $K = K_h(0)$;

while $n \leq N$ **do**

1. Propose $\theta_i \sim g(\theta)$;
2. Simulate $\mathbf{x}_i \sim \mathcal{M}(\theta_i)$;
3. Set $\theta_n = \theta_i$ with probability $\frac{K_h(||\mathbf{x}_i-\mathbf{x}_{obs}||)}{K}$, and increment n;

end

Result: Samples: $\theta_1, ..., \theta_N \sim p^h(\theta|\mathbf{x}_{obs})$

This sampler is pivotal from a conceptual viewpoint. It provides an avenue towards Bayesian inference over parameters for any arbitrary simulation model $\mathcal{M}$, a conceptual breakthrough that spurred enormous amounts of methodological research. The rest of the discussion now focuses on how to use this conceptual breakthrough and make it in fact operational for large array of real world applications.

The ABC Rejection Sampler as proposed has one severe shortcoming. In realistic scenarios, the sample acceptance probabilities tend to be so low that computational cost is unreasonably burdensome for any but trivial applications. For a simple illustration of how acceptance probabilities might drop to minuscule levels consider a sequence of bernoulli trials (a sequence of heads or tails from repeated fair coin flips) of length 100, $\mathbf{x}_{obs} = \{1, 0, ..., 0\}$. We now use a bernoulli simulator and apply the ABC rejection sampler. To accept a sample from the simulator at all, we need to match our sequence $\mathbf{x}_{obs}$ exactly. Hence even if we simulate with $p = 0.5$ (the ground truth since our coin was fair to begin with), we have an approximate chance on the order of 2^{-100} to match our original sequence exactly. This simple coin tossing example suffices to render application of ABC rejection sampling infeasible.

The traditional developments to remedy this fact can be roughly split into two categories, both of which work in tandem to improve on the trade-off between computational burden and degree of approximation,

1. Choice of sampling methodology
2. Dimensionality reduction of $\mathbf{x}$ to stabilize distance computations $||\mathbf{x}-\mathbf{x}_{obs}||$

2.4.2 Choice of sampler

We discuss first the choice of sampler, bypassing for now the acknowledged instability of $K_h(||\mathbf{x}-\mathbf{x}_{obs}||)$. Two basic streams of advancement can be identified. First, the application of Markov Chain Monte Carlo samplers to LFI (Marjoram et al., 2003), including advanced, MCMC based, samplers such as Hamiltonian Monte Carlo (Meeds, Leenders, and Welling, 2015).

The basic transition step for such an ABC-MCMC sampler can be described as,

Algorithm 5: ABC-MCMC Algorithm

Input: Initial Parameter: θ_0;
Simulator: $\mathcal{M}$;
Prior distribution: $\pi_{prior}(\theta)$;
Proposal Distribution: $q(\theta^{'}|\theta^{*})$;
Smoothing Kernel: $K_h(||.||)$;
Desired Number of samples: N;

initialization; $n = 1$;

while $n \leq N$ **do**

1. Propose $\theta_n \sim q(\theta|\theta_{n-1})$;
2. Generate $\mathbf{x}_n \sim p(\mathbf{x}|\theta_n)$;
3. Calculate acceptance probability: $\alpha = min\left(1, \frac{K_h(||\mathbf{x}_n-\mathbf{x}_{obs}||)\pi_{prior}(\theta_n)q(\theta_n|\theta_{n-1})}{K_h(||\mathbf{x}_{n-1}-\mathbf{x}_{obs}||)\pi_{prior}(\theta_{n-1})q(\theta_{n-1}|\theta_n)}\right)$;
4. Accept θ_n with probability α. Else set $\theta_n = \theta_{n-1}$. Increment n;

end

Result: Samples: $\theta_1, ..., \theta_N \sim p^h_{ABC}(\theta|\mathbf{x}_{obs})$

where the acceptance probability α so defined ensures the *detailed balance property* Metropolis et al., 1953; Hastings, 1970. For large enough N, this sampler will therefore successfully sample from the target distribution $p^h_{ABC}(\theta|\mathbf{x}_{obs}$, however general shortcomings endemic to Markov Chain Monte Carlo methods, such as widely acknowledged difficulties to sample from multi-modal target distributions remain.

The second stream of sampling techniques employed for ABC methods grows out of basic *importance sampling*. Importance sampling is based on the following basic identity,

$$\int f(\theta)\, d\theta = \int \frac{f(\theta)}{g(\theta)} g(\theta)\, d\theta = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \frac{f(\theta_i)}{g(\theta_i)}, \;\; \theta \sim g \tag{2.12}$$

where $f(\theta)$ is our intractable *target* distribution from which we would like to sample and $g(\theta)$ is a *proposal* distribution from which we can easily draw samples. The re-normalized *importance weights*, $w_i = \frac{\frac{f(\theta_i)}{g(\theta_i)}}{\sum_{i=1}^{N} \frac{f(\theta_i)}{g(\theta_i)}}$ are used as the basis for sampling from $f(\theta)$. In our problems, $f(\theta) = p^h_{ABC}(\theta|\mathbf{x}_{obs})$, and the importance weights take the form $w^i_{ABC} \propto \frac{K_h(||\mathbf{x_i}-\mathbf{x}_{obs}||)\pi_{prior}(\theta)}{g(\theta)}$.

Without delving into further details of importance samplers and related techniques, I instead note the guiding principles that lead the improvements. In common with rejection samplers, the efficiency of importance sampling techniques is strongly dependent on a good choice of the proposal distribution $g(\theta)$. The closer $g(\theta)$ is to the target, the higher the efficiency of the sampler. For rejection samplers

this efficiency is measured using the *rejection rate* of the sampler, while for importance sampling the quality of $g(\theta)$ is usually tracked using the *effective sample size* $ESS = \left(\frac{1}{\sum_{n=1} w_i^2}\right)$, $1 \leq ESS \leq N$. (J. S. Liu, Chen, and W. H. Wong, 1998; J. S. Liu, Chen, and Logvinenko, 2001). Sophisticated importance sampling techniques, such as population monte carlo (PMC) (Cappé et al., 2004), sequential monte carlo (SMC) (Sisson, Fan, and Tanaka, 2007), sequential importance sampling (SIS) (Neal, 2001) tend to find smart ways to gradually adjust $g(\theta)$ towards $f(\theta)$. Note that while importance sampling can be an extremely successfull (efficient) method to sampling from distributions of low to moderate dimensionality, endemic to the method is a limitation in overcoming the curse of dimensionality. Importance sampling is therefore not advisable in high dimensions, which damages applicability for higher dimensional models as well as parameter inflated inference problems such as in hierarchical models.

Both likelihood free MCMC and Importance sampling offer great improvements over the basic ABC rejection sampling.

2.4.3 Dimensionality Reduction: Summary Statistics

While the the choice of sampling technique does matter, a basic shortcoming embedded in all approaches discussed thus far is the instability of distance computations in high dimensions. This is a common problem for approaches in machine learning and directly related to the curse of dimensionality (Friedman, Hastie, and Tibshirani, 2001). The problem faced by ABC methods lies in the inheritance of this phenomenon concerning all computations involving the smoothing kernel $K_h(||\mathbf{x} - \mathbf{x}_{obs}||)$.

To overcome this, in general, traditional ABC methods rely on lower dimensional summary statistics which substitute for the potentially high-dimensional dataset (M. A. Beaumont, Zhang, and Balding, 2002). Instead of the data $\mathbf{x}$, we can base our computations on the summary statistics $\mathbf{s}$, with $dim(\mathbf{x}) >> dim(\mathbf{s})$, to stabilize the distance metrics.

While the curse of dimensionality is addressed, the choice of summary statistics now needs to be dealt with. The guiding principle in the search for summary statistics is the following: We wish to minimize the $dim(\mathbf{s})$, while maximizing the information $\mathbf{s}$ carries about it's underlying parameters.

We will discuss some theoretical background on the notion of 'good' summary statistics, and broadly circumscribe the landscape of traditional ABC approaches to finding summary statistics. In the statistical literature we can identify what one may call a gold standard for what constitutes a good summary statistic. This gold standard is represented by the notion of a *minimal sufficient statistics*.

Definition 2.4.1 (Sufficient Statistic)**.** Given a parametric model $\mathcal{M}(\theta)$, $\theta \in \Theta$, a *statistic* $\mathbf{s}(\mathbf{x})$ is *sufficient* for the parameter θ, given some dataset $\mathbf{x} \sim \mathcal{M}(\theta)$ if the conditional probability of $p(\mathbf{x}|\mathbf{s}(\mathbf{x}))$ is independent of θ.

Colloquially, a sufficient statistic is a function of the data that captures all information about the data generating parameters of a given model $\mathcal{M}$. Even more straightforwardly, we lose no information about the parameters θ, if instead of $\mathbf{x}$ we perform our computations with $\mathbf{s}(\mathbf{x})$.

Note that for a given model there can be a (possibly infinitely large) set of possible sufficient statistics $\mathbf{S}$ (a trivial sufficient statistic being the data $\mathbf{x}$ itself for example). A *minimal sufficient statistic* further requests that a given sufficient statistic has the lowest dimensionality amongst all possible sufficient statistics that can be defined for a given model $\mathcal{M}$.

Somewhat dampening our hopes of achieving this gold standard easily however, the *Pitman-Koopman-Darmois* theorem (Pitman, 1936; Koopman, 1936; Darmois, 1935), central to the theory of statistics, asserts that only if we restrict ourselves to the *exponential family of probability distributions* can we find sufficient statistics of dimensionality independent of the number of datapoints in our dataset $\mathbf{x}$. For example in any of the models in Figure 2.5, we would want to naturally request that a set of summary statistics needs to be applicable across a range of numbers of trials for a given experiment. The *Pitman-Koopman-Darmois* theorem tells us that this hope could be fulfilled only if the first-passage distributions of those models belong to the exponential family, which is most often not the case.

ABC methods however are essentially a priori designed to be applicable exactly when we are not dealing with tractable likelihoods such as may be found in the exponential family. The best we can hope for, is thus to find what we may call *highly informative approximately minimal* summary statistics. I discuss a number of methods designed for this purpose, with focus on the ideas that are relevant for the deep learning methods developed in section 2.5.

Consider $\theta_1^{ABC}, ..., \theta_n^{ABC} \sim \pi_{ABC}^h(\theta|\mathbf{s}_{obs}^*)$, as samples resulting from a given ABC posterior, where computations are based on the summary statistic vector s_{obs}^*. The goal is to find s^*, so that inference performance is maximized. Naturally this entails keeping $dim(s^*)$ as small as possible. Traditional ABC methods generally start with the assumption that we have access to a number of *candidate summary statistics* $\{s_1, ..., s_m\}$ on which to base our analysis (a shortcoming which the methods discussed in section 2.5 lift). Blum et al., 2013, classify and discuss various approaches proposed in the traditional LFI-ABC literature to carry out this process. For purposes of this discussion, the crucial aspect of these methods is the lack of mechanism to learn a data to summary mapping. Section 2.5, highlights how LFI-NN algorithms can incorporate such mappings quite naturally.

2.5 LFI-NN

In recent years the basic principles of LFI have been increasingly cross-pollinated with ideas growing out of the machine learning literature. Specifically the application of Neural Networks approaches to all parts of the LFI pipeline have slowly become the dominant paradigm, a field transcending trend, which can be explained by the flexible nature of Neural Networks as universal function approximators (Cybenko, 1989; Hornik, Stinchcombe, White, et al., 1989). In this section I will discuss how Neural Network based approaches have reshaped the LFI literature. I will first point out the basic building blocks as far as Neural Network architectures are concerned, and relate these to the relevant parts of LFI. Then I discuss different classes of algorithms which, utilizing these building blocks, attack the LFI problem in varying ways. I will discuss two fundamental kinds of algorithms: First, algorithms

that target the posterior distribution and second, algorithms that target the likelihood function.

2.5.1 Basic Building Block: Density Estimators

Many Neural Network based algorithms for LFI rely on density estimation as the main workhorse. In this section I discuss some of the fundamental ideas behind Neural Network based density estimation, with focus on two broad classes: Mixture Density Networks (MDN) and Flows.

Mixture density networks

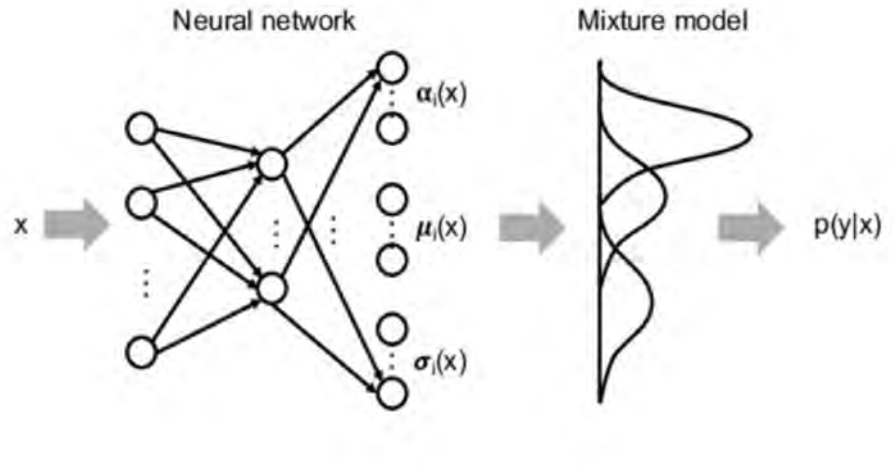

Figure 2.9. Graphical illustration of a Mixture Density Network (MDN). Taken from Vossen, Feron, and Monti, 2018

MDNs (Bishop, 1994) rest on the following simple idea. Given any Neural Network architecture, we can turn its output nodes to represent the parameterization of a *probabilistic mixture model*. Conceptually the output nodes are split into two pools. One pool of C nodes represents the mixture probabilities α_i, $i \in \{1, .., C\}$, while the other pool of $C * P$ nodes represent the $P - dimensional$ parameters θ_i for each of the component distributions, which we can abstractly denote as $\mathcal{D}(\theta_i)$, with associated densities $p_{\mathcal{D}}(y|\theta_i)$.

For features $X \in \mathcal{X}$ and labels $Y \in \mathcal{Y}$, we therefore model the distribution of y given x as,

$$p(y|x) = \sum_{i=1}^{C} \alpha_i(x) p_{\mathcal{D}}(y|\theta_i(x)) \tag{2.13}$$

Figure 2.9 illustrates the setup graphically. Naturally the suitable training loss is simply the (log) likelihood under the probability model.

A common choice for the mixture components, and one which we will encounter in our discussion of algorithms, are (multivariate) Gaussian distributions, specializing the formulation so that, $\theta_i =$

$\{\mu_i, \Sigma_i\}$ and

$$p_{\mathcal{D}_i}(\mathbf{y}|\theta_i(x)) = (2\pi)^{\frac{d}{2}}|\Sigma_i^{-1}|^{\frac{1}{2}} \exp(-\frac{1}{2}(\mathbf{y}-\mu_i)^T\Sigma_i^{-1}(\mathbf{y}-\mu_i)) \tag{2.14}$$

where d refers to the dimensionality of $\mathbf{y}$.

The MDN framework is quite general, in principle allowing for mixture distributions with components from heterogeneous base distributions. Similar to the universal approximation theorem for Neural Networks in general (Hornik, Stinchcombe, White, et al., 1989; Cybenko, 1989), it holds true that a Mixture of Gaussians (MoG), given enough components, is a universal approximator in the space of probability distributions (Goodfellow, Y. Bengio, and Courville, 2016), serving as a motivation for the application of such MoG formulations. Despite their conceptual appeal, training MDNs is notoriously difficult due to instability with tendency to mode collapse in high dimensions (Makansi et al., 2019) and necessitates numerical as well as pre-training related (Hjorth and Nabney, 1999) tricks since the likelihood functions does not easily decompose under the log() transformation (we are taking a log over a sum) (Kruse, 2020).

As it forms a conceptual part of algorithms discussed in later sections, I describe here a particular approach to make the training of MDNs (Papamakarios and I. Murray, 2016) more robust. At the core of this method lies the idea that adding uncertainty over weights, in other words formulating the MDN as a Bayesian Neural Network (Neal, 2012), will prevent getting stuck in early local modes.

For a given architecture with MDN with network parameters (weights) $\mathbf{w}$, we represent the parameters as deriving from independent Gaussian distributions. We define the (stochastic) function that is executed by the MDN as $q_{\mathbf{w}}(\mathbf{y}|\mathbf{x})$, which defines a *likelihood* of output vector $\mathbf{y}$, given input vector $\mathbf{x}$. In other words, we augmented $\mathbf{w}$ to $\{\mathbf{w}_\mu, \mathbf{w}_\sigma\}$, the mean and standard deviation respectively. We can then write,

$$\mathbf{w} = \mathbf{w}_\mu + \mathbf{w}_\sigma * \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \tag{2.15}$$

This defines a distribution over weights $q(\mathbf{w}) \sim \mathcal{N}(\mathbf{w}_\mu, diag(\mathbf{w}_\sigma))$. Defining a prior over weights as $\pi(\mathbf{w}) \sim \mathcal{N}(0, \tau^{-1}\mathbf{I})$, we can then train the network using Stochastic Variational Inference (SVI) (Hoffman, D. M. Blei, et al., 2013) minimizing the following loss with respect to its arguments,

$$\mathcal{L}_{SVI}^{MDN}(\mathbf{w}_\mu, \mathbf{w}_\sigma) = -\sum_{n=1}^{N} \mathbb{E}_{q(\mathbf{w})}^{MDN}\left[q_{\mathbf{w}}(\mathbf{y}^n|\mathbf{x}^n)\right] + D_{KL}\left(q(\mathbf{w})||\pi(\mathbf{w})\right) \tag{2.16}$$

While the standard loss for MDNs,

$$\mathcal{L}_{ML}^{MDN}(\mathbf{w}) = -\sum_{n=1}^{N} q_{\mathbf{w}}(y^n|x^n) \tag{2.17}$$

learns the density estimator $q_w(y^n|x^n)$, via a *maximum likelihood criterion*, using $\mathcal{L}_{SVI}$ optimizes a variational lower bound (ELBO) (Jordan et al., 1999). One can make the connection between $\mathcal{L}_{SVI}(\mathbf{w}_\mu, \mathbf{w}_\sigma)$ and the ELBO precise as,

$$q^* = \underset{\mathbf{q}\in\mathcal{Q}}{argmin}\ D_{KL}(q(\mathbf{w})||p(\mathbf{w}|\mathbf{x}^n,\theta^n)) \tag{2.18}$$

$$= -\mathbb{E}_{q(\mathbf{w})}\left[\log(p(\mathbf{x}^n,\theta^n|\mathbf{w}))\right] + D_{KL}(q(\mathbf{w})||p(\mathbf{w})) \tag{2.19}$$

$$= -\mathbb{E}_{q(\mathbf{w})}\left[\sum_{i=1}^{N}\log(p(\mathbf{x}_i,\theta_i|\mathbf{w}))\right] + D_{KL}(q(\mathbf{w})||p(\mathbf{w})) \tag{2.20}$$

$$= -\mathbb{E}_{q(\mathbf{w})}\left[\sum_{i=1}^{N}\log(q_{\mathbf{w}}(\theta_i|\mathbf{x}_i))\right] + D_{KL}(q(\mathbf{w})||p(\mathbf{w})) \tag{2.21}$$

$$= -\sum_{i=1}^{N}\mathbb{E}_{q(\mathbf{w})}\left[\log(q_{\mathbf{w}}(\theta_i|\mathbf{x}_i))\right] + D_{KL}(q(\mathbf{w})||p(\mathbf{w})) \tag{2.22}$$

Flows

The second large class of useful density estimators employed in modern LFI algorithms is the class of flows (Rezende and Mohamed, 2015).

I will describe the main ideas behind these flow methods, with focus on a few examples and an eye for distinguishing two main classes of approaches which I will call *autoregressive* and *joint* estimators.

In general flow based methods approach density estimation from the point of view of a transformation of random variables. Starting from an underlying *base (or latent) distribution* (for our purposes we stick with the common isotropic unit multivariate Gaussian, $\pi(u) = \mathcal{N}(0, \mathbf{I})$), a flow is an invertible, differentiable, transformation (IDT) $f_{\mathbf{w}} : \mathcal{U} \to \mathcal{X}$, where $\mathbf{w}$ refers to the transformations parameters. Commonly $f_{\mathbf{w}}$ is composed of a succession of such IDTs so that,

$$f_{\mathbf{w}}(u) = f_{n,\mathbf{w}_n} \circ f_{n-1,\mathbf{w}_{n-1}} \circ ... \circ f_{1,\mathbf{w}_1}(u) \tag{2.23}$$

The idea here is that for an IDT, we can apply the *transformations of variables formula* from basic probability, to access the *pdf* of $f_{\mathbf{w}}(u)$. Taking $x = f_{\mathbf{w}}(u)$ we have,

$$p(\mathbf{x}) = \pi(f_{\mathbf{w}}^{-1}(\mathbf{x}))\left|det\left(\frac{\partial f_{\mathbf{w}}^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right| \tag{2.24}$$

where,

$$f_w^{-1}(x) = f_{1,\mathbf{w}_1}^{-1} \circ f_{2,\mathbf{w}_2}^{-1} \circ ... \circ f_{n,\mathbf{w}_n}^{-1}(x) \tag{2.25}$$

This is useful because via the inverse mapping we can apply a tractable loss, where otherwise we would have an intractable likelihood to evaluate. On the other hand, once a mapping is learned, we can easily sample from $\mathbf{x} \sim p(\mathbf{x})$, by transforming a vector of standard Gaussian variables $\mathbf{u}$.

Specifically, if our problem demands us to learn the density $p(\mathbf{x})$ from samples $\{\mathbf{x}_1, .., \mathbf{x}_N\}$, via a flow $f_{\mathbf{w}}(.)$, we can now use stochastic gradient descent (SGD) (Robbins and Monro, 1951; Bottou,

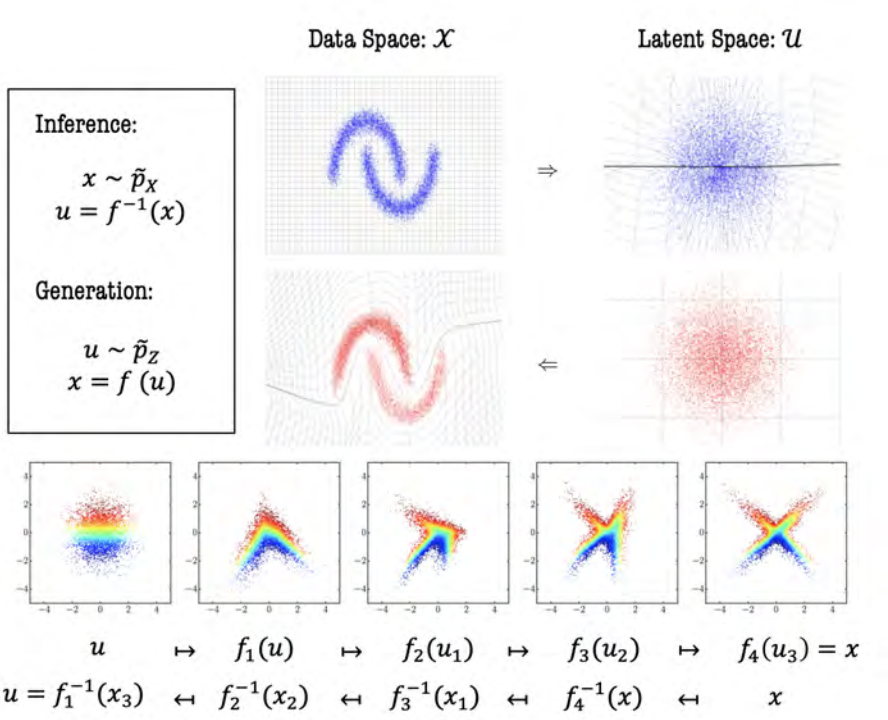

Figure 2.10. This figure illustrates the basic idea behind invertible flow transformations. Adapted from Dinh, Sohl-Dickstein, and S. Bengio, 2016 and Papamakarios, Nalisnick, et al., 2019

2012) or any of it's commonly used variants (Diederik P Kingma and Ba, 2014; Ioffe and Szegedy, 2015) to attack the following minimization problem,

$$\mathbf{w}^* = \underset{\mathbf{w}}{argmin} \frac{1}{N} \sum_{i=1}^{N} \left[-\log(\pi(f_w^{-1}(\mathbf{x}_i))) - \log\left(det \left| \frac{\partial f_{\mathbf{w}}^{-1}(\mathbf{x}_i)}{\partial \mathbf{x}} \right| \right) \right] \tag{2.26}$$

I turn the focus now to specific mappings (or sequences of mappings) $f : u \to \mathcal{X}$. As a general principle, research efforts has gone into devising *maximally expressive* such mappings f, while *maintaining computational tractability of the determinant terms in the loss functions.* I first consider a main example of *joint* estimators, specifically the RNVP approach (Dinh, Sohl-Dickstein, and S. Bengio, 2016), which extends it's predecessor algorithm NICE Dinh, Krueger, and Y. Bengio, 2014.

The basic workhorse of RNVPs is the *coupling layer.* A coupling layer first partitions the D dimensional vector $\mathbf{u}$ into two parts, $\mathbf{u}_{1:d}$ and $\mathbf{u}_{d+1:D}$ (where $d > \frac{D}{2}$) and then applies,

$$f_i(\mathbf{u}) = \begin{cases} x_{1:d} & = u_{1:d} \\ x_{d+1:D} & = u_{d+1:D} \odot \exp(s(u_{1:d})) + t(u_{1:d}) \end{cases} \tag{2.27}$$

where $\odot$ signifies *element-wise multiplication.* This function has inverse,

$$f_i^{-1}(\mathbf{x}) = \begin{cases} u_{1:d} & = x_{1:d} \\ u_{d+1:D} & = (x_{d+1:D} - t(x_{1:d})) \odot \exp(-s(x_{1:d})) \end{cases} \tag{2.28}$$

where $s(.)$ and $t(.)$ are arbitrary Neural Networks with appropriate output dimensionality. Now crucially, by design we then have,

$$\frac{\partial f(\mathbf{u})}{\partial \mathbf{u}^T} = \frac{\partial \mathbf{x}}{\partial \mathbf{u}^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:D}}{\partial \mathbf{u}_{1:d}^T} & diag(\exp(s(\mathbf{u}_{1:d})) \end{bmatrix} \tag{2.29}$$

with,

$$\log det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{u}^T} \right) = \sum_{i=1}^{D-d} s(\mathbf{u}_{1:d})_i \tag{2.30}$$

which makes the determinant computation computationally trivial (naively $O(n^3)$. Noting that a *permutation* is invertible, with determinant 1, we can then construct a Flow by stacking permutation and coupling layers into a deep network which can be trained using Eq. 2.26 (where $\mathbf{w}$ now refers collectively to the weights of the Neural Networks s and t used in the stacked coupling layers).

Specific tractable layers which can be used as alternatives to or in conjunction with coupling layers in deep *joint* flows form an active area of research (Rezende and Mohamed, 2015; Rezende, Papamakarios, et al., 2020; Huang, Dinh, and Courville, 2020a; Nielsen et al., 2020; Sorrenson, Rother, and Köthe, 2020; Berg et al., 2018; Hoogeboom et al., 2020; Tran, Vafa, et al., 2019), including continuous versions which are based on a differential equation framework instead of iterative mappings as described in the example of RNVP Flows (Huang, Dinh, and Courville, 2020b; Dupont, Doucet, and Teh, 2019).

We now turn to the second branch of Flows, *autoregressive* flows. The connection between autoregressive Neural Networks and density estimation has been developed since roughly a decade (Larochelle and I. Murray, 2011; Uria, I. Murray, and Larochelle, 2013; Durk P Kingma et al., 2016; Raiko et al., 2014; Huang, Krueger, et al., 2018), however early conceptual roots may be found earlier (Frey, Hinton, and Dayan, 1996).

We discuss in more detail *masked autoregressive flows* (MAFs) (Papamakarios, Pavlakou, and I. Murray, 2017), developed as an extension of the ideas in Germain et al., 2015. The most recent iteration, dubbed neural spline flows (NSFs) (Durkan, Bekasov, et al., 2019), is available in state-of-the-art software for Simulation Based Inference (Tejero-Cantero et al., 2020), and builds on the foundational ideas implicit in MAFs. The details of NSFs however are not conducive to the illustration of the basic principles here.

A MAF layer i is defined by the following mapping $f : \mathcal{U} \to \mathcal{X}$.

$$f_i(\mathbf{u})_j = \mathbf{u}_j \exp(s_j(\mathbf{x}_{1:j-1})) + t_j(\mathbf{x}_{1:j-1}), \quad j \in \{1, ..., D\} \tag{2.31}$$

with inverse,

$$f_i^{-1}(\mathbf{x})_j = (\mathbf{x}_j - t_j(\mathbf{x}_{1:j-1}))\exp(-s_j(\mathbf{x}_{1:j-1})), \quad j \in \{1, ..., D\} \tag{2.32}$$

where $\mathbf{u} \sim \mathcal{N}(0, \mathbb{I})$), and s_j, t_j are in general arbitrary Neural Networks (however in applications architecturally constrained). Note the major difference to the layers for RNVPs defined above: MAF layers are *autoregressive*, in that the jth output, depends on input $\mathbf{x}_{1:j-1}$, whereas no such property is enforced for RNVP layers. The autoregressive property automatically renders the log determinant of the Jacobian tractable as the sum over the log diagonal,

$$\log det\left(\frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}^T}\right) = -\sum_{j=1}^{D} s_j(\mathbf{x}_{1:j-1}) \tag{2.33}$$

Moreover, one may note that layers in MAFs are designed to have x_j depend on $\mathbf{x}_{1:j-1}$ whereas RNVP layers make $\mathbf{x}$ depend exclusively on $\mathbf{u}$. Using *masking of connections* to preserve the autoregressive property in an otherwise standard feedforward Neural Network (Germain et al., 2015) MAF layers can compute $f^{-1}(\mathbf{x}) = \mathbf{u}$ in one forward pass (as RNVP), however sampling $\mathbf{x}$ demands D steps of recursion (Papamakarios, Pavlakou, and I. Murray, 2017).

As with RNVP layers, MAF layers can be stacked to allow for more complex generative models. More expressive than RNVPs (Papamakarios, Pavlakou, and I. Murray, 2017), MAFs are explicitly designed to allow fast density evaluation for a new datapoint from the target distribution $p(\mathbf{x})$, but are slower than RNVPs when the goal is to use them as generative models. This trade-off emerges as relevant in applications to LFI, however I delay the discussion of this aspect for now.

Autoregressive density estimators as *joint* density estimators form an active area of research (Bhattacharyya et al., 2020; Marino et al., 2020; Khemakhem et al., 2020). In general, as flexible density estimators for intractable probability distributions (implicit models) flows are widely applied and the development of ever new variations is ongoing (Kobyzev, Prince, and Brubaker, 2020; Papamakarios, Nalisnick, et al., 2019; J. Liu et al., 2019; Rezende, Papamakarios, et al., 2020; Durkan, Bekasov, et al., 2019). Papamakarios, Nalisnick, et al., 2019 provide a very detailed review dedicated to flow based methods with a view towards probabilistic modeling.

2.5.2 Basic Building Block: Ratio Estimators

Ratio estimators re-frame our estimation problem in the context of LFI as a classification task. Instead of targeting a density we target a likelihood ratio, bringing the idea in connection with the framework of hypothesis testing (Cranmer, Pavez, and Louppe, 2015). This representation can be beneficial if our algorithm (specifically the SNRE algorithm detailed later) attempts posterior inference by learning approximate likelihoods. For a given dataset $\mathbf{x}$, specifying a family of generative probability models $p_\theta(), \theta \in \Theta$ and denoting the likelihood of $\mathbf{x}$, given parameter θ, by $p_\theta(\mathbf{x})$, we define the *likelihood ratio* $r(.|.)$ as,

$$r(\mathbf{x}|\theta_0, \theta_1) := \frac{p_{\theta_0}(\mathbf{x})}{p_{\theta_1}(\mathbf{x})} \tag{2.34}$$

We can simulate (as per assumption) $\mathbf{x} \sim p_\theta(\mathbf{x})$ for any $\theta \in \Theta$. Now we wish to define a function (our classifier) $d_{\mathbf{w}} : \mathbb{R}^D \to [0, 1]$ that learns to separate samples $\mathbf{x} \sim p_{\theta_0}(\mathbf{x})$ from samples $\mathbf{x} \sim p_{\theta_1}(\mathbf{x})$. It can be shown (under assumption of a prior $\pi(\theta_0) = \pi(\theta_1) = \frac{1}{2}$) (Cranmer, Pavez, and Louppe, 2015) that the optimal solution $\mathbf{w}^*$, given $d_{\mathbf{w}}(.)$ is expressive enough,

$$d_{\mathbf{w}^*}(\mathbf{x}) = p(\theta_0|\mathbf{x}) = \frac{p(\mathbf{x}|\theta_0)}{p(\mathbf{x}|\theta_0) + p(\mathbf{x}|\theta_1)} \tag{2.35}$$

so that,

$$r(\mathbf{x}|\theta_0, \theta_1) = \frac{d_{\mathbf{w}^*}(\mathbf{x})}{1 - d_{\mathbf{w}^*}(\mathbf{x})} \tag{2.36}$$

This estimator (including extensions and variations) is widely used in LFI (Michael U Gutmann, Dutta, et al., 2018; Thomas et al., 2020; Brehmer et al., 2020; Rhodes, Xu, and Michael U. Gutmann, 2020), with another prominent application being as the discriminator loss in *generative adversarial networks* (GANs) (Goodfellow, Pouget-Abadie, et al., 2014; Uehara et al., 2016; Mohamed and Lakshminarayanan, 2016).

We focus on an extension by (Hermans, Begy, and Louppe, 2020), that is especially useful for amortization strategies to LFI. Hermans, Begy, and Louppe, 2020 propose to instead learn $d_{\mathbf{w}}(.,.) : \mathbb{R}^{D_x} \times \mathbb{R}^{D_\theta} \to [0, 1]$, where $d_{\mathbf{w}}(.,.)$ attempts to distinguish *joint samples* $(\mathbf{x}, \theta) \sim p(\mathbf{x}, \theta)$ from *independent samples* $(\mathbf{x}, \theta) \sim p(\mathbf{x})p(\theta)$. Again for an expressive enough $d_{\mathbf{w}}(.,.)$, the optimal classifier can be shown to respect

$$d_{\mathbf{w}^*}(\mathbf{x}, \theta) = \frac{p(\mathbf{x}, \theta)}{p(\mathbf{x}, \theta) + p(\mathbf{x})p(\theta)} \tag{2.37}$$

with the corresponding ratio estimator,

$$\frac{d_{\mathbf{w}^*}(\mathbf{x}, \theta)}{1 - d_{\mathbf{w}^*}(\mathbf{x}, \theta)} = \frac{p(\mathbf{x}, \theta)}{p(\mathbf{x})p(\theta)} = \frac{p(\mathbf{x}|\theta)}{p(\mathbf{x})} = r(\mathbf{x}|\theta) \tag{2.38}$$

now estimating the *likelihood to evidence ratio*, rather than a traditional likelihood ratio.

Hermans, Begy, and Louppe, 2020 show that training under *binary cross entropy*, where we draw training samples as triples $\{(\mathbf{x}, \theta, \theta^m)_i\}_{i=1}^N$, where $(\mathbf{x}, \theta)_i \sim p(\mathbf{x}, \theta)$ and $\theta_i^m \sim p(\theta)$, yields the optimal solution defined in Eq. 2.37. The loss is then defined as

$$\mathcal{L}^{Ratio}(\mathbf{w}) = \sum_{i=1}^{N} -\log(d(\mathbf{x}_i, \theta_i)) - \log(1 - d(\mathbf{x}_i, \theta_i^m)) \tag{2.39}$$

These ratio estimators are a highly flexible building blocks for LFI-NN algorithms, specifically those designed to focus the learning problem on the likelihood instead of directly on the posterior. Detailed description of a selection of relevant algorithms follows in the section dedicated to *algorithms that target the likelihood*. Figure 2.11 shows how the idea of ratio estimation may be utilized in the context of MCMC algorithms Metropolis et al., 1953; Hastings, 1970.

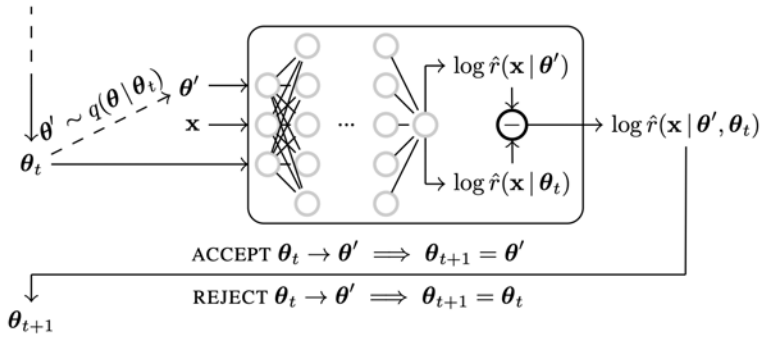

Figure 2.11. This figure illustrates how the ratio estimators may be utilized in an MCMC algorithm for a downstream task. The corresponding network is set up to use two ratio estimators, so that the output is a likelihood ratio between $\ell(\theta_t|\mathbf{x})$ and $\ell(\theta^{'}|\mathbf{x})$. This likelihood ratio is all that needs to be evaluated in the context of MCMC algorithms. Taken from Hermans, Begy, and Louppe, 2020.

2.5.3 Basic Building Block: Networks Relevant for Summary Statistics

Discussed previously in the context of traditional ABC methods, *summary statistics* play an important role for many of the Neural Network based LFI approaches. Some NN-LFI algorithms implicitly assume a relatively low-dimensional observed data-vector $\mathbf{x}$, essentially outsourcing the problem of finding such low-dimensional summaries, before application of the sampling algorithm Papamakarios and I. Murray, 2016; Papamakarios, Pavlakou, and I. Murray, 2017. Others propose explicitly an end-to-end trainable pipeline which incorporates a *summary statistics network* (e.g. S. T. Radev, Mertens, et al., 2020).

Generally, finding appropriate summary statistics (low dimensional summaries) for high-dimensional data-vectors is a problem that has attracted attention from the statistics and machine learning communities in its own right (Friedman, Hastie, and Tibshirani, 2001). One can distinguish between two approaches. First, finding summary statistics via a target that is relevant for *model based inference*. Second, finding data compressions by means divorced from the downstream inference context.

Concerning the second approach, for purposes of this review I constrain myself to mention that it is an active research area, with promising approaches based on recent advances in methods for neural estimation of mutual information criteria (Battiti, 1994; Paninski, 2003; Hjelm et al., 2018; Belghazi et al., 2018; Chan et al., 2019).

We will now consider in some detail the approach of finding summary statistics for purposes of model based inference. A simple such method was devised by Jiang et al., 2017.

For a given (likelihood free) probabilistic model $p_{\mathcal{M}}(\mathbf{x}|\theta)$, the authors suggest to learn a mapping $f_{\mathbf{w}} : \mathbf{x} \to \theta$, which targets $\mathbb{E}_{p_{\mathcal{M}}}[\theta|\mathbf{x}]$ by minimizing the *mean squared error*,

$$\mathcal{L}^{RMSE}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} ||f_{\mathbf{w}}(\mathbf{x}_i) - \theta_i||_2^2 \tag{2.40}$$

Under assumption of convergence $f_{\mathbf{w}^*}(\mathbf{x}) = \mathbb{E}_{p_{\mathcal{M}}}[\theta|\mathbf{x}]$. The authors show that in this case, the downstream application of an ABC sampler will yield an approximate posterior $\pi^{\epsilon}_{ABC}(\theta) = \pi(\theta \mid ||f_{\mathbf{w}^*}(\mathbf{x}) - f_{\mathbf{w}^*}(\mathbf{x}_{obs})|| < \epsilon)$ which, in the limit $\epsilon \to 0$, respects,

$$\lim_{\epsilon \to 0} \mathbb{E}_{\pi^{\epsilon}_{ABC}}[\theta] = \mathbb{E}_{\pi}[\theta|\mathbf{x} = \mathbf{x}_{obs}] \tag{2.41}$$

In other words, using this approach to learn summary statistics targets posteriors which are correct in their *first moment*.

Concretely (however not essential for application of the basic principle of this approach), Jiang et al., 2017 suggest to apply *Convolutional Neural Networks* (CNNs) (LeCun et al., 1989), which implicitly allows inputs of variable size to be handled. While this approach was a promising first step, it suffers from a basic shortcoming. We are constrained to compress our data-set $\mathbf{x}_{obs}$ into a fixed length vector of length $dim(\theta)$. In light of the discussion on *sufficient statistics* earlier, specifically the *Pitman-Koopman-Darmois Theorem*, we must immediately doubt sufficiency and moreover the consistency of posterior behavior across data-set sizes. It seems therefore in general of interest to allow the *embedding dimensionality* to be dissociated from dimensionality of the parameter space of our model $\mathcal{M}$.

This dissociation can be accomplished by separating the loss target from $dim(\theta)$. (we note that the aforementioned methods based on mutual information estimation accomplish this dissociation too, as a major principal advantage over the approach proposed by Jiang et al., 2017).

Current LFI-NN algorithms, allow for this via an end-to-end differentiable pipeline, which bases the loss directly on the posterior, so that an intermediate *encoding* or *embedding* network is learned implicitly. This will be discussed more in the respective section on NN-LFI algorithms, however it matters here insofar as it allows focus on basic properties of summary networks that one may sensibly want to request, without worrying about the loss function on the summary network directly.

In general, the architecture of summary networks should follow the inherent structure of data generated by our simulator model $\mathcal{M}$. If $\mathcal{M}$ generates time-series data, we may choose a *Recurrent Neural Network* (RNN) (Rumelhart, Hinton, and Williams, 1986; Hochreiter and Schmidhuber, 1997) architecture. An example from neuroscience would be a SDE driven neural firing model, for which we can observe the firing rate over the course of an experimental trial (K.-F. Wong and X.-J. Wang, 2006). If the model generates 2D structures akin to images, such as e.g. the Ising model (Cipra, 1987; Kadirvelu, Hayashi, and Nasuto, 2017), a natural choice (as pursued as a toy example in Jiang et al., 2017) may be *Convolutional Neural Networks* (CNNs) (LeCun et al., 1989), which are designed to summarize spatial information (usually for a final classification purpose). If $\mathcal{M}$ generates relational structures (e.g. random graphs), a suitable architecture may be *Graph Neural Networks* (GNNs) (Scarselli et al., 2008; Zhou et al., 2018).

A secondary but crucial aspect of data-sets $\mathbf{x}_{obs}$, is that commonly we are not dealing with single data-points from $\mathcal{M}$, but instead with multiple data-points, which are under $\mathcal{M}$ either *independent and identically distributed* (i.i.d), or as a related Bayesian notion *exchangeable* (Aldous, 1985). Single data-points may be time-series or any of the other data-types mentioned in the previous paragraph,

however collectively, the data-points act not as a *sequence*, but as a *set*. The implication is that over this set, we desire a network architecture which is *permutation invariant*. Formally, if we represent $\mathbf{x}_{obs}$ as $\{\mathbf{x}_{obs,1}, ..., \mathbf{x}_{obs,n}\}$, and our summary network as an overall mapping $f_{\mathbf{w}} : \mathbf{x}_{obs} \rightarrow \mathbf{s} \in \mathbb{R}^{dim(S)}$, where $dim(S)$ is the desired dimension of the summary statistic, then we request that,

$$f^I_{\mathbf{w}}(\{\mathbf{x}_{obs,1}, ..., \mathbf{x}_{obs,n}\}) = f^I_{\mathbf{w}}(\{\mathbf{x}_{obs,\pi(1)}, ..., \mathbf{x}_{obs,\pi(n)}\}) \tag{2.42}$$

for any permutation $\pi(.)$ of $\{1, .., n\}$. The importance of permutation invariance across a variety of application domains has led to active research into the characterization and development of Neural Network architecture which embed this principle a priori (Zaheer et al., 2017; Charles R Qi et al., 2017; Charles Ruizhongtai Qi et al., 2017; J. Lee et al., 2019; Vaswani et al., 2017). A general, constructive characterization of how to build permutation invariant networks through the perspective of probabilistic invariances (Kallenberg, 2006), was developed by Bloem-Reddy and Teh, 2020.

The authors suggest to build a permutation invariant Neural Network by stacking a sequence of *equivariant* modules, topped by a final *invariant* module. An module equivariant to permutations, represented by a function $f^E_{\mathbf{w}} : \mathbf{x} \rightarrow \mathbf{y}$, has the following property,

$$f^E_{\mathbf{w}}(\{\mathbf{x}_{obs,\pi(1)}, ..., \mathbf{x}_{obs,\pi(n)}\}) = \{\mathbf{y}_{obs,\pi(1)}, ..., \mathbf{y}_{obs,\pi(n)}\} \tag{2.43}$$

Figure 2.12, illustrates the structure of such networks in graphical form. Of importance to us is the following characterization (we specialize to deterministic functions, however note that the results of Bloem-Reddy and Teh, 2020 generalize to stochastic modules),

$$f^I(\mathbf{x}) = f_{\mathbf{w}_I}(\sum_{i=1}^{N} \phi_{\mathbf{w}_\phi}(\mathbf{x}_i)), \ \textit{Invariant Module} \tag{2.44}$$

$$\big(f^E(\mathbf{x})\big)_i = f_{\mathbf{w}_E}(\mathbf{x}_i, f^I(\mathbf{x})), \ \textit{Equivariant Module} \tag{2.45}$$

where $f_{\mathbf{w}_I}(.)$, $\phi_{\mathbf{w}_\phi}(.)$ and $f_{\mathbf{w}_E}(.)$ are Neural Networks. Note that invariant modules demand a pooling operation before application of $f_{\mathbf{w}_I}(.)$, where $\phi_{\mathbf{w}_\phi}(.)$, prescribes the *embedding dimension*. Moreover, note that the equivariant module is applied dimensions wise.

2.5.4 Algorithms that target the Posterior

State of the art LFI-NN algorithms which target the posterior distribution directly, can be traced back to a conceptually foundational paper by Papamakarios and I. Murray, 2016, titled: Fast ε-free Inference of Simulation Models with Bayesian Conditional Density Estimation. Their approach was later taxonomized as *sequential neural posterior estimation* (SNPE), and is now embedded in an arc of improvement from SNPE-A over SNPE-B, to the most current SNPE-C. Since this sequence of refinements sits firmly on the conceptual foundations laid by SNPE-A, I explain the SNPE-A approach in some detail.

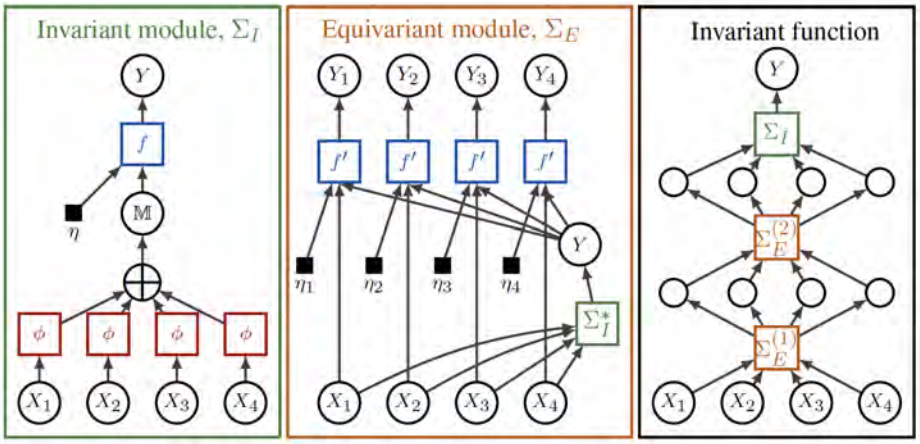

Figure 2.12. The figure provides the schemata on how to construction invariant functions via iterations (stacking) of *equivariant modules*, topped by an *invariant module*. *Circles* denote random variables. *Blue squares*, denote arbitrary functions (possibly with outsourced noise terms η). *Red squares* denote arbitrary embedding functions. The $\oplus$ denotes symmetric pooling operations. Figure taken from Bloem-Reddy and Teh, 2020.

Papamakarios and I. Murray, 2016 suggest to iteratively learn the posterior $p(\theta|\mathbf{s}_{obs})$ in form of a Gaussian-MDN $q_{\mathbf{w}}^{MDN}(\theta|\mathbf{x})$. The iterative (sequential) part to the algorithms lies in the progressive refinement of the prior distribution $\tilde{\pi}_i(\theta)$ (at iteration i), towards the posterior true posterior $p(\theta|\mathbf{s}_{obs})$. The idea is to *guide simulations* in an attempt to minimize wasted computations, which is achieved by simulating training data only *where it matters* (parameters for which the true posterior has mass). At iteration i we simulate training data according to,

$$\theta_n \sim \tilde{\pi}_i(\theta) \quad , \quad \mathbf{s}_n \sim \mathcal{M}(\theta) \tag{2.46}$$

Training-data so generated, given an expressive enough Gaussian-MDN, the conditional density estimator $q_{\mathbf{w},i}^{MDN}(\theta|\mathbf{s})$ can then be trained via $\mathcal{L}_{SVI}$ or $\mathcal{L}_{ML}$ as described earlier in the section dedicated to MDNs. Such a network will then converge to yield (*Proposition 1*, Papamakarios and I. Murray, 2016),

$$q_{\mathbf{w},i}^{MDN}(\theta|\mathbf{s}) \tilde{\propto} \frac{\tilde{\pi}_i(\theta)}{\pi(\theta)} p(\theta|\mathbf{s}) \tag{2.47}$$

or,

$$p(\theta|\mathbf{s}) \tilde{\propto} \frac{\pi(\theta)}{\tilde{\pi}_i(\theta)} q_{\mathbf{w},i}^{MDN}(\theta|\mathbf{s}) \tag{2.48}$$

where $\pi(\theta)$ is the actual prior we would like to apply to the underlying inference problem. Choosing the true prior to be a Gaussian Mixture Model (or otherwise Uniform), approximation i to the posterior $p(\theta)$ can then be found via analytical computations (a Gaussian conjugate problem in Bayesian Analysis), by plugging in $\mathbf{s}_{obs}$ for $\mathbf{s}$. This yields again a Gaussian mixture as an approximate posterior and renders sampling trivial.

The resulting iterative algorithm is now commonly encountered under the acronym SNPE-A. While conceptually important, and elegant as far as the involved analytical computations are concerned, the algorithm was subsequently refined (Lueckmann, Goncalves, et al., 2017; Greenberg, Nonnenmacher, and Macke, 2019) via SNPE-B and SNPE-C to allow for a wider range of distributional shapes and overcome endogenous numerical problems, which were acknowledged already in Papamakarios and I. Murray, 2016.

Algorithm 6: SNPE-A

Input: Prior: $\tilde{\pi}_0$;
Simulator: $\mathcal{M}$;
Initialize: $q_{\mathbf{w},1}(\theta|\mathbf{x})$ with K components;
Set: i = 1;

while *not converged* **do**
  for $n = 1..N$ **do**
    Sample $\theta_n \sim \tilde{\pi}_i(\theta)$,
    Sample $\mathbf{x}_n \sim \mathcal{M}(\theta_n)$
  end
  train $q_{\mathbf{w},i}(\theta|\mathbf{x})$ with $\{\theta, \mathbf{x}\}_{i=1}^{N}$;
  set $\tilde{\pi}_{i+1}(\theta) \leftarrow \frac{\pi(\theta)}{\tilde{\pi}_i(\theta)} q_{\mathbf{w},i}(\theta|\mathbf{s}_{obs})$;
  $i = i + 1$;
  check convergence;;
end

Result: $\tilde{\pi}_{i+1}(\theta)$

With SNPE-B, Lueckmann, Goncalves, et al., 2017 add two innovations to the SNPE-A algorithm, which respectively increases the efficiency of the training procedure by avoiding training restart at every iteration and increases modeling flexibility, allowing for more complex priors and density estimators by avoiding a reliance on full analytical tractability. SNPE-B uses a density estimator $q_{\mathbf{w}}(\theta|\mathbf{s})$ (as opposed to SNPE-A, the restriction to MDNs is lifted) to directly represent the posterior $p(\theta|\mathbf{s})$.

Changed is the optimization objective. For iteration (round) r of the the optimization process, the loss function is defined to be,

$$\mathcal{L}_r^{SNPE-B}(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\frac{\pi(\theta_i)}{\tilde{\pi}_r(\theta_i)}\log q_{\mathbf{w}}(\theta_i|\mathbf{s}_i) + \frac{1}{N}D_{KL}\left(\pi_r(\mathbf{w})||\pi_{r-1}(\mathbf{w})\right) \tag{2.49}$$

which combines the VI loss for MDNs with an importance weighting on the loss of $q_{\mathbf{w}}(\theta|\mathbf{s})$. The KL-divergence part allows training to be informed by past rounds, by making the posterior over weights in round $r-1$, the prior over weights in round r. The importance weights adjust the training loss so that instead of,

$$q_{\mathbf{w},i}(\theta|\mathbf{s}) \tilde{\propto} \frac{\tilde{\pi}_i(\theta)}{\pi(\theta)} p(\theta|\mathbf{s}) \tag{2.50}$$

our density estimator is directly proportional to the posterior,

$$q_{\mathbf{w},i}(\theta|\mathbf{s}) \propto p(\theta|\mathbf{s}) \tag{2.51}$$

Hence easy sampling from the posterior does not depend on analytic tractability of Eq. 2.50, as in SNPE-A. The resulting SNPE-B algorithm is

Algorithm 7: SNPE-B

Input: Prior: $\tilde{\pi}_0$;
Simulator: $\mathcal{M}$;
Initialize: $q_{\mathbf{w},1}(\theta|\mathbf{x})$ with K components;
Set: i = 1;

while *not converged* **do**
  for $n = 1..N$ **do**
    Sample $\theta_n \sim \tilde{\pi}(\theta)$;
    Sample $\mathbf{x}_n \sim \mathcal{M}(\theta_n)$
  end
  continue training $q_{\mathbf{w},i}(\theta|\mathbf{x})$ with $\{\theta, \mathbf{x}\}_{i=1}^{N}$ and $\mathcal{L}_i^{SNPE-B}(\mathbf{w})$;
  set $\tilde{\pi}_{i+1}(\theta) \leftarrow q_{\mathbf{w},i}(\theta|\mathbf{s}_{obs})$;
  $i = i + 1$;
  check convergence;;
end

Result: $\tilde{\pi}_{i+1}(\theta)$

These innovations allow for greater flexibility, however the approach was criticized for added training instability due to the high variability of importance ratios in realistic settings ($\tilde{\pi}_r(.)$ can be arbitrarily different from $\pi(.)$, Greenberg, Nonnenmacher, and Macke, 2019). This problem can lead to failure modes. As can be discerned, the final algorithm is only very slightly different from SNPE-A.

SNPE-C Greenberg, Nonnenmacher, and Macke, 2019, as of this moment part of the state of the art algorithms for LFI-NN, aims to maintain the flexbility of SNPE-B, while however overcoming the issues with importance weights. Moreover the authors improve on the cross-rounds training introduced with SNPE-B.

The authors instead start from the following identity,

$$\tilde{p}(\theta|\mathbf{s}) = p(\theta|\mathbf{s}) \frac{\tilde{\pi}(\theta)}{\pi(\theta)} \frac{\pi(\mathbf{s})}{\tilde{\pi}(\mathbf{s})} \tag{2.52}$$

and define,

$$\tilde{q}_{\mathbf{w}}(\theta|\mathbf{s}) = q_{\mathbf{w}}(\theta|\mathbf{s}) \frac{\tilde{\pi}(\theta)}{\pi(\theta)} \frac{1}{Z(\mathbf{s}, \mathbf{w})} \tag{2.53}$$

for some density estimator $q_{\mathbf{w}}(\theta, \mathbf{s})$ and normalization constant, $Z(\mathbf{s}, \mathbf{w})$. The idea is to use $\tilde{q}_{\mathbf{w}}(\theta|\mathbf{s})$ for the loss function,

$$\mathcal{L}^{SNPE-C}(\mathbf{w}) = -\sum_{i=1}^{N} \log\left(\tilde{q}_{\mathbf{w}}(\theta_i|\mathbf{s}_i)\right) \tag{2.54}$$

however we train the parameters $\mathbf{w}$ of $q_{\mathbf{w}}(\theta|\mathbf{s})$, which will finally be proportional to $p(\theta|\mathbf{s})$.

As Greenberg, Nonnenmacher, and Macke, 2019 concede, we rarely have access to $Z(\mathbf{s}, \mathbf{w})$, which motivates an extension to *atomic proposals*. Using $\tilde{\pi}(\theta) = \mathcal{U}(\Theta)$ (a uniform distribution over the set Θ), with $\{\theta_1, .., \theta_m\} = \Theta$, where $\Theta \sim \mathcal{V}$ for some *hyperproposal* $\mathcal{V}$, we can define,

$$\tilde{q}_{\mathbf{w}}(\theta|\mathbf{s}) = \frac{q_{\mathbf{w}}(\theta|\mathbf{s})/\pi(\theta)}{\sum_{\theta' \in \Theta} q_{\mathbf{w}}(\theta'|\mathbf{s})/\pi(\theta')} \tag{2.55}$$

as a discrete distribution for which,

$$\mathbf{w}^* = \underset{\mathbf{w}}{argmin}\ \mathbb{E}_{\Theta\sim\mathcal{V},\theta\sim\mathcal{U}(\Theta),\mathbf{s}\sim p(\mathbf{s}|\theta)}\left[\mathcal{L}^{SNPE-C}(\mathbf{w})\right] \tag{2.56}$$

will yield $q_{\mathbf{w}^*}(\theta|\mathbf{s}) = p(\theta|\mathbf{s})$ (*Proposition 1* in Greenberg, Nonnenmacher, and Macke, 2019). Connections have been draw between this procedure and the ratio estimators describes as basic building blocks(Durkan, I. Murray, and Papamakarios, 2020).

Algorithm 8: SNPE-C

Input: Initialize: $q_{\mathbf{w},1}(\theta|\mathbf{s})$;
Prior: $p(\theta)$;
Simulator: $\mathcal{M}$;
$\mathbf{s}_{obs}$;
N simulations per training round ;
R number of rounds ;
M number $|\Theta|$ of atoms ;
Set: c = 0;

for $r = 1..R$ **do**
  for $n = 1..N$ **do**
    increment c ;
    sample $\theta_c \sim \tilde{\pi}_r(\theta)$;
    sample $\mathbf{s}_c \sim \mathcal{M}(\theta_c)$
  end
  Set up distribution $\mathcal{V}_r(\Theta) := \binom{c}{M}^{-1}$ *for* $\Theta = \{\theta_{b_1}, .., \theta_{b_M}\}$, $1 \leq b_1 \leq ... \leq b_M \leq c$;
  $\mathbf{w}_r^* = \underset{\mathbf{w}}{argmin}\ \mathbb{E}_{\Theta\sim\mathcal{V}_r(\Theta)}\left[\mathcal{L}^{SNPE-C}(\mathbf{w})\right]$;
  $\tilde{\pi}_{r+1}(\theta) := q_{\mathbf{w}^*,r}(\theta|\mathbf{s}_{obs})$;
end

Result: $q_{\mathbf{w}^*,R}(\theta|\mathbf{s})$

SNPE-C is a highly flexible algorithm, however a weakness remains. The learned posterior $q_w^*(\theta|\mathbf{s}_{obs})$ can leak mass outside of the support of $p(\theta)$. Hence, we cannot sample from $q_w^*(\theta|\mathbf{s}_{obs})$ naively, but have to consider a procedure such as *rejection sampling*, which may incur substantial

computational loss if rejection rates are high (possibly exacerbated by the dimensionality of the original posterior inference problem).

Another recent approach, closely aligned with the lineage of relevant concepts of SNPE, however with a focus on global amortization of the posterior (across parameter space and relevant dataset sizes) is BayesFlow (S. T. Radev, Mertens, et al., 2020). This approach provides an end-to-end pipeline which combines invertible flows with permutation invariant embedding (summary statistic) networks, amortizing a complete *inference scenario*. Conditioned on an experimental (or otherwise observed) dataset, sampling is achieved simply by drawing random samples from the learned conditional flow. Specifically, the authors use RNVP-Flows (as described earlier), and explicitly build summary statistic networks according to the guidelines derived from Bloem-Reddy and Teh, 2020. For completeness I list the resulting algorithm below, and show a graphic schemata in Figure 2.13,

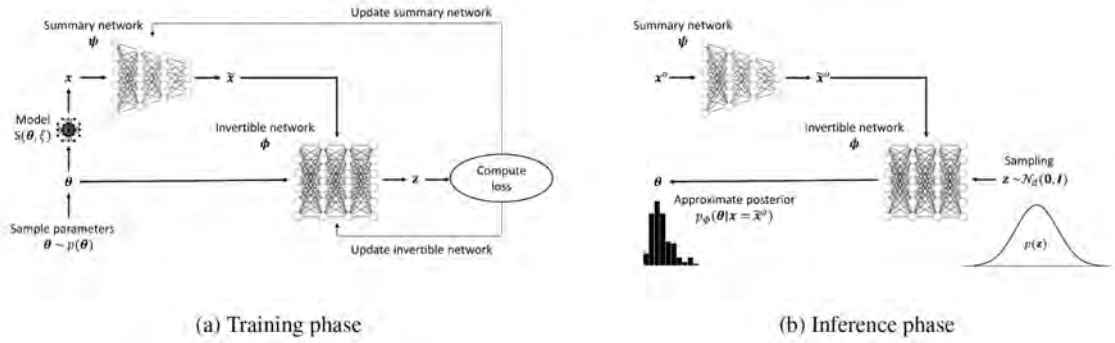

Figure 2.13. Schemata for the BayesFlow algorithm. During the training phase, one jointly trains a summary statistics network and a invertible conditional flow (ICF) from simulation data. At inference, one passes the observed dataset (here x^0) through the learned summary statistic network and samples from the inverted ICF, which is conditioned on the respective summary statistic ($\tilde{x}^0$). Figure taken from S. T. Radev, Mertens, et al., 2020

Algorithm 9: BayesFlow

Input: Initialize: $q_{\mathbf{w}_{Flow}}(\theta|\mathbf{s})$ (here RNVP), $f_{\mathbf{w}_{Summary}}(\mathbf{x})$ (summary network) ;
prior $p(\theta)$;
simulator $\mathcal{M}(\theta)$;
N simulations per training round ;
B Batch size ;
N_{min}, N_{max} maximum and minimum numbers of observations allowed ;

repeat
  Sample current number of observations $N \sim \mathcal{U}(N_{min}, N_{max})$;
  for $b = 1..B$ **do**
    sample $\theta_b \sim \tilde{\pi}_r(\theta)$;
    for $n = 1..N$ **do**
      sample $\mathbf{x}_n^b \sim \mathcal{M}(\theta_b)$
    end
    Compute $\mathbf{s}^b = f_{\mathbf{w}_{Summary}}(\mathbf{x}_{1:N}^b)$;
    Compute $\mathbf{z}^b = q_{\mathbf{w}_{Flow}}(\theta^b|\mathbf{s}^b)$;
  end
  SGD update on $(\mathbf{w}_{Flow}, \mathbf{w}_{Summary})$ using $\{\theta^b, \mathbf{s}^b, \mathbf{z}^b\}_{1:B}$
until *until convergence to* $\mathbf{w}^*_{Flow}$, $\mathbf{w}^*_{Summary}$;
Result: $q_{\mathbf{w}^*,R}(\theta|\mathbf{s})$

BayesFlow is conceptually very closely aligned with the SNPE approaches, the explicit incorporation of the embedding network not signifying a difference in kind, but simply emphasis. SNPE-C and SNPE-B in principle allow for joint learning of an embedding network as well. Global amortization is moreover possible with the SNPE approach as well, again signifying a difference in emphasis rather than a difference in kind. To use SNPE for global amortization one would simply constrain training to the equivalent of *round 1*, and not specialize the proposal prior across rounds (since the specialization across rounds in conditioned on the observed data).

Both SNPE and BayesFlow allow for post hoc adjustments of prior distributions via e.g. rejection sampling, however in spirit both approaches aim at learning the posterior distribution for a single prior / likelihood pair. This focus on the amortization of posteriors allows very fast sampling from the resulting approximate posterior, however it does incur some implicit costs.

First, the direct focus on posteriors prevents likelihood based model comparison exercises (e.g. via Bayes Factors) (Kass and Raftery, 1995) to apply directly. Second and more severely, posterior amortization makes sense for *single inference scenarios*, which imposes several limitations. If one deals with experimental datasets, and would like to do inference across a number of experimental conditions, assuming that a subset of our model parameters are fixed, while others are affected by the experimental manipulations, then it is necessary to amortize the posterior distribution for every single experimental design one might wish to consider. Moreover we may want to use hierarchical inference to pool information about model parameters across subjects. For every number of subjects N, we

would have to amortize a different posterior. The last examples are especially damning from the viewpoint of flexible application, since they go hand in hand with an inflation of the dimensionality of the parameter space. One of the fundamental limitations of both SNPE and Bayesflow is that these methods are not suited for problems that involve high dimensional (as a rough guide $dim(\theta) > 30$) parameter spaces.

We will discuss the relative shortcomings and benefits of posterior amortization methods and turn them into practical suggestions for applications in section 3.6. However we keep in mind that the mentioned shortcomings can be viewed through the lens of problem representation. A different compartmentalization of the amortization strategy, with a focus on likelihoods, incurs a different set of trade-offs. I will discuss this strategy next.

2.5.5 Algorithms that target the Likelihood

Instead of focusing the amortization on the posterior distribution, we may focus on amortizing the likelihood function. The immediate benefit of amortizing likelihood functions lies in the potential for flexible deployment across inference scenarios. This is an important distinction from posterior targeting LFI-NN techniques, which I will discuss in more depth in section 3.6.

This section discusses two basic strategies for this approach and discusses some of the strengths and weaknesses of each. Specifically, I focus on two algorithms which are rooted in the same conceptual framework as the SNPE algorithms discussed in the previous section. These approaches are somewhat explicitly built around a Neural Network framework (to adhere to our LFI-NN classification) and are part of recent benchmarking efforts to inventory the current state of the art (Lueckmann, Boelts, et al., 2021). A number of other related approaches exist, which may not be based on Neural Networks as the basic workhorse, however conceptually maintain a close relationship (Michael U Gutmann, Dutta, et al., 2018; Dinev and Michael U Gutmann, 2018; Thomas et al., 2021).

The first approach I discuss is known as *sequential neural likelihood* (SNL) (Papamakarios, Sterratt, and I. Murray, 2019). This follows directly in the footsteps of SNPE, however one uses a density estimator (either of the ones discusses previously) $q_{\mathbf{w}}(\mathbf{s}_{obs}|\theta)$ to target the likelihood. In the framework of a SNPE algorithm, only minimal adjustments are needed. Defining, as we have seen for SNPE, a round-wise prior $\tilde{\pi}_r(\theta)$, and a corresponding round-wise joint $\tilde{p}_r(\theta, \mathbf{s})$, each round we can simply minimize,

$$\mathcal{L}^{SNL}(\mathbf{w}) = -\sum_{i=1}^{N} \log(q_{\mathbf{w}}(\mathbf{s}|\theta)), \quad (\mathbf{s}, \theta)_i \sim \tilde{p}_r(\theta, \mathbf{s}) \tag{2.57}$$

The key difference lies in the how updated priors are defined,

$$\tilde{\pi}_r(\theta) \propto q_{\mathbf{w},r}(\mathbf{s}|\theta)\pi(\theta) \tag{2.58}$$

which implies that sampling from $\tilde{\pi}_r$ needs to proceed by e.g. MCMC.

Restricting this approach to a single round, as for SNPE algorithms, allows for amortization independent of $\mathbf{s}_{obs}$. Note that a similar approach to SNL, which focuses (however is not necessarily

Algorithm 10: SNL

Input: Initialize: $q_{\mathbf{w},1}(\mathbf{s}|\theta)$;
Prior: $\tilde{\pi}_1(\theta) = \pi(\theta)$;
Simulator: $\mathcal{M}$;
Cumulative Training Data Set: $\mathcal{D} = \{\}$;
Number of training rounds: R;
Training sampler per round: N;

for $r = 1..R$ **do**
  for $n = 1..N$ **do**
    sample $\theta_n \sim \tilde{\pi}_r(\theta)$ with MCMC;
    sample $\mathbf{s}_n \sim \mathcal{M}(\theta_n)$;
    Add $(\theta_n, \mathbf{s}_n)$ to $\mathcal{D}$;
  end
  train $q_{\mathbf{w},r}(\mathbf{s}|\theta)$ on $\mathcal{D}$ with $\mathcal{L}^{SNL}(\mathbf{w})$;
  set $\tilde{\pi}_{r+1}(\theta) \leftarrow q_{\mathbf{w},r}(\mathbf{s}_{obs}|\theta)\pi(\theta)$;
end

Result: $q_{\mathbf{w},r}(\mathbf{s}|\theta)$

restricted to) on MDNs as density estimators was proposed by Lueckmann, Bassetto, et al., 2019. While very similar in spirit, the authors suggest to train an *ensemble* of density estimators, to approach the choice of parameters for simulations efficiently via *active learning*. I do not pursue this line of reasoning further, however note that it is promising as a generally applicable method for minimizing the amount of simulation runs needed for a desired level of posterior accuracy. This mirrors the principles found in the literature on LFI which bases likelihood / posterior approximations on Gaussian Processes (Meeds and Welling, 2014; Järvenpää, Michael U Gutmann, Vehtari, et al., 2021). However I refrain from pursuing this lead in more detail here.

The last method I consider was recently proposed by Hermans, Begy, and Louppe, 2020, and uses the ratio estimators as discussed in the sections on basic building blocks. Taking into account the explanations provided in said section leads us directly to an amortization algorithm for *likelihood-to-evidence* ratios $r_{\mathbf{w}}(\mathbf{s}|\theta)$ via learning the classifier $d_{\mathbf{w}}(\mathbf{s}, \theta)$,

$$r_{\mathbf{w}}(\mathbf{s}|\theta) = \frac{d_{\mathbf{w}}(\mathbf{s}, \theta)}{1 - d_{\mathbf{w}}(\mathbf{s}, \theta)} \tag{2.59}$$

Following (Lueckmann, Boelts, et al., 2021), the resulting algorithm will be called *neural ratio estimation* (NRE) in the following. I show the details in algorithm 11. Conceived as a global amortization algorithm, it can be specialized to specific datasets following Durkan, I. Murray, and Papamakarios, 2020, and turned into a sequential version that closely resembles SNL/SNPE and is consequently given the acronym SNRE (for *sequential neural ratio estimation*).

While in tendency slower in terms of sampling from the approximate posterior at inference, a focus on likelihoods, if made possible by the given application, has several advantages over direct posterior amortization. These derive from the simple fact that once the likelihood is amortized, it can be reused freely across inference contexts. The parameter inflation resulting from e.g. a hierarchical model, with subject-wise likelihoods, does not make the likelihood amortization any

Algorithm 11: NRE

Input: Initialize: $d_{\mathbf{w}}(\mathbf{s}, \theta)$;
Prior: $\tilde{\pi}_1(\theta) = \pi(\theta)$;
Simulator: $\mathcal{M}(\theta)$;
Cumulative Training Data Set: $\mathcal{D} = \{\}$;
Set: Batch size B

while *not converged* **do**
  for $n = 1..B$ **do**
    sample $\theta_b \sim \pi(\theta)$;
    sample $\theta_b^{'} \sim \pi(\theta)$;
    sample $\mathbf{x}_b \sim \mathcal{M}(\theta_b)$;
  end
  Train-step: $d_{\mathbf{w}}(\mathbf{s}|\theta)$ on *batch* with $\mathcal{L}^{NRE}(\mathbf{w})$;
end

Result: $d_{\mathbf{w}^*}(\mathbf{s}|\theta)$

more complicated, but instead outsources the high dimensional inference part to the MCMC sampler. Moreover, while the jury is still out (see Lueckmann, Boelts, et al., 2021), learning likelihood ratios as a classification problem instead of relying on density estimators could be beneficial for performance and training stability.

2.6 Exploiting LFI-NN in Cognitive Neuroscience

LFI methods are gaining traction in computational cognitive and neuroscience. The detailed overview of the early historical roots of LFI methods (traditional ABC), gave a thorough circumscription of the basic problems of Simulation Based Inference and early approaches and algorithms to attack them.

The key problems and trade-offs identified by traditional ABC methods remain challenges to drive the improvement of modern techniques and can be characterized as follows:

1. Increase efficiency of sampling mechanisms
2. Find optimal low dimensional summary statistics (approximate sufficient statistics)
3. Target the Likelihood or target the Posterior ?
4. Minimize the amount of simulations needed for one-off problems
5. Amortize simulations for re-usability

In the context of LFI-ABC we have seen how kernel smoothing and MCMC approaches improved the original ABC rejection sampler. Second, we have seen approaches to select among candidate summary statistics. While original LFI-ABC methods targeted the posterior directly, the synthetic likelihood approach developed by Wood, 2010 is an early indication of the use of targeting likelihoods

instead. LFI-ML approaches have done so via GPs. For LFI-ABC methods, I did not discuss the problem of simulation run minimization as distinct from point 1. I note however that it has been considered, via LFI-ML methods specifically using GPs, which allow for a straightforward mechanism of simulation parameter selection since GPs provide uncertainty estimates out of the box (Acerbi, 2020; Wilkinson, 2014; Meeds and Welling, 2014; Järvenpää, Michael U Gutmann, Vehtari, et al., 2021).

Lastly, LFI-ABC methods did traditionally not focus very much on the aspect of amortization. The storage of simulation data in lookup tables is a common technique to aide re-usability across datasets, however this methods suffers from various shortcomings. Reasonably large parameter spaces (say $|\theta| > 10$), for which we may want to store a table of summary statistics to parameter pairs will impose unreasonable storage requirements. Moreover, to remain feasible such storage demands a (rather coarse) notion of discretization of the parameter space, prohibiting sampling techniques such as MCMC from performing at full resolution, which in turn leads to problems with mixing. Incorporation of hierarchical inference setting, or rather general considerations of flexibility with respect to inference scenarios is not commonly discussed.

I then discussed a range of modern LFI-NN techniques which have several advantages. I showed how neural density estimators (e.g. Flows, MDNs), once trained on a particular problem, allow for fast posterior sampling. Moreover, the mentioned frameworks of SNPE, SNRE as well as the Bayesflow algorithm are all amenable to joint learning of low dimensional embedding (summary) networks, which allows us to use the power of emerging Neural Network architectures specifically designed for learning embeddings that consider the statistical and structural properties of application relevant datasets. However, just as with LFI-ABC approaches, summary statistics can also be derived from any other algorithm (e.g. Belghazi et al., 2018; Hjelm et al., 2018) and simply supplied as a predefined embedding to any of the aforementioned algorithms. I have shown how LFI-NN algorithms can be employed for both, the purpose of targeting the likelihood, as well as the purpose of targeting the posterior distribution directly. LFI-NN algorithms, specifically the sequential variants SNPE and SNRE, by design attempt to minimize the amount of simulations necessary to solve a given posterior approximation problem. Lastly, all of the LFI-NN algorithms discussed are directly applicable when global amortization of inference scenarios is the goal of deployment of an LFI technique.

The development of LFI-NN techniques is relatively young, gaining traction with the paper of Papamakarios and I. Murray, 2016, but draws on a solid historical arc of research on neural density estimation (Bishop, 1994; Uria, I. Murray, and Larochelle, 2013; Larochelle and I. Murray, 2011; Frey, Hinton, and Dayan, 1996; Germain et al., 2015; Raiko et al., 2014). LFI-NN benefits from the already significant but still developing interest of the Neural Network community in probabilistic approaches and uncertainty estimation, partly spurned by the advent of GANs (Goodfellow, Pouget-Abadie, et al., 2014). I emphasized the basic building blocks of LFI-NN, which form an exploitable interface to large research enterprises in the development to more and more powerful neural density estimators, ratio estimation techniques and more. Several working algorithms of great flexibility, with at best modest exaggeration spanning the whole of the problem-space previously considered by LFI-ABC

and LFI-ML methods, have shown its potential as a framework. Furthermore, there is a large current momentum towards the implementation of user friendly programming interfaces to many of the by now standard LFI-NN algorithms, which will lead to wider adoption and through the increased exposure to real world datasets, further refinement and innovation on the algorithm side. Recent papers, targeting the neuroscience and computational cognitive science communities, have already pointed out the benefits which may accrue from these developments.

In the remainder of this section I will further discuss these benefits. I will make explicit how the LFI-NN algorithms discussed in previous sections interface with a range of inference scenarios and point out which of the discussed algorithms may respectively be applied for best results.

We will maintain an emphasis on end-to-end algorithms, which where necessary include learned embedding (summary) networks. Providing a priori defined summary statistics is a special case.

The benefit of using such end-to-end algorithms stems from our ability to use a very broad range of Neural Network architectures to serve as embeddings, while deriving the training signal (gradient) strictly from the inference performance, which drives the training loss functions across our LFI-NN methods. This is important since generative (mechanistic) models in neuroscience and computational cognitive science produce a large variety of structural data types ranging from simple single i.i.d vectors to complex time series data, across a range of modalities from reaction times and choices over neural spiking patterns to time-courses of spatial attention allocation (Jang, Sharma, and Drugowitsch, 2020; Frank, Gagne, et al., 2015; K.-F. Wong and X.-J. Wang, 2006). As mentioned in the section on summary networks, suitable Neural Network architectures to produce useful embeddings for each of these classes of datastructures are easily found in the Neural Network literature, hence no fundamental roadblock persists for a wide range of proof of concept applications (S. T. Radev, Mertens, et al., 2020). Successful training of a resulting end-to-end pipeline may however strongly depend on the respective hyperparameters.

As hopefully convincingly argued, apart from a broad restriction on the number of underlying parameters of a mechanistic model (current algorithms may not work well for models that with $|\Theta| > 15$), hardly any limitations on the use of LFI-NN algorithms remain.
Matching the goals of a project with the right algorithm however remains an important step for the effective application of LFI-NN. Following a few general remarks, I illustrate this with our SSMs at hand section 2, to make precise some of the considerations that may arise.

We can draw a substantial line between the use of LFI-NN for local (for a specific dataset) or global purposes (amortization across a generous range of the parameter space). Specializing our algorithms for a specific dataset may be useful under the following circumstances. First, we may have a simulator which makes it prohibitively expensive (given a computing infrastructure) to attempt amortization across a large parameter space. Second, we are interested in analysing our data with a one-off exploratory model which may not have a high a priori expectation for reuse.

If computational limits allow, the global approach may be desirable however. First, once a likelihood or posterior is amortized globally for a given mechanistic model, one is able to re-use it across a wide range of future datasets and importantly, distribute the resulting network for use

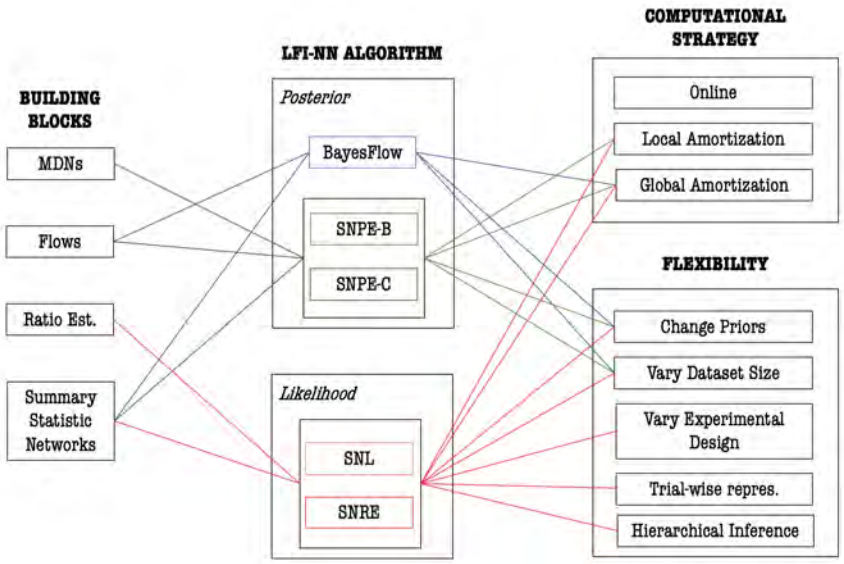

Figure 2.14. This figure summarizes aspects of the discussion in section 3.6. It shows the links between the building blocks (*left*) discussed in sections 2.5.1, 2.5.2, and 2.5.3, and the LFI-NN algorithms discussed in sections 2.5.4 and 2.5.4 (*middle*). The algorithms are linked with the computational strategies discussed in 2.3, and aspects of deployment flexibility (*right*).

by other researchers who may be interested in testing their data against such model. This has the immense benefit of systematically accruing use for the entire research community by amortizing a model. Combined with the potential for downstream easy to use software packages that make such models, including surrounding inference infrastructure, available to the community, this holds the seeds of a revolutionary change in how resources for computational modeling may be shared across the discipline. Our work on likelihood approximation networks (LANs) serves as one proof of concept realization of this vision (Fengler et al., 2020), however novel toolboxes for Simulation Based Inference (e.g. Tejero-Cantero et al., 2020) have equal promise in being utilized for such purposes. Second, apart from re-usability, global amortization is useful for the downstream analysis of model properties. When performing inference on synthetic datasets generated from across the parameter space, do parameter trade-offs follow a consistent pattern, or can we partition the generative parameter space according to well identifiable classes of pattern in the resulting posteriors? Are parameters consistently identifiable / non-identifiable across the parameter space? Such analyses are crucial especially when proposing new models to the community. Since I consider the benefits of the global approach to be dominant, especially the potential benefits to the community, I generally recommend to attempt global amortization whenever possible. LFI-NN however, as shown remains a very useful approach for local posterior inference as well.

The second substantial dividing line separates model classes along their utility across separate inference scenarios. Is it sensible to envision a given simulation model as being applied on experimental data stemming from a variety of complex experimental designs? Could it be used for hierarchical inference, varying the number of subject, and or the dataset-sizes by subject?

If the answer to these questions is no, the benefits of targeting posterior distributions directly may outweigh those that may result from targeting the likelihood. A primary purpose of an analysis could lie for example in investigating the properties of a given model, such a parameter identifiability. In these cases case amortizing the posterior may be preferred, since sampling from such amortized posteriors can be significantly faster than sampling via the likelihood. In the case of the BayesFlow algorithm, we simply draw samples from a conditional invertible flow, an operation that can provide 1000s of posterior sampling within seconds. Up to defective cases resulting from posterior leakage outside of the allowed prior range, SNPE-C behaves similarly. SNL and SNRE however may use MCMC algorithms to generate posterior samples. The notion of sampling directly from an amortized flow does not apply here. The ratio or density estimator are primarily used for evaluation of the density / ratio itself.

Furthermore, some models seem to make likelihood amortization inherently hard as compared to targeting the posterior. The converse is true for other models (see Papamakarios, Sterratt, and I. Murray, 2019 for examples of both). This trade-off may predetermine the focus by virtue of leaving us only a single valid choice. If likelihood amortization is accessible however and one can answer the above questions in the affirmative, the resulting added flexibility provides immense benefits.

Consider for example the Levy Flight model, or a DDM with collapsing bounds (see Figure 2.5). These model are suitable for the joint analysis of reaction time and choice data across a large variety of experimental designs for which simpler versions, such as the basic DDM, have often been used out of convenience rather than strong de facto theoretical commitments (Frank, Gagne, et al., 2015; Cavanagh, Wiecki, et al., 2014; Cavanagh and Frank, 2014; Mads Lund Pedersen, Frank, and Biele, 2017). The dissemination of SSMs as a useful framework of analysis of such large varieties of experimental data has in turn produced the necessity of expanding the flexibility of connected inference algorithms. Examples of such demands are the ability to include latent covariate processes that may drive the model parameters on a trial by trial basis. Examples include linking EEG or fMRI signals as conditional or trial level covariates via a regression model (Frank, Gagne, et al., 2015; Cavanagh and Frank, 2014). Another established example consists of the nesting of a reinforcement learning process to the model parameters of a DDM. These connections are just as valid for any of the models listed in Figure 2.5. Standard inference procedures may allow a number of parameters to vary across conditions, while theoretical commitments are made to fix other parameters which a priori are judged to have no conceptual link to a given condition manipulation. Experimental datasets may include any number of subjects, subject level datasets, highly variable numbers of trials etc. Posterior amortization algorithms not only make many of these inference scenarios impossible to amortize (e.g. trial by trial covariate processes). Scenarios for which we may successfully apply posterior amortization (e.g. condition level effects) result in a highly specialized posterior samplers which demand retraining for any slight variation in setup. This in turn defeats the ability for amortization to lead to effective democratization of once incurred computational cost.

The polar opposite, is represented by learning trial wise likelihoods (suitable algorithms may be SNRE or SNL), an approach emphasized in the authors prior work (Fengler et al., 2020). Once we

have access to amortized trial wise likelihoods, any of the above mentioned inference scenarios are immediately accessible. One may describe this approach as *maximal modularization*. I argue that where possible, this is the amortization strategy to apply, since it simultaneously maximizes the potential for democratization mentioned above. However, even intermediate levels of modularization will reap benefits as to flexibility. As an example, we may want to generate subject level (condition level, group level) likelihoods by virtue of data-set summary statistics. We have seen how the summary statistic networks can by used to learn dataset embeddings (exploiting inherent structure such as exchangeability etc.). One may use this approach for any of the SSMs listed in Figure 2.5, to learn the likelihood function over summary statistics of single datasets (with varying size). This may not always represent an easier problem than learning trial wise likelihoods, however summary statistics can be better behaved. Once such subject level likelihoods are amortized, there may still be flexibility with respect to experimental design considerations, which do not rely on trial by trial level manipulations.

Lastly, focusing instead on amortization of a likelihood function in general decouples the amortization problem from other parts of an inference pipeline (e.g. MCMC samplers). Downstream one may interface with the whole ecosystem of posterior sampling algorithms. This adds significant degrees of freedom for end-to-end solutions. Figure 2.14 summarizes the main points in the preceding discussion in a simple graphic. It shows the dependence of specific LFI-NN algorithms on basic building blocks, connects the algorithms with the computational strategies discussed in section 2.3 and emphasizes the degree of flexibility they provide downstream.

2.7 The software Landscape

The software landscape for ABC is steadily growing and in the foreseeable future I expect this trend to continue, given that research in the area is multiplying.

This section is not meant to provide an exhaustive list of available software, rather I omit highly specialized packages that implement primarily a single publication's methods and focus instead on the already popular general purpose packages, as well as emerging software with a certain ambition towards generality.

One can roughly divide the ABC software terrain into three categories,

1. General purpose LFI-ABC packages with focus on variants of traditional ABC methods
2. Software that focuses on LFI-NN methods
3. Other software that may be co-opted for ABC computations

In the first category, designed for the R programming language there are the ABCtoolbox (Wegmann et al., 2010), EasyABC (Jabot, Faure, and Dumoulin, 2013) and abc (Csilléry, François, and Blum, 2012) packages. These implement a variety of traditional ABC samplers and especially the abc package provides extra routines for simple ABC model selection including cross-validation

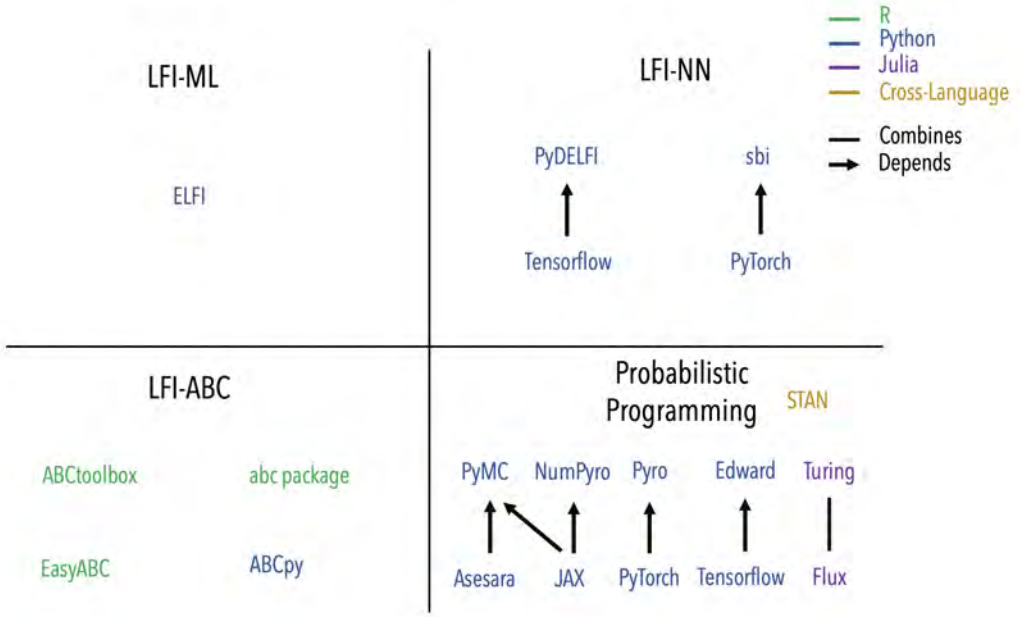

Figure 2.15. Figure illustrates the software landscape as described in section 2.7. I emphasize that this is *not* an *exhaustive* enumeration of all related software in existence.

for trained model selection schemes. A counterpart to these R packages for the Python programming language is the ABCpy package which offers quite a comprehensive combination of a variety of traditional ABC samplers and includes an interface to Pytorch (Paszke et al., 2019) for Neural Network based summary statistic learning. Another notable library for python is the Engine for Likelihood free Inference (ELFI) (Lintusaari et al., 2018), which has a focus on the BOLFI (Bayesian Optimization for Likelihood Free Inference) (Michael U Gutmann and Corander, 2016) algorithm and therefore is a good choice when working with very expensive simulators for which minimization of the number of simulation runs is of highest priority.

LFI-NN methods, including pipelines for joint learning of summary statistics together with posterior distributions and moreover global amortization of posteriors and likelihoods via neural density estimators, are implemented in the pyDELFI (Tensorflow) (Alsing et al., 2019) and sbi (Pytorch) (Tejero-Cantero et al., 2020) packages for the Python programming language. The main focus of these packages is to provide a user friendly interface to variatiations of the SNPE / SNLE / SNRE paradigm. The ABCpy, pyDELFI and sbi libraries are under active development and will likely expand in capability in the recent future.

We now consider the third category of software packages. First, options for basic Simulation Based Inference are included in the pytorch based probabilist programming language Pyro (Bingham et al., 2019), as well as in the popular PYMC3 library for probabilistic modeling in Python (Salvatier, Wiecki, and Fonnesbeck, 2016). Second, when imposing a restriction to *differentiable simulators* one can utilize the probabilistic inference engines emerging directly as part of the PyTorch (distributions)

and Tensorflow (Tensorflow probability). The rapid development of the PyTorch as well as Tensorflow libraries suggest further extensions in their respective capabilities in quick succession, and render a detailed taking stock of their current state somewhat redundant. It can however be projected that the opportunities to co-opt PyTorch and Tensorflow for Likelihood Free Inference (LFI) will continue to increase over time, especially with the emergence of neural density estimation on the center state of Approximate Bayesian Computation techniques. Last but not least, the STAN platform for statistical computing (Carpenter et al., 2017) is highlighted. STAN is an established as a major player in the probabilistic computing world, however has explicit focus on Hamiltonian Monte Carlo (HMC) (Hoffman and Gelman, 2014), and Variational Inference algorithms (Hoffman, D. M. Blei, et al., 2013). An interface to STAN is offered for R, Python, Julia and other programming languages. As of now other approaches to ABC are not yet explicitly supported in STAN, this may however happen in the future.

2.8 Conclusion and Limitations

The goal of this review is to serve as a detailed introduction to likelihood free methods, from basic conceptual ideas to the modern state of the art, with an emphasis on their utility for the computational cognitive- and neuroscience communities. To serve as an entry-point into the LFI literature, I discussed in detail several relevant algorithms spanning more than two decades of research on LFI methods preceded by a general overview of the fields vast amount of techniques. The arguments were constructed to aid understanding by developing basic conceptual building blocks along the way. I generally built up to complex methods from simpler ones, often aided by the analogous chronology of original development of the techniques. These developments were woven together in a discussion on the benefits of LFI-NN methods over traditional LFI-ABC with emphasis on the relative strengths and weaknesses of state of the art LFI-NN procedures. I stressed the benefits of LFI-NN techniques which focus on likelihood amortization, and provided some guidelines on how to choose a method for a given problem. To assist the reader, the discussion referred to the class of SSMs as a running example. I ended with a short survey of currently available software packages for LFI inference. This review adds to other similar reviews which recently appeared in the literature. Cranmer, Brehmer, and Louppe, 2020, give a useful high level overview of current approaches to Simulation Based Inference. The authors survey what may be considered design patterns for LFI algorithms, and broadly discuss strengths, weaknesses and structural limitations of these patterns respectively. Their paper however lacks discussion of any methods in sufficient detail to lay bare the basic technical building blocks for an uninitiated reader. Papamakarios, Nalisnick, et al., 2019, review flow based Neural Networks in great detail, but do not primarily focus their discussion on LFI. Palestro, Sederberg, et al., 2018 discuss the power of LFI, but neither consider amortization strategies, nor consider LFI-NN methods, implicitly ignoring this rising trend. S. T. Radev, A. Voss, et al., 2020 provide a high level cognitive science focused, forward looking account of LFI-NN algorithms for amortization. However the authors focus on the amortization of posteriors and give neither a detailed

nor sufficiently broad account of the LFI-NN methods available today. None of the aforementioned reviews consider a detailed account of the emerging marriage between global amortization strategies and complex experimental designs, a main emphasis and point of focus of this review.

Last, to aid the interested reader I make explicit how I narrowed the scope of the discussion to achieve the goals of this chapter. To maintain focus on Neural Network based techniques, I de-emphasized the parallel and ongoing development of LFI-ML algorithms. Due to their potential for amortization techniques, I believe LFI-NN will develop to serve as the dominant framework, however specific problems may reasonably call for a particular LFI-ML methods instead. I consider aiding such a choice outside the scope of this review. Further, I do not engage in any discussion of problems concerning Bayesian inference apart from likelihood and posterior approximation and sampling methods as they pertain to the respective approximation algorithms. Such related problems include calibration (Talts et al., 2018) of the inference procedure, evidence calculation (Sisson, Fan, and M. Beaumont, 2018; Didelot et al., 2011; Wieschen, A. Voss, and S. Radev, 2020), and model comparison in general. These form an integral part of a fully developed end-to-end approach to likelihood inference, and play an important role in day to day computational science. A sufficiently detailed discussion however would have significantly expanded the scope of the discussion.

References

Acerbi, Luigi (2019). "An exploration of acquisition and mean functions in Variational Bayesian Monte Carlo". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 1–10.

— (2020). "Variational Bayesian Monte Carlo with Noisy Likelihoods". In: *arXiv preprint arXiv:2006.08655*.

Aldous, David J (1985). "Exchangeability and related topics". In: *École d'Été de Probabilités de Saint-Flour XIII—1983*. Springer, pp. 1–198.

Alsing, Justin et al. (2019). "Fast likelihood-free cosmology with neural density estimators and active learning". In: *Monthly Notices of the Royal Astronomical Society* 488.3, pp. 4440–4458.

Badre, David et al. (2012). "Rostrolateral prefrontal cortex and individual differences in uncertainty-driven exploration". In: *Neuron* 73.3, pp. 595–607.

Bakkour, Akram et al. (2018). "Value-based decisions involve sequential sampling from memory". In: *BioRxiv*, p. 269290.

Balci, Fuat et al. (2011). "Acquisition of decision making criteria: reward rate ultimately beats accuracy". In: *Attention, Perception, & Psychophysics* 73.2, pp. 640–657.

Barthelmé, Simon and Nicolas Chopin (2011). "ABC-EP: expectation propagation for likelihoodfree Bayesian computation". In: *ICML*.

Battiti, Roberto (1994). "Using mutual information for selecting features in supervised neural net learning". In: *IEEE Transactions on neural networks* 5.4, pp. 537–550.

Beaumont, Mark A (2010). "Approximate Bayesian computation in evolution and ecology". In: *Annual review of ecology, evolution, and systematics* 41, pp. 379–406.

Beaumont, Mark A, Wenyang Zhang, and David J Balding (2002). "Approximate Bayesian computation in population genetics". In: *Genetics* 162.4, pp. 2025–2035.

Belghazi, Mohamed Ishmael et al. (2018). "Mine: mutual information neural estimation". In: *arXiv preprint arXiv:1801.04062*.

Berg, Rianne van den et al. (2018). "Sylvester normalizing flows for variational inference". In: *arXiv preprint arXiv:1803.05649*.

Bhattacharyya, Apratim et al. (2020). "Haar Wavelet based Block Autoregressive Flows for Trajectories". In: *arXiv preprint arXiv:2009.09878*.

Bingham, Eli et al. (2019). "Pyro: Deep universal probabilistic programming". In: *The Journal of Machine Learning Research* 20.1, pp. 973–978.

Bishop, Christopher M (1994). *Mixture density networks*. Tech. rep. Microsoft Research.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). "Variational inference: A review for statisticians". In: *Journal of the American statistical Association* 112.518, pp. 859–877.

Bloem-Reddy, Benjamin and Yee Whye Teh (2020). "Probabilistic symmetries and invariant neural networks". In: *Journal of Machine Learning Research* 21.90, pp. 1–61.

Blum, Michael GB et al. (2013). "A comparative review of dimension reduction methods in approximate Bayesian computation". In: *Statistical Science* 28.2, pp. 189–208.

Bottou, Léon (2012). "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, pp. 421–436.

Brehmer, Johann et al. (2020). “Mining gold from implicit models to improve likelihood-free inference”. In: *Proceedings of the National Academy of Sciences* 117.10, pp. 5242–5249.

Brown, Scott D and Andrew Heathcote (2008). “The simplest complete model of choice response time: Linear ballistic accumulation”. In: *Cognitive psychology* 57.3, pp. 153–178.

Busemeyer, Jerome R and James T Townsend (1993). “Decision field theory: a dynamic-cognitive approach to decision making in an uncertain environment.” In: *Psychological review* 100.3, p. 432.

Cappé, Olivier et al. (2004). “Population monte carlo”. In: *Journal of Computational and Graphical Statistics* 13.4, pp. 907–929.

Carpenter, Bob et al. (2017). “Stan: a probabilistic programming language.” In: *Grantee Submission* 76.1, pp. 1–32.

Cavanagh, James F and Michael J Frank (2014). “Frontal theta as a mechanism for cognitive control”. In: *Trends in cognitive sciences* 18.8, pp. 414–421.

Cavanagh, James F, Thomas V Wiecki, et al. (2014). “Eye tracking and pupillometry are indicators of dissociable latent decision processes.” In: *Journal of Experimental Psychology: General* 143.4, p. 1476.

Chan, Chung et al. (2019). “Neural entropic estimation: A faster path to mutual information estimation”. In: *arXiv preprint arXiv:1905.12957*.

Cipra, Barry A (1987). “An introduction to the Ising model”. In: *The American Mathematical Monthly* 94.10, pp. 937–959.

Cisek, Paul, Geneviève Aude Puskas, and Stephany El-Murr (2009). “Decisions in changing conditions: the urgency-gating model”. In: *Journal of Neuroscience* 29.37, pp. 11560–11571.

Collins, Anne GE and Michael J Frank (2014). “Opponent actor learning (OpAL): modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive.” In: *Psychological review* 121.3, p. 337.

Cranmer, Kyle, Johann Brehmer, and Gilles Louppe (2020). “The frontier of simulation-based inference”. In: *Proceedings of the National Academy of Sciences* 117.48, pp. 30055–30062.

Cranmer, Kyle, Juan Pavez, and Gilles Louppe (2015). “Approximating likelihood ratios with calibrated discriminative classifiers”. In: *arXiv preprint arXiv:1506.02169*.

Csilléry, Katalin, Olivier François, and Michael GB Blum (2012). “abc: an R package for approximate Bayesian computation (ABC)”. In: *Methods in ecology and evolution* 3.3, pp. 475–479.

Cybenko, George (1989). “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

Darmois, Georges (1935). “Sur les lois de probabilitéa estimation exhaustive”. In: *CR Acad. Sci. Paris* 260.1265, p. 85.

Dayan, Peter and Yael Niv (2008). “Reinforcement learning: the good, the bad and the ugly”. In: *Current opinion in neurobiology* 18.2, pp. 185–196.

Didelot, Xavier et al. (2011). “Likelihood-free estimation of model evidence”. In: *Bayesian analysis* 6.1, pp. 49–76.

Dinev, Traiko and Michael U Gutmann (2018). "Dynamic likelihood-free inference via ratio estimation (DIRE)". In: *arXiv preprint arXiv:1810.09899*.

Dinh, Laurent, David Krueger, and Yoshua Bengio (2014). "Nice: Non-linear independent components estimation". In: *arXiv preprint arXiv:1410.8516*.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2016). "Density estimation using real nvp". In: *arXiv preprint arXiv:1605.08803*.

Dragalin, Vladimir P, Alexander G Tartakovsky, and Venugopal V Veeravalli (2000). "Multihypothesis sequential probability ratio tests. II. Accurate asymptotic expansions for the expected sample size". In: *IEEE Transactions on Information Theory* 46.4, pp. 1366–1383.

Draglia, VP, Alexander G Tartakovsky, and Venugopal V Veeravalli (1999). "Multihypothesis sequential probability ratio tests. I. Asymptotic optimality". In: *IEEE Transactions on Information Theory* 45.7, pp. 2448–2461.

Dupont, Emilien, Arnaud Doucet, and Yee Whye Teh (2019). "Augmented neural odes". In: *Advances in Neural Information Processing Systems*, pp. 3140–3150.

Durkan, Conor, Artur Bekasov, et al. (2019). "Neural spline flows". In: *Advances in Neural Information Processing Systems*, pp. 7511–7522.

Durkan, Conor, Iain Murray, and George Papamakarios (2020). *On Contrastive Learning for Likelihood-free Inference*. arXiv: `2002.03712 [stat.ML]`.

Evans, Nathan J, Jennifer S Trueblood, and William R Holmes (2020). "A parameter recovery assessment of time-variant models of decision-making". In: *Behavior research methods* 52.1, pp. 193–206.

Fengler, Alexander et al. (2020). "Likelihood Approximation Networks (LANs) for Fast Inference of Simulation Models in Cognitive Neuroscience". In: *bioRxiv*. DOI: `10.1101/2020.11.20.392274`. eprint: `https://www.biorxiv.org/content/early/2020/11/22/2020.11.20.392274.full.pdf`. URL: `https://www.biorxiv.org/content/early/2020/11/22/2020.11.20.392274`.

First, Michael B (2013). *DSM-5 handbook of differential diagnosis*. American Psychiatric Pub.

Fisher, Geoffrey (2017). "An attentional drift diffusion model over binary-attribute choice". In: *Cognition* 168, pp. 34–45.

Fontanesi, Laura, Sebastian Gluth, et al. (2019). "A reinforcement learning diffusion decision model for value-based decisions". In: *Psychonomic bulletin & review* 26.4, pp. 1099–1121.

Fontanesi, Laura, Stefano Palminteri, and Maël Lebreton (2019). "Decomposing the effects of context valence and feedback information on speed and accuracy during reinforcement learning: a meta-analytical approach using diffusion decision modeling". In: *Cognitive, Affective, & Behavioral Neuroscience* 19.3, pp. 490–502.

Foster, Kendal and Henrik Singmann (2021). *Another Approximation of the First-Passage Time Densities for the Ratcliff Diffusion Decision Model*. arXiv: `2104.01902 [stat.AP]`.

Frank, Michael J, Bradley B Doll, et al. (2009). "Prefrontal and striatal dopaminergic genes predict individual differences in exploration and exploitation". In: *Nature neuroscience* 12.8, p. 1062.

Frank, Michael J, Chris Gagne, et al. (2015). "fMRI and EEG predictors of dynamic decision parameters during human reinforcement learning". In: *Journal of Neuroscience* 35.2, pp. 485–494.

Frank, Michael J, Lauren C Seeberger, and Randall C O'reilly (2004). "By carrot or by stick: cognitive reinforcement learning in parkinsonism". In: *Science* 306.5703, pp. 1940–1943.

Frey, Brendan J, Geoffrey E Hinton, and Peter Dayan (1996). "Does the wake-sleep algorithm produce good density estimators?" In: *Advances in neural information processing systems*, pp. 661–667.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2001). *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York.

Fu, Michael C (2008). "What you should know about simulation and derivatives". In: *Naval Research Logistics (NRL)* 55.8, pp. 723–736.

Fu, Michael C and Jian-Qiang Hu (1995). "Sensitivity analysis for Monte Carlo simulation of option pricing". In.

Geana, Andra et al. (2022). "Using Computational Modeling to Capture Schizophrenia-Specific Reinforcement Learning Differences and Their Implications on Patient Classification". In: *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging* 7.10, pp. 1035–1046.

Germain, Mathieu et al. (2015). "Made: Masked autoencoder for distribution estimation". In: *International Conference on Machine Learning*, pp. 881–889.

Gläscher, Jan et al. (2010). "States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning". In: *Neuron* 66.4, pp. 585–595.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*.

Goodfellow, Ian, Jean Pouget-Abadie, et al. (2014). "Generative adversarial nets". In: *Advances in neural information processing systems* 27, pp. 2672–2680.

Greenberg, David, Marcel Nonnenmacher, and Jakob Macke (2019). "Automatic posterior transformation for likelihood-free inference". In: *International Conference on Machine Learning*. PMLR, pp. 2404–2414.

Gutmann, Michael U and Jukka Corander (2016). "Bayesian optimization for likelihood-free inference of simulator-based statistical models". In: *The Journal of Machine Learning Research* 17.1, pp. 4256–4302.

Gutmann, Michael U, Ritabrata Dutta, et al. (2018). "Likelihood-free inference via classification". In: *Statistics and Computing* 28.2, pp. 411–425.

Hastings, W Keith (1970). "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57 (1).

Hermans, Joeri, Volodimir Begy, and Gilles Louppe (2020). "Likelihood-free mcmc with amortized approximate ratio estimators". In: *International Conference on Machine Learning*. PMLR, pp. 4239–4248.

Hjelm, R Devon et al. (2018). "Learning deep representations by mutual information estimation and maximization". In: *arXiv preprint arXiv:1808.06670*.

Hjorth, Lars U and Ian T Nabney (1999). "Regularisation of mixture density networks". In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99.(Conf. Publ. No. 470)*. Vol. 2. IET, pp. 521–526.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hoffman, Matthew D, David M Blei, et al. (2013). "Stochastic variational inference". In: *The Journal of Machine Learning Research* 14.1, pp. 1303–1347.

Hoffman, Matthew D and Andrew Gelman (2014). "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623.

Holmes, William R, Jennifer S Trueblood, and Andrew Heathcote (2016). "A new framework for modeling decisions about changing information: The Piecewise Linear Ballistic Accumulator model". In: *Cognitive psychology* 85, pp. 1–29.

Hoogeboom, Emiel et al. (2020). "The Convolution Exponential and Generalized Sylvester Flows". In: *arXiv preprint arXiv:2006.01910*.

Hornik, Kurt, Maxwell Stinchcombe, Halbert White, et al. (1989). "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5, pp. 359–366.

Huang, Chin-Wei, Laurent Dinh, and Aaron Courville (2020a). "Augmented normalizing flows: Bridging the gap between generative flows and latent variable models". In: *arXiv preprint arXiv:2002.07101*.

— (2020b). "Solving ode with universal flows: Approximation theory for flow-based models". In: *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

Huang, Chin-Wei, David Krueger, et al. (2018). "Neural autoregressive flows". In: *arXiv preprint arXiv:1804.00779*.

Huys, Quentin JM, Tiago V Maia, and Michael J Frank (2016). "Computational psychiatry as a bridge from neuroscience to clinical applications". In: *Nature neuroscience* 19.3, p. 404.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR, pp. 448–456.

Ishida, EEO et al. (2015). "Cosmoabc: likelihood-free inference via population Monte Carlo approximate Bayesian computation". In: *Astronomy and Computing* 13, pp. 1–11.

Jabot, Franck, Thierry Faure, and Nicolas Dumoulin (2013). "Easy ABC: performing efficient approximate B ayesian computation sampling schemes using R". In: *Methods in Ecology and Evolution* 4.7, pp. 684–687.

Jang, Anthony I, Ravi Sharma, and Jan Drugowitsch (2020). "Optimal policy for attention-modulated decisions explains human fixation behavior". In: *bioRxiv*.

Järvenpää, Marko, Michael U Gutmann, Arijus Pleska, et al. (2019). "Efficient acquisition rules for model-based approximate Bayesian computation". In: *Bayesian Analysis* 14.2, pp. 595–622.

Järvenpää, Marko, Michael U Gutmann, Aki Vehtari, et al. (2021). "Parallel Gaussian process surrogate Bayesian inference with noisy likelihood evaluations". In: *Bayesian Analysis* 16.1, pp. 147–178.

Jiang, Bai et al. (2017). "Learning summary statistic for approximate Bayesian computation via deep neural network". In: *Statistica Sinica*, pp. 1595–1618.

Jordan, Michael I. et al. (1999). "An Introduction to Variational Methods for Graphical Models". In: *Journal of Machine Learning Research* 1.37, pp. 183–233.

Kadirvelu, Balasundaram, Yoshikatsu Hayashi, and Slawomir J Nasuto (2017). "Inferring structural connectivity using Ising couplings in models of neuronal networks". In: *Scientific reports* 7.1, pp. 1–12.

Kallenberg, Olav (2006). *Probabilistic symmetries and invariance principles*. Springer Science & Business Media.

Kass, Robert E and Adrian E Raftery (1995). "Bayes factors". In: *Journal of the american statistical association* 90.430, pp. 773–795.

Kayser, Andrew S et al. (2015). "Dopamine, locus of control, and the exploration-exploitation tradeoff". In: *Neuropsychopharmacology* 40.2, pp. 454–462.

Khemakhem, Ilyes et al. (2020). "Causal Autoregressive Flows". In: *arXiv preprint arXiv:2011.02268*.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kingma, Diederik P, Tim Salimans, and Max Welling (2015). "Variational dropout and the local reparameterization trick". In: *arXiv preprint arXiv:1506.02557*.

Kingma, Durk P et al. (2016). "Improved variational inference with inverse autoregressive flow". In: *Advances in neural information processing systems* 29, pp. 4743–4751.

Kobyzev, Ivan, Simon Prince, and Marcus Brubaker (2020). "Normalizing flows: An introduction and review of current methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Koopman, Bernard Osgood (1936). "On distributions admitting a sufficient statistic". In: *Transactions of the American Mathematical society* 39.3, pp. 399–409.

Krajbich, Ian, Carrie Armel, and Antonio Rangel (2010). "Visual fixations and the computation and comparison of value in simple choice". In: *Nature neuroscience* 13.10, pp. 1292–1298.

Krajbich, Ian, Dingchao Lu, et al. (2012). "The attentional drift-diffusion model extends to simple purchasing decisions". In: *Frontiers in psychology* 3, p. 193.

Kruse, Jakob (2020). "Technical report: Training Mixture Density Networks with full covariance matrices". In: *arXiv preprint arXiv:2003.05739*.

Larochelle, Hugo and Iain Murray (2011). "The neural autoregressive distribution estimator". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 29–37.

LeCun, Yann et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

Lee, Juho et al. (2019). "Set transformer: A framework for attention-based permutation-invariant neural networks". In: *International Conference on Machine Learning*. PMLR, pp. 3744–3753.

Lieder, Falk and Thomas L Griffiths (2020). "Resource-rational analysis: understanding human cognition as the optimal use of limited computational resources". In: *Behavioral and Brain Sciences* 43.

Lintusaari, Jarno et al. (2018). "Elfi: Engine for likelihood-free inference". In: *The Journal of Machine Learning Research* 19.1, pp. 643–649.

Liu, Jenny et al. (2019). "Graph normalizing flows". In: *Advances in Neural Information Processing Systems*, pp. 13578–13588.

Liu, Jun S, Rong Chen, and Tanya Logvinenko (2001). "A theoretical framework for sequential importance sampling with resampling". In: *Sequential Monte Carlo methods in practice*. Springer, pp. 225–246.

Liu, Jun S, Rong Chen, and Wing Hung Wong (1998). "Rejection control and sequential importance sampling". In: *Journal of the American Statistical Association* 93.443, pp. 1022–1031.

Lueckmann, Jan-Matthis, Giacomo Bassetto, et al. (2019). "Likelihood-free inference with emulator networks". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 32–53.

Lueckmann, Jan-Matthis, Jan Boelts, et al. (2021). "Benchmarking Simulation-Based Inference". In: *arXiv preprint arXiv:2101.04653*.

Lueckmann, Jan-Matthis, Pedro J Goncalves, et al. (2017). "Flexible statistical inference for mechanistic models of neural dynamics". In: *arXiv preprint arXiv:1711.01861*.

Maia, Tiago V and Michael J Frank (2011). "From reinforcement learning models to psychiatric and neurological disorders". In: *Nature neuroscience* 14.2, pp. 154–162.

Makansi, Osama et al. (2019). "Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7144–7153.

Marino, Joseph et al. (2020). "Improving sequential latent variable models with autoregressive flows". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 1–16.

Marjoram, Paul et al. (2003). "Markov chain Monte Carlo without likelihoods". In: *Proceedings of the National Academy of Sciences* 100.26, pp. 15324–15328.

McCoy, Brónagh et al. (2019). "Dopaminergic medication reduces striatal sensitivity to negative outcomes in Parkinson's disease". In: *Brain* 142.11, pp. 3605–3620.

Meeds, Edward, Robert Leenders, and Max Welling (2015). "Hamiltonian abc". In: *arXiv preprint arXiv:1503.01916*.

Meeds, Edward and Max Welling (2014). "GPS-ABC: Gaussian process surrogate approximate Bayesian computation". In: *arXiv preprint arXiv:1401.2838*.

Metropolis, Nicholas et al. (1953). "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6, pp. 1087–1092.

Miletić, Steven, Russell J Boag, and Birte U Forstmann (2020). "Mutual benefits: Combining reinforcement learning with sequential sampling models". In: *Neuropsychologia* 136, p. 107261.

Milosavljevic, Milica et al. (2010). "The drift diffusion model can account for value-based choice response times under high and low time pressure". In: *Judgment and Decision Making* 5.6, pp. 437–449.

Minka, Thomas P (2013). "Expectation propagation for approximate Bayesian inference". In: *arXiv preprint arXiv:1301.2294*.

Minka, Tom et al. (2005). *Divergence measures and message passing*. Tech. rep. Citeseer.

Mohamed, Shakir and Balaji Lakshminarayanan (2016). "Learning in implicit generative models". In: *arXiv preprint arXiv:1610.03483*.

Navarro, Daniel J and Ian G Fuss (2009). "Fast and accurate calculations for first-passage times in Wiener diffusion models". In: *Journal of mathematical psychology* 53.4, pp. 222–230.

Neal, Radford M (2001). "Annealed importance sampling". In: *Statistics and computing* 11.2, pp. 125–139.

— (2012). *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media.

Nielsen, Didrik et al. (2020). "Survae flows: Surjections to bridge the gap between vaes and flows". In: *Advances in Neural Information Processing Systems* 33.

O'Connell, Redmond G et al. (2018). "Bridging neural and computational viewpoints on perceptual decision-making". In: *Trends in neurosciences* 41.11, pp. 838–852.

Palestro, James J, Per B Sederberg, et al. (2018). *Likelihood-free methods for cognitive science*. Springer.

Palestro, James J, Emily Weichart, et al. (2018). "Some task demands induce collapsing bounds: Evidence from a behavioral analysis". In: *Psychonomic bulletin & review* 25.4, pp. 1225–1248.

Paninski, Liam (2003). "Estimation of entropy and mutual information". In: *Neural computation* 15.6, pp. 1191–1253.

Papamakarios, George and Iain Murray (2016). "Fast ε-free Inference of Simulation Models with Bayesian Conditional Density Estimation". In: *Advances in Neural Information Processing Systems* 29, pp. 1028–1036.

Papamakarios, George, Eric Nalisnick, et al. (2019). "Normalizing flows for probabilistic modeling and inference". In: *arXiv preprint arXiv:1912.02762*.

Papamakarios, George, Theo Pavlakou, and Iain Murray (2017). "Masked autoregressive flow for density estimation". In: *Advances in Neural Information Processing Systems*, pp. 2338–2347.

Papamakarios, George, David Sterratt, and Iain Murray (2019). "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 837–848.

Paszke, Adam et al. (2019). "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems*, pp. 8026–8037.

Pedersen, Mads L et al. (2021). "Computational phenotyping of brain-behavior dynamics underlying approach-avoidance conflict in major depressive disorder". In: *PLoS computational biology* 17.5, e1008955.

Pedersen, Mads Lund, Michael J Frank, and Guido Biele (2017). "The drift diffusion model as the choice rule in reinforcement learning". In: *Psychonomic bulletin & review* 24.4, pp. 1234–1251.

Pessiglione, Mathias et al. (2006). "Dopamine-dependent prediction errors underpin reward-seeking behaviour in humans". In: *Nature* 442.7106, pp. 1042–1045.

Pitman, Edwin James George (1936). "Sufficient statistics and intrinsic accuracy". In: vol. 32. Cambridge University Press, pp. 567–579.

Poggio, Tomaso, Steve Smale, et al. (2003). "The mathematics of learning: Dealing with data". In: *Notices of the AMS* 50.5, pp. 537–544.

Pritchard, Jonathan K et al. (1999). "Population growth of human Y chromosomes: a study of Y chromosome microsatellites." In: *Molecular biology and evolution* 16.12, pp. 1791–1798.

Qi, Charles R et al. (2017). "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660.

Qi, Charles Ruizhongtai et al. (2017). "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems*, pp. 5099–5108.

Radev, Stefan T, Ulf K Mertens, et al. (2020). "BayesFlow: Learning complex stochastic models with invertible neural networks". In: *arXiv preprint arXiv:2003.06281*.

Radev, Stefan T, Andreas Voss, et al. (2020). "Amortized Bayesian Inference for Models of Cognition". In: *arXiv preprint arXiv:2005.03899*.

Raiko, Tapani et al. (2014). "Iterative neural autoregressive distribution estimator nade-k". In: *Advances in neural information processing systems* 27, pp. 325–333.

Ranganath, Rajesh, Dustin Tran, and David Blei (2016). "Hierarchical variational models". In: *International Conference on Machine Learning*. PMLR, pp. 324–333.

Rasmussen, Carl Edward (2003). "Gaussian processes in machine learning". In: *Summer school on machine learning*. Springer, pp. 63–71.

Ratcliff, Roger (1978). "A theory of memory retrieval." In: *Psychological review* 85.2, p. 59.

Ratcliff, Roger and Michael J Frank (2012). "Reinforcement-based decision making in corticostriatal circuits: mutual constraints by neurocomputational and diffusion models". In: *Neural computation* 24.5, pp. 1186–1229.

Ratcliff, Roger, Cynthia Huang-Pollock, and Gail McKoon (2018). "Modeling individual differences in the go/no-go task with a diffusion model." In: *Decision* 5.1, p. 42.

Ratcliff, Roger and Gail McKoon (2008). "The diffusion decision model: theory and data for two-choice decision tasks". In: *Neural computation* 20.4, pp. 873–922.

Raynal, Louis et al. (2019). "ABC random forests for Bayesian parameter inference". In: *Bioinformatics* 35.10, pp. 1720–1728.

Rezende, Danilo Jimenez and Shakir Mohamed (2015). "Variational inference with normalizing flows". In: *arXiv preprint arXiv:1505.05770*.

Rezende, Danilo Jimenez, George Papamakarios, et al. (2020). "Normalizing flows on tori and spheres". In: *arXiv preprint arXiv:2002.02428*.

Rhodes, Benjamin, Kai Xu, and Michael U. Gutmann (2020). *Telescoping Density-Ratio Estimation*. arXiv: 2006.12204 [stat.ML].

Robbins, Herbert and Sutton Monro (1951). "A stochastic approximation method". In: *The annals of mathematical statistics*, pp. 400–407.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, pp. 533–536.

Salvatier, John, Thomas V Wiecki, and Christopher Fonnesbeck (2016). "Probabilistic programming in Python using PyMC3". In: *PeerJ Computer Science* 2, e55.

Scarselli, Franco et al. (2008). "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80.

Schafer, Chad M and Peter E Freeman (2012). "Likelihood-free inference in cosmology: Potential for the estimation of luminosity functions". In: *Statistical Challenges in Modern Astronomy V*. Springer, pp. 3–19.

Shinn, Maxwell, Norman H Lam, and John D Murray (2020). "A flexible framework for simulating and fitting generalized drift-diffusion models". In: *Elife* 9, e56938.

Sisson, Scott A, Yanan Fan, and Mark Beaumont (2018). *Handbook of approximate Bayesian computation*. CRC Press.

Sisson, Scott A, Yanan Fan, and Mark M Tanaka (2007). "Sequential monte carlo without likelihoods". In: *Proceedings of the National Academy of Sciences* 104.6, pp. 1760–1765.

Sobolev, Artem and Dmitry Vetrov (2019). "Importance weighted hierarchical variational inference". In: *arXiv preprint arXiv:1905.03290*.

Sorrenson, Peter, Carsten Rother, and Ullrich Köthe (2020). "Disentanglement by Nonlinear ICA with General Incompressible-flow Networks (GIN)". In: *arXiv preprint arXiv:2001.04872*.

Srivastava, Vaibhav et al. (2017). "A martingale analysis of first passage times of time-dependent Wiener diffusion models". In: *Journal of Mathematical Psychology* 77, pp. 94–110.

Sullivan, Nicolette et al. (2015). "Dietary self-control is related to the speed with which attributes of healthfulness and tastiness are processed". In: *Psychological science* 26.2, pp. 122–134.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Talts, Sean et al. (2018). "Validating Bayesian inference algorithms with simulation-based calibration". In: *arXiv preprint arXiv:1804.06788*.

Tavaré, Simon et al. (1997). "Inferring coalescence times from DNA sequence data". In: *Genetics* 145.2, pp. 505–518.

Tejero-Cantero, Alvaro et al. (2020). "Sbi: a toolkit for simulation-based inference". In: *Journal of Open Source Software* 5.52, p. 2505.

Thomas, Owen et al. (2020). "Likelihood-free inference by ratio estimation". In: *Bayesian Analysis*.

— (2021). "Likelihood-free inference by ratio estimation". In: *Bayesian Analysis*.

Tran, Dustin, Rajesh Ranganath, and David M Blei (2017). "Hierarchical implicit models and likelihood-free variational inference". In: *arXiv preprint arXiv:1702.08896*.

Tran, Dustin, Keyon Vafa, et al. (2019). "Discrete flows: Invertible generative models of discrete data". In: *Advances in Neural Information Processing Systems*, pp. 14719–14728.

Turner, Brandon M and Per B Sederberg (2014). "A generalized, likelihood-free method for posterior estimation". In: *Psychonomic bulletin & review* 21.2, pp. 227–250.

Turner, Brandon M and Trisha Van Zandt (2012). "A tutorial on approximate Bayesian computation". In: *Journal of Mathematical Psychology* 56.2, pp. 69–85.

— (2014). "Hierarchical approximate Bayesian computation". In: *Psychometrika* 79.2, pp. 185–209.

— (2018). "Approximating Bayesian inference through model simulation". In: *Trends in Cognitive Sciences* 22.9, pp. 826–840.

Uehara, Masatoshi et al. (2016). "Generative adversarial nets from a density ratio estimation perspective". In: *arXiv preprint arXiv:1610.02920*.

Uria, Benigno, Iain Murray, and Hugo Larochelle (2013). "RNADE: The real-valued neural autoregressive density-estimator". In: *Advances in Neural information processing systems* 26, pp. 2175–2183.

Usher, Marius and James L McClelland (2001). "The time course of perceptual choice: the leaky, competing accumulator model." In: *Psychological review* 108.3, p. 550.

Vandekerckhove, Joachim and Francis Tuerlinckx (2008). "Diffusion model analysis with MATLAB: A DMAT primer". In: *Behavior research methods* 40.1, pp. 61–72.

Vandekerckhove, Joachim, Francis Tuerlinckx, and Michael D Lee (2011). "Hierarchical diffusion models for two-choice response times." In: *Psychological methods* 16.1, p. 44.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008.

Voss, Andreas and Jochen Voss (2007). "Fast-dm: A free program for efficient diffusion model analysis". In: *Behavior research methods* 39.4, pp. 767–775.

Vossen, Julian, Baptiste Feron, and Antonello Monti (2018). "Probabilistic forecasting of household electrical load using artificial neural networks". In: *2018 IEEE International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*. IEEE, pp. 1–6.

W., Feller and Feller V. (1968). *An Introduction to Probability Theory and its Applications Vol 1*. Vol. 1. Wiley.

Wagenmakers, Eric-Jan, Han L. J. Van der Maas, and Raoul P. P. P. Grasman (2007). "An EZ-diffusion model for response time and accuracy". In: *Psychonomic Bulletin & Review* 14, pp. 3–22.

Wald, Abraham and Jacob Wolfowitz (1948). "Optimum character of the sequential probability ratio test". In: *The Annals of Mathematical Statistics*, pp. 326–339.

Wang, Bo and D Michael Titterington (2005). "Inadequacy of interval estimates corresponding to variational Bayesian approximations." In: *AISTATS*. Citeseer, pp. 373–380.

Wegmann, Daniel et al. (2010). "ABCtoolbox: a versatile toolkit for approximate Bayesian computations". In: *BMC bioinformatics* 11.1, p. 116.

Wiecki, Thomas V, Chrystalina A Antoniades, et al. (2016). "A computational cognitive biomarker for early-stage Huntington's disease". In: *PLoS One* 11.2, e0148409.

Wiecki, Thomas V, Jeffrey Poland, and Michael J Frank (2015). "Model-based cognitive neuroscience approaches to computational psychiatry: clustering and classification". In: *Clinical Psychological Science* 3.3, pp. 378–399.

Wiecki, Thomas V, Imri Sofer, and Michael J Frank (2013). "HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python". In: *Frontiers in neuroinformatics* 7, p. 14.

Wieschen, Eva Marie, Andreas Voss, and Stefan Radev (2020). "Jumping to conclusion? a lévy flight model of decision making". In: *TQMP* 16.2, pp. 120–132.

Wilkinson, Richard (2014). "Accelerating ABC methods using Gaussian processes". In: *Artificial Intelligence and Statistics*. PMLR, pp. 1015–1023.

Wong, Kong-Fatt and Xiao-Jing Wang (2006). "A recurrent network mechanism of time integration in perceptual decisions". In: *Journal of Neuroscience* 26.4, pp. 1314–1328.

Wood, Simon N (2010). "Statistical inference for noisy nonlinear ecological dynamic systems". In: *Nature* 466.7310, pp. 1102–1104.

Zaheer, Manzil et al. (2017). "Deep sets". In: *Advances in neural information processing systems*, pp. 3391–3401.

Zajkowski, Wojciech K, Malgorzata Kossut, and Robert C Wilson (2017). "A causal role for right frontopolar cortex in directed, but not random, exploration". In: *Elife* 6, e27430.

Zhou, Jie et al. (2018). "Graph neural networks: A review of methods and applications". In: *arXiv preprint arXiv:1812.08434*.

Chapter 3

Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience

In cognitive neuroscience, computational modeling can formally adjudicate between theories and affords quantitative fits to behavioral/brain data. Pragmatically, however, the space of plausible generative models considered is dramatically limited by the set of models with known likelihood functions. For many models, the lack of a closed-form likelihood typically impedes Bayesian inference methods. As a result, standard models are evaluated for convenience, even when other models might be superior. Likelihood-free methods exist but are limited by their computational cost or their restriction to particular inference scenarios. Here, we propose neural networks that learn approximate likelihoods for arbitrary generative models, allowing fast posterior sampling with only a one-off cost for model simulations that is amortized for future inference. We show that these methods can accurately recover posterior parameter distributions for a variety of neurocognitive process models. We provide code allowing users to deploy these methods for arbitrary hierarchical model instantiations without further training.

3.1 Introduction

Computational modeling has gained traction in cognitive neuroscience in part because it can guide principled interpretations of functional demands of cognitive systems while maintaining a level of tractability in the production of quantitative fits of brain-behavior relationships. For example, models of reinforcement learning are frequently used to estimate the neural correlates of the exploration/exploitation tradeoff, of asymmetric learning from positive versus negative outcomes, or of model-based vs model-free control (Schoennberg et al., 2007; Niv et al., 2012; Frank, Samanta, et al., 2007; Zajkowski, Kossut, and Wilson, 2017; Badre et al., 2012; Nathaniel D Daw et al., 2011). Similarly, models of dynamic decision-making processes are commonly used to disentangle the strength of the evidence for a given choice from the amount of that evidence needed to commit to any choice, and how such parameters are impacted by reward, attention, and neural variability across species (Rangel, Camerer, and Montague, 2008; Forstmann et al., 2010; Krajbich and Rangel, 2011; Frank, Gagne, et al., 2015; Yartsev et al., 2018; Doi et al., 2020). Parameter estimates might also be used as a theoretically-driven method to reduce the dimensionality of brain/behavioral data that can be used for prediction of e.g. clinical status in computational psychiatry (Huys, Maia, and Frank, 2016).

Interpreting such parameter estimates requires robust methods that can estimate their generative values, ideally including their uncertainty. For this purpose, Bayesian statistical methods have gained traction. The basic conceptual idea in Bayesian statistics is to treat parameters θ and data $\mathbf{x}$ as stemming from a joint probability model $p(\theta, \mathbf{x})$. Statistical inference proceeds by using Bayes' rule,

$$p(\theta|\mathbf{x}) = \frac{p(\mathbf{x}|\theta)p(\theta)}{p(\mathbf{x})}$$

to get at $p(\theta|\mathbf{x})$, the posterior distribution over parameters given data. $p(\theta)$ is known as the prior distribution over the parameters θ, and $p(\mathbf{x})$ is known as the evidence or just probability of the data (a quantity of importance for model comparison). The term $p(\mathbf{x}|\theta)$, the probability (density) of the dataset $\mathbf{x}$ given parameters θ, is known as the likelihood (in accordance with usual notation, we will also write $\ell(\theta|\mathbf{x})$ in the following). It is common in cognitive science to represent likelihoods as $p(\mathbf{x}|\theta, s)$, where s specifies a particular stimulus. We suppress s in our notation, but note that our approach easily generalizes when explicitly conditioning on trial-wise stimuli. Bayesian parameter estimation is a natural way to characterize uncertainty over parameter values. In turn, it provides a way to identify and probe parameter trade-offs. While we often don't have access to $p(\theta|\mathbf{x})$ directly, we can draw samples from it instead, for example via Markov Chain Monte Carlo (Robert and Casella, 2013; Diaconis, 2009; Robert and Casella, 2011).

Bayesian estimation of the full posterior distributions over model parameters contrast with maximum likelihood estimation methods often used to provide single best parameter values, without considering their uncertainty or whether other parameter estimates might give similar fits. Bayesian methods naturally extend to settings that assume an implicit hierarchy in the generative model in which parameter estimates at the individual level are informed by the distribution across a group, or

even to assess within an individual how trial-by-trial variation in (for example) neural activity can impact parameter estimates (commonly known simply as hierarchical inference). Several toolboxes exist for estimating the parameters of popular models like the drift diffusion model of decision making and are widely used by the community for this purpose (Wiecki, Sofer, and Frank, 2013; Heathcote et al., 2019; Turner, Van Maanen, and Forstmann, 2015b; Ahn, Haines, and Zhang, 2017). Various studies have used these methods to characterize how variability in neural activity, and manipulations thereof, alter learning and decision parameters that can quantitatively explain variability in choice and response time distributions (Cavanagh et al., 2011; Frank, Gagne, et al., 2015; Herz et al., 2016; Pedersen and Frank, 2020).

Traditionally, however, posterior sampling or maximum likelihood estimation for such models required analytical likelihood functions: a closed-form mathematical expression providing the likelihood of observing specific data (reaction times and choices) for a given model and parameter setting. This requirement limits the application of any likelihood-based method to a relatively small subset of cognitive models chosen for so-defined convenience instead of theoretical interest. Consequently, model comparison and estimation exercises are constrained, as many important but likelihood-free models were effectively *untestable* or required weeks to process a single model formulation. Testing any slight adjustment to the generative model (e.g., different hierarchical grouping or splitting conditions) requires a repeated time investment of the same order. For illustration, we focus on the class of sequential sampling models applied to decision-making scenarios, with the most common variants of the drift diffusion model (DDM). The approach is, however, applicable to any arbitrary domain.

In the standard DDM, a two-alternative choice decision is modeled as a noisy accumulation of evidence toward one of two decision boundaries (Ratcliff and McKoon, 2008). This model is widely used as it can flexibly capture variations in choice, error rates, and response time distributions across a range of cognitive domains and its parameters have both psychological and neural implications. While the likelihood function is available for the standard DDM and some variants including inter-trial variability of its drift parameter, even seemingly small changes to the model form, such as dynamically varying decision bounds (Cisek, Puskas, and El-Murr, 2009; Hawkins et al., 2015) or multiple choice alternatives (Krajbich and Rangel, 2011), are prohibitive for likelihood-based estimation, and instead require expensive Monte Carlo simulations, often without providing estimates of uncertainty across parameters.

In the last decade and a half, Approximate Bayesian Computation (ABC) algorithms have grown to prominence (Sisson, Fan, and Beaumont, 2018). These algorithms enable one to sample from posterior distributions over model parameters, where models are defined only as simulators, which can be used to construct empirical likelihood distributions (Sisson, Fan, and Beaumont, 2018). ABC approaches have enjoyed successful application across life and physical sciences (e.g., Akeret et al., 2015), and notably, in cognitive science (Turner and Van Zandt, 2018), enabling researchers to estimate theoretically interesting models that were heretofore intractable. However, while there have been many advances without sacrificing information loss in the posterior distributions (Turner and

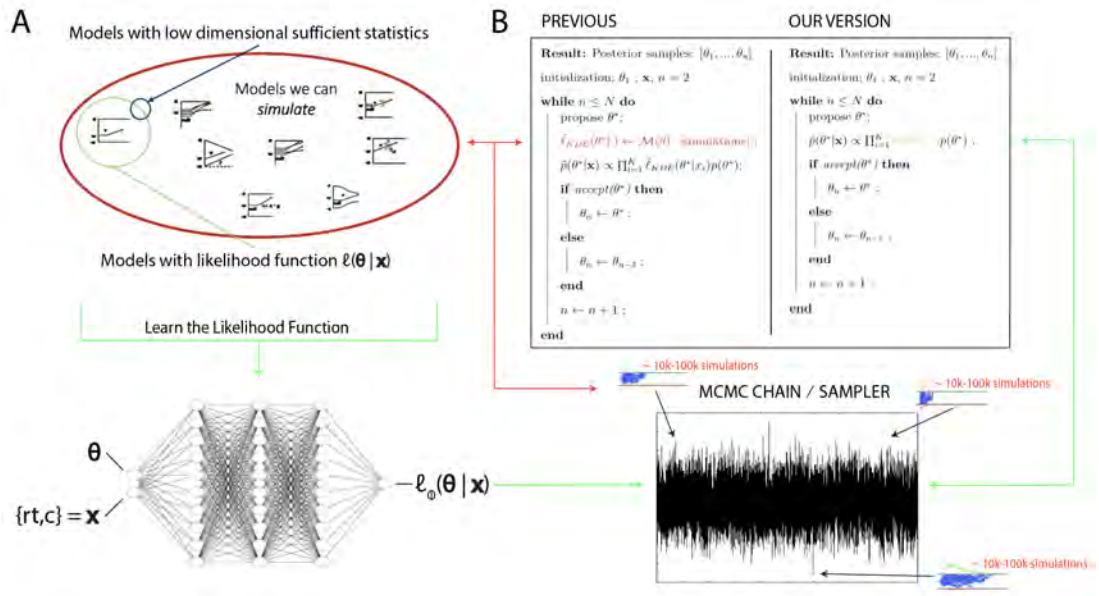

Figure 3.1. **A** The space of theoretically interesting models in the cognitive neurosciences (red) is much larger than the space of mechanistic models with analytic likelihood functions (green). Traditional ABC methods require models that have low dimensional sufficient statistics (blue). **B** Illustrates how LANs can be used in lieu of online simulations for efficient posterior sampling. The left panel shows the predominant PDA method used for ABC in the cognitive sciences (Turner, Van Maanen, and Forstmann, 2015a). For each step along a Markov chain, 10K-100K simulations are required to obtain a single likelihood estimate. The right panel shows how we can avoid the simulation steps during inference using amortized likelihood networks that have been pretrained using empirical likelihood functions (operationally in this paper: Kernel density estimates and discretized histograms).

Sederberg, 2014; Holmes, 2015), such ABC methods typically require many simulations to generate synthetic or empirical likelihood distributions, and hence can be computationally expensive (in some cases prohibitive – it can take weeks to estimate parameters for a single model). This issue is further exacerbated when embedded within a sequential Markov Chain Monte Carlo (MCMC) sampling scheme, which is needed for unbiased estimates of posterior distributions. For example, one typically needs to simulate between 10,000 - 100,000 times (the exact number varies depending on the model) for each proposed combination of parameters (i.e., for each sample along a Markov chain, which may itself contain tens of thousands of samples), after which they are discarded. This situation is illustrated in Figure 3.1, where the red arrows point at the computations involved in the approach suggested by Turner, Van Maanen, and Forstmann, 2015a.

To address this type of issue, the statistics and machine learning communities have increasingly focused on strategies for the amortization of simulation-based computations (Gutmann et al., 2018; Papamakarios and I. Murray, 2016; Papamakarios, Nalisnick, et al., 2019; Lueckmann et al., 2019a; Lueckmann et al., 2019b; S. T. Radev, Mertens, Voss, and Köthe, 2020; S. T. Radev, Mertens, Voss, Ardizzone, et al., 2020; Gonçalves et al., 2020). The aim is generally to use model simulations

up-front and learn a reusable approximation of the function of interest (targets can be the likelihood, the evidence, or the posterior directly).

In this article, we develop a general ABC method (and toolbox) that allows users to perform inference on a significant number of neurocognitive models without repeatedly incurring substantial simulation costs. To motivate our particular approach and situate it in the context of other methods, we outline the following key desiderata:

1. First, the method needs to be easily and rapidly deployable to end-users for Bayesian inference on various models hitherto deemed computationally intractable. This desideratum naturally leads us to an amortization approach, where end-users can benefit from costs incurred upfront.

2. Second, our method should be sufficiently flexible to support arbitrary inference scenarios, including hierarchical inference, estimation of latent covariate (e.g., neural) processes on the model parameters, arbitrary specification of parameters across experimental conditions, and without limitations on data-set sizes. This desideratum leads us to amortize the likelihood functions, which (unlike other amortization strategies) can be immediate applied to arbitrary inference scenarios without further cost.

3. Third, we desire approximations that do not apriori sacrifice covariance structure in the parameter posteriors, a limitation often induced for tractability in variational approaches to approximate inference (Blei, Kucukelbir, and McAuliffe, 2017).

4. Fourth, end-users should have access to a convenient interface that integrates the new methods seamlessly into pre-existing workflows. The aim is to allow users to get access to a growing database of amortized models through this toolbox and enable increasingly complex models to be fit to experimental data routinely, with minimal adjustments to the user's working code. For this purpose, we will provide an extension to the widely used HDDM toolbox (Wiecki, Sofer, and Frank, 2013; Pedersen and Frank, 2020) for parameter estimation of DDM and RL models.

5. Fifth, our approach should exploit modern computing architectures, specifically parallel computation. This leads us to focus on the encapsulation of likelihoods into neural network architectures, which will allow batch-processing for posterior inference.

Guided by these desiderata, we developed two amortization strategies based on Neural Networks (NNs) and empirical likelihood functions. Rather than simulate during inference, we instead train NNs as parametric function approximators to learn the likelihood function from an initial set of *apriori* simulations across a wide range of parameters. By learning (log) likelihood functions directly, we avoid posterior distortions that result from inappropriately chosen (user defined) summary statistics and corresponding distance measures as applied in traditional ABC methods (Sisson, Fan, and Beaumont, 2018). Once trained, likelihood evaluation only requires a forward pass through the NN (as if it were an analytic likelihood) instead of necessitating simulations. Moreover, any algorithm can be used to facilitate posterior sampling, maximum likelihood estimation (MLE), or maximum a posteriori estimation (MAP).

For generality, and because they each have their advantages, we use two classes of architectures, Multi-layered-perceptrons (MLPs) and Convolutional Neural Networks (CNNs), and two different posterior sampling methods (MCMC and Importance sampling). We show proofs of concepts using posterior sampling and parameter recovery studies for a range of cognitive process models of interest. The trained neural networks provide the community with a (continually expandable) bank of encapsulated likelihood functions that facilitate consideration of a larger (previously computationally inaccessible) set of cognitive models, with orders of magnitude speedup relative to simulation-based methods. This speedup is possible because costly simulations only have to be run once per model upfront and henceforth be avoided during inference: previously executed computations are amortized and then shared with the community.

Moreover, we develop standardized amortization-pipelines that allow the user to apply this method to arbitrary models, requiring them to provide only a functioning simulator of their model of choice.

In Section 3.2 we situate our approach in the greater context of approximate Bayesian computation (ABC), with a brief review of online and amortization algorithms. Section 3.3 describes the amortization-pipelines we propose as well as suggested use-cases. Section 3.4 discusses the cognitive process models that we utilized as relevant test-beds for our methods. Section 3.5 shows proof-of-concept parameter recovery studies for the two proposed algorithms, demonstrating that the method accurately recovers both the posterior mean and variance (uncertainty) of generative model parameters, and that it does so at a run-time speed of orders of magnitude faster than traditional ABC approaches without further training. In Section 3.5.4 we further demonstrate an application to hierarchical inference, in which our trained networks can be imported into widely used toolboxes for arbitrary inference scenarios. In Section 3.6 and Section 3.7, we further situate our work in the context of other ABC amortization strategies and discuss limitations and future work.

3.2 Approximate Bayesian Computation

ABC methods apply when one has access to a parametric stochastic simulator (also referred to as generative model), but, unlike the usual setting for statistical inference, no access to an explicit mathematical formula for the likelihood of observations given the simulator's parameter setting.

While likelihood functions for a stochastic stimulator can in principle be mathematically derived, this can be exceedingly challenging even for some historically famous models such as the Ornstein-Uhlenbeck process (Lipton and Kaushansky, 2018), and may be intractable in many others. Consequently, the statistical community has increasingly developed ABC tools that enable posterior inference of such "likelihood-free" stochastic models, while completely bypassing any likelihood derivations (Cranmer, Brehmer, and Louppe, 2020).

Given a parametric stochastic simulator model $\mathcal{M}$ and dataset $\mathbf{x}$, instead of exact inference based on $p_{\mathcal{M}}(\theta|\mathbf{x})$, these methods attempt to draw samples from an approximate posterior $\tilde{p}_{\mathcal{M}}(\theta|\mathbf{x})$. Consider the following general equation:

$$\tilde{p}_{\mathcal{M}}(\theta|\mathbf{x}) \propto \tilde{p}_{\mathcal{M}}(\mathbf{x}|\theta)\pi(\theta) \propto \int K_h\left(||s_{\mathcal{M}} - s_{\mathbf{x}}||\right) p_{\mathcal{M}}(s_{\mathcal{M}}|\theta)\, ds_{\mathcal{M}}\ \pi(\theta)$$

where $s_{\mathcal{M}}$ refers to sufficient statistics (roughly, summary statistics of a dataset that retain sufficient information about the parameters of the generative process). [1] $K_h(||s_{\mathcal{M}} - s_{\mathbf{x}}||)$, refers to a kernel-based distance measure / cost function, which evaluates a probability density function for a given distance between the observed and expected summary statistics $||s_{\mathcal{M}} - s_{\mathbf{x}}||$. The parameter h (commonly known as bandwidth parameter) modulates the cost gradient. Higher values of h lead to more graceful decreases in cost (and therefore a worse approximation of the true posterior).

By generating simulations, one can use such summary statistics to obtain approximate likelihood functions, denoted as $\tilde{p}_{\mathcal{M}}(\mathbf{x}|\theta)$, where approximation error can be mitigated by generating large numbers of simulations. The caveat is that the amount of simulation runs needed to achieve a desired degree of accuracy in the posterior can render such techniques computationally infeasible.

With a focus on amortization, our goal is to leverage some of the insights and developments in the ABC community to develop neural network architectures that can learn approximate likelihoods deployable for any inference scenario (and indeed any inference method, including MCMC, variational inference, or even maximum likelihood estimation) without necessitating repeated training. We next describe our particular approach, and we return to a more detailed comparison to existing methods in section .

3.3 Learning the Likelihood with simple Neural Network Architectures

In this section we outline our approach to amortization of computational costs of large numbers of simulations required by traditional ABC methods. Amortization approaches, incur a one-off (potentially quite large) simulation cost to enable cheap, repeated inference for any dataset. Recent research has lead to substantial developments in this area (Cranmer, Brehmer, and Louppe, 2020). The most straightforward approach is to simply simulate large amounts of data and compile a database of how model parameters are related to observed summary statistics of the data (Mestdagh et al., 2019). This database can then be used during parameter estimation in empirical datasets using a combination of nearest neighbor search and local interpolation methods. However, this approach suffers from the curse of dimensionality with respect to storage demands (a problem that is magnified with increasing model parameters). Moreover, its reliance on summary statistics (Sisson, Fan, and Beaumont, 2018) does not naturally facilitate flexible re-use across inference scenarios (e.g. hierarchical models, multiple conditions while fixing some parameters across conditions etc.).

[1] We note two observations regarding sufficient statistics. First, a sufficient statistic is, in principle, simply a function of the data $f(\{x_1, ..., x_n\}) \mapsto \{s_1, ..., s_m\}$, where hopefully $m << n$. The identity map is one such function, whence we can consider the data itself as a special case (s can be x). Second, it is not common to have well-defined, low dimensional sufficient statistics for any given $\mathcal{M}$.

To fulfill all desiderata outlined in the introduction, we focus on directly encapsulating the likelihood function over empirical observations (i.e., choices. RTs) of a simulation model so that likelihood evaluation is substantially cheaper than constructing (empirical) likelihoods via model simulation online during inference. This strategy then allows for flexible reuse of such approximate likelihood functions $\hat{\ell}(\theta|\mathbf{x})$, in a large variety of inference scenarios applicable to common experimental design paradigms. Specifically, we encapsulate $\hat{\ell}(\theta|\mathbf{x})$ as a feed-forward neural network, which allows for parallel evaluation by design. We refer to these networks as likelihood approximation networks (LANs).

Figure 3.1 spells out the setting (panel **A**) and usage (panel **B**) of such a method. The LAN architectures used in this paper are simple, small in size and are made available for download for local usage. While this approach does not allow for instantaneous posterior inference, it does considerably reduce computation time (by up to three orders of magnitude; see "Run-time" section below) when compared to approaches that demand simulations at inference. Notably, this approach also substantially speeds up inference even for models that are not entirely likelihood free but nevertheless require costly numerical methods to obtain likelihoods. Examples are the full DDM with inter-trial variability in parameters (for which likelihoods can be obtained via numerical integration, as is commonly done in software packages such as HDDM but with substantial cost), but also other numerical methods for generalized DDMs (Shinn, Lam, and J. D. Murray, 2020; Drugowitsch, 2016). At the same time we maintain the high degree of flexibility with regards to deployment across arbitrary inference scenarios. As such, LANs can be treated as a highly flexible plugins to existing inference algorithms and remain conceptually simple and lightweight.

Before elaborating our LAN approach, we briefly situate it in the context of some related work.

One branch of literature which interfaces ABC with deep learning attempts to amortize posterior inference directly in end-to-end neural networks (S. T. Radev, Mertens, Voss, and Köthe, 2020; S. T. Radev, Mertens, Voss, Ardizzone, et al., 2020; Papamakarios and I. Murray, 2016; Papamakarios, Nalisnick, et al., 2019; Gonçalves et al., 2020). Here, neural network architectures are trained with a large number of simulated datasets to produce posterior distributions over parameters, and once trained, such networks can be applied to directly estimate parameters from new datasets without need for further simulations. However, the goal to directly estimate posterior parameters from data requires the user to first train a neural network for the very specific inference scenario in which it is applied empirically. Such approaches are not easily deployable if a user wants to test multiple inference scenarios (e.g., parameters may vary as a function of task condition or brain activity, or in hierarchical frameworks) and consequently, they do not achieve our second desideratum needed for a user-friendly toolbox making inference cheap and simple.

We return to discuss the relative merits and limitations of these and further alternative approaches in the Discussion.

Formally, we use model simulations to learn a function $f_\Phi(x, \theta)$, where $f_\Phi(.)$ is the output of a neural network with parameter vector Φ (weights, biases). The function $f_\Phi(\mathbf{x}, \theta)$ is used as an approximation $\hat{\ell}(\theta|\mathbf{x})$ of the likelihood function $\ell(\theta|\mathbf{x})$. Once learned, we can use such f as a plug-in

to Bayes Rule,

$$\hat{p}(\theta|\mathbf{x}) \propto \prod_{i=1}^{N} \hat{\ell}(\theta|x_i)p(\theta)$$

To perform posterior inference we can now use a broad range of existing Monte Carlo (MC) or Markov Chain Monte Carlo (MCMC) algorithms. We note that, fulfilling our second desideratum, an extension to hierarchical inference is as simple as plugging in our neural network into a probabilistic model of the form,

$$\hat{p}(\theta|\mathbf{x}) \propto \prod_{j=1}^{J} \prod_{i=1}^{N} \hat{\ell}(\theta_j|x_{ji})p(\theta_j|\alpha)p(\alpha|\gamma)$$

where α refers to generic group level global parameters, and γ serves as a generic fixed hyperparameter vector.

We provide proofs of concepts for two types of LANs. While we use multi-layered perceptrons (MLPs) and convolutional neural networks (CNNs), of conceptual importance is the distinction between the two problem representations they tackle, rather than the network architectures *per se*.

The first problem representation, which we call the pointwise approach, considers the functions $f_\Phi(x|\theta)$, where θ is the parameter vector of a given stochastic simulator model, and x is a single data point (trial outcome). The pointwise approach is a mapping from the input dimension $|\Theta| + |\mathbf{x}|$, where $|.|$ refers to the cardinality, to the one dimensional output. The output is simply the log-likelihood of the single datapoint x given the parameter vector θ. As explained in the next section, for this mapping we chose simple MLPs.

The second problem representation, which we will refer to as the histogram approach, instead aims to learn a function $f_\Phi(.|\theta)$, which maps a parameter vector θ to the likelihood over the full (discretized) dataspace (i.e., the likelihood of the entire RT distributions at once). We represent the output space as an outcome histogram with dimensions $n \times m$ (where in our applications n is the number of bins for a discretization of reaction times, and m refers to the number of distinct choices). Thus our mapping has input dimension $|\Theta|$ and output dimension $|m \times n|$. Representing the problem this way, we chose CNNs as the network architecture.

In each of the above cases, we pursued the architectures that seemed to follow naturally, without any commitment to their optimality. The pointwise approach operates on low dimensional inputs *and* outputs. With network evaluation speed being of primary importance to us, we chose a relatively shallow MLP to learn this mapping, given that it was expressive enough. However, when learning a mapping from a parameter vector θ to the likelihood over the full dataspace, as in the histogram approach, we map a low dimensional data manifold to a much higher dimensional one. Using an MLP for this purpose would imply that the number of neural network parameters needed to learn this function would be orders of magnitude larger than using a CNN. Not only would this mean that forward passes through the network would take longer, but also an increased propensity to overfit on an identical data budget.

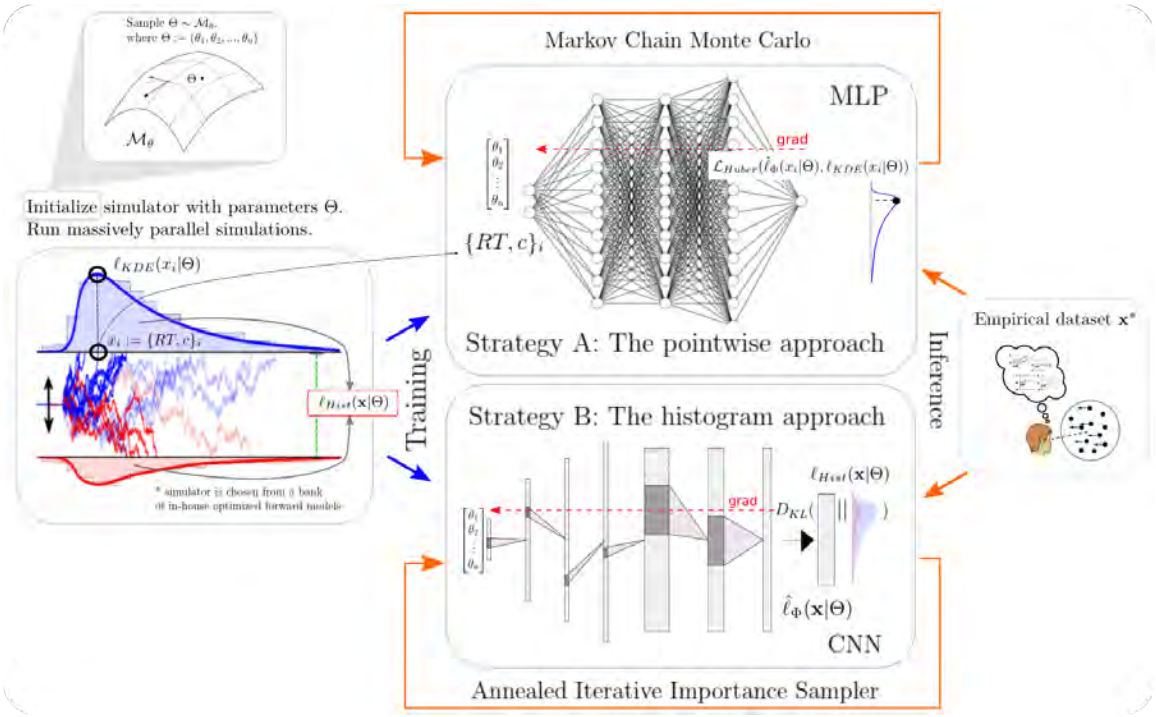

Figure 3.2. High level overview of our approaches. For a given model $\mathcal{M}$, we sample model parameters θ from a region of interest (upper left), and run $100k$ simulations (mid left). We use those simulations to construct a KDE-based empirical likelihood, and a discretized (histogram-like) empirical likelihood. The combination of parameters and the respective likelihoods is then used to train the likelihood networks (middle). Once trained we can use the MLP and CNN for posterior inference given an empirical / experimental dataset (right).

The next two sections will give some detail regarding the networks chosen for the pointwise and histogram LANs. Figure 3.1 illustrates the general idea of our approach while Figure 3.2 gives a conceptual overview of the exact training and inference procedure proposed. These details are thoroughly discussed in the last section of the paper.

3.3.1 Pointwise approach: Learn likelihoods of individual observations with MLPs

As a first approach, we use simple Multi-layered Perceptrons (MLPs), to learn the likelihood function of a given stochastic simulator. The network learns the mapping $\{\theta, \mathbf{x}\} \rightarrow \log \ell_\Phi(\theta|\mathbf{x})$, where θ represents the parameters of our stochastic simulator, $\mathbf{x}$ are the simulator outcomes (in our specific examples below, $\mathbf{x}$, refers to the tuple (rt, c) of reaction times and choices), and $\log \ell(\theta|\mathbf{x})$ is the log-likelihood of the respective outcome. The trained network then serves as the approximate likelihood function $\log \hat{\ell}_\Phi(\theta|\mathbf{x})$. We emphasize that the network is trained as a function approximator, to provide us with a computationally cheap likelihood estimate, not to provide a surrogate simulator. Biological plausibility of the network is therefore not a concern when it comes to architecture selection. Fundamentally this approach attempts to learn the log-likelihood via a non-linear regression, for

which we chose an MLP. Log-likelihood labels for training were derived from empirical likelihoods functions (details in section 3.8) which in turn were constructed as kernel density estimates (KDEs). The construction of KDEs roughly followed Turner, Van Maanen, and Forstmann, 2015a. We chose the Huber loss function (detailed in section 3.8) because it is more robust to outliers and thus less susceptible to distortions that can arise in the tails of distributions.

3.3.2 Histogram approach: Learn likelihoods of entire dataset distributions with CNNs

Our second approach is based on a convolutional neural network (CNN) architecture. Whereas the MLP learned to output a single scalar likelihood output for each data point ("trial", given a choice, reaction time and parameter vector), the goal of the CNN was to evaluate, for given model parameters, the likelihood of an arbitrary number of datapoints via one forward pass through the network. To do so, the output of the CNN was trained to produce a probability distribution over a discretized version of the dataspace, given a stochastic model and parameter vector. The network learns the mapping $\theta \rightarrow \log \ell_{\Phi}(\theta|.)$.

There is a side-benefit, in line with the methods proposed by Lueckmann et al., 2019b; Papamakarios, Sterratt, and I. Murray, 2019. The CNN can in fact act as a surrogate simulator, for the purpose of this paper we do not exploit this possibility however. Since here we attempt to learn distributions, labels were simple binned empirical likelihoods, and as a loss function we chose the KL-divergence between the networks output distribution and the label distribution (details in section 3.8).

3.3.3 Training Specifics

Most of the specifics regarding training procedures are discussed in detail in section 3.8, however we mention some aspects here to aid readability.

We used $1.5M$, and $3M$ parameter vectors (based on 100k simulations each), to train the MLP and CNN approach respectively. We chose these numbers consistently across all models and in fact trained on less examples for the MLP only due to RAM limitations we faced on our machines (which in principle can be circumvented). These numbers are purposely high (but in fact quite achievable with access to a computing cluster, simulations for each model was on the order of hours only) since we were interested in a workable proof of concept. We did not investigate that systematic minimization of training data, however some crude experiments indicate that a decrease by an order of magnitude did not seriously affect performance.

We emphasize that this is in line with the expressed philosophy of our approach. The point of amortization is to throw a lot of resources at the problem once, so that downstream inference is made accessible even on basic setups (a usual laptop). In case simulators are prohibitively expensive even for reasonably sized compute clusters, minimizing training data may gain more relevance. In such scenarios, training the networks will be very cheap as compared to simulation time, which implies

that retraining with progressively more simulations until one observes asymptotic test performance is a viable strategy.

3.4 Test Beds

We choose variations of sequential sampling models (SSMs) common in the cognitive neurosciences as our test-bed (Figure 3.3). The range of models we consider permits great flexibility in allowable data distributions (choices and response times). We believe that initial applications are most promising for such SSMs, because (i) analytic likelihoods are available for the most common variants (and thus provide an upper-bound benchmark for parameter recovery), and (ii) there exist many other interesting variants for which no analytic solution exists.

We note that there is an intermediate case in which numerical methods can be applied to obtain likelihoods for a broader class of models (e.g., Shinn, Lam, and J. D. Murray, 2020). These methods are nevertheless computationally expensive and do not necessarily afford rapid posterior inference. Therefore amortization via LANs is attractive even for these models. Figure 3.3 further outlines this distinction.

We emphasize that our methods are quite general and any model that generates discrete choices and response times from which simulations can be drawn within a reasonable amount of time can be suitable to the amortization techniques discussed in this paper (given that the model has parameter vectors of dimension roughly < 15). In fact, LANs are not restricted to models of reaction time and choice to begin with, even though we focus on these as test-beds.

As a general principle, all models tested below are based on stochastic differential equations of the following form,

$$d\mathbf{X}_t = a(t, x)\, dt + b(t, x)\, d\mathbf{B}_t, \quad \mathbf{X}_0 = w$$

where we are concerned with the probabilistic behavior of the particle (or vector of particles) $\mathbf{X}$. The behavior of this particle is driven by $a(t, x)$, an underlying drift function, $b(t, x)$ an underlying noise transformation function, B_t an incremental noise process, and $X_0 = w$ a starting point.

Of interest to us are specifically the properties of the first-passage-time-distributions (FPTD) for such processes, which are needed to compute the likelihood of a given response time/choice pair $\{rt, c\}$. In these models, the exit region of the particle (i.e., the specific boundary it reaches) determines the choice, and the time point of that exit determines the response time. The joint distribution of choices and reaction times are referred to as a FPTDs.

Given some exit-region $\mathcal{E}$, such FPTD's are formally defined as,

$$f_{\mathcal{E}}(t) = p(\inf_{\tau}\{(X_\tau \in \mathcal{E})\} = t)$$

In words, a first passage time, for a given exit region, is defined as the first time point of entry into the exit region, and the FPTD is the probability, respectively of such an exit happening at any

specified time t (Ratcliff, 1978; W. and V., 1968). Partitioning the exit region into subsets $\mathcal{E}_1, ..., \mathcal{E}_n$ (for example representing n choices), we can now define the set of defective distributions,

$$\{f_{\mathcal{E}_1}(t;\theta), ..., f_{\mathcal{E}_N}(t;\theta)\}$$

where $\theta \in \Theta$ describes the collection of parameters driving the process. For every $\mathcal{E}_i$,

$$\int_{[0,+\infty]} f_{\mathcal{E}_i}(t;\theta)dt = P(\mathbf{X} \text{ exits into } \mathcal{E}_i) = P(\ i \text{ gets chosen})$$

$\{f_{\mathcal{E}_\rangle}, ..., f_{\mathcal{E}_\backslash}\}$ jointly define the FPTD such that,

$$\sum_{i=1}^{n} \int_{[0,+\infty]} f_{\mathcal{E}_i}(t;\theta)dt = 1$$

These functions $f_{\mathcal{E}_i}$, jointly serve as the likelihood function s.t.,

$$\ell(\theta; \{rt, c\}) = f_{\mathcal{E}_c}(t;\theta)$$

For illustration we focus the general model formulation above to the standard DDM. Details regarding the other models in our test-bed are relegated to the methods section at the end of the paper.

To obtain the DDM from the general equation above, we set $a(t,x) = v$ (a fixed drift across time), $b(t,x) = 1$ (a fixed noise variance across time), and $\Delta\mathbf{B} \sim \mathcal{N}(0, \Delta t)$. The DDM applies to the two alternative decision case, where decision correspond to to particle crossings of an upper or lower fixed boundary. Hence $\mathcal{E}_1 = \{\mathbb{R} \geq a\}$ and $\mathcal{E}_2 = \{\mathbb{R} \leq -a\}$ where a is a parameter of the model. The DDM also includes a normalized starting point w (capturing potential response biases or priors), and finally a non-decision time τ (capturing the time for perceptual encoding and motor output). Hence, the parameter vector for the DDM is then $\theta = (v, a, w, \tau)$. The SDE is defined as,

$$d\mathbf{X}_{\tau+t} = v\, dt + \ d\mathbf{W},\ \mathbf{X}_\tau = w$$

The DDM serves principally as a basic proof of concept for us, in that it is a model for which we can compute the exact likelihoods analytically (W. and V., 1968; Navarro and Fuss, 2009).

The other models chosen for our test-bed systematically relax some of the fixed assumptions of the basic DDM, as illustrated in Figure 3.3.

We note that many new models can be constructed from the components tested here. As an example of this modularity, we introduce inhibition/excitation to the race model, which gives us the Leaky Competing Accumulator (LCA) (Usher and McClelland, 2001). We could then further extend this model by introducing parameterized bounds. We could introduce reinforcement learning parameters to a DDM (Pedersen and Frank, 2020) or in combination with any of the other decision models. Again we emphasize that while these diffusion-based models provide a large test-bed for our proposed methods, applications are in no way restricted to this class of models.

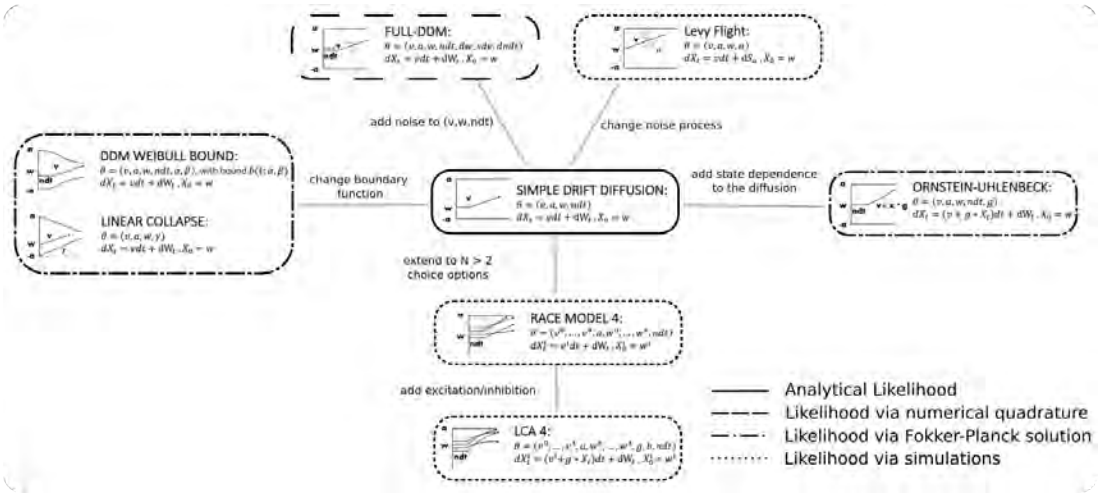

Figure 3.3. Pictorial representation of the stochastic simulators that form our test-bed. Our point of departure is the standard simple drift diffusion model (DDM) due its analytical tractability and its prevalence as the most common SSM in cognitive neuroscience. By systematically varying different facets of the DDM, we test our LANs across a range of SSMs for parameter recovery, goodness of fit (posterior predictive checks) and inference runtime. We divide the resulting models into four classes as indicated by the legend. We consider the simple DDM in the *analytical likelihood* **(solid line)** category although, strictly speaking, the likelihood involves an infinite sum and thus demands an approximation algorithm introduced by Navarro & Fuss, but this algorithm is sufficiently fast to evaluate so that it is not a computational bottleneck. The Full-DDM model needs *numerical quadrature* **(dashed line)** to integrate over variability parameters, which inflates the evaluation time by one to two orders of magnitude compared to the simple DDM. Similarly, likelihood approximations have been derived for a range of models using the *Fokker-Planck* equations **(dotted-dashed line)**, which again incurs non-negligible evaluation cost. Finally, for some models no approximations exist and we need to resort to computationally expensive *simulations* for likelihood estimates **(dotted line)**. Amortizing computations with LANs can substantially speed up inference for all but the *analytical likelihood* category (but see run-time for how it can even provide speed up in that case for large datasets).

3.5 Results

3.5.1 Networks learn likelihood function manifolds

Across epochs of training, both training and validation loss decrease rapidly and remain low (Figure 3.4A), which suggests that over-fitting is not an issue, which is sensible in this context. The low validation loss further shows that the network can interpolate likelihoods to specific parameter values it has not been exposed to (with the caveat that it has to be exposed to the same range; no claims are made about extrapolation).

Indeed, a simple interrogation of the learned likelihood manifolds shows that they smoothly vary in an interpretable fashion with respect to changes in generative model parameters (Figure 3.4B). Moreover, Figure 3.4C shows that the MLP-likelihoods mirror those obtained by KDEs using 100,000 simulations, even though the model parameter vectors were drawn randomly and thus not trained *per se*. We also note that the MLP-likelihoods appropriately filter out simulation noise (random

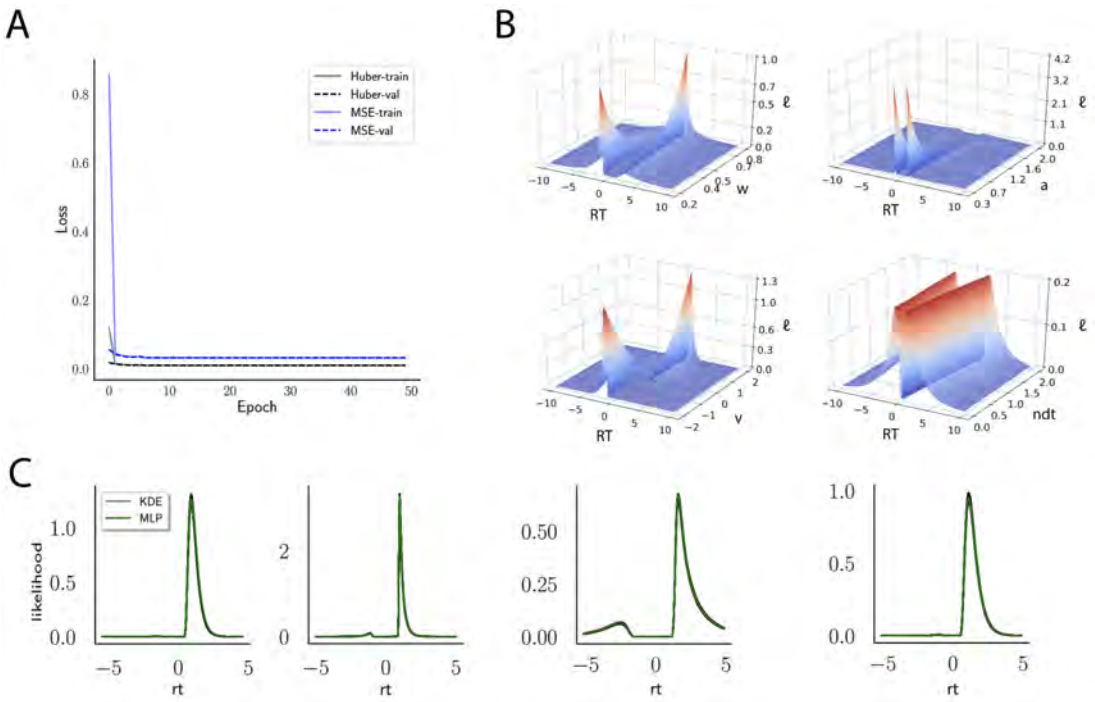

Figure 3.4. **A** Shows the training and validation loss for the MLP for the DDM model across epochs. Training was driven by the Huber loss. The MLP learned the mapping $\{\theta, rt, c\} \mapsto log\ell(\theta|rt, c)$, i.e., the log likelihood of a single choice/RT data point given the parameters. Training error declines rapidly, and validation loss trailed training loss without further detriment (no overfitting). Please see Figure 3.2 and section 3.8 for more details about training procedures. **B** Illustrates the marginal likelihood manifolds for choices and RTs, by varying one parameter in the trained region. Reaction times are mirrored for choice options -1, and 1 respectively, to aid visualization. **C** shows MLP likelihoods in green for four random parameter vectors, overlaid on top of a sample of 100 KDE-based empirical likelihoods derived from 100k samples each. The MLP mirrors the KDE likelihoods despite not having been explicitly trained on these parameters. Moreover, the MLP likelihood sits firmly at the mean of sample of 100 KDE's. Negative and positive reaction times are to be interpreted as for **B**.

fluctuations in the KDE empirical likelihoods across separate simulation runs of $100K$ samples each). This observation can also be gleaned from Figure 3.4C, which shows the learned likelihood to sit right at the center of sampled KDEs (note that for each subplot 100 such KDEs were used). As illustrated in the appendix, these observations hold across all tested models. One perspective on this is to consider the MLP-likelihoods as equivalent to KDE likelihoods derived from a much larger number of underlying samples and interpolated. The results for the CNN (not shown to avoid redundancy) mirror the MLP results. Finally, while Figure 3.4 depicts the learned likelihood for the simple DDM for illustration purposes, the same conclusions apply to the learned manifolds for all of the tested models (as shown in the appendix Figures 3.13–3.18). Indeed inspection of those manifolds is insightful for facilitating interpretation of the dynamics of the underlying models, how they differ from each other, and the corresponding RT distributions that can be captured.

3.5.2 Parameter Recovery

Benchmark: Analytical Likelihood Available

While the above inspection of the learned manifolds is promising, a true test of the method is to determine whether one can perform proper inference of generative model parameters using the MLP and CNN. Such parameter recovery exercises are typically performed to determine whether a given model is identifiable for a given experimental setting (e.g., number of trials, conditions, etc). Indeed, when parameters are collinear, recovery can be imperfect even if the estimation method itself is flawless (Wilson and Collins, 2019; Nilsson, Rieskamp, and Wagenmakers, 2011; N. D. Daw et al., 2011). A Bayesian estimation method, however, should properly assign uncertainty to parameter estimates in these circumstances, and hence it is also important to evaluate the posterior variances over model parameters.

Thus as a benchmark, we first consider the basic DDM for which an arbitrarily close approximation to the analytic likelihood is available (Navarro and Fuss, 2009). This benchmark allows us to compare parameter recovery given (1) the analytic likelihood, (2) an approximation to the likelihood specified by training an MLP on the analytical likelihood (thus evaluating the potential loss of information incurred by the MLP itself), (3) an approximation to the likelihood specified by training an MLP on KDE-based empirical likelihoods (thus evaluating any further loss incurred by the KDE reconstruction of likelihoods), and (4) an approximate likelihood resulting from training the CNN architecture, on empirical histograms. Figure 3.5 shows the results for the DDM.

For the simple DDM and analytic likelihood, parameters are nearly perfectly recovered given $N = 1024$ data points ("trials") (Figure 3.5A). Notably, these results are mirrored when recovery is performed using the MLP trained on the analytic likelihood (Figure 3.5B). This finding corroborates, as visually suggested by the learned likelihood manifolds, the conclusion that globally the likelihood function was well behaved. Moreover, only slight reductions in recoverability were incurred when the MLP was trained on the KDE likelihood estimates (Figure 3.5C), likely due to the known small biases in KDE itself (Turner, Van Maanen, and Forstmann, 2015a). Similar performance is achieved using the CNN instead of MLP (Figure 3.5 D).

As noted above, an advantage of Bayesian estimation is that we obtain an estimate of the posterior uncertainty in estimated parameters. Thus, a more stringent requirement is to additionally recover the correct posterior variance, for a given dataset $\mathcal{D}$ and model $\mathcal{M}$. One can already see visually in Figure 3.5C, D that posterior uncertainty is larger when the mean is further from the ground truth (lighter shades of grey indicate higher posterior variance). However to be more rigorous one can assess whether the posterior variance is precisely what it should be.

The availability of an analytical likelihood for the DDM, together with our use of sampling methods (as opposed to variational methods which can severely bias posterior variance), allows us to obtain the "ground truth" uncertainty in parameter estimates. Figure 3.6 shows that the sampling from a MLP trained on analytical likelihoods, a MLP trained on KDE-based likelihoods and a CNN all yield excellent recovery of the variance. For an additional run that involved datasets of size

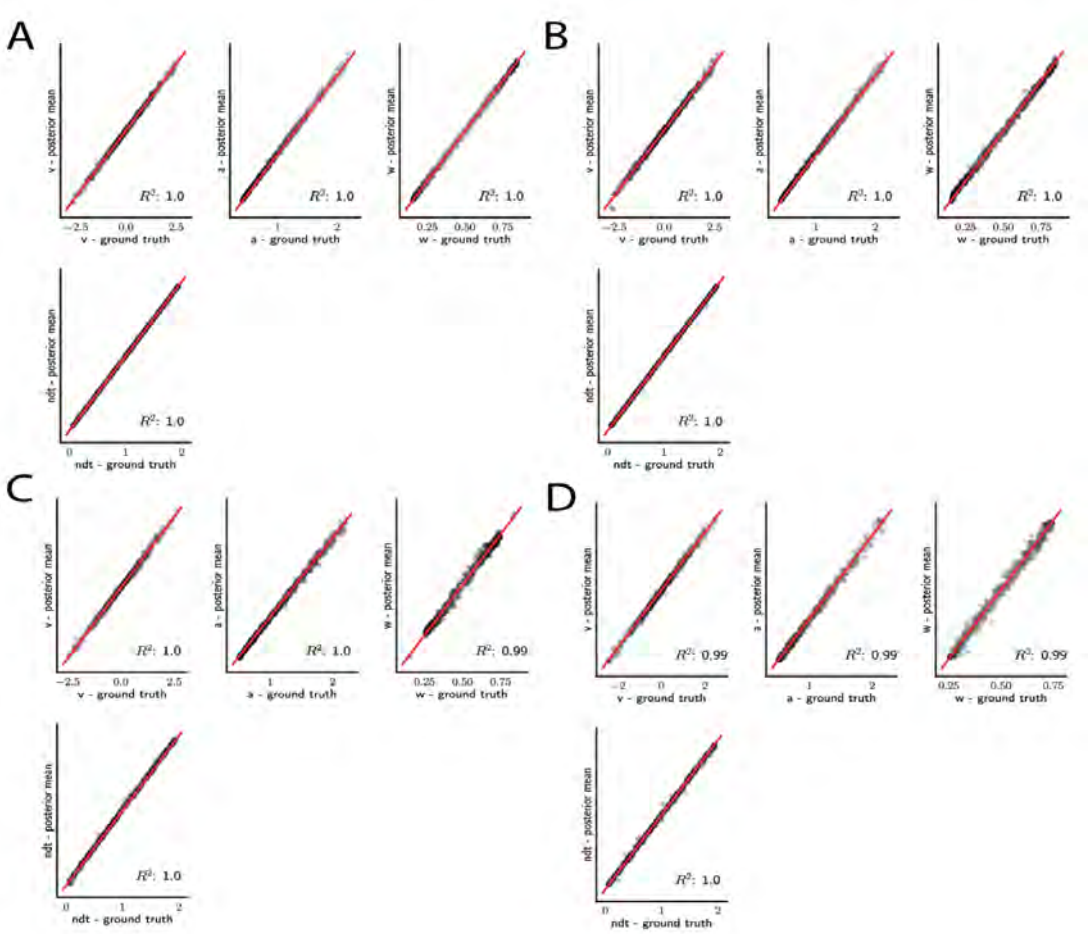

Figure 3.5. Simple DDM Parameter recovery results for, **A** analytic likelihood (ground truth), **B** MLP trained on analytic likelihood, **C** MLP trained on KDE-based likelihoods (100K simulations per KDE), **D** CNN trained on binned likelihoods. The results represent posterior means, based on inference over datasets of size $N_1 = 1024$ "trials". Dot shading is based on parameter-wise normalized posterior variance, with lighter shades indicating larger posterior uncertainty of the parameter estimate.

$n = 4096$ instead of $n = 1024$, we observed a consistent decrease in posterior variance across all methods (not shown) as expected.

No Analytical Likelihood Available

As a proof of concept for the more general ABC setting, we show parameter recovery results for two non-standard models, the LC and WEIBULL models as described in the the Test Bed section. The results are summarized in Figure 3.7 and 3.8 and described in more detail in the following two paragraphs.

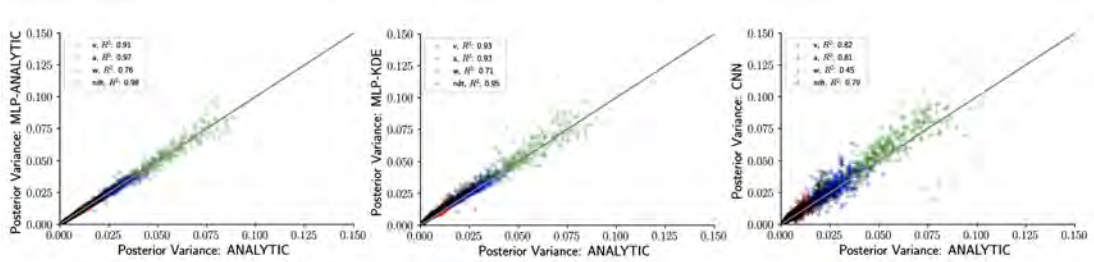

Figure 3.6. Inference using LANs recovers posterior uncertainty. Here we leverage the analytic solution for the DDM to plot the "ground truth" posterior variance on the x-axis, against the posterior variance from the LANs on the y-axis. Left. MLPs trained on the analytic likelihood. Middle. MLPs trained on KDE-based emprical likelihoods. Right. CNNs trained on binned empirical likelihoods. Datasets were equivalent across methods for each model (left to right) and involved $n = 1024$ samples.

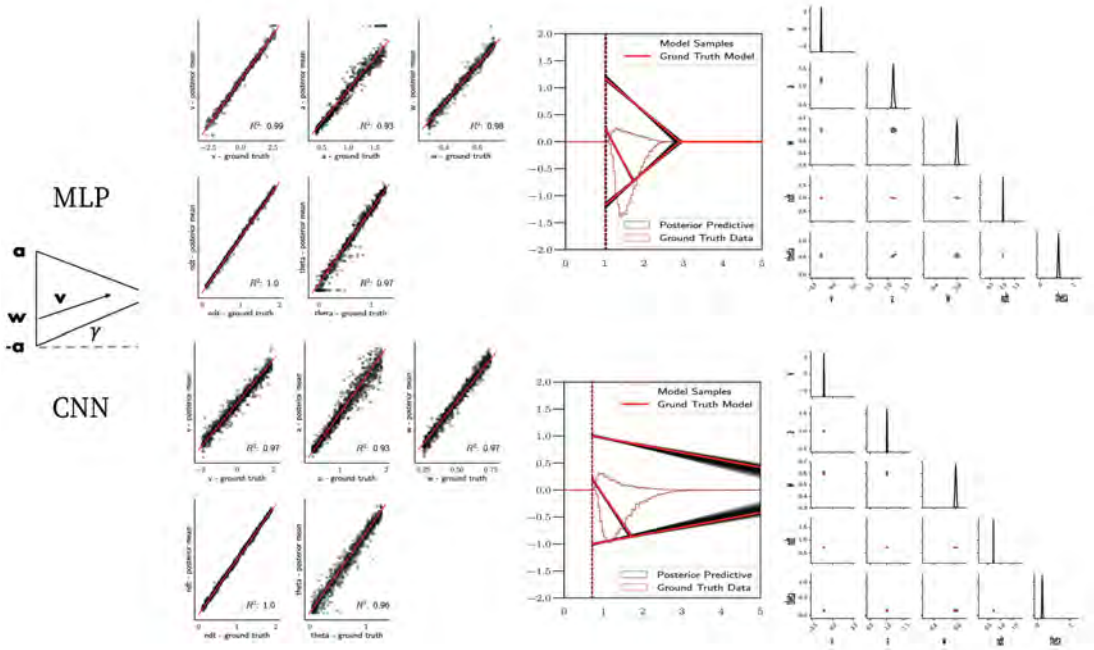

Figure 3.7. LC model parameter recovery and posterior predictives. **Left.** Parameter recovery results for the MLP (top) and CNN (bottom). **Right.** Posterior predictive plots for two representative datasets. Model samples of all parameters (black) match those from the true generative model (red), but one can see that for the lower dataset, the bound trajectory is somewhat more uncertain (more dispersion of the bound). In both cases, the posterior predictive (black histograms) is shown as predicted choice proportions and RT distributions for upper and lower boundary responses, overlaid on top of the ground truth data (red; hardly visible since overlapping / matching).

Parameter Recovery Figure 3.7 shows that both the MLP and CNN methods consistently yield very good to excellent parameter recovery performance for the LC model, with parameter-wise regression coefficients globally above $R^2 > 0.9$. As shown in Figure 3.8 parameter recovery for the WEIBULL model, is less successful however, particularly for the weibull collapsing bound parameters. The drift parameter v, the starting point bias w and the non-decision time are estimated well, however

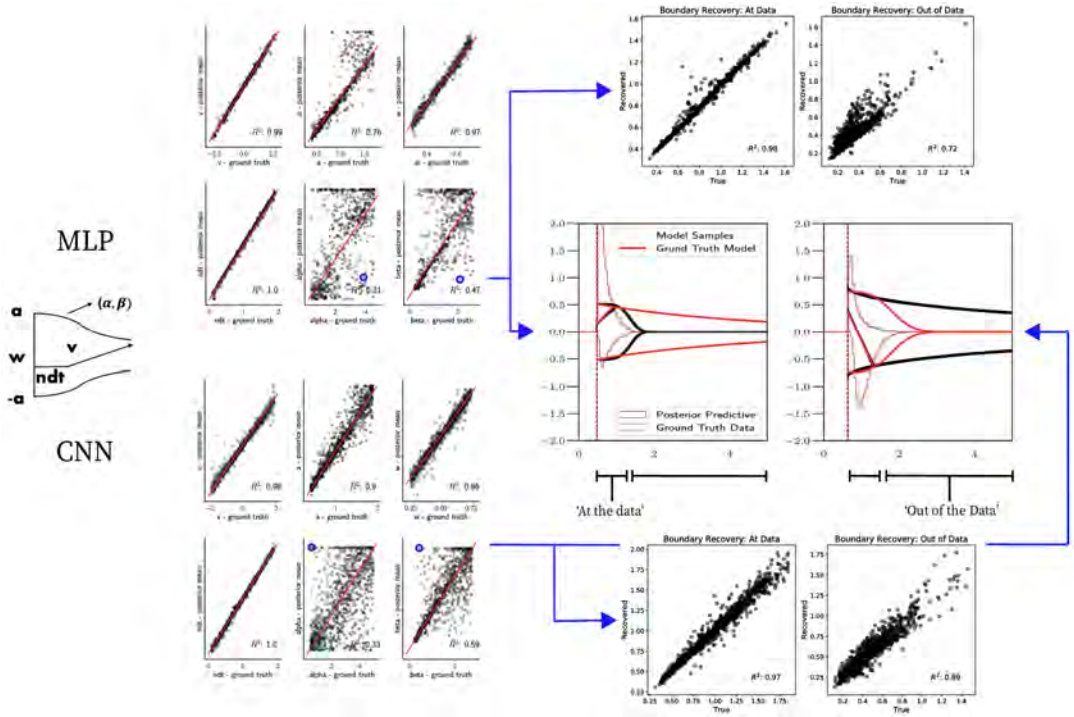

Figure 3.8. WEIBULL model parameter recovery and posterior predictives.**Left.** Parameter recovery results for the MLP (top) and CNN (bottom). **Right.** Posterior predictive plots for two representative datasets in which parameters were poorly estimated (denoted in blue on the left). In these examples, model samples (black) recapitulate the generative parameters (red) for the non-boundary parameters, the recovered bound trajectory is poorly estimated relative to the ground truth, despite excellent posterior predictives in both cases (RT distributions for upper and lower boundary, same scheme as Figure 3.7). Nevertheless, one can see that the net decision boundary is adequately recovered within the range of the RT data that are observed. Across all datasets, the net boundary $B(t) = a * \exp\left(-\frac{t}{\beta}^{\alpha}\right)$ is well recovered within the range of the data observed, and somewhat less so outside of the data, despite poor recovery of individual Weibull parameters α and β.

the boundary parameters a, α and β are less well recovered by the posterior mean. Judging by the parameter recovery plot, the MLP seems to perform slightly less well on the boundary parameters when compared to the CNN.

To interrogate the source of the poor recovery of α and β parameters, we considered the possibility that the model itself may have issues with identifiability, rather than poor fit. Figure 3.8 shows that indeed, for two representative datasets in which these parameters are poorly recovered, the model nearly perfectly reproduces the ground truth data in the posterior predictive RT distributions. Moreover, we find that whereas the individual Weibull parameters are poorly recovered, the net boundary $B(t)$ is very well recovered, particularly when evaluated within the range of the observed dataset. This result is reminiscent of the literature on sloppy models (Gutenkunst et al., 2007), where sloppiness implies that various parameter configurations can have the same impact on the data. Moreover, two further conclusions can be drawn from this analysis. First, when fitting the

WEIBULL model, researchers should interpret the bound trajectory as a latent parameter rather than the individual α and β parameters *per se*. Second, the WEIBULL model may be considered as viable only if the estimated bound trajectory varies sufficiently within the range of the empirical RT distributions. If the bound is instead flat or linearly declining in that range, the simple DDM or LC models may be preferred, and their simpler form would imply that they would be selected by any reasonable model comparison metric. Lastly, given our results the WEIBULL model could likely benefit from re-parameterization if the desire is to recover individual parameters rather than the bound trajectory $B(t)$. Given the common use of this model in collapsing bound studies (Hawkins et al., 2015) and that the bound trajectories are nevertheless interpretable, we leave this issue for future work.

The appendix shows parameter recovery studies on a number of other stochastic simulators with non-analytic likelihoods, described in the Test Bed section. The appendices show tables of parameter wise recovery R^2 for all models tested. In general, recovery ranges from good to excellent. Given the WEIBULL results above, we attribute the less good recovery for some of these models to identifiability issues and specific dataset properties rather than to the method *per se*. We note that our parameter recovery studies here are in general constrained to the simplest inference setting equivalent to a single subject, single condition experimental design. Moreover we use uninformative priors for all parameters of all models. Thus these results provide a lower bound on parameter recoverability, provided of course that the datasets were generated from the valid parameter ranges on which the Networks were trained; please see section 3.5.4 for how recovery can benefit from more complex experimental designs with additional task conditions, which more faithfully represents the typical inference scenario deployed by cognitive neuroscientists. Lastly, some general remarks about the parameter recovery performance. A few factors can negatively impact how well one can recover parameters. First, if the model generally suffers from identifiability issues, the resulting trade-offs in the parameters can lead the MCMC chain to get stuck on the boundary for one or more parameters. This issue is endemic to all models and unrelated to likelihood free methods or LANs, and should at best be attacked at the level of reparameterization (or a different experimental design that can disentangle model parameters). Second, if the generative parameters of a dataset are too close to (or beyond) the bounds of the trained parameter space we may also end with a chain that gets stuck on the boundary of the parameter space. We confronted this problem by training on parameter spaces that yield response time distributions that are broader than typically observed experimentally for models of this class, while also excluding obviously defective parameter setups. Defective parameter setups were defined in the context of our applications as parameter vectors which generate data that never allow one or the other choice to occur (as in $p(c) << \frac{1}{100{,}000}$, data which concentrates more than half of the reaction times within a single $1ms$ bin and data that generated mean reaction times beyond $10s$. These guidelines were chosen as a mix of basic rationale and domain knowledge regarding usual applications of DDMs to experimental data. As such, the definition of defective data may depend on the model under consideration.

3.5.3 Runtime

A major motivation for this work is the amortization of network training time during inference, affording researchers the ability to test a variety of theoretically interesting models for linking brain and behavior without large computational cost. To quantify this advantage we provide some results on the posterior sampling run-times using (1) the MLP with Slice Sampling (Neal, 2003) and (2) CNN with iterated importance sampling.

The MLP timings are based on slice sampling (Neal, 2003), with a minimum of $n = 2000$ samples. The sampler was stopped at some $n >= 2000$, for which the Geweke statistic (Geweke, 1992) indicated convergence (the statistic was computed once every 100 samples for $n >= 2000$). Using an alternative sampler, based on DEMCMC and stopped when the Gelman Rubin $\hat{R} < 1.1$ (Gelman, Rubin, et al., 1992) yielded very similar timing results and was omitted in our figures.

For the reported importance sampling runs we used $200K$ importance samples per iteration, starting with γ values of 64, which was first reduced to 1 where in iteration i, $\gamma_i = \frac{64}{2^{i-1}}$, before a stopping criterion based on relative improvement of the confusion metric was used.

Figure 3.9**A** shows that all models can be estimated in the order of hundreds of seconds (minutes), comprising a speed improvement of at least two orders of magnitude compared to traditional ABC methods using KDE during inference (i.e., the PDA method motivating this work (Turner, Van Maanen, and Forstmann, 2015a)). Indeed, this estimate is a lower bound on the speed-improvement: we extrapolate only the observed difference between network evaluation and online simulations, ignoring the additional cost of constructing and evaluating the KDE-based likelihood. We decided to use this benchmark because it provides a fairer comparison to more recent PDA approaches in which the KDE evaluations can be sped up considerably Holmes, 2015.

Notably, due to its potential for parallelization (especially on GPUs), our neural network methods can even induce performance speed-ups relative to analytic likelihood evaluations. Indeed, figure 3.9**B** shows that as the dataset grows, runtime is significantly faster than even a highly optimized cython implementation of the Navarro Fuss algorithm (Navarro and Fuss, 2009) for evaluation of the analytic DDM likelihood. This is also noteworthy in light of the Full-DDM model (as described in the test-bed section), for which it is currently common to compute the likelihood term via quadrature methods, in turn based on repeated evaluations of the Navarro Fuss algorithm. This can easily inflate the evaluation time by 1 to 2 orders of magnitude. In contrast, evaluation times for the MLP and CNN are only marginally slower (as a function of the slightly larger network size in response to higher dimensional inputs). We confirm (omitted as separate Figure) from experiments with the HDDM python toolbox, that our methods end up approximately $10 - 50$ times faster for the Full-DDM than the current implementation based on numerical integration, maintaining comparable parameter recovery performance. We strongly suspect there to be additional remaining potential for performance optimization.

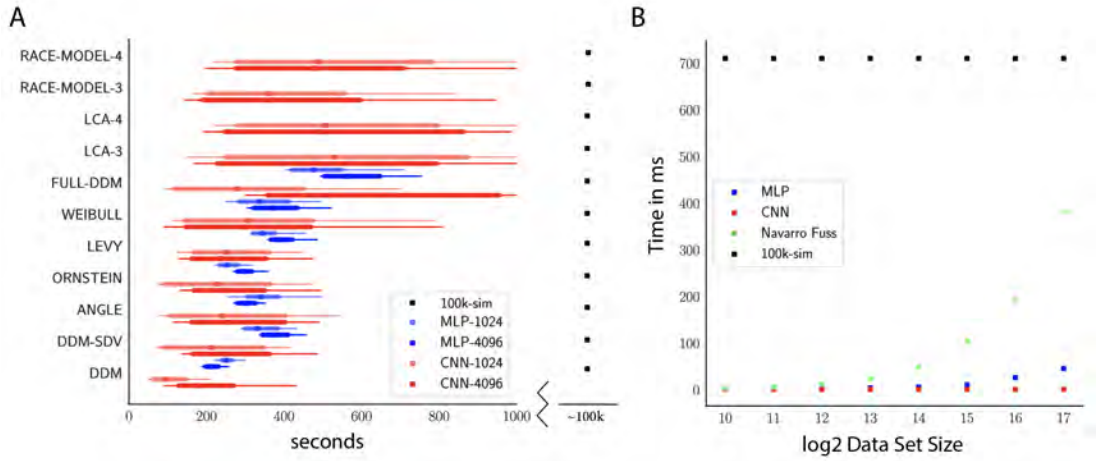

Figure 3.9. **A** Comparison of sampler timings for the MLP and CNN methods, for datasets of size 1024 and 4096 (respectively MLP-1024, MLP-4096, CNN-1024, CNN-4096). For comparison, we include a lower bound estimate of the sample timings using traditional PDA approach during online inference (using 100k online simulations for each parameter vector).100K simulations were used because we found this to be required for sufficiently smooth likelihood evaluations and is the number of simulations used to train our networks; fewer samples can of course be used at the cost of worse estimation, and only marginal speed up since the resulting noise in likelihood evaluations tends to prevent chain-mixing; see Holmes, 2015). We arrive at $100k$ seconds via simple arithmetic. It took our slice samplers on average approximately $200k$ likelihood evaluations to arrive at 2000 samples from the posterior. Taking $500ms * 200000$ gives the reported number. Note that this is a generous but rough estimate, since the cost of data-simulation varies across simulators (usually quite a bit higher than the DDM simulator). Note further that these timings scale linearly with the number of participants and task conditions for the online method, but not for LANs, where they can be in principle be parallelized. **B** Compares the timings for obtaining a single likelihood evaluation for a given dataset. MLP and CNN refer to Tensorflow implementations of the corresponding networks. Navarro Fuss, refers to a cython (Behnel et al., 2010) (cpu) implementation of the algorithm suggested Navarro and Fuss, 2009 for fast evaluation of the analytical likelihood of the DDM. 100k-sim refers to the time it took a highly optimized cython (cpu) version of a DDM-sampler to generate 100k simulations (averaged across 100 parameter vectors).

3.5.4 Hierarchical inference

One of the principal benefits of LANs is that they can be directly extended – without further training – to essentially arbitrary hierarchical inference scenarios, including those in which (i) individual participant parameters are drawn from group distributions; (ii) some parameters are pooled and others separated across task conditions and (iii) neural measures are estimated as regressors on model parameters (Figure 3.10). Hierarchical inference is critical for improving parameter estimation particularly for realistic cognitive neuroscience datasets in which thousands of trials are not available for each participant, and/or where one estimates impacts of noisy physiological signals onto model parameters (Wiecki, Sofer, and Frank, 2013; Boehm et al., 2018; Vandekerckhove, Tuerlinckx, and Lee, 2011; Ratcliff and Childers, 2015).

To provide a proof of concept, we developed an extension to the HDDM python toolbox (Wiecki,

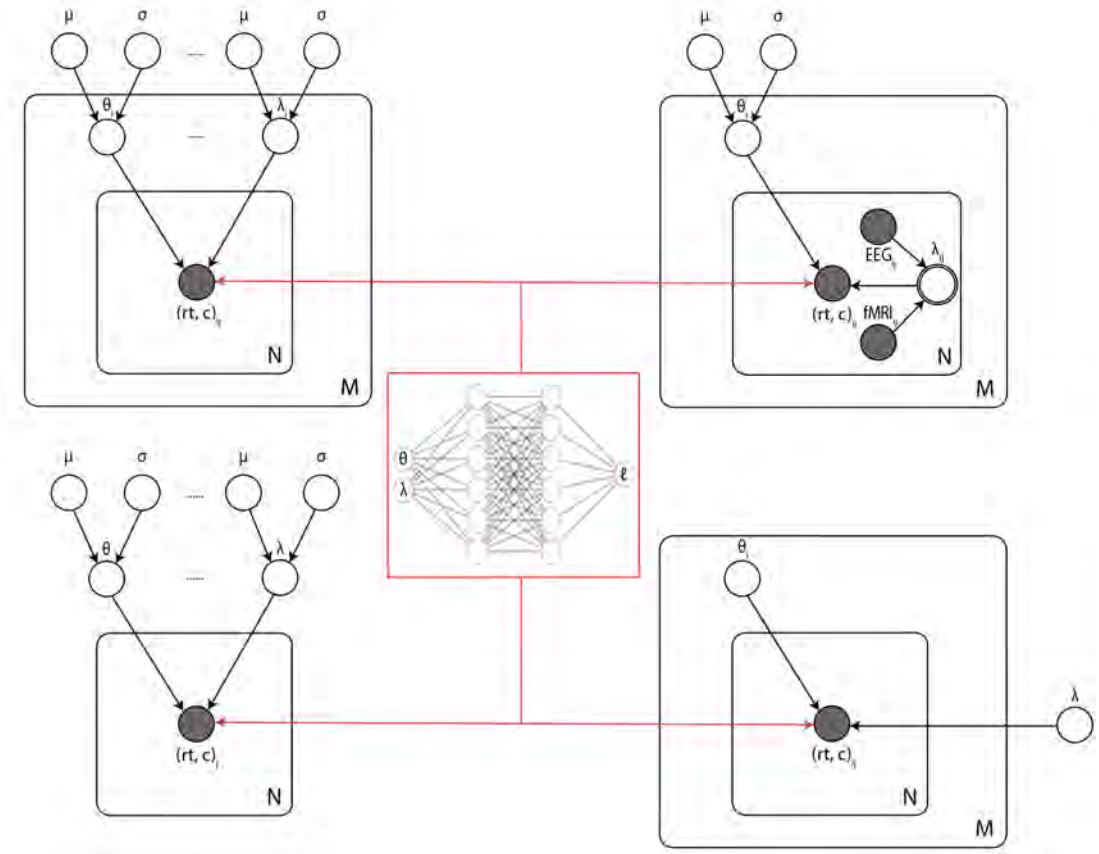

Figure 3.10. Illustrates common inference scenarios applied in the cognitive neurosciences and enabled by our amortization methods. The figure uses standard plate notation for probabilistic graphical models. White single circles represent random variables, white double circles represent variables computed deterministically from their inputs, and grey circles represent observations. For illustration we split the parameter vector of our simulator model (which we call θ in the rest of the paper) into two parts θ and λ, since some but not all parameters may sometimes vary across conditions and / or come from global distribution. (Upper left) basic hierarchical model across **M** participants, with **N** observations (trials) per participant. Parameters for individuals are assumed to be drawn from group distributions. (Upper right) hierarchical models which further estimate the impact of trial-wise neural regressors onto model parameters. (Lower left) non-hierarchical, standard model estimating one set of parameters across all trials. (Lower right), common inference scenario in which a subset of parameters (θ) are estimated to vary across conditions **M**, while others (λ) are global. LANs can be immediately re-purposed for all of these scenarios (and more) without further training.

Sofer, and Frank, 2013), widely used for hierarchical inference of the DDM applied to such settings. Lifting the restriction of previous versions of HDDM to only DDM variants with analytical likelihoods, we imported the MLP likelihoods for all two-choice models considered in this paper. Note that GPU-based computation is supported out of the box, which can easily be exploited with minimal overhead using free versions of Google's Colab notebooks. We generally observed GPUs to improve

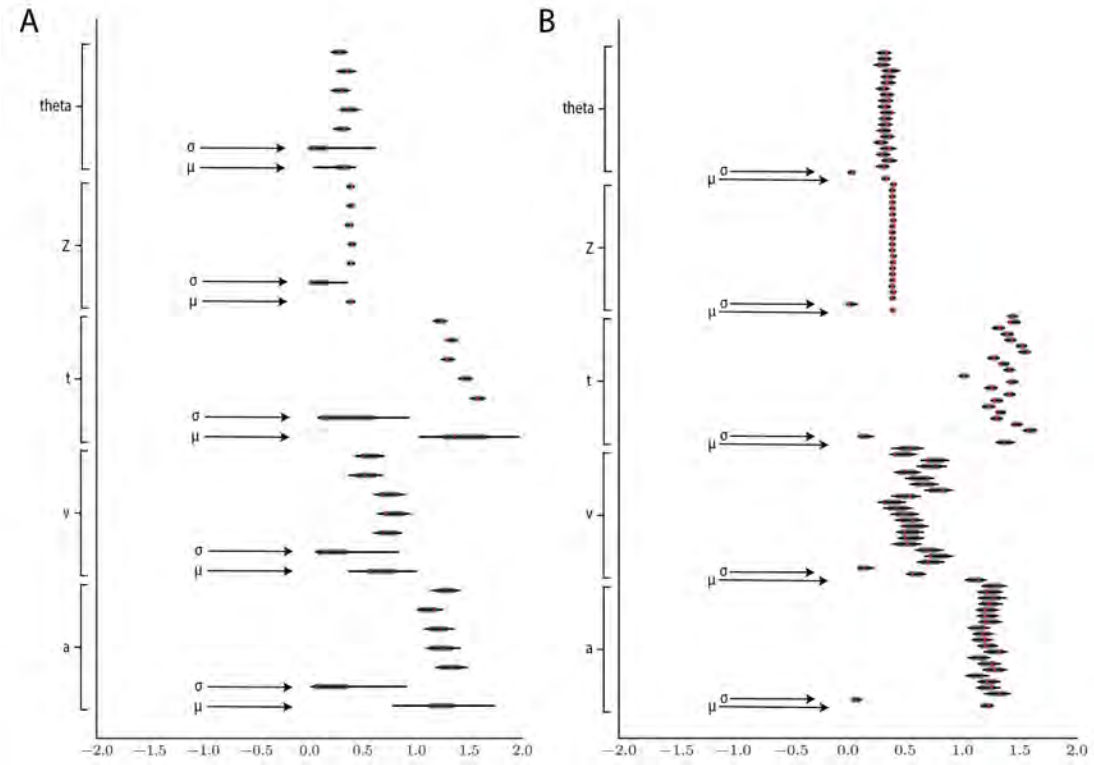

Figure 3.11. Hierarchical inference results using the MLP likelihood imported into the HDDM package. **A** posterior inference for the LC model on a synthetic dataset with 5 participants and 500 trials each. Posterior distributions are shown with caterpillar plots (thick lines correspond to $5-95$ percentiles, thin lines correspond to $1-99$ percentiles) grouped by parameters (ordered from above $\{subject_1, ..., subject_n, \mu_{group}\sigma_{group}\}$). Ground truth simulated values denoted in red. **B** Hierarchical inference for synthetic data comprising 20 participants and 500 trials each. μ and σ indicate the group level mean and variance parameters. Estimates of group level posteriors improve with more participants as expected with hierarchical methods. Individual level parameters are highly accurate for each participant in both scenarios.

speed approximately five-fold over CPU-based setups for the inference scenarios we tested [2].

Figure 3.11 shows example results from hierarchical inference using the LC model, applied to synthetic datasets comprising 5 and 20 subjects (a superset of participants). Recovery of individual parameters was adequate even for 5 participants, and we also observe the expected improvement of recovery of the group level parameters μ and σ for 20 participants.

Figure 3.12 shows an example that illustrates how parameter recovery is affected when a dataset contains multiple experimental conditions (e.g., different difficulty levels). It is common in such scenarios to allow task conditions to affect a single (or subset) of model parameters (in the cases shown: v), while other model parameters are pooled across conditions. As expected, for both the Full-DDM (panel **A**) and the Levy model (panel **B**), the estimation of global parameters is improved

[2]Preliminary access to this interface and corresponding instructions can be found under `https://github.com/lnccbrown/lans/tree/master/hddmnn-tutorial`.

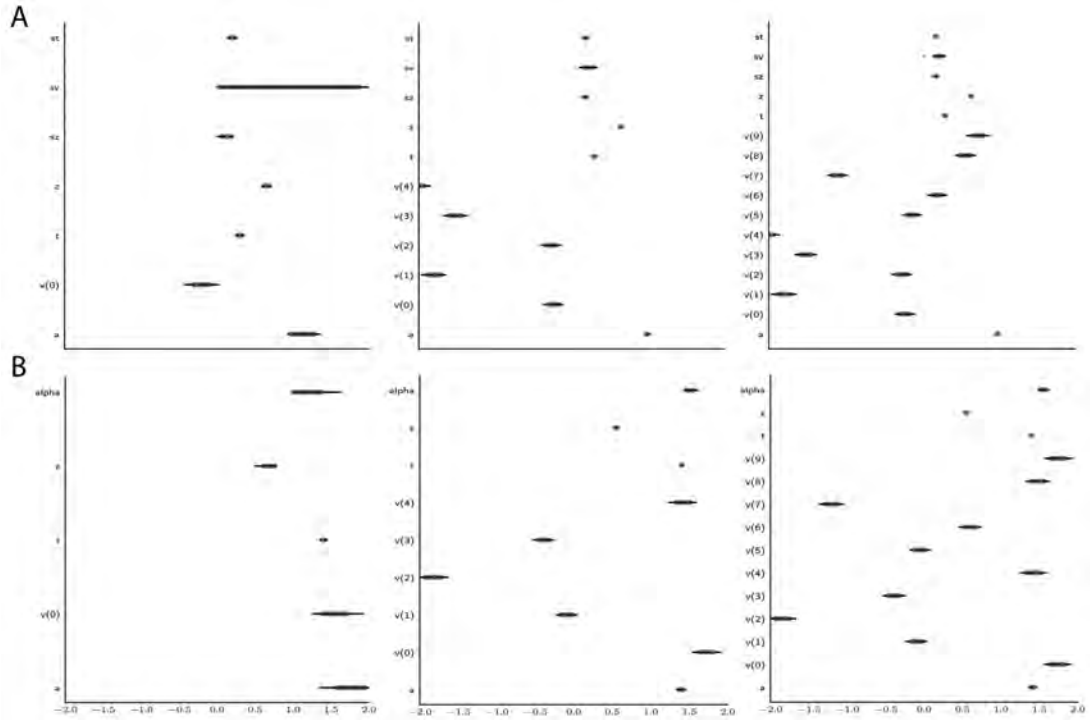

Figure 3.12. Effect of multiple experimental conditions on inference. The panel shows an example of posterior inference for **1**, (left), **5** (middle) and **10** (right) conditions. **A** and **B**, refer to the Full-DDM and Levy models respectively. The drift parameter v is estimated to vary across conditions, while the other parameters are treated as global across conditions. Inference tends to improve for all global parameters when adding experimental conditions. Importantly this is particularly evident for parameters that are otherwise notoriously difficult to estimate, such as sv (trial by trial variance in drift in the Full-DDM model) and α (the noise distribution in the Levy model). Red stripes show the ground truth values of the given parameters.

when increasing the number of conditions from 1 to 5 to 10 (left to right, where the former are subsets of the latter datasets. These experiments confirm that one can more confidently estimate parameters that are otherwise difficult to estimate, such as the noise α in the Levy model and sv the standard deviation of the drift in the Full-DDM.

Both of these experiments provide evidence that our MLPs provide approximate likelihoods which behave in accordance with what is expected from proper analytic methods, while also demonstrating their robustness to other samplers (i.e., we used HDDM slice samplers without further modification for all models).

We expect that proper setting of prior distributions (uniform in our examples) and further refinements to the slice sampler settings (to help mode discovery), can improve these results even further. We include only the MLP method in this section, since it is most immediately amenable to the kind of trial-by-trial level analysis that HDDM is designed for. We plan to investigate the feasibility of including the CNN method into HDDM in future projects.

3.6 Discussion

Our results demonstrate the promise and potential of amortized likelihood approximation networks for Bayesian parameter estimation of neurocognitive process models. Learned manifolds and parameter recovery experiments showed successful inference using a range of network architectures and posterior sampling algorithms, demonstrating the robustness of the approach.

Although these methods are extendable to any model of similar complexity, we focused here on a class of sequential sampling models, primarily because the most popular of them – the DDM – has an analytic solution, and is often applied to neural and cognitive data. Even slight departures from the standard DDM framework (e.g., dynamic bounds, or changes in the noise distribution) are often not considered for full Bayesian inference due to the computational complexity associated with traditional ABC methods. We provide access to the learned likelihood functions (in the form of network weights) and code to enable users to fit a variety of such models with orders of magnitude speed up (minutes instead of days). In particular, we provided an extension to the commonly used HDDM toolbox (Wiecki, Sofer, and Frank, 2013) that allows users to apply these models to their own datasets immediately. We also provide access to code that would allow users to train their own likelihood networks and perform recovery experiments, which can then be made available to the community.

We offered two separate approaches with their own relative advantages and weaknesses. The MLP is suited for evaluating likelihoods of individual observations (choices, response times) given model parameters, and as such can be easily extended to hierarchical inference settings and trial-by-trial regression of neural activity onto model parameters. We showed that importing the MLP likelihood functions into the HDDM toolbox affords fast inference over a variety of models without tractable likelihood functions. Moreover, these experiments demonstrated that use of the neural network likelihoods even confers a performance speed up over the analytic likelihood function – particularly for the Full-DDM, which otherwise required numerical methods on top of the analytic likelihood function for the simple DDM.

Conversely, the CNN approach is well suited for estimating likelihoods across parameters for entire datasets in parallel, as implemented with importance sampling. More generally and implying potential further improvements, any Sequential Monte Carlo (SMC) method may be applied instead. These methods offer a more robust path to sampling from multimodal posteriors compared to MCMC, at the cost of the curse of dimensionality, rendering them potentially less useful for highly parameterized problems, such as those that require hierarchical inference. Moreover, representing the problem directly as one of learning probability distributions, and enforcing the appropriate constraints by design endows the CNN approach with a certain conceptual advantage. Finally we note that in principle (with further improvements) trial-level inference is possible with the CNN approach, and vice-versa, importance sampling can be applied to the MLP approach.

In this work, we employed sampling methods (MCMC and importance sampling) for posterior inference, because in the limit they are well known to allow for accurate estimation of posterior distributions on model parameters, including not only mean estimates but their variances and

covariances. Accurate estimation of posterior variances is critical for any hypothesis testing scenario, because it allows one to be confident about the degree of uncertainty in parameter estimates. Indeed, we showed that for the simple DDM, we found that posterior inference using our networks yielded nearly perfect estimation of the variances of model parameters (which are available due to the analytic solution). Of course, our networks can also be deployed for other estimation methods even more rapidly: they can be immediately used for maximum likelihood estimation via gradient descent, or within other approximate inference methods, such as variational inference (see Acerbi, 2020 for a related approach).

Other approaches exist for estimating generalized diffusion models. A recent example, not discussed thus far, is the pyDDM Python toolbox (Shinn, Lam, and J. D. Murray, 2020), which allows maximum likelihood estimation of generalized drift diffusion models (GDDM). The underlying solver is based on the Fokker-Planck equations, which allow access to approximate likelihoods (where the degree of approximation is traded off with computation time / discretization granularity) for a flexible class of diffusion-based models, notably allowing arbitrary evidence trajectories, starting point and non-decision time distributions. However, to incorporate trial-by-trial effects would severely inflate computation time (on the order of the number of trials), since the solver would have operate on a trial by trial level. Moreover, any model that is not driven by Gaussian Diffusion, such as the Levy model we considered here, or the linear ballistic accumulator, is out of scope with this method. In contrast LANs can be trained to estimate any such model, limited only by the identifiability of the generative model itself. Finally, pyDDM does not afford full Bayesian estimation and thus quantification of parameter uncertainty and covariance.

We moreover note that our LAN approach can be useful even if the underlying simulation model admits other likelihood approximations, regardless of trial-by-trial effect considerations, since a forward pass through a LAN may be speedier. Indeed, we observed substantial speed ups in HDDM for using our LAN method to the full DDM, for which numerical methods were previously needed to integrate over inter-trial variability.

We emphasize that our test-bed application to sequential sampling models does not delimit the scope of application of LANs. Neither are reasonable architectures restricted to MLPs and CNNs (see Lueckmann et al., 2019b; Papamakarios, Sterratt, and I. Murray, 2019 for related approaches which use completely different architectures). Models with high-dimensional (roughly > 15) parameter spaces may present a challenge for our global amortization approach, due to the curse of dimensionality. Further, models with discrete parameters of high cardinality may equally present a given network with training difficulties. In such cases other methods may be preferred over likelihood amortization generally (e.g., Acerbi, 2020), given that this is an open and active area of research we can expect surprising developments which may do in fact turn the tide again in the near future.

Despite some constraints this still leaves a vast array of models in reach for LANs, of which our test-bed can be considered only a small beginning.

By focusing on likelihood approximation networks, our approach affords the flexibility of networks serving as plug-ins for hierarchical or arbitrarily complex model extensions. In particular, the

networks can be immediately transferred, without further training, to essentially arbitrary inference scenarios in which researchers may be interested in evaluating links between neural measures and model parameters, and to compare various assumptions about whether parameters are pooled and split across experimental manipulations. This flexibility in turn sets our methods apart from other amortization and neural network based ABC approaches offered in the statistics, machine learning and computational neuroscience literature (Papamakarios and I. Murray, 2016; Papamakarios, Nalisnick, et al., 2019; Gonçalves et al., 2020; Lueckmann et al., 2019a; S. T. Radev, Mertens, Voss, and Köthe, 2020; S. T. Radev, Mertens, Voss, Ardizzone, et al., 2020) , while staying conceptually extremely simple. Instead of focusing on extremely fast inference for very specialized inference scenarios, our approach focuses on achieving speedy inference while not implicitly compromising modeling flexibility through amortization step.

Closest to our approach is the work of Lueckmann et al., 2019b, and Papamakarios, Sterratt, and I. Murray, 2019, both of which attempt to target the likelihood with a neural density estimator. While flexible, both approaches imply the usage of summary statistics, instead of a focus on trial-wise likelihood functions. Our work can be considered a simple alternative with explicit focus on trial-wise likelihoods.

Besides deep learning based approaches, another major machine learning inspired branch of the ABC literature, concerns log-likelihood and posterior approximations via Gaussian Process Surrogates (GPSs) (Meeds and Welling, 2014; Järvenpää et al., 2018; Järvenpää et al., 2021; Acerbi, 2020). A major benefit of GPSs lies in the ability for clever training data selection via active learning, since such GPSs allow uncertainty quantification out of the box, which in turn can be utilized for the purpose of targeting high uncertainty regions in parameter space. GPS based computations scale with the number of training examples however, which make them much more suitable for minimizing the computational cost for a given inference scenario, than facilitating global amortization as we suggest in this paper (for which one usually need larger sets of training data than can traditionally be handled efficiently by GPS). Again when our approach is applicable, it will offer vastly greater flexibility, once a LAN is trained.

3.7 Limitations and Future Work

There are several limitations of the methods presented in this article, which we hope to address in future work. While allowing for great flexibility, the MLP approach suffers from the drawback that we don't enforce (or exploit) the constraint that $\hat{\ell}(\theta|\mathbf{x})$ is a valid probability distribution, and hence the networks have to learn this constraint implicitly and approximately. Enforcing this constraint, has the potential to improve estimation of tail probabilities (a known issue for KDE approaches to ABC more generally (Turner, Van Maanen, and Forstmann, 2015a)).

The CNN encapsulation exploits the fact that $\int_{\mathcal{X}} \hat{\ell}(\theta|\mathbf{x})d\mathbf{x} = 1$, however makes estimation of trial-by-trial effects more resource hungry. We plan to investigate the potential of the CNN for trial-by-trial estimation in future research.

A potential solution that combines the strengths of both the CNN and MLP methods, is to utilize Mixture Density Networks to encapsulate the likelihood functions. We are currently exploring this avenue. Mixture density networks have been successfully applied in the context of ABC (Papamakarios and I. Murray, 2016), however training can be unstable without extra care (Guillaumes, 2017). Similarly, invertible flows (Rezende and Mohamed, 2015) and / or mixture density networks Bishop, 1994 may be used to learn likelihood functions (Papamakarios, Sterratt, and I. Murray, 2019; Lueckmann et al., 2019b), however the philosophy remains focused on distributions of summary statistics for single datasets. While impressive improvements have materialized at the intersection of ABC and Deep Learning methods (Papamakarios, Nalisnick, et al., 2019; Greenberg, Nonnenmacher, and Macke, 2019; Gonçalves et al., 2020) (showing some success with posterior amortization for models up to 30 parameters,but restricted to a local region of high posterior density in the resulting parameter space), generally less attention has been paid to amortization methods that are not only of case-specific efficiency but sufficiently modular to serve a large variety of inference scenarios (e.g., Figure 3.10). This is an important gap, which we believe the popularization of the powerful ABC framework in the domain of experimental science hinges upon. A second, and short-term avenue for future work is the incorporation of our presented methods into the HDDM Python Toolbox (Wiecki, Sofer, and Frank, 2013) to extend its capabilities to a larger variety of SSMs. Initial work in this direction is completed, alpha version of the extension being available in form of a tutorial under `https://github.com/lnccbrown/lans/tree/master/hddmnn-tutorial`.

Our current training pipeline can be further optimized on two fronts. First, no attempt was made to minimize the size of the network needed to reliably approximate likelihood functions so as to further improve computational speed. Second, little attempt was made to optimize the amount of training provided to networks. For the models explored here, we found it sufficient to simply train the networks for a very large number of simulated datapoints such that interpolation across the manifold was possible. However, as model complexity increases, it would be useful to obtain a measure of the networks' uncertainty over likelihood estimates for any given parameter vector. Such uncertainty estimates would be beneficial for multiple reasons. One such benefit would be to provide a handle on the reliability of sampling, given the parameter region. Moreover, such uncertainty estimates could be used to guide the online generation of training data to train the networks in regions with high uncertainty. At the intersection of ABC and Neural Networks active learning has been explored via uncertainty estimates based on network ensembles (Lueckmann et al., 2019a). We plan to additionally explore the use of Bayesian neural networks, which provide uncertainty over their weights, for this purpose (Neal, 1995).

One more general shortcoming of our methods is the reliance on empirical likelihoods for training, which in turn are based on a fixed number of samples across parameter vectors, just as the PDA method proposed by Turner, Van Maanen, and Forstmann, 2015a. Recently this approach has been criticized fundamentally on grounds of producing bias in the generated KDE based likelihood estimates (Opheusden, Acerbi, and Ma, 2020). A reduction of the approximate likelihood problem to one of inverse binomial sampling was proposed (Opheusden, Acerbi, and Ma, 2020), which will

generate unbiased likelihood estimates. To address this concern, we will investigate adaptive strategies for the selection of the simulations count n. We however highlight two points here which aim to put the promise of unbiased likelihoods in perspective. First, our networks add interpolation to the actual estimation of a likelihood. Likelihoods close in parameter space therefore share information, which translates into an effectively higher simulation count than the $100k$ chosen to construct each empirical likelihood used for training. Quantifying this benefit precisely we leave for future research, however we suspect, as suggested by Figure 3.4, that it may be substantial. Second, while we generally acknowledge that bias in the tails remains somewhat of an issue in our approach, resolution is at best partial even in the proposed methods of (Opheusden, Acerbi, and Ma, 2020). For the estimation of parameters for which a given datapoint is extremely unlikely (i.e., the data is generally unlikely under the model), the authors suggest to threshold the simulation count so that their algorithm is guaranteed to stop. This effectively amounts to explicitly allowing for bias again. As another alternative the authors suggest to introduce a lapse rate in the generative model, which the LAN approach can accommodate as well. However the introduction of a lapse rate doesn't deal with tail events directly either, but rather assumes that tail events are unrelated to the process of interest. This in turn will render a lower but fixed number of simulations N feasible for training LANs as well. This is notwithstanding the desirable minimization of simulation times even for high likelihood events, especially when trial wise simulations are in fact necessary (which tends to be in cases where amortization with LANs is a priori not a good computational strategy to begin with). Hence, although the inverse binomial sampling approach is elegant conceptually, excessive computation remains an issue when we need accurate estimates of the probability of actual tail-events. Generally however, we maintain it desirable and important for future work to make use of the otherwise great potential of adaptive sampling to minimize total computational cost.

Furthermore we relegate to future research proper exploitation of the fact that LANs are by design differentiable in the parameters. We are currently working on an integration of LANs with tensorflow probability (Abadi et al., 2016), utilizing autograd to switch our MCMC method to the gradient-based NUTS sampler (Hoffman and Gelman, 2014). Main benefits of this sampler are robust mixing behavior, tolerance for high levels of correlations in the parameter space, while at the same time maintaining the ability to sample from high dimensional posteriors. High level of correlations in posteriors are traditionally an Achilles heel of the otherwise robust coordinate wise slice samplers. DEMCMC and Iterated Importance samplers are somewhat more robust in this regards, however both may not scale efficiently to high dimensional problems. Robustness concerns aside, initial numerical results additionally show some promising further speed-ups.

Another important branch for future work lies in the utilization of LANs for model comparison. Initial results are promising in that we obtained satisfactory model recovery using the Deviance Information Criterion (DIC) used for model selection in the standard HDDM package. However this issue demands much more attention to evaluate other model selection metrics and extensive further numerical experiments, which we relegate to future work.

Lastly, in contrast to the importance sampler driving the posterior inference for the CNN, we

believe that some of the performance deficiencies of the MLP, are the result of our Markov Chains not having converged to the target distribution. A common problem seems to be that the sampler hits the bounds of the constrained parameter space and does not recover from that. As we show in Figure 3.7 and Figure 3.8, even ostensibly bad parameter recoveries follow a conceptual coherence and lead to good posterior predictive performance. We therefore may be under-reporting the performance of the MLP and plan to test the method on an even more comprehensive suite of MCMC samplers, moreover including thus far neglected potential for re-parameterization.

Acknowledgements

This work was funded by NIMH grants P50 MH 119467-01 and R01 MH084840-08A1. We thank Michael Shvartsman, Matthew Nassar and Thomas Serre for helpful comments and discussion regarding earlier versions of this manuscript. Furthermore, we thank Mads Lund Pederson for help with integrating our methods into the HDDM python toolbox. Lastly we would like to thank the two reviewers of the manuscript for helpful suggestions, which improved the readability of the manuscript.

3.8 Materials and Methods

3.8.1 Test-Beds

General Information

All models were simulated using the Euler-Maruyama method, which for some fixed discretization step-size Δt, evolves the process as,

$$X_{t+\Delta t} = X_t + a(t,x)\Delta t + b(t,x)\Delta \mathbf{B}$$

where the definition of $\Delta \mathbf{B}$ depends on the noise process. For simple Brownian Motion this translates into Gaussian displacements, specifically $\Delta \mathbf{B} \sim \mathcal{N}(0, \Delta t)$, which is commonly denoted as $d\mathbf{W}$. More generally the noise need not be Gaussian, and indeed we later apply our methods to the Levy Flight model for which the noise process is an alpha stable distribution, denoted as $\mathbf{L}_\alpha$ s.t. $\Delta \mathbf{L}_\alpha \sim (\Delta t)^{\frac{1}{\alpha}} \mathbf{L}(\alpha, 0, 1, 0)$.

The models chosen for our test-bed systematically vary different aspects of complexity, as illustrated in 3.3. The DDM provides a benchmark and a sanity check since we can compute its likelihood analytically. The Full-DDM provides us with a model for which analytical computations are still based on the analytical likelihood of the DDM, however evaluation is slowed by the necessity for numerical integration. This forms a first test for the speed of evaluation of our methods. For the Ornstein Uhlenbeck, Levy, Race and DDM with parameterized boundary models we cannot base our calculations on an analytical likelihood, but we can nevertheless perform parameter recovery and compare to other methods that utilize empirical likelihoods. The Ornstein Uhlenbeck Model adds state-dependent behavior to the diffusion while the Levy Model adds variation in the noise process and the Race Models expand the output dimensions according to the number of choices.

Full Drift Diffusion Model

The Full Drift Diffusion Model (Full-DDM) maintains the same specification for the driving SDE, but also allows for trial-to-trial variability in three parameters (Ratcliff and McKoon, 2008). We allow the drift rate v to vary trial by trial, according to a normal distribution, $v \sim \mathcal{N}(0, \sigma_v)$, the non-decision-time τ to vary according to a uniform distribution $\tau \sim \mathbf{U}[-\epsilon_\tau, \epsilon_\tau]$ and the starting point w to vary according to a uniform distribution as well $w \sim \mathbf{U}[-\epsilon_w, \epsilon_w]$. The parameter vector for the Full-DDM model is then $\theta = (v, a, w, \tau, \sigma_v, \epsilon_\tau, \epsilon_w)$.

To calculate the FPTD for this model, we can use the analytical likelihood expression from the DDM. However, we need to use numerical integration to take into account the random parameters (Wiecki, Sofer, and Frank, 2013). This inflates execution time by a factor equivalent to the number of executions needed to compute the numerical integral.

Ornstein Uhlenbeck Model

The Ornstein Uhlenbeck model introduces a state-dependency on the drift rate v. Here $a(t, x) = v + g * x$, where g is an inhibition / excitation parameter. If $g < 0$ it acts as a leak (the particle is mean reverting). If $g > 0$ the particle accelerates away from the 0 state, as in an attractor model. At $g = 0$ we recover the simple DDM process. This leaves us with a parameter vector $\theta = (v, a, w, \tau, g)$. The corresponding SDE is defined as,

$$d\mathbf{X}_{\tau+t} = (v + g * \mathbf{X}_t)\, dt + d\mathbf{W},\ \mathbf{X}_\tau = w$$

This model does not have an analytical likelihood function that can be employed for cheap inference (Mullowney and Iyengar, 2006). We discuss alternatives, other than our proposed methods, to simple analytical likelihoods later. For our purposes approximate inference is necessary for this model. [3].

Levy Flights

The Levy Flight (Wieschen, Voss, and S. Radev, 2020; Reynolds and Rhodes, 2009) model dispenses with the Gaussian noise assumption in that the incremental noise process instead follows an alpha-stable distribution $\mathcal{L}_\alpha$. Specifically, we consider distributions $\mathcal{L}(\alpha, 0, 1, 0)$ which are centered at 0, symmetric and have unitary scale parameter. These distributions have a first moment for $\alpha \in (1, 2]$, but infinite variance for $\alpha < 2$. An important special case is $\alpha = 2$, where $\mathcal{L}(2, 0, 1, 0) = \mathcal{N}(0, 2)$. The parameter vector for this process is $\theta = (v, a, w, \tau, \alpha)$. We fix $a(t, x) = v$ and $b(t, x) = 1$. The SDE is defined as,

$$d\mathbf{X}_{\tau+t} = v\, dt + d\mathbf{L}_\alpha,\ \ \mathbf{X}_\tau = w$$

[3]The Ornstein Uhlenbeck model is usually defined only for $g < 0$, our parameter space makes it strictly speaking a relaxation

The Levy Flight is a flexible model used across disciplines for some of its theoretical optimality properties (Wosniack et al., 2017) despite not possessing closed-form FPTD's. We add it here, as it is different from the other models under consideration; in principle, it could also capture decision-making scenarios in which there are sudden jumps in the accumulation of evidence (e.g., due to internal changes in attention). Its behavior is shaped by altering the properties of the incremental noise process directly.

Parameterized Collapsing Decision Bounds

We will consider variations of the DDM in which the decision boundary is not fixed but is time-varying (represented by a boundary parameter a with a parameterized boundary function $h(t;\theta_h)$). In such cases we augment the parameter vector θ with the set θ_h and drop a. Such variations are optimal in a variety of settings (for example, when there are response deadlines (Frazier and Angela, 2008) or distributions of trial-types with different difficulties (Malhotra et al., 2018; Palestro et al., 2018)), and also better reflect the underlying dynamics of decision bounds within biologically inspired neural models (O'Reilly and Frank, 2006; Ratcliff and Frank, 2012; Wiecki and Frank, 2013). The boundary functions considered in the following are the Weibull bound (WEIBULL),

$$b_{WB}(t;a,\alpha,\beta) = a * \exp\left(-\frac{t}{\beta}^{\alpha}\right)$$

and the linear collapse bound (LC),

$$b_{LC}(t;a,\theta) = a - \left(t * \frac{\sin(\theta)}{\cos(\theta)}\right)$$

Race Models: N > 2

The Race Model departs from previous model formulations in that it has a particle for each of N choice options, instead of a single particle representing the evidence for one option over another. The function $f_{E_i}(t,\theta)$ now represents the probability of particle i to be the first of all particle to cross the bound a at time t. We consider race models for which the drift and starting point can vary for each particle separately. Treating the boundary as a constant a leaves us with a parameter vector $\theta = (v_1,..,v_n,a,w_1,...,w_n,ndt)$. The SDE is defined, for each particle separately (or in vector form) as,

$$d\mathbf{X}^i_{\tau+t} = v^i\,dt + d\mathbf{W}, \quad \mathbf{X}^i_0 = ... = \mathbf{X}^i_\tau = w^i$$

These models represent the most straightforward extension to a multi-choice scenario.

3.8.2 MLP

Network Specifics

We apply the same simple architecture consistently across all example contexts in this paper. Our networks have 3 hidden layers, $\{L_1, L_2, L_3\}$ of sizes $\{100, 100, 120\}$, each using $tanh(.)$ activation

functions. The output layer consists of a single node with linear activation function.

Training Process

Training Hyperparameters: The network is trained via stochastic back-propagation using the Adam (Kingma and Ba, 2014) optimization algorithm. As a loss function, we utilize the huber loss (Huber, 1992) defined as,

$$f(|y-\hat{y}|) = \begin{cases} 0.5 * |y-\hat{y}|^2 & \text{if } |y-\hat{y}| \leq 1 \\ 0.5 + |y-\hat{y}| & \text{if } |y-\hat{y}| > 1 \end{cases}$$

Training Data: We used the following approach to generate training data across all examples shown below.

First, we generate $100K$ simulations from the stochastic simulator (or model $\mathcal{M}$)), for each of $1.5M$ parameter configurations. Since for the examples we consider, the stochasticity underlying the models are in the form of a stochastic differential equation (SDE), all simulations were conducted using the simple Euler-Maruyama method with timesteps δt of $0.001s$. The maximum time we allowed the algorithms to run was $20s$, much more than necessary for a normal application of the simulator models under consideration.

Based on these simulations, we then generate empirical likelihood functions using kernel density estimators (KDEs) (Turner, Van Maanen, and Forstmann, 2015a). KDEs use atomic datapoints $\{x_0, ..., x_N\}$ and reformulate them into a continuous probability distribution $f(y; \mathbf{x}) = \sum_i^N K(\frac{y-x}{h})$, where we choose $K(.)$ as a standard Gaussian kernel $f(x) = \frac{1}{\sqrt{2\pi}} \exp -\frac{x^2}{2}$, and h the so-called bandwidth parameter, is set by utilizing Silverman's rule of thumb (Silverman, 1986). Where the data made Silverman's rule inapplicable we set a lower bound on h as 10^{-3}. Additionally we follow (Charpentier and Flachaire, 2015) in transforming our KDE to accommodate positive random variables with skewed distributions (in adherence to the properties of data resulting from the response time models forming our examples).

To ensure that the networks accurately learn likelihoods across a range of plausible data, for each parameter set we trained the networks by sampling 1000 datapoints from a mixture distribution with three components (mixture probabilities respectively $\{0.8, 0.1, 0.1\}$. The first component draws samples directly from the KDE-distributions. The second component is uniform on $[0s, 20s]$, and the third component samples uniformly on $[-1s, 0s]$. The aim of this mixture is to allow the network to see, for each parameter setting of the stochastic simulator, training examples of three kinds: (1) "Where it matters", that is where the bulk of the probability mass is given the generative model. (2) Regions of low probability to inform the likelihood estimate in those regions (i.e. to prevent distortion of likelihood estimates for datapoints that are unlikely to be generated under the model). (3) Examples on the negative real line to ensure that it is learned to consistently drive likelihood predictions to 0 for datapoints close to 0.

The supervision signal for training has two components. For positive datapoints (reaction times in our examples), we evaluate the log likelihood according to our KDE. Likelihoods of negative

datapoints were set to an arbitrary low value of 10^{-29} (a log likelihood of -66.79). 10^{-29} also served as the lower bounds on likelihood evaluations. While this constrains our accuracy on the very tails of distributions, extremely low evaluations unduly affect the training procedure. Since the generation of training data can easily be parallelized across machines, we simply front-loaded the data generation accordingly. We refer back to Figure 3.2 for a conceptual overview.

This procedure yields $1.5B$ labeled training examples on which we train the network. We applied early stopping upon a lack of loss improvement for more than 5 epochs of training. All model were implemented using Tensorflow (Abadi et al., 2016).

We note here that this amount of training examples is likely an overshoot by potentially one or more orders of magnitude. We did not systematically test for the minimum amount of training examples needed to train the networks. Minimal experiments we ran showed that roughly one tenth of the training examples lead to very much equivalent training results. Systematic minimization of the training data is left for future numerical experiments, since we don't deem it essential for purposes of a proof of concept.

Sampling algorithms

Once trained, we can now run standard Markov Chain Monte Carlo schemes, where instead of an analytical likelihood, we evaluate $f_w(\mathbf{x}, \theta)$ as a forward pass through the MLP. Figure 3.1 B schematically illustrates this approach (following the green arrows), and contrasts with currently applied methods (red arrows). We report multiple so-conducted parameter recovery experiments in the Results section and validate the approach first with models with known analytic likelihood functions.

Regarding sampling we utilized two MCMC algorithms, which showed generally very similar results. In contrast to the importance sampling algorithm used for the CNN (described below), MCMC methods are known for having trouble with multimodal posteriors. Running our experiments across algorithms was a safeguard against incorporating sampler specific deficiencies into our analysis. We however acknowledge that even more extensive experiments may be necessary for comprehensive guarantees. First, having an ultimate implementation of our method into the HDDM Python toolbox (Wiecki, Sofer, and Frank, 2013) in view, we use slice sampling (as used by the toolbox), specifically the step-out procedure following Neal, 2003. Second, we used a custom implementation of the DE-MCMC algorithm (Ter Braak, 2006), known for being robust in higher dimensional parameter spaces. Our DE-MCMC implementation adds reflecting boundaries to counteract problematic behavior when the sampler attempts to move beyond the parameter space, which is truncated by the (broad) range of parameters in which the MLP was trained. The number of chains we use is consistently determined as $5 * |\theta|$, five times the number of parameters of a given stochastic model. Samplers were initialized, by using slight perturbations of 5 maximum likelihood estimates, computed via differential evolution optimization (Storn and Price, 1997; Virtanen et al., 2020). Since results were very similar across samplers, we restrict ourselves mostly to reporting results derived from the slice sampler, given that

this sampler forms the back-end of the HDDM user interface we envision. [4].

Additional Notes

Note that we restricted parameter recovery for the MLP to datasets which distributed at least 5% of choices to the less frequently chosen option. This modest filtering accommodates the fact that such data-sets were also excluded form the training data for the MLP model, since they (1) present difficulties for the KDE estimator, (2) lead to generally less stable parameter estimates (i.e., it is not advisable to use diffusion models when choices are deterministic).

3.8.3 CNN

Network Specifics

The CNN takes as an input a parameter vector θ, giving as output a discrete probability distribution over the relevant dataspace. In the context of our examples below the output space is of dimensions $\mathcal{R}^{N_c} \times \mathcal{R}^{N_d}$, where N_c is the number of relevant choice alternatives, and N_d is the number of bins for the reaction time for each choice ($N_d = 512$ for all examples below). The network architecture consists of a sequence of three fully connected up-sampling layers, $\{L_1^{FC}, L_2^{FC}, L_3^{FC}\}$, of respectively $\{64, 256, 1024\}$ nodes. These are followed by a sequence of three convolutional layers $\{L_1^{C}, L_2^{C}, L_3^{C}\}$ with 1×5 kernels, and a final fully connected layer with softmax activation. The network size was not minimized through architecture search, which along with other potential further speed improvements we leave for future research.

Training Process

For the CNN, we use $100K$ simulations from the stochastic simulator for each of $3M$ parameter vectors, and bin the simulation outcomes as normalized counts into $\mathcal{R}^{N_c} \times \mathcal{R}^{N_d}$ slots respectively (looking ahead to our examples, N_c concerns the number of choice outcomes, and N_d the number of bins into which the reaction time outcomes are split for a given simulator). The resultant relative frequency histograms (empirical likelihood functions) $\ell_{empirical}(\theta|\mathbf{x})\ \forall\theta \in \Theta, \forall x \in \mathcal{X}$, then serve as the target labels during training, with the corresponding parameters θ serving as feature vectors. For a given parameter vector θ the CNN gives out a histogram $\hat{\ell_\phi}(\theta|\mathbf{x})$, where ϕ are the network parameters. The network is then trained by minimizing the KL-divergence between observed and generated histograms,

$$\mathcal{D}(\hat{\ell}(\theta|\mathbf{x})\|\ell_{empirical}(\theta|\mathbf{x})) = \sum_{i=0}^{c}\sum_{j=0}^{d}\left[\hat{\ell}(\theta|x_{ij})\log\frac{\hat{\ell}(\theta|x_{ij})}{\ell_{empirical}(\theta|x_{ij})}\right]$$

[4]Implementations of the MLP method, the samplers we used as well as the training pipeline can be found at `https://github.com/lnccbrown/lans/tree/master/al-mlp`.

Training $\mathcal{D}(\hat{\ell}(\theta|\mathbf{x})\|\ell_{empirical}(\theta|\mathbf{x}))$, is not the only option. We note that it would have been a valid choice to train on $\mathcal{D}(\ell_{empirical}(\theta|\mathbf{x})\|\hat{\ell}(\theta|\mathbf{x}))$ (Minka, 2013), or the symmetrized Kullback Leibler Divergence instead. Training results however were good enough for our present purposes to leave a precise performance comparison across those loss functions for future research, leaving room for further improvements.

As for the MLP, we use the Adam optimizer (Kingma and Ba, 2014), and implemented the network in Tensorflow (Abadi et al., 2016).

Sampling Algorithm

One benefit in using the CNN lies in the enhanced potential for parallel processing across large number of parameter configurations and datapoints. To fully exploit this capability, instead of running a (sequential) MCMC algorithm for our parameter recovery studies, we use iterated importance sampling, which can be done in parallel. Specifically, we use adaptive importance sampling based on mixtures of t-distributions, following a slightly adjusted version of the suggestions in (Cappé et al., 2008; Wraith et al., 2009).

While importance sampling is well established, for clarity and the setting in which we apply it, we explain some of the details here. Importance sampling algorithms are driven by the basic equality,

$$\int f(\theta)d\theta = \int \frac{f(\theta)}{g(\theta)} g(\theta)d\theta$$

which holds for any pair of probability distributions such that $g(\theta) > 0$, where $f(\theta) > 0$. $f(\theta)$ is our posterior distribution, and $g(\theta)$ is the proposal distribution. We now sample N tuples θ according to $g(\theta)$, and assign each θ_i an importance weight $w_i = \frac{f(\theta_i)}{g(\theta_i)}$.

To get samples from the posterior distribution, we sample with replacement the θ from the set $\{\theta_0, ..., \theta_n\}$, with probabilities assigned as the normalized weights $\{\tilde{w_0}, ..., \tilde{w_n}\}$. We note that importance sampling is exact for $N \to \infty$. However for finite N, the performance is strongly dependent on the quality of the proposal distribution $g(.)$. A bad match of $f(.)$ and $g(.)$ leads to high variance in the importance weights, which drives down performance of the algorithm, as commonly measured by the effective sample size (Liu, 2008),

$$\hat{ESS} = \frac{1}{\sum_{n=1}^{N} \bar{w}_n^2}$$

Iterated importance sampling uses consecutive importance sampling rounds, to improve the proposal distribution $g(.)$. A final importance sampling round is used to get the importance sample we use as our posterior sample. Specifically, we start with a mixture of t-distributions $g_0(.)$, where $\mathcal{M}$ is the number of mixture components. Each component of $g_0(.)$ is centered at the MAP according to a optimization run (again we used differential evolution). The component-covariance-matrix is estimated by a numerical approximation of the Hessian at the respective MAP. Each round i, based on the importance sample $\{\mathbf{x}, \mathbf{w}\}_i$, we update the proposal distribution (to a new mixture of t-distributions), using the update equations derived in (Cappé et al., 2008).

As suggested by Cappé et al., 2008, convergence is assessed using the normalized perplexity statistic (the exponentiated Shannon entropy of the importance weights). For run i this is computed as $\exp^{\frac{H^{k,N}}{/N}}$, where $H^{k,N} = -\sum_{i=1}^{N} \bar{w}_{k,i} \log \bar{w}_{k,i}$.

To help convergence, we depart from the basic setup suggested in Cappé et al., 2008 in the following way. We apply an annealing factor $\gamma_k = \max 2^{z-k}, 1 \; z \in \{1, 2, 4, .., \}$, so that for iteration k of the importance sampler, we are operating on the target $f(x)^{\frac{1}{\gamma_k}}$. Smoothing the target during the first iterations helps with successfully adjusting the proposal distribution $g(.)$. Figure 3.2 visualizes the CNN approach. Again, we emphasize that more numerical experiments using a larger variety of sampling algorithms are desirable, but are out of the scope for the current paper. [5]

3.8.4 Strengths and Weaknesses

In this section we clarify a few strengths and weaknesses of the two presented methods and their respective use cases. First, representing the likelihood function datapoint-wise as an MLP output, or globally via the CNN output histogram, affects the potential for parallelization. As exploited by the choice of sampler, the CNN is very amenable to parallelization across parameters, since inputs are parameter tuples only. Since the output is represented as a global likelihood histogram, the dataset-likelihood is computed as the summation of the elementwise multiplied of bin-log-likelihoods, with a correspondingly binned dataset (counts over bins). This has the highly desirable property of making evaluation cost (time) independent of dataset size. While the MLP in principle allows parallel processing of inputs, the data-point wise representation of input values ($\{\theta, x\}$) makes the potential for cross-parameter parallelization dependent on data-set sizes. While a single evaluation of the CNN is more costly, cross-parameter batch processing can make it preferable to the MLP. Second, the CNN has an advantage during training, where the representation of the output as a softmax layer, and corresponding training via minimization of the KL divergence, provides a more robust training signal to ensure probability distributions compared to the purely local one in which the MLP learns a scalar likelihood output as a simple regression problem. Third, and conversely, the MLP formulation is more natural for trial-wise parameter estimates, since the histogram representations may be redundant in case data-points are in fact evaluated one by one (given datapoint-wise parameters induced by trial-by-trial effects on parameter vectors). Give equivalent success in learning likelihoods, we see potential for speed-up when using the point-wise approach in this case. In principle both approaches however allow one to estimate the impact of trial-wise regressors on model parameters during inference, without further training. It is for example common in the cognitive neuroscience literature to allow the cross-trial time-course of EEG, fMRI, or spike signals to be modelled as a trial-by-trial regressor on model parameters of e.g. drift diffusion models (Wiecki, Sofer, and Frank, 2013; Frank, Gagne, et al., 2015; Cavanagh et al., 2011; Herz et al., 2016; Pedersen and Frank, 2020). Another relevant example is the incorporation of latent learning dynamics. If a subject's choice behavior are driven by reinforcement learning across stimuli, we can translate this into trial

[5] Implementations of the CNN method, the samplers we used as well as the training pipeline can be found at `https://github.com/lnccbrown/lans/tree/master/al-cnn`.

by trial effects on the parameter vectors of a generative process model (Pedersen and Frank, 2020). These applications are implicitly enabled at no extra cost with the MLP method, while the trial by trial split multiplies the necessary computations for the CNN by the number N of data-points when compared to scenarios that only need dataset-wise parameters. We stress again however that in general both the CNN, as well as the MLP can directly be used for hierarchical inference scenarios. The preceding discussion pertains to further potential for optimization and relative strengths, not categorical potential for application to a given scenario. With respect to the latter, both methods are essentially equal.

3.9 Appendix

3.9.1 Parameter Recovery

Here we provide additional figures concerning parameter recovery studies. Table 3.1 summarizes the parameter-wise R^2 between ground truth and the posterior mean estimates for each tested model and for each the CNN and MLP (where applicable) methods in turn. For the MLP, results are based on a reference run which used training data constructed from KDE empirical likelihoods utilizing $100k$ simulations each, and a slice sampler stopped with help of the Geweke Diagnostic. Results in the paper are based on slice samplers as well as Slice samplers, which explains why not all R^2 values match exactly the ones found in other Figures. Our findings were however generally robust across samplers.

3.9.2 Manifolds / Likelihoods

We show the some examples of the likelihood manifolds for the various models that we tested.

DDM-SDV

Figure 3.13.

Linear Collapse

Figure 3.14.

WEIBULL

Figure 3.15.

LEVY

Figure 3.16.

DDM		N	v	a	w	ndt							
R^2	MLP	1024	1.0	1.0	0.99	1							
		4096	1.0	1.0	0.99	1							
	CNN	1024	1	0.94	0.98	1							
		4096	1	1	0.99	1							
DDM-SDV			v	a	w	ndt	sdv						
R^2	MLP	1024	0.95	0.94	0.96	1	0.57						
		4096	0.94	0.95	0.97	1	0.58						
	CNN	1024	0.98	0.97	0.98	1	0.79						
		4096	0.99	0.98	0.99	1	0.87						
LC			v	a	w	ndt	θ						
R^2	MLP	1024	0.99	0.93	0.97	1	0.98						
		4096	0.99	0.94	0.98	1	0.97						
	CNN	1024	0.96	0.94	0.97	1	0.97						
		4096	0.97	0.94	0.98	1	0.97						
OU			v	a	w	ndt	g						
R^2	MLP	1024	0.98	0.89	0.98	0.99	0.12						
		4096	0.99	0.79	0.95	0.99	0.03						
	CNN	1024	0.99	0.94	0.97	1	0.41						
		4096	0.99	0.95	0.98	1	0.45						
LEVY			v	a	w	ndt	α						
R^2	MLP	1024	0.96	0.94	0.84	1	0.33						
		4096	0.97	0.91	0.61	1	0.2						
	CNN	1024	0.99	0.97	0.9	1	0.71						
		4096	0.99	0.98	0.95	1	0.8						
WEIBULL			v	a	w	ndt	α	β					
R^2	MLP	1024	0.99	0.82	0.96	1	0.2	0.43					
		4096	0.99	0.8	0.98	0.99	0.26	0.41					
	CNN	1024	0.98	0.91	0.96	1	0.4	0.69					
		4096	0.98	0.91	0.97	1	0.37	0.63					
FULL-DDM			v	a	w	ndt	dw	sdv	dndt				
R^2	MLP	1024	0.95	0.94	0.88	1	0	0.28	0.47				
		4096	0.93	0.94	0.88	1	0	0.25	0.38				
	CNN	1024	0.98	0.98	0.93	1	0	0.62	0.79				
		4096	0.99	0.99	0.97	1	0	0.8	0.91				
RACE 3			v0	v1	v2	a	w0	w1	w2	ndt			
R^2	CNN	1024	0.88	0.86	0.89	0.19	0.49	0.51	0.5	0.99			
		4096	0.93	0.91	0.93	0.18	0.49	0.47	0.47	1			
RACE 4			v0	v1	v2	v3	a	w0	w1	w2	w3	ndt	
R^2	CNN	1024	0.73	0.68	0.71	0.73	0.11	0.49	0.5	0.48	0.49	0.99	
		4096	0.79	0.76	0.77	0.81	0.18	0.5	0.5	0.51	0.55	0.99	
LCA 3			v0	v1	v2	a	w0	w1	w2	g	b	ndt	
R^2	CNN	1024	0.58	0.56	0.58	0.47	0.7	0.72	0.68	0.27	0.57	1	
		4096	0.51	0.5	0.52	0.44	0.67	0.67	0.66	0.23	0.52	1	
LCA 4			v0	v1	v2	v3	a	w0	w1	w2	w3	g	b	ndt
R^2	CNN	1024	0.5	0.46	0.54	0.51	0.51	0.71	0.69	0.69	0.67	0.18	0.7	0.99
		4096	0.42	0.42	0.46	0.42	0.52	0.67	0.63	0.68	0.65	0.15	0.64	1

Table 3.1. Parameter Recovery for a variety of test-bed models

ORNSTEIN

Figure 3.17.

FULL-DDM

Figure 3.18.

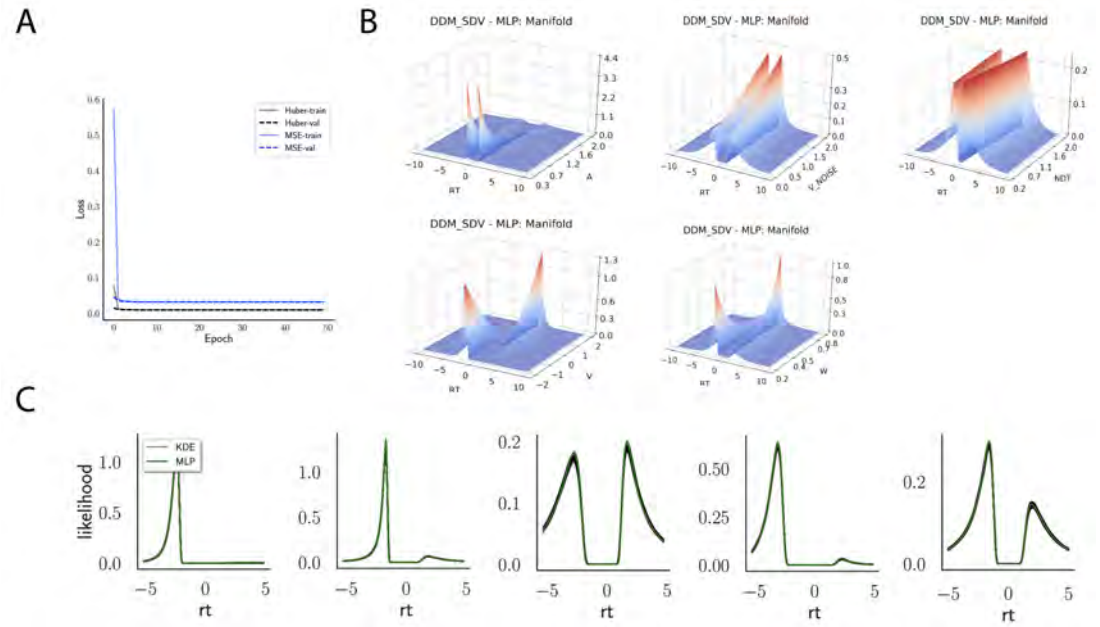

Figure 3.13. **A** Shows the training and validation loss for Huber as well as MSE for the DDM-SDV model. Training was driven by the Huber loss. **B** Illustrates the likelihood manifolds, by varying one parameter in the trained region. **C** shows MLP likelihoods in green, on top of a sample of 50 KDE-based empirical likelihoods derived from 20k samples each.

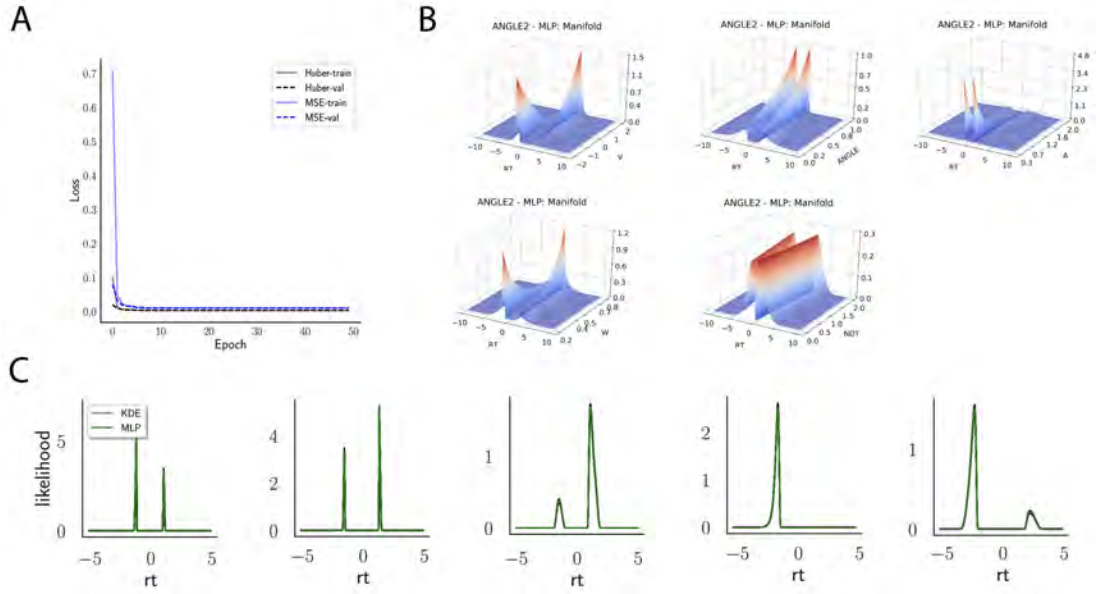

Figure 3.14. **A** Shows the training and validation loss for Huber as well as MSE for the LC model. Training was driven by the Huber loss. **B** Illustrates the likelihood manifolds, by varying one parameter in the trained region. **C** shows MLP likelihoods in green, on top of a sample of 50 KDE-based empirical likelihoods derived from 20k samples each.

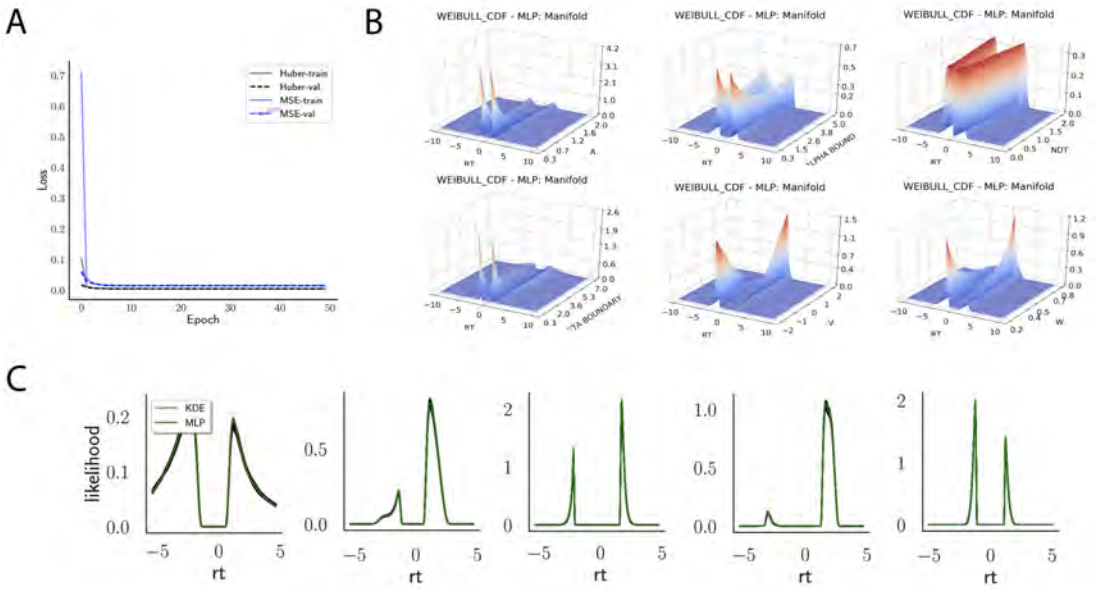

Figure 3.15. **A** Shows the training and validation loss for Huber as well as MSE for the WEIBULL model. Training was driven by the Huber loss. **B** Illustrates the likelihood manifolds, by varying one parameter in the trained region. **C** shows MLP likelihoods in green, on top of a sample of 50 KDE-based empirical likelihoods derived from 100k samples each.

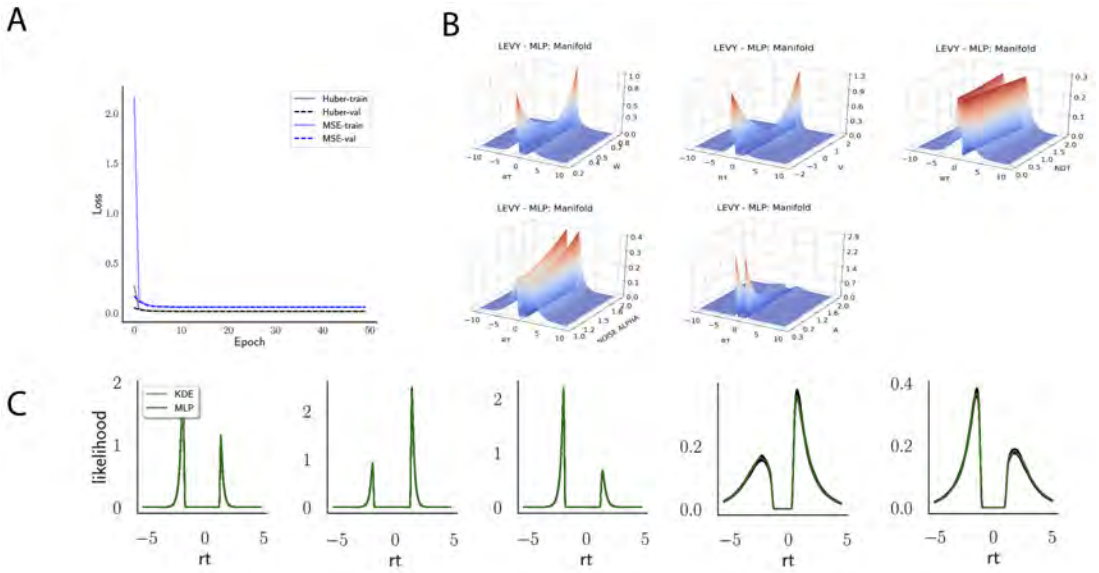

Figure 3.16. **A** Shows the training and validation loss for Huber as well as MSE for the Levy model. Training was driven by the Huber loss. **B** Illustrates the likelihood manifolds, by varying one parameter in the trained region. **C** shows MLP likelihoods in green, on top of a sample of 50 KDE-based empirical likelihoods derived from 100k samples each.

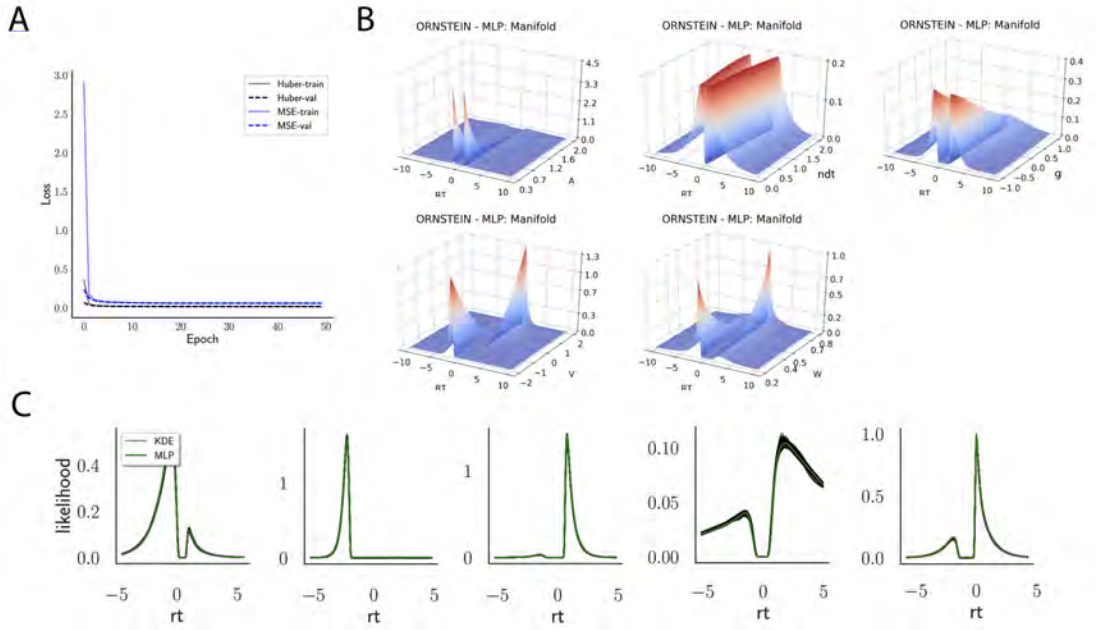

Figure 3.17. **A** Shows the training and validation loss for Huber as well as MSE for the ORNSTEIN model. Training was driven by the Huber loss. **B** Illustrates the likelihood manifolds, by varying one parameter in the trained region. **C** shows MLP likelihoods in green, on top of a sample of 50 KDE-based empirical likelihoods derived from 100k samples each.

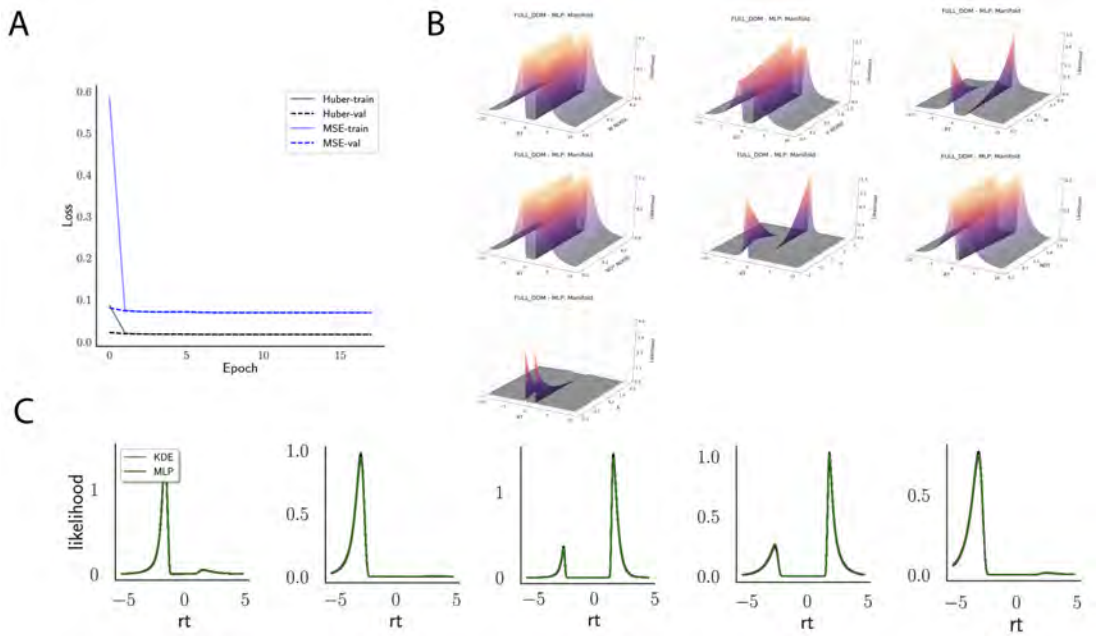

Figure 3.18. **A** Shows the training and validation loss for Huber as well as MSE for the Full-DDM model. Training was driven by the Huber loss. **B** Illustrates the likelihood manifolds, by varying one parameter in the trained region. **C** shows MLP likelihoods in green, on top of a sample of 50 KDE-based empirical likelihoods derived from 100k samples each.

References

Abadi, Martın et al. (2016). "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283.

Acerbi, Luigi (2020). "Variational Bayesian Monte Carlo with Noisy Likelihoods". In: *arXiv preprint arXiv:2006.08655*.

Ahn, Woo-Young, Nathaniel Haines, and Lei Zhang (2017). "Revealing neurocomputational mechanisms of reinforcement learning and decision-making with the hBayesDM package". In: *Computational Psychiatry* 1, pp. 24–57.

Akeret, Joël et al. (2015). "Approximate Bayesian computation for forward modeling in cosmology". In: *Journal of Cosmology and Astroparticle Physics* 2015.08, p. 043.

Badre, David et al. (2012). "Rostrolateral prefrontal cortex and individual differences in uncertainty-driven exploration". In: *Neuron* 73.3, pp. 595–607.

Behnel, Stefan et al. (2010). "Cython: The best of both worlds". In: *Computing in Science & Engineering* 13.2, pp. 31–39.

Bishop, Christopher M (1994). *Mixture density networks*. Tech. rep. Microsoft Research.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). "Variational inference: A review for statisticians". In: *Journal of the American statistical Association* 112.518, pp. 859–877.

Boehm, Udo et al. (2018). "Estimating across-trial variability parameters of the Diffusion Decision Model: Expert advice and recommendations". In: *Journal of Mathematical Psychology* 87, pp. 46–75.

Cappé, Olivier et al. (2008). "Adaptive importance sampling in general mixture classes". In: *Statistics and Computing* 18.4, pp. 447–459.

Cavanagh, James F et al. (2011). "Subthalamic nucleus stimulation reverses mediofrontal influence over decision threshold". In: *Nature neuroscience* 14.11, pp. 1462–1467.

Charpentier, Arthur and Emmanuel Flachaire (2015). "Log-transform kernel density estimation of income distribution". In: *L'actualité économique* 91.1-2, pp. 141–159.

Cisek, Paul, Geneviève Aude Puskas, and Stephany El-Murr (2009). "Decisions in changing conditions: the urgency-gating model". In: *Journal of Neuroscience* 29.37, pp. 11560–11571.

Cranmer, Kyle, Johann Brehmer, and Gilles Louppe (2020). "The frontier of simulation-based inference". In: *Proceedings of the National Academy of Sciences* 117.48, pp. 30055–30062.

Daw, N. D. et al. (2011). "Trial-by-trial data analysis using computational models". In: *Decision making, affect, and learning: Attention and performance XXIII* 23.1.

Daw, Nathaniel D et al. (2011). "Model-based influences on humans' choices and striatal prediction errors". In: *Neuron* 69.6, pp. 1204–1215.

Diaconis, Persi (2009). "The markov chain monte carlo revolution". In: *Bulletin of the American Mathematical Society* 46.2, pp. 179–205.

Doi, Takahiro et al. (2020). "The caudate nucleus contributes causally to decisions that balance reward and uncertain visual information". In: *ELife* 9, e56694.

Drugowitsch, Jan (2016). "Fast and accurate Monte Carlo sampling of first-passage times from Wiener diffusion models". In: *Scientific reports* 6, p. 20490.

Forstmann, Birte U et al. (2010). "Cortico-striatal connections predict control over speed and accuracy in perceptual decision making". In: *Proceedings of the National Academy of Sciences* 107.36, pp. 15916–15920.

Frank, Michael J, Chris Gagne, et al. (2015). "fMRI and EEG predictors of dynamic decision parameters during human reinforcement learning". In: *Journal of Neuroscience* 35.2, pp. 485–494.

Frank, Michael J, Johan Samanta, et al. (2007). "Hold your horses: impulsivity, deep brain stimulation, and medication in parkinsonism". In: *science* 318.5854, pp. 1309–1312.

Frazier, Peter I and J Yu Angela (2008). "Sequential hypothesis testing under stochastic deadlines". In: *Advances in neural information processing systems*, pp. 465–472.

Gelman, Andrew, Donald B Rubin, et al. (1992). "Inference from iterative simulation using multiple sequences". In: *Statistical science* 7.4, pp. 457–472.

Geweke, John (1992). "Evaluating the accuracy of sampling-based approaches to the calculations of posterior moments". In: *Bayesian statistics* 4, pp. 641–649.

Gonçalves, Pedro J et al. (2020). "Training deep neural density estimators to identify mechanistic models of neural dynamics". In: *Elife* 9, e56261.

Greenberg, David, Marcel Nonnenmacher, and Jakob Macke (2019). "Automatic posterior transformation for likelihood-free inference". In: *International Conference on Machine Learning*. PMLR, pp. 2404–2414.

Guillaumes, Axel Brando (2017). "Mixture density networks for distribution and uncertainty estimation". PhD thesis. Universitat Politècnica de Catalunya. Facultat d'Informàtica de Barcelona.

Gutenkunst, R. N. et al. (2007). "Sloppy models and parameter indeterminancy in systems biology". In: *PLoS Computational Biology* 3 (10).

Gutmann, Michael U et al. (2018). "Likelihood-free inference via classification". In: *Statistics and Computing* 28.2, pp. 411–425. DOI: 10.1007/s11222-017-9738-6.

Hawkins, Guy E et al. (2015). "Revisiting the evidence for collapsing boundaries and urgency signals in perceptual decision-making". In: *Journal of Neuroscience* 35.6, pp. 2476–2484.

Heathcote, Andrew et al. (2019). "Dynamic models of choice". In: *Behavior research methods* 51.2, pp. 961–985.

Herz, Damian M et al. (2016). "Neural correlates of decision thresholds in the human subthalamic nucleus". In: *Current Biology* 26.7, pp. 916–920.

Hoffman, Matthew D and Andrew Gelman (2014). "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623.

Holmes, William R (2015). "A practical guide to the Probability Density Approximation (PDA) with improved implementation and error characterization". In: *Journal of Mathematical Psychology* 68, pp. 13–24.

Huber, Peter J (1992). "Robust estimation of a location parameter". In: *Breakthroughs in statistics*. Springer, pp. 492–518.

Huys, Quentin JM, Tiago V Maia, and Michael J Frank (2016). "Computational psychiatry as a bridge from neuroscience to clinical applications". In: *Nature neuroscience* 19.3, p. 404.

Järvenpää, Marko et al. (2018). "Gaussian process modelling in approximate Bayesian computation to estimate horizontal gene transfer in bacteria". In: *Annals of Applied Statistics* 12.4, pp. 2228–2251.

— (2021). "Parallel Gaussian process surrogate Bayesian inference with noisy likelihood evaluations". In: *Bayesian Analysis* 16.1, pp. 147–178.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint* arXiv:1412.6980.

Krajbich, Ian and Antonio Rangel (2011). "Multialternative drift-diffusion model predicts the relationship between visual fixations and choice in value-based decisions". In: *Proceedings of the National Academy of Sciences* 108.33, pp. 13852–13857.

Lipton, Alexander and Vadim Kaushansky (2018). "On the first hitting time density of an Ornstein-Uhlenbeck process". In: *arXiv preprint* arXiv:1810.02390.

Liu, Jun S (2008). *Monte Carlo strategies in scientific computing*. Springer Science & Business Media.

Lueckmann, Jan-Matthis et al. (2019a). "Likelihood-free inference with emulator networks". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 32–53.

— (2019b). "Likelihood-free inference with emulator networks". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 32–53.

Malhotra, Gaurav et al. (2018). "Time-varying decision boundaries: insights from optimality analysis". In: *Psychonomic bulletin & review* 25.3, pp. 971–996.

Meeds, Edward and Max Welling (2014). "GPS-ABC: Gaussian process surrogate approximate Bayesian computation". In: *arXiv preprint arXiv:1401.2838*.

Mestdagh, Merijn et al. (2019). "Prepaid parameter estimation without likelihoods". In: *PLoS computational biology* 15.9, e1007181.

Minka, Thomas P (2013). "Expectation propagation for approximate Bayesian inference". In: *arXiv preprint arXiv:1301.2294*.

Mullowney, P. and S. Iyengar (2006). *Maximum Likelihood Estimation and Computation for the Ornstein-Uhlenbeck Process*. URL: `https://www.stat.pitt.edu/si/Satish,Paul%5C%20paper1.pdf`.

Navarro, Daniel J and Ian G Fuss (2009). "Fast and accurate calculations for first-passage times in Wiener diffusion models". In: *Journal of mathematical psychology* 53.4, pp. 222–230.

Neal, Radford M (1995). "Bayesian learning for neural networks". PhD thesis. University of Toronto.

— (2003). "Slice sampling". In: *Annals of statistics*, pp. 705–741.

Nilsson, Håkan, Jörg Rieskamp, and Eric-Jan Wagenmakers (2011). "Hierarchical Bayesian parameter estimation for cumulative prospect theory". In: *Journal of Mathematical Psychology* 55.1, pp. 84–93.

Niv, Yael et al. (2012). "Neural prediction errors reveal a risk-sensitive reinforcement-learning process in the human brain". In: *Journal of Neuroscience* 32.2, pp. 551–562.

O'Reilly, Randall C and Michael J Frank (2006). "Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia". In: *Neural computation* 18.2, pp. 283–328.

Opheusden, Bas van, Luigi Acerbi, and Wei Ji Ma (2020). "Unbiased and efficient log-likelihood estimation with inverse binomial sampling". In: *PLOS Computational Biology* 16.12, e1008483.

Palestro, James J et al. (2018). "Some task demands induce collapsing bounds: Evidence from a behavioral analysis". In: *Psychonomic bulletin & review* 25.4, pp. 1225–1248.

Papamakarios, George and Iain Murray (2016). "Fast ε-free inference of simulation models with bayesian conditional density estimation". In: *Advances in Neural Information Processing Systems*, pp. 1028–1036.

Papamakarios, George, Eric Nalisnick, et al. (2019). "Normalizing flows for probabilistic modeling and inference". In: *arXiv preprint arXiv:1912.02762*.

Papamakarios, George, David Sterratt, and Iain Murray (2019). "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 837–848.

Pedersen, Mads L and Michael J Frank (2020). "Simultaneous Hierarchical Bayesian Parameter Estimation for Reinforcement Learning and Drift Diffusion Models: a Tutorial and Links to Neural Data". In: *Computational Brain & Behavior* 3, pp. 458–471.

Radev, Stefan T, Ulf K Mertens, Andreas Voss, Lynton Ardizzone, et al. (2020). "BayesFlow: Learning complex stochastic models with invertible neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15.

Radev, Stefan T, Ulf K Mertens, Andreas Voss, and Ullrich Köthe (2020). "Towards end-to-end likelihood-free inference with convolutional neural networks". In: *British Journal of Mathematical and Statistical Psychology* 73.1, pp. 23–43.

Rangel, Antonio, Colin Camerer, and P Read Montague (2008). "A framework for studying the neurobiology of value-based decision making". In: *Nature reviews neuroscience* 9.7, pp. 545–556.

Ratcliff, Roger (1978). "A theory of memory retrieval." In: *Psychological review* 85.2, p. 59.

Ratcliff, Roger and Russ Childers (2015). "Individual differences and fitting methods for the two-choice diffusion model of decision making". In: *Decision* 2.4, p. 237.

Ratcliff, Roger and Michael J Frank (2012). "Reinforcement-based decision making in corticostriatal circuits: mutual constraints by neurocomputational and diffusion models". In: *Neural computation* 24.5, pp. 1186–1229.

Ratcliff, Roger and Gail McKoon (2008). "The diffusion decision model: theory and data for two-choice decision tasks". In: *Neural computation* 20.4, pp. 873–922.

Reynolds, Andy M and Christopher J Rhodes (2009). "The Lévy flight paradigm: random search patterns and mechanisms". In: *Ecology* 90.4, pp. 877–887.

Rezende, Danilo and Shakir Mohamed (2015). "Variational inference with normalizing flows". In: *International Conference on Machine Learning*. PMLR, pp. 1530–1538.

Robert, Christian and George Casella (2011). "A short history of Markov chain Monte Carlo: Subjective recollections from incomplete data". In: *Statistical Science*, pp. 102–115.

— (2013). *Monte Carlo statistical methods*. Springer Science & Business Media.

Schoennberg, Tom et al. (2007). "Reinforcement learning signals in the human striatum distinguish learners from nonlearners during reward-based decision making". In: *Journal of Neuroscience* 27.47, pp. 12860–12867.

Shinn, Maxwell, Norman H Lam, and John D Murray (2020). "A flexible framework for simulating and fitting generalized drift-diffusion models". In: *Elife* 9, e56938.

Silverman, Bernard W (1986). *Density estimation for statistics and data analysis*. Vol. 26. CRC press.

Sisson, Scott A, Yanan Fan, and Mark Beaumont (2018). *Handbook of approximate Bayesian computation*. CRC Press.

Storn, Rainer and Kenneth Price (1997). "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11.4, pp. 341–359.

Ter Braak, Cajo JF (2006). "A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces". In: *Statistics and Computing* 16.3, pp. 239–249.

Turner, Brandon M and Per B Sederberg (2014). "A generalized, likelihood-free method for posterior estimation". In: *Psychonomic bulletin & review* 21.2, pp. 227–250.

Turner, Brandon M, Leendert Van Maanen, and Birte U Forstmann (2015a). "Informing cognitive abstractions through neuroimaging: The neural drift diffusion model." In: *Psychological review* 122.2, p. 312.

— (2015b). "Informing cognitive abstractions through neuroimaging: the neural drift diffusion model." In: *Psychological review* 122.2, p. 312.

Turner, Brandon M and Trisha Van Zandt (2018). "Approximating Bayesian inference through model simulation". In: *Trends in Cognitive Sciences* 22.9, pp. 826–840.

Usher, Marius and James L McClelland (2001). "The time course of perceptual choice: the leaky, competing accumulator model." In: *Psychological review* 108.3, p. 550.

Vandekerckhove, Joachim, Francis Tuerlinckx, and Michael D Lee (2011). "Hierarchical diffusion models for two-choice response times." In: *Psychological methods* 16.1, pp. 44–62.

Virtanen, Pauli et al. (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods*.

W., Feller and Feller V. (1968). *An Introduction to Probability Theory and its Applications Vol 1*. Vol. 1. Wiley.

Wiecki, Thomas V and Michael J Frank (2013). "A computational model of inhibitory control in frontal cortex and basal ganglia." In: *Psychological review* 120.2, p. 329.

Wiecki, Thomas V, Imri Sofer, and Michael J Frank (2013). "HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python". In: *Frontiers in neuroinformatics* 7, p. 14.

Wieschen, Eva Marie, Andreas Voss, and Stefan Radev (2020). "Jumping to conclusion? a lévy flight model of decision making". In: *The Quantitative Methods for Psychology* 16.2, pp. 120–132.

Wilson, Robert C and Anne GE Collins (2019). "Ten simple rules for the computational modeling of behavioral data". In: *Elife* 8, e49547.

Wosniack, Marina E et al. (2017). "The evolutionary origins of Lévy walk foraging". In: *PLoS computational biology* 13.10, e1005774.

Wraith, Darren et al. (2009). "Estimation of cosmological parameters using adaptive importance sampling". In: *Physical Review D* 80.2, p. 023507.

Yartsev, Michael M et al. (2018). "Causal contribution and dynamical encoding in the striatum during evidence accumulation". In: *Elife* 7, e34929.

Zajkowski, Wojciech K, Malgorzata Kossut, and Robert C Wilson (2017). "A causal role for right frontopolar cortex in directed, but not random, exploration". In: *Elife* 6, e27430.

Chapter 4

Beyond Drift Diffusion Models: Fitting a Broad Class of Decision and Reinforcement Learning Models with HDDM

Computational modeling has become a central aspect of research in the cognitive neurosciences. As the field matures, it is increasingly important to move beyond standard models to quantitatively assess models with richer dynamics that may better reflect underlying cognitive and neural processes. For example, sequential sampling models (SSMs) are a general class of models of decision making intended to capture processes jointly giving rise to reaction time distributions and choice data in n-alternative choice paradigms. A number of model variations are of theoretical interest, but empirical data analysis has historically been tied to a small subset for which likelihood functions are analytically tractable. Advances in methods designed for likelihood-free inference have recently made it computationally feasible to consider a much larger spectrum of sequential sampling models. In addition, recent work has motivated the combination of SSMs with reinforcement learning (RL) models, which had historically been considered in separate literature. Here we provide a significant addition to the widely used HDDM Python toolbox and include a tutorial for how users can easily fit and assess a (user extensible) wide variety of SSMs, and how they can be combined with RL models. The extension comes batteries included, including model visualization tools, posterior predictive checks, and ability to link trial-wise neural signals with model parameters via hierarchical Bayesian regression.

4.1 Introduction

The drift diffusion model (DDM, also called diffusion decision model or Ratcliff diffusion model) (Ratcliff, 1978; Ratcliff, Smith, et al., 2016), and more generally the framework of sequential sampling models (SSMs) (Hawkins et al., 2015; Ratcliff, Smith, et al., 2016; Voss et al., 2019; Tillman, Trish Van Zandt, and Logan, 2020; Heathcote, Matzke, and Heathcote, 2022) have become a mainstay of the cognitive scientist's model arsenal in the last two decades (Trisha Van Zandt, Colonius, and Proctor, 2000; Lawlor et al., 2020; Wieschen, Voss, and Radev, 2020).

SSMs are used to model neurocognitive processes that jointly give rise to choice and reaction time data in a multitude of domains, spanning from perceptual discrimination to memory retrieval to preference-based choice (Ratcliff, 1978; Smith, Ratcliff, and Sewell, 2014; Ratcliff, Thapar, and McKoon, 2006; Krajbich and Rangel, 2011; Krajbich, Lu, et al., 2012) across species (Gold and Shadlen, 2007; Yartsev et al., 2018; Doi et al., 2020). Moreover, researchers are often interested in the underlying neural dynamics that give rise to such choice processes. As such, many studies include additional measurements such as EEG, fMRI or eyetracking signals as covariates, which act as latent variables and connect to model parameters (e.g. via a regression model) to drive trial specific parameter valuations (Rangel, Camerer, and Montague, 2008; Forstmann et al., 2010; Frank et al., 2015; Yartsev et al., 2018; Doi et al., 2020). See Figure 4.1 for an illustration of the DDM and some canonical experimental paradigms.

The widespread interest and continuous use of SSMs across the research community has spurred the development of several software packages targeting the estimation of such models (Vandekerckhove and Tuerlinckx, 2008; L. Fontanesi, 2022; Heathcote, Lin, et al., 2019). For a hierarchical Bayesian approach to parameter estimation, the HDDM toolbox in Python (Wiecki, Sofer, and Frank, 2013) (available at https://github.com/hddm-devs/hddm) is widely used and the backbone of hundreds of studies published in peer reviewed journals.

HDDM allows users to conveniently specify and estimate DDM parameters for a wide range of experimental designs, including the incorporation of trial-by-trial covariates via regression models targeting specific parameters. As an example, one may use this framework to estimate whether trial-by-trial drift rates in a DDM co-vary with neural activity in a given region (and/or temporal dynamic), pupil dilation or eye gaze position. Moreover, by using hierarchical Bayesian estimation, HDDM optimizes the inference about such parameters at the individual subject and group levels.

Nevertheless, until now, HDDM and other such toolboxes have been largely limited to fitting the 2-alternative choice DDM (albeit allowing for the full DDM with inter-trial parameter variability). The widespread interest in SSMs has however also spurred theoretical and empirical investigations into various alternative model variants. Notable examples are, amongst others, race models with more than 2 decision options, the leaky competing accumulator model (Usher and McClelland, 2001), SSMs with dynamic decision boundaries (Cisek, Puskas, and El-Murr, 2009; Ratcliff and Frank, 2012; Trueblood et al., 2021) and more recently SSMs based on Levy flights rather than basic Gaussian diffusions (Wieschen, Voss, and Radev, 2020). Moreover, as mentioned earlier, SSMs naturally extend to n-choice paradigms.

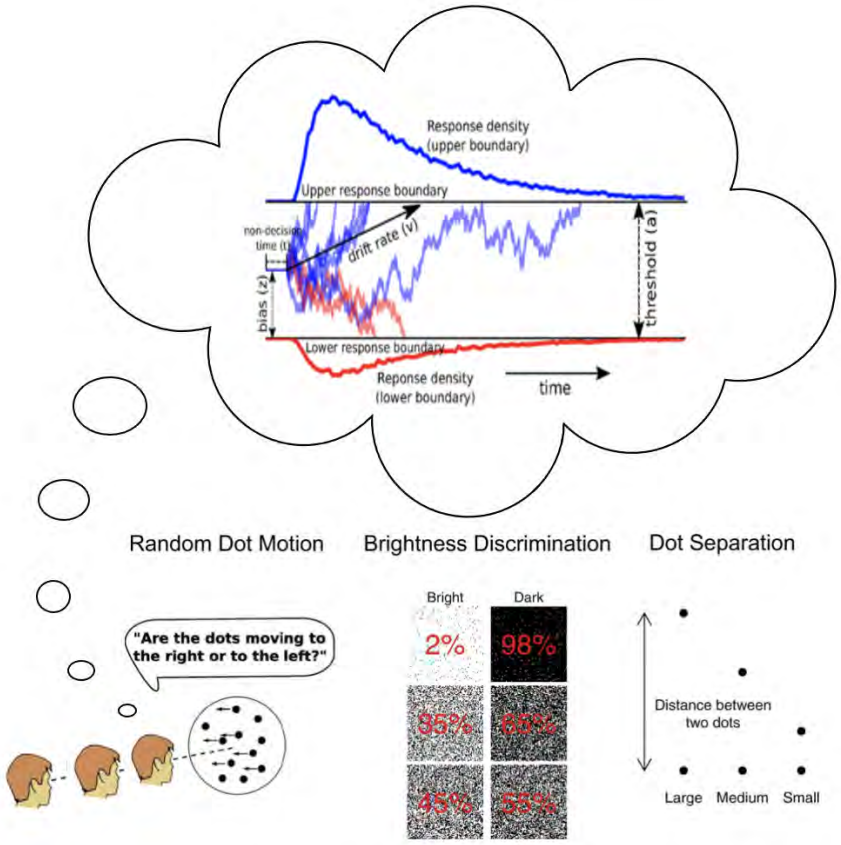

Figure 4.1. Drift Diffusion Model and some example applications.

A similar state of affairs is observed for another class of cognitive models which aim to simultaneously model the dynamics of a feedback-based learning across trials as well as the within-trial decision process. One way to achieve this is by replacing the choice rule in a reinforcement learning (RL) process, in itself an important theoretical framework in the study of learning behavior across trials (Dowd et al., 2016; McDougle and Collins, 2021; Eckstein, Wilbrecht, and Collins, 2021; Collins and Shenhav, 2022) with a cognitive process models such as SSMs. This forms a powerful combination of modeling frameworks. While recent studies moved into this direction (Mads Lund Pedersen, Frank, and Biele, 2017; Turner, 2019; Mads L Pedersen and Frank, 2020; Laura Fontanesi et al., 2019; L. Fontanesi, 2022), they have again been limited to an application of the basic DDM.

Despite the great interest in these classes of models, tractable inference and therefore, widespread adoption of such models has been hampered by the lack of easy to compute likelihood functions (including essentially all of the examples provided above). In particular, while many interesting models are straightforward to simulate, often researchers want to go the other way: from the observed data to infer the most likely parameters. For all but the simplest models, such likelihood functions are analytically intractable, and hence previous approaches required computationally costly simulations and/or lacked flexibility in applying such methods to different scenarios (Turner and Sederberg, 2014; Turner and Trisha Van Zandt, 2018; Palestro et al., 2019; Shinn, Lam, and J. D. Murray, 2020;

Boehm et al., 2021). We recently developed a novel approach using artificial neural networks which can, given sufficient training data, approximate likelihoods for a large class of SSM variants, thereby amortizing the cost and enabling rapid, efficient and flexible inference (Fengler et al., 2021). We dubbed such networks LANs, for *likelihood approximation networks*.

The core idea behind computation amortization is to run an expensive process only once, so that the fruits of this labor can later be reused and shared with the rest of the community. Profiting from the computational labor incurred in other research groups enables researchers to consider a larger bank of generative models and to sharpen conclusions that may be drawn from their experimental data. The benefit is three-fold. Experimenters will be able to adjudicate between a rising number of competing models (theoretical accounts), capture richer dynamics informed by neural activity, and at the same time new models proposed by theoreticians can find wider adoption and be tested against data much sooner.

Just as streamlining the analysis of simple SSMs (via e.g., the HDDM toolbox and others) allowed an increase in adoption, streamlining the production and inference pipeline for amortized likelihoods, we hope, will drive the embrace of SSM variations in the modeling and experimental community by making a much larger class of models ready to be fit to experimental data.

Here we develop an extension to the widely used HDDM toolbox, which generalizes it to allow for flexible simulation and estimation of a large class of SSMs by reusing amortized likelihood functions.

Specifically, this extension incorporates,

- LAN (Fengler et al., 2021) based likelihoods for a variety of SSMs (batteries included)
- LAN-driven extension of the Reinforcement Learning (RL) - DDM capabilities, which allows RL learning rules to be applied to all included SSMs
- New plots which focus on visual communications of results across models
- An easy interface for users to import and incorporate their own models and likelihoods into HDDM

This paper is formulated as a tutorial to support application of the HDDM LAN extension for data analysis problems involving SSMs.

The rest of the paper is organized as follows. In section 4.2 we start by providing some basic overview of the capabilities of HDDM. Section 4.3 gives a brief overview of LANs (Fengler et al., 2021). Section 4.4 constitutes a tutorial with a detailed introduction on how to use these new features in HDDM. We conclude in section 4.5 embedding the new features into a broader agenda. Lastly we mention limitations and preview future developments in section 4.6.

4.2 HDDM: The basics

The HDDM Python package (Wiecki, Sofer, and Frank, 2013) was designed to make hierarchical Bayesian inference for drift diffusion models simple for end-users with some programming experience

in Python. The toolbox has been widely used for this purpose by the research community and the feature set evolves to accommodate new use-cases. This section serves as a minimal introduction to HDDM to render the present tutorial self-contained. To get a deeper introduction to HDDM itself, please refer to the original paper (Wiecki, Sofer, and Frank, 2013), an extension paper specifically concerning reinforcement learning capabilities (Mads L Pedersen and Frank, 2020), and the documentation of the package. Here we concern ourselves with a very basic workflow that uses the HDDM package for inference.

Data HDDM expects a dataset, provided as a `pandas DataFrame` (McKinney, 2010) with three basic columns. A `'subj_idx'` column which identifies the subject, a `'response'` column which specifies the choice taken in a given trial (usually coded as 1 for *correct* choices and 0 for *incorrect* choices) and a `'rt'` column which stores the trial-wise reaction times (in seconds). Other columns can be added, for example to be used as covariates (task condition or additional measurements such as trial-wise neural data). Here we take the example of a dataset which is provided with the HDDM package. Codeblock 4.1 shows how to load this dataset into a Python interpreter, which looks as follows,

```
cav_data = hddm.load_csv(hddm.__path__[0] + '/examples/cavanagh_theta_nn.csv')
```

```
      subj_idx stim     rt  response     theta  dbs conf
0            0   LL  1.210       1.0  0.656275    1   HC
1            0   WL  1.630       1.0 -0.327889    1   LC
2            0   WW  1.030       1.0 -0.480285    1   HC
3            0   WL  2.770       1.0  1.927427    1   LC
4            0   WW  1.140       0.0 -0.213236    1   HC
...        ...  ...    ...       ...       ...  ...  ...
3983        13   LL  1.450       0.0 -1.237166    0   HC
3984        13   WL  0.711       1.0 -0.377450    0   LC
3985        13   WL  0.784       1.0 -0.694194    0   LC
3986        13   LL  2.350       0.0 -0.546536    0   HC
3987        13   WW  1.250       1.0  0.752388    0   HC

[3988 rows x 7 columns]
```

Codeblock 4.1. Loading package-included data

HDDM Model Once we have our data in the format expected by HDDM, we can now specify a HDDM model. We focus on a simple example here: a basic hierarchical model, which estimates separate drift rates (v) as a function of task condition, denoted by the `'stim'` column, and moreover estimates the starting point bias z. (Boundary separation, otherwise known as decision threshold, a and non-decision time t, are also estimated by default).

This model assumes that the subject level z, a and t parameters are each drawn from the respective group distributions, the parameters of which are also inferred. The v parameters derive

```
basic_hddm_model = hddm.HDDM(cav_data, include=['z'], depends_on={'v': ['stim']})
```

Codeblock 4.2. Initializing HDDM model

from separate group distributions for each value of `'stim'`. Details about the choices of *group priors* and *hyperparameters* can be found in the original toolbox paper (Wiecki, Sofer, and Frank, 2013). Codeblocks 4.2 and 4.3 show how to construct and sample from such a model.

Sample and Analyze With the HDDM model defined, the goal is to fit this model to a given dataset. In a Bayesian context this implies obtaining a posterior distribution over model parameters. For completeness, we note that such posterior distributions are defined via Bayes' rule,

$$p(\theta|\mathbf{D}) \propto p(\mathbf{D}|\theta)p(\theta)$$

where $\mathbf{D}$ is our *data*, θ is our set of parameters, $p(\mathbf{D}|\theta)$ defines the *likelihood* (analytic in the case of the standard `HDDM` class) of our dataset under the model and $p(\theta)$ defines our initial *prior* over the parameters.

HDDM uses the probabilistic programming toolbox PyMC (Patil, Huard, and C. J. Fonnesbeck, 2010) to generate samples from the posterior distribution via *Markov chain Monte Carlo* (MCMC) (specifically, using coordinate-wise slice samples (Neal, 1995)). To generate samples from the posterior we simply type,

```
basic_hddm_model.sample(1000, burn=500)
```

Codeblock 4.3. Sampling from a basic HDDM model

HDDM then provides access to a variety of tools to analyze the posterior and generate quantities of interest, including:

1. *chain summaries*: To get a quick glance at mean posterior estimates (and their uncertainty) for parameters.

2. *trace-plots* and the *Gelman-Rubin statistic* (Brooks and Gelman, 1998): To understand issues with chain-convergence (i.e., whether one can trust that the estimates are truly drawn from the posterior).

3. *the deviance information criterion* (DIC) (Spiegelhalter et al., 2014) : As a score to be used for purposes of model comparison (with caution).

4. *posterior predictive plots*: To check for the absolute fit of a given model to data (potentially as a function of task condition, etc).

The HDDM LAN extension maintains this basic HDDM workflow, which we hope facilitates seamless transition for current users of HDDM. After some brief explanations concerning *approximate likelihoods*, which form the spine of the extension, we will expose the added capabilities in detail.

4.3 Approximate Likelihoods

Approximate Bayesian inference is an active area of research. Indeed, the last decade has seen a multitude of proposals for new algorithms, many of which rely in one way or another on popular deep learning techniques (Gutmann et al., 2018; Papamakarios and I. Murray, 2016; Papamakarios, Nalisnick, et al., 2019; Papamakarios, Sterratt, and I. Murray, 2019; Lueckmann, Bassetto, et al., 2019; Greenberg, Nonnenmacher, and Macke, 2019; Tejero-Cantero et al., 2020). Relevant to our goals here are algorithms which can estimate trial-by-trial likelihoods for a given model. The main idea is to replace the *likelihood* term in Bayes' Rule, with an approximation $\hat{p}(\mathbf{D}|\theta)$, which can be evaluated via a forward pass through a simple neural network. Once the networks are trained, these "amortized" likelihoods can then be used as a plug-in (replacing the analytical likelihood function) to run approximate inference. Having access to approximate likelihoods, the user will now be able to apply HDDM to a broad variety of sequential sampling models.

The HDDM extension described here is based on a specific likelihood amortization algorithm, which we dubbed *likelihood approximation networks* (LANs) (Fengler et al., 2021). Details regarding this LAN approach, including methods, parameter recovery studies and thorough tests, can be found in Fengler et al., 2021. Note that in principle, our extension supports the integration of *any* approximate (or exact) likelihood, in the context of a now simple interface for adding models to HDDM. The scope remains limited only insofar as HDDM remains specialized towards choice / reaction time modeling. Figure 2 provides some visual intuition regarding concerning the ideas behind LANs.

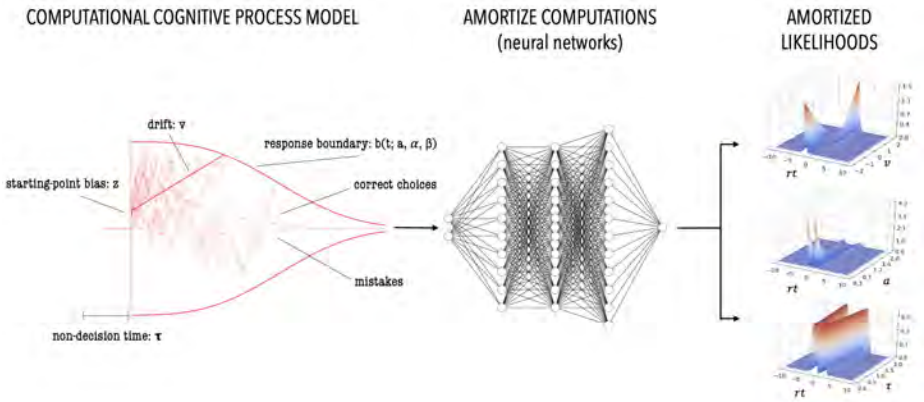

Figure 4.2. Depiction of the general idea behind *likelihood approximation networks*. We use a *simulator* of a *likelihood-free* cognitive process model to generate training data. This training data is then used to train a *neural network* which predicts the *log-likelihood* for a given feature vector consisting of model parameters, as well as a particular choice and reaction time. This neural network then acts as a stand-in for a *likelihood function* facilitating *approximate Bayesian inference*. Crucially these networks are then fully and flexibly reusable for inference on data derived from any experimental design.

4.4 HDDM Extension: Step by Step

A Central Database For Models: `hddm.model_config`

To accommodate the multitude of new models, HDDM > 0.9 now uses a model specification dictionary to extract data about a given model that is relevant for inference. The `model_config` module contains a central dictionary with which the user can interrogate to inspect models that are currently supplied with HDDM. Codeblock 4.4 shows how to list the models included by name. For each model, we have a specification dictionary. Codeblock 4.5 provides an example for the simple DDM.

```
hddm.model_config.model_config.keys()
```

Codeblock 4.4. `model_config` - list available models.

```
hddm.model_config.model_config["ddm"] =
    {
    "params": ["v", "a", "z", "t"],
    "params_trans": [0, 0, 1, 0],
    "param_bounds": [[-3.0, 0.3, 0.1, 1e-3], [3.0, 2.5, 0.9, 2.0]],
    "boundary": hddm.simulators.bf.constant,
    "hddm_include": ["z"],
    "choices": [-1, 1],
    "params_default": [0.0, 1.0, 0.5, 1e-3],
    "params_std_upper": [1.5, 1.0, None, 1.0],
    }
```

Codeblock 4.5. DDM specfications in `model_config`

We focus on the most important aspects of this dictionary (more options are available). Under `"params"` the parameter names for the given model are listed. `"params_trans"` specifies if the sampler should *transform* the parameter at the given position (Transforming parameters can be helpful for convergence, especially if the parameter space is strongly constrained a priori, e.g. between 0 and 1). The order follows the list supplied under `"param"`. `"param_bounds"` lists the parameter-wise lower and upper bounds of parameters that the sampler can explore. This is important in the context of LAN based likelihoods, which are only valid in the range of parameters which were observed during training. We trained the LANs included in HDDM on a broad range of parameters (spanning quite a large range of sensible data, you can inspect the training bounds in the `hddm.model_config.model_config` dictionary under the `param_bounds` key). However, it cannot be guaranteed that these were broad enough for any given empirical dataset. If the provided LANs are deemed inappropriate for a given dataset (e.g., if parameter estimates hit the bounds upon fitting), it is always possible to retrain on an even broader range of parameters. Ruling out convergence issues however, should be the first order of business in such cases.

HDDM uses the *inverse logistic* (or *logit*) transformation for the sampler to operate on an unconstrained parameter space. For a parameter θ and parameter bounds $[a, b]$, this transformation

takes θ from a value in $[a, b]$ to a value x in $(-\infty, \infty)$ via,

$$x = \ln\left(\frac{\theta - a}{b - \theta}\right)$$

A given SSM usually has a `"decision boundary"` which is supplied as a function that can be evaluated over time-points $(t_0, ..., t_n)$, given boundary parameters (supplied implicitly via `"params"`). The values representing each choice are reported as a `list` under `"choices"`. A note of caution: if a user wants to estimate a new model that is not currently in HDDM, a new LAN (or generally likelihood) has to be created, for it to be added to the `model_config` dictionary. Simply changing a setting in an existing `model_config` dictionary will not work. Under the `"hddm_include"` key, a list holds a working default for the `include` argument expected from the `HDDM` classes. Lastly, `"params_default"` specify the parameter values that are fixed (*not fit*) by HDDM and `"params_std_upper"` specify upper bounds on group level standard deviations for each parameter (optional, but this can help constrain the parameter ranges proposed by the sampler, making it more efficient).

These `model_config` dictionaries provide a scaffolding for model specification which is applied throughout all of the new functionalities discussed in the next sections.

Batteries Included: `hddm.simulators`, `hddm.network_inspectors`

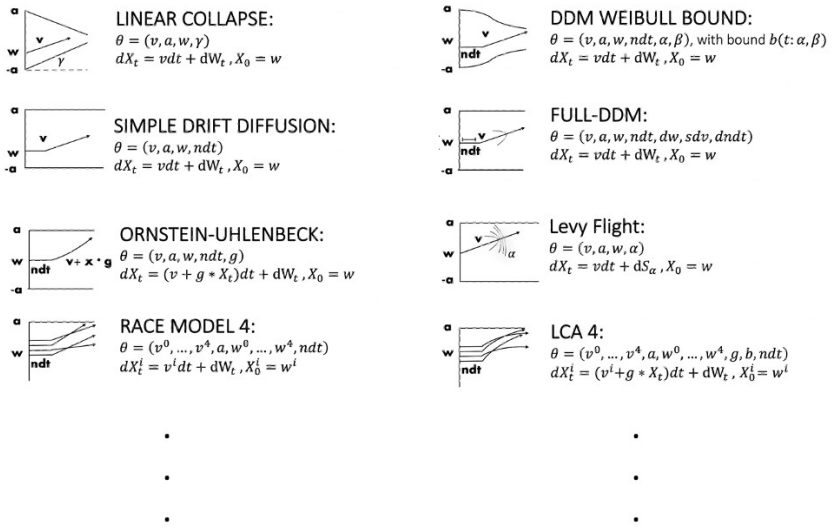

Figure 4.3. Graphical examples for some of the sequential sampling included in HDDM.

The new `HDDMnn` (where *nn* is for *neural network*), `HDDMnnRegressor` and `HDDMnnStimCoding` classes have access to a (growing) stock of supplied SSMs, including rapid compiled (Behnel et al., 2010) simulators, and rapid likelihood evaluation via LANs (Fengler et al., 2021) and their implementation in PyTorch (Paszke et al., 2019). We will discuss how to fit these models to data in the next section. Here we describe how one can access the low level simulators and LANs directly, in

case one wants to adopt them for custom purposes. We also show how to assess the degree to which the LAN approximates the true (empirical) likelihood for a given model. Users who just want to apply existing SSMs in HDDM to fit data can skip to the next section.

As described in the previous section, the user can check which models are currently available by using the `model_config` dictionary. Figure 3, provides some pictorial examples. For a given model, a doc-string includes some information (and possible warnings) about usage. As an example, let us pick the **angle** model, which is a SSM that allows for the decision boundary to decline linearly across time with some estimated angle (note that although other aspects of the model are standard DDM, even in this case the likelihood is analytically intractable. Nevertheless, we previously observed that inference using LANs yields good parameter recovery, as per Fengler et al., 2021).

```
print(hddm.model_config.model_config['angle']['doc'])
```

```
Model formulation is described in the documentation under LAN Extension.
Meant for use with the extension.
```

Codeblock 4.6. `model_config` doc-string for the `angle` model

```
from hddm.simulators import simulator
from hddm.model_config import model_config
out = simulator(model='angle', theta=model_config['angle']['params_default'],
                n_samples=100, delta_t=0.001)
```

Codeblock 4.7. Using the `simulator` simulator for generating synthetic data

Codeblock 4.6, illustrates such a doc-string. Codeblock 4.7 shows how we can simulate synthetic data from this model. Following the code, the variable `out` is now a three-tuple. The first element contains an array of *reaction times*, the second contains an array of *choices* and finally the third element returns a *dictionary of metadata* concerning the simulation run. Next, we can access the LAN corresponding to our **angle** model directly by typing the code in codeblock 4.8. We can utilize the `get_torch_mlp` function, which is defined in the `network_inspectors` submodule.

```
from hddm.network_inspectors import get_torch_mlp
lan_angle = get_torch_mlp(model='angle')
```

Codeblock 4.8. Loading a torch network from the package

The `lan_angle` object defined in codeblock 4.8 is in fact a method, which defines the forward pass through a given LAN. It expects as input a matrix where each row defines a parameter vector suitable for the SSM of choice (here **angle**, so we need a value for each of the parameters `['v', 'a', 'z', 't', 'theta']` which can be found in our `model_config` dictionary). Two elements are then added: a *reaction time* and a *choice* at which we would like to evaluate our likelihood. Codeblock 4.9 provides a full example.

```
# Make some random parameter set
from hddm.simulators import make_parameter_vectors_nn

parameter_df = make_parameter_vectors_nn(model='angle',
                        param_dict=None, n_parameter_vectors=1)

parameter_matrix = np.tile(np.squeeze(parameter_df.values), (200, 1))

# Initialize network input
network_input = np.zeros((parameter_matrix.shape[0], parameter_matrix.shape[1] + 2))

# Note the + 2 on the right we append the parameter vectors with
# reaction times (+1 columns) and choices (+1 columns)

# Add reaction times
network_input[:, -2] = np.linspace(0, 3, parameter_matrix.shape[0])

# Add choices
network_input[:, -1] = np.repeat(np.random.choice([-1, 1]), parameter_matrix.shape[0])

# Note: The networks expects float32 inputs
network_input = network_input.astype(np.float32)

# Show example output
print('Some network outputs')
print(lan_angle(network_input)[:10]) # printing the first 10 outputs
print('Shape')
print(lan_angle(network_input).shape) # original shape of output
```

```
Some network outputs
[[-2.9323568]
 [ 2.078088 ]
 [ 0.4104141]
 [-0.5943402]
 [-1.1136726]
 [-1.6901499]
 [-2.3512228]
 [-3.080151 ]
 [-3.8215086]
 [-4.4257374]]
Shape
(200, 1)
```

Codeblock 4.9. Check forward pass of supplied `angle` network.

To facilitate a simple sanity check, we provide the `kde_vs_lan_likelihoods` plot, which can be accessed from the `network_inspectors` submodule. This plot lets the user compare LAN likelihoods against empirical likelihoods from simulator data for a given matrix of parameter vectors (Fengler

et al., 2021). The empirical likelihoods are defined via kernel density estimators (KDEs) (Silverman, 1986). We show an example in codeblock 4.10. Figure 4.4 shows the output.

```
from hddm.network_inspectors import kde_vs_lan_likelihoods

# Make a set of parameter vectors
parameter_df = make_parameter_vectors_nn(model=model, param_dict=None,
                        n_parameter_vectors=6)

# Generate plot
kde_vs_lan_likelihoods(parameter_df=parameter_df, model=model,
                       n_samples=1000, n_reps=10, font_scale=1.25)
```

Codeblock 4.10. Example usage of the `kde_vs_lan_likelihood()` function to compare LAN likelihoods to empirical kernel-density estimates.

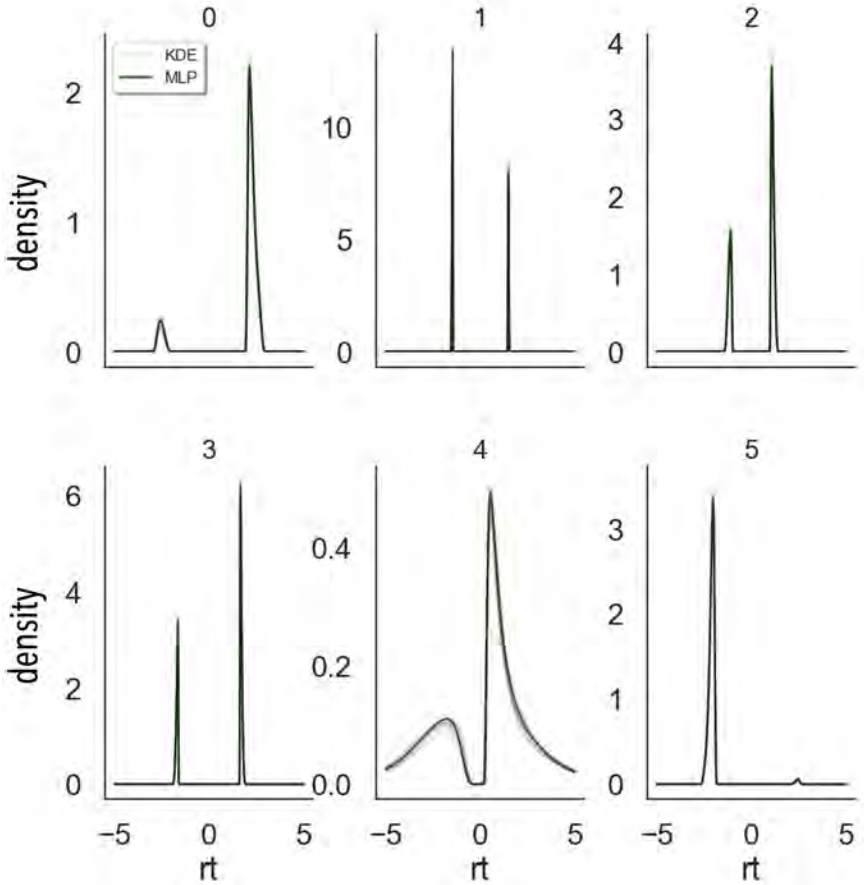

Figure 4.4. Example of a `kde_vs_lan_likelihoods` plot. If the green (deterministic) and gray (stochastic) lines overlap, then the approximate likelihood (MLP for multilayered perceptron, the neural network that provides our LAN) is a good fit to the actual likelihood.

Fitting data using `HDDMnn`, `HDDMnnRegressor`, and `HDDMnnStimCoding` classes

Using the `HDDMnn`, `HDDMnnRegressor` and `HDDMnnStimCoding` classes, we can follow the general workflow established by the basic `HDDM` package to perform Bayesian inference. In this section we will fit the **angle** model to the example dataset provided with the HDDM package. Codeblock 4.11 shows us how to load the correponding dataset, after which we can set up our HDDM model, and draw 1000 MCMC samples using the code in codeblock 4.12.

```
cav_data = hddm.load_csv(hddm.__path__[0] + '/examples/cavanagh_theta_nn.csv')
```

```
      subj_idx stim     rt  response     theta  dbs conf
0            0   LL  1.210       1.0  0.656275    1   HC
1            0   WL  1.630       1.0 -0.327889    1   LC
2            0   WW  1.030       1.0 -0.480285    1   HC
3            0   WL  2.770       1.0  1.927427    1   LC
4            0   WW  1.140       0.0 -0.213236    1   HC
...        ...  ...    ...       ...       ...  ...  ...
3983        13   LL  1.450       0.0 -1.237166    0   HC
3984        13   WL  0.711       1.0 -0.377450    0   LC
3985        13   WL  0.784       1.0 -0.694194    0   LC
3986        13   LL  2.350       0.0 -0.546536    0   HC
3987        13   WW  1.250       1.0  0.752388    0   HC

[3988 rows x 7 columns]
```

Codeblock 4.11. Loading package supplied **cavanagh** dataset.

```
hddmnn_model_cav = hddm.HDDMnn(cav_data, model='angle', include=['z', 'theta'],
                               is_group_model = True)
hddmnn_model_cav.sample(1000, burn=500)
```

```
 [-----------------100%-----------------]
 1001 of 1000 complete in 365.3 sec
```

Codeblock 4.12. Sampling from a HDDMnn model.

We note a few differences between a call to construct a `HDDMnn` class and a standard `HDDM` class. First, the supply of the `model` argument specifying which SSM to fit (requires that this model is already available in HDDM; see above). Second, the inclusion of model-specific parameters under the `include` argument. The workflow is otherwise equivalent, a fact that is conserved for the `HDDMnnRegressor` and `HDDMnnStimCoding` classes. A third difference concerns the choice of argument defaults. The `HDDMnn` class uses non-informative priors, instead of the informative priors derived from the literature which form the default for the basic `HDDM` class. Since, as per our earlier discussions, variants of SSMs are historically rarely fit to experimental data, we can not easily derive reasonable informative priors from the literature and therefore choose to remain agnostic in our beliefs about the parameters underlying a given dataset. If the research community starts fitting SSM variants

to experimental data, this state of affairs may evolve through collective learning. At this point we caution the user to however not use these new models blindly. We strongly encourage conducting appropriate parameter recovery studies, specific to the experimental dataset under consideration. We refer to the section on *inference validation tools* below, for how HDDM might help in this procedure.

New Visualization Plots: `hddm.plotting`

Based on our model fit from the previous section, we illustrate a few new informative plots, which are now included in HDDM. We can generally distinguish between two types of plots. Plots which use the traces only (to display posterior parameter estimates) and plots which make use of the model simulators (to display how well the model can reproduce empirical data given posterior parameters). The first such plot is produced by the the `plot_caterpillar` function, which presents an approximate posterior 99%-HDI (specifically we show the 1% to 99% range in the cumulative distribution function of the posterior), for each parameter. Codeblock 4.13 shows us how to invoke this function and Figure 4.5 illustrates the resulting plot.

```
from hddm.plotting import plot_caterpillar
plot_caterpillar(hddm_model=hddmnn_model_cav, figsize=(8, 8), columns=3)
```

Codeblock 4.13. Example usage of the `caterpillar_plot()` function.

The second such plot is the a posterior pair plot, called via the `plot_posterior_pair` function. This plot shows the pairwise posterior distribution, subject by subject (and, if provided, condition by condition). Codeblock 4.14 illustrates how to call this function, and Figure 4.6 exemplifies the resulting output.

```
from hddm.plotting import plot_posterior_pair
plot_posterior_pair(hddmnn_model_cav, samples=500, figsize=(6, 6))
```

Codeblock 4.14. Example concerning usage of the `plot_posterior_pair()` function.

A last very useful plot addition is what we call the **model plot**, an extension to the standard posterior predictive plot, which can be used to visualize the impact of the parameter posteriors on decision dynamics. For example, if one is estimating a linearly collapsing bound, instead of just interpreting the posterior angle parameter, one can see how that translates to the evolving decision bound over time in tandem with the estimating drift rate, etc. It is an extension of the `plot_posterior_predictive` function. This function operates by manipulating `matplotlib axes` objects, via a supplied *axes manipulator*. The novel *axis manipulator* in the example show in Codeblock 4.15 is the `_plot_func_model` function. Figure 4.7 show the resulting plot.

We use this moment to illustrate how the **angle** model in fact outperforms the **DDM** on this example dataset. For this purpose we take an example subject from Figure 4.7 and contrast the posterior predictive of the **angle** model with the posterior predictive of the **DDM** side by side in Figure 4.8. We clearly see that the **DDM** model has trouble capturing the leading edge and the tail

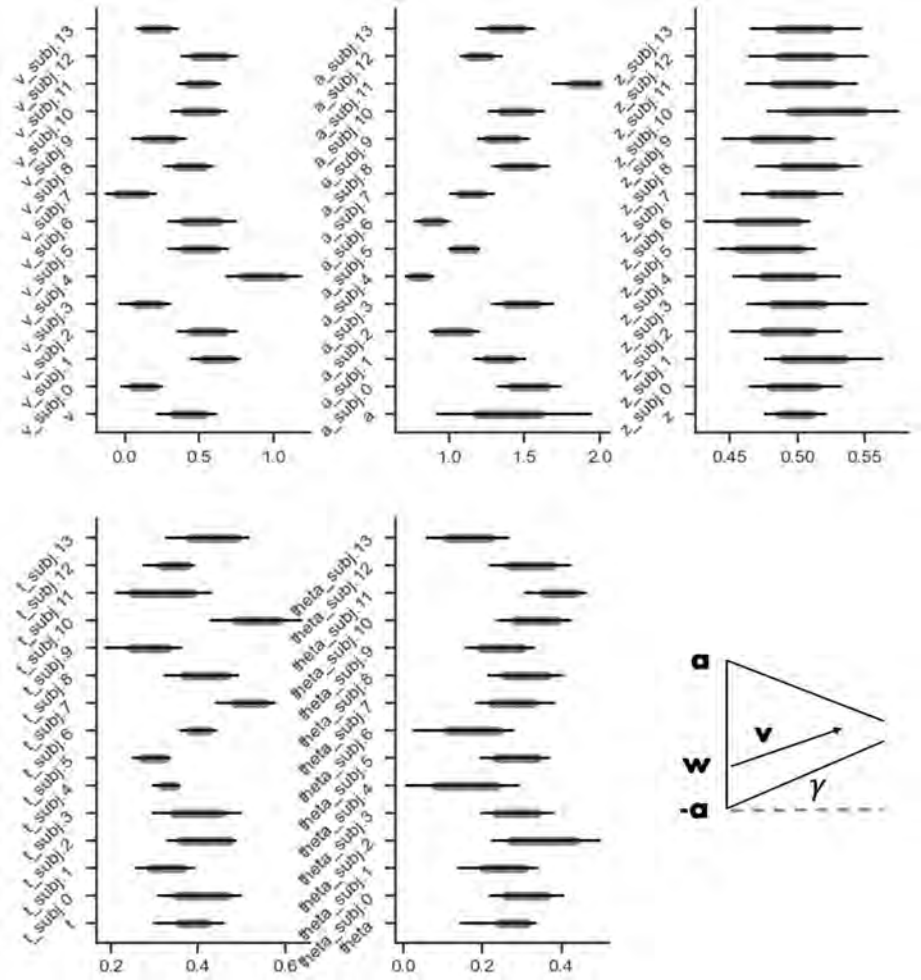

Figure 4.5. Example of a `caterpillar_plot`. The plot, split by model parameters, shows the 99% (line-ends) and 95% (gray band ends) highest density intervals (HDIs) of the posterior for each parameter. Multiple styling options exist.

```
from hddm.plotting import plot_posterior_predictive

plot_posterior_predictive(model = hddmnn_model_cav, columns=3,  figsize=(10, 12),
                          groupby=['subj_idx'], value_range=np.arange(0.0, 3, 0.1),
                          plot_func=hddm.plotting._plot_func_model,
                          **{'alpha': 0.01, 'ylim': 3, 'samples': 200,
                             'legend_fontsize': 7., 'legend_location': 'upper left',
                             'add_posterior_uncertainty_model': True,
                             'add_posterior_uncertainty_rts': False,
                             'subplots_adjust': {'top': 0.94, 'hspace': 0.35, 'wspace': 0.3}
                          })
```

Codeblock 4.15. Example usage of the `plot_posterior_predictive()` function

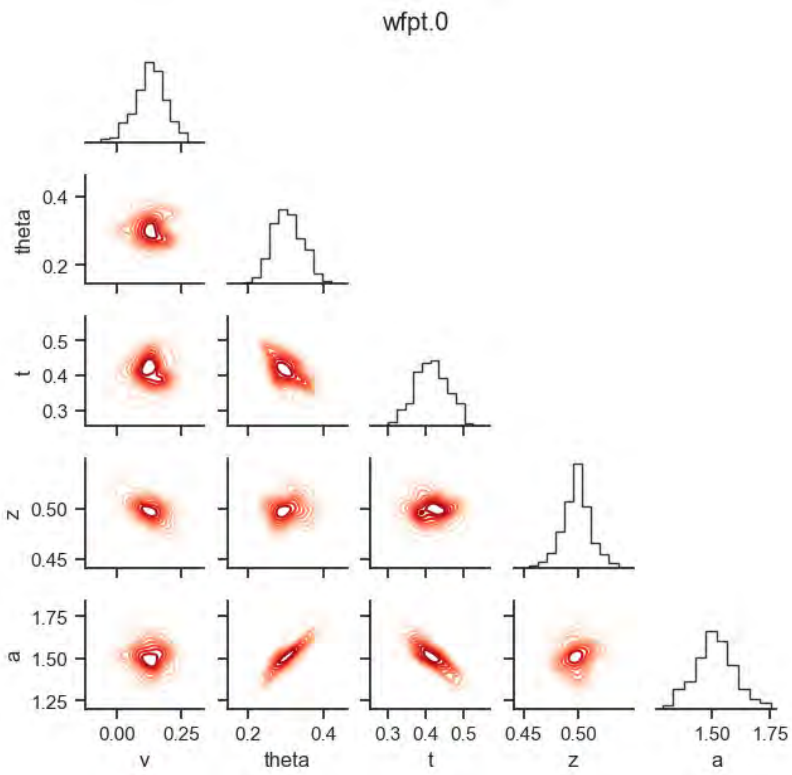

Figure 4.6. Example of a `posterior_pair_plot` in the context of parameter recovery. The plot is organized per stochastic node (here, grouped by the `'subj_idx'` column where in this example `'subj_idx' = '0'`). The diagonal shows the *marginal posterior* of a given parameter as a histogram. The elements below the diagonal show pair-wise posteriors via (approximate) level curves. These plots are especially useful to identify parameter collinearities, which indicate parameter-tradeoffs and can hint at issues with identifiability. This example shows how the `theta` (boundary collapse) and `a` (boundary separation) parameters as well as the `t` (non-decision time) and `a` parameters trade-off in the posterior. We refer to Fengler et al., 2021 for parameter recovery results using the underlying *angle* SSM. We note that such parameter trade-offs and attached identifiability issues derive not just from a given likelihood model, but are also affected by the data and parameter structure as task design and modeling choices.

behavior of the rt distributions simultaneously, while the **angle** model strikes a much better balance. While this example does not present a fully rigorous model comparison (DIC scores for example however bear out the same conclusion) exercise, it provides a hint at the benefits one may expect from utilizing an expanded model space.

Inference Validation Tools: `simulator_h_c()`

Validating that a model is identifiable on simulated data is an important aspect of a trustworthy inference procedure (Tran et al., 2021; Evans, Trueblood, and Holmes, 2020; Holmes and Trueblood, 2018; Wilson and Collins, 2019). We have two layers of uncertainty in this regard. First, LANs are approximate likelihoods. A model that is otherwise identifiable could in principle lose this

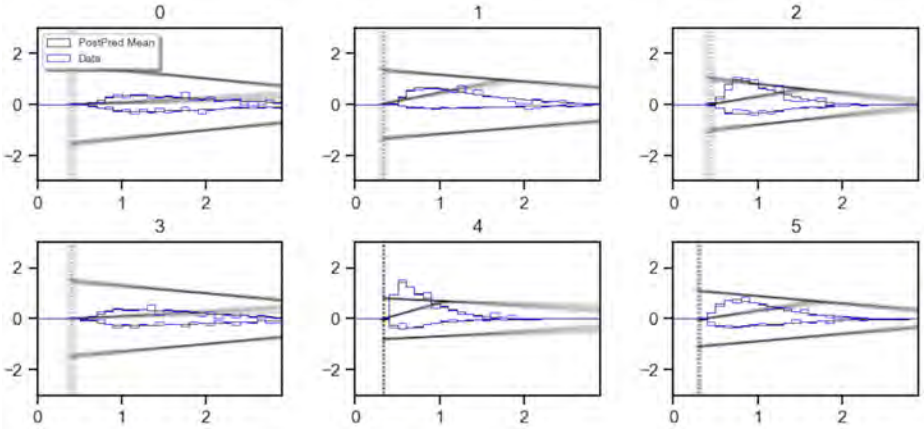

Figure 4.7. Example of a `model_plot`. This plot shows the underlying data in blue, choices and reaction times presented as histograms (positive y-axis for choice option 1, negative y-axis or choice option 0 or −1). The black histograms show the reaction times and choices under the parameters corresponding to the posterior mean. In addition the plot shows a graphical depiction of the model corresponding to parameters drawn from the posterior distribution in black. Various options exist to add and drop elements from this plot; the provided example corresponds to what we consider the most useful settings for purposes of illustration. Note that, in the interest of space, we only illustrate the first six subjects here.

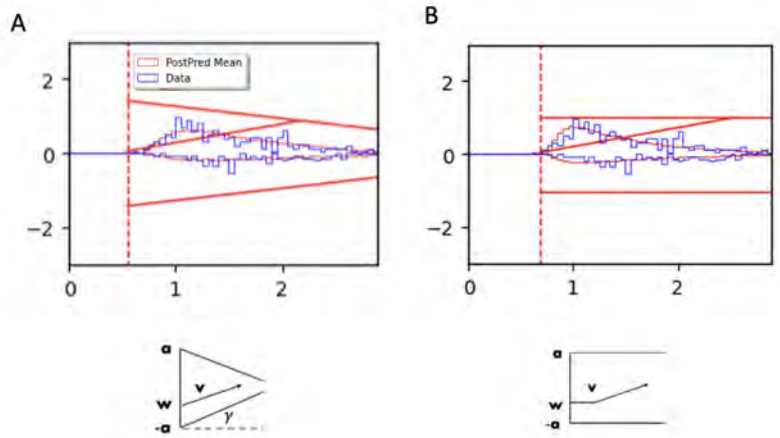

Figure 4.8. Contrasting the posterior predictive of the **angle** and **DDM** model on an example subject. **A**) shows the **angle** model and **B**) shows the **DDM**. While the fits are not dramatically better for this dataset (in our experience more extreme differences can be seen in other cases), the **angle** model shows two characteristic differences to the **DDM** model fit. First, it better captures the graceful initial increase in density for short reaction times. Second, it captures the slower decrease in density for longer reaction times, as compared to the **DDM**, for which the reaction time density falls of quicker than is apparent in the data. Both of these effects are directly produced by allowing a collapsing bound, instead of the **DDM's** static, parallel bounds.

property when using LANs to estimate its parameters from a dataset, should the LAN not have been trained adequately. Second, a given model can inherently be unidentifiable for a given dataset and or theoretical commitments (regardless of whether its likelihood is analytic or approximate). As a simple

example consider an experimental dataset, which does not include enough samples to identify the parameter of a model of interest with any degree of accuracy. Slightly more involved, the posterior could tend to be multi-modal, a problem for MCMC samplers that can lead to faulty inference. While increasing the number of trials in an experiment and/or increasing the number of participants can help remedy this situation, this is not a guarantee. Apart from the size and structure of the empirical dataset, our modeling commitments play an important role for identifiability too. As an example, we might have experimental data from a random dot motion task and we are interested in modeling the *choices* and *reaction times* of participating subjects with our **angle** model. A reasonable assumption is that the v parameter (a rough proxy for processing speed) differs depending on the difficulty of the trial. However, the parameters t and a may not depend on the difficulty since we do not have a good a-priori theoretical reason to suspect that the *non-decision time* (t) and the initial the *boundary separation* a (the degree of evidence expected to take a decision) will differ across experimental conditions. These commitments are embedded in the model itself (they are assumptions on the data generating process imposed by the modeler), and determine jointly with an experimental dataset whether inference can be successful. For a modeler it is therefore of paramount importance to check whether their chosen combinations of theoretical commitments and experimental dataset jointly lead to an inference procedure that is accurate. Since the space of models incorporated into HDDM has been significantly expanded with the LAN extension, we provide a few tools to help facilitate parameter recovery studies which are relevant to real experimental data analysis and plan to supplement these tools even further in the future.

First, we provide the `simulator_h_c` function, in the `hddm_dataset_generators` submodule. The function is quite flexible, however we will showcase a particularly relevant use-case. Taking our `cav_data` dataset loaded previously, we would like to generate data from our **angle** model in such a way that we encode assumptions about our model into the generated dataset. In the example below we assume that the v and *theta* parameters vary as a function of the `"stim"` column. For each value of `"stim"` a group-level μ and σ (defining the *mean* and *standard deviation* of a group level Normal distribution) are generated and subject-level parameters are sampled from this group distribution. This mirrors exactly the modeling assumptions when specifying a HDDM model with the `depends_on` argument set to `{'v': 'stim', 'theta': 'stim'}`. Codeblock 4.16 provides an example on how to call this function. The `simulator_h_c` function returns the respective dataset (here `sim_data`) exchanging values in the previous *rt* and *response* columns with simulation data. Trial-by-trial parameters are attached to the dataframe as well. The `parameter_dict` dictionary contains all the parameters of the respective hierarchical model which was used to generate the synthetic data. This parameter dictionary follows the parameter naming conventions of HDDM exactly. We can fit this data using the `HDDMnn` class as illustrated in Codeblock 4.17.

The plots defined in the previous section allow us to specify a `parameter_recovery_mode` which we can utilize to check how well our estimation worked on our synthetic dataset. Codeblocks 4.18, 4.19 and 4.20 and Figures 4.9, 4.10 and 4.11 show respectively code and plot examples.

Note how both the `plot_posterior_pair` function as well as the `plot_posterior_predictive`

```
# Generate some data
from hddm.simulators.hddm_dataset_generators import simulator_h_c
sim_data, parameter_dict = simulator_h_c(data=cav_data, model='angle',
                                         p_outlier=0.00,
                                         depends_on={'v': ['stim'], 'theta': ['stim']},
                                         regression_models=None,
                                         regression_covariates=None,
                                         group_only_regressors=False,
                                         group_only=None, fixed_at_default=None)

sim_data
```

```
      subj_idx stim        rt  response     theta  dbs conf         v  \
0            0   LL  2.020890       1.0  0.442532    1   HC -0.474451
1            0   WL  2.075889       1.0  0.844691    1   LC -0.865643
2            0   WW  2.119889       1.0  0.661660    1   HC  0.752663
3            0   WL  1.804893       0.0  0.844691    1   LC -0.865643
4            0   WW  2.410885       1.0  0.661660    1   HC  0.752663
...        ...  ...       ...       ...       ...  ...  ...       ...
3983        13   LL  2.874057       1.0  0.402371    0   HC -0.473813
3984        13   WL  2.169051       0.0  0.972350    0   LC -1.001207
3985        13   WL  1.798055       0.0  0.972350    0   LC -1.001207
3986        13   LL  1.709054       1.0  0.402371    0   HC -0.473813
3987        13   WW  2.115052       1.0  0.911009    0   HC  0.824063

             a         z         t
0     1.402356  0.577363  1.468893
1     1.402356  0.577363  1.468893
2     1.402356  0.577363  1.468893
3     1.402356  0.577363  1.468893
4     1.402356  0.577363  1.468893
...        ...       ...       ...
3983  1.283326  0.616165  1.618054
3984  1.283326  0.616165  1.618054
3985  1.283326  0.616165  1.618054
3986  1.283326  0.616165  1.618054
3987  1.283326  0.616165  1.618054

[3988 rows x 11 columns]
```

Codeblock 4.16. Using the `simulator_h_c()` function.

function take the `parameter_recovery_mode` argument to add a ground truth to the visualization automatically (the ground truth is expected to be included in the dataset attached to the HDDM model itself). The `plot_caterpillar` function needs a `ground_truth_parameter_dict` argument to add the ground truth parameters. The `simulator_h_c` function provides such a compatible dictionary of ground truth parameters. Using the set of tools in this section, we hope that HDDM conveniently facilitates application relevant parameter recovery studies.

```
hddmnn_model_sim = hddm.HDDMnn(sim_data, model='angle',
                               is_group_model=True, p_outlier=0.00,
                               include=['v', 'a', 't', 'z', 'theta'],
                               depends_on = {'v': ['stim'], 'theta': ['stim']},
                              )
hddmnn_model_sim.sample(1000, burn=500)
```

```
 [-----------------100%-----------------]
 1001 of 1000 complete in 1436.2 sec
```

Codeblock 4.17. Fitting a `HDDMnn` model to synthetic data.

```
from hddm.plotting import plot_caterpillar
plot_caterpillar(hddm_model=hddmnn_model_sim,
                 ground_truth_parameter_dict=parameter_dict,
                 figsize=(10, 15), y_tick_size=6, columns=3)
```

Codeblock 4.18. `caterpillar plot` on fit to simulated data.

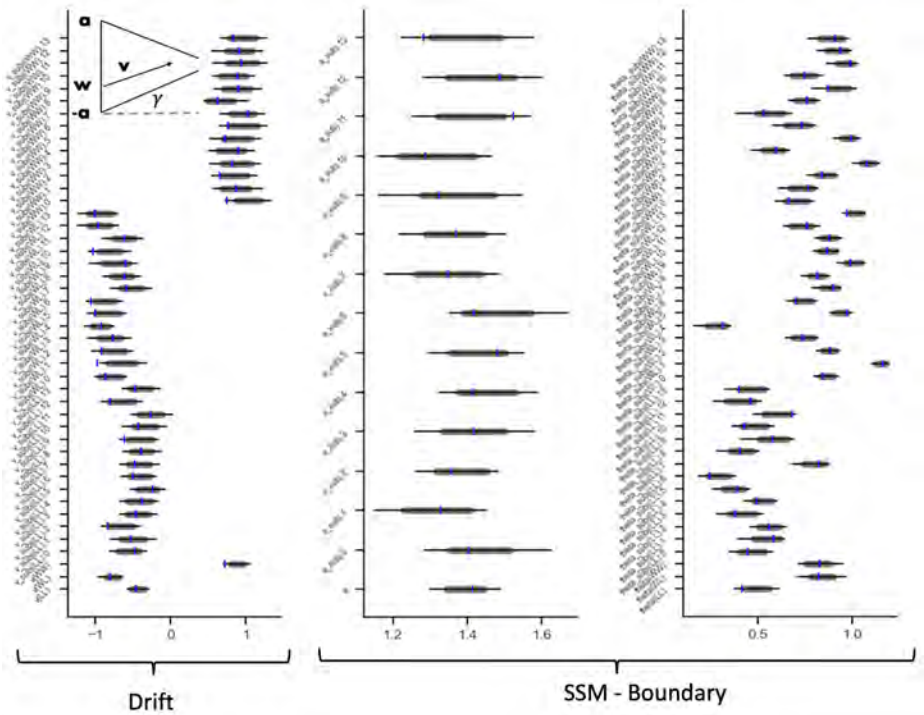

Figure 4.9. Example of a `caterpillar_plot`. The plot is split by model parameter kind, showing parameter-wise, the 99% (line-ends) and 95% (gray band ends) highest density intervals (HDIs) of the posterior. In the context of parameter recovery studies, the user can provide ground-truth parameters to the plot, which will be shown as blue tick-marks on top of the HDIs. Multiple styling options exist. Note, in the interest of space, we show only three of the five basic parameters here. `['v', 'a', 'theta']` of the underlying model (leaving out `['z', 't']`).

```
from hddm.plotting import plot_posterior_predictive
plot_posterior_predictive(model = hddmnn_model_sim, columns = 3, figsize = (10, 12),
                          groupby = ['subj_idx'], value_range = np.arange(0.0, 3, 0.1),
                          plot_func = hddm.plotting._plot_func_model,
                          parameter_recovery_mode = True,
                          **{'alpha': 0.01, 'ylim': 3,
                          'add_model': True, 'samples': 200,
                          'legend_fontsize': 7., 'legend_location': 'upper left',
                          'add_posterior_uncertainty_rts': False,
                          'add_posterior_uncertainty_model': True,
                          'add_posterior_mean_model': True,
                          'add_posterior_mean_rts': True,
                          'subplots_adjust': {'top': 0.94, 'hspace': 0.35, 'wspace': 0.3}
                          })
```

Codeblock 4.19. `model_plot` for fit to simulated data.

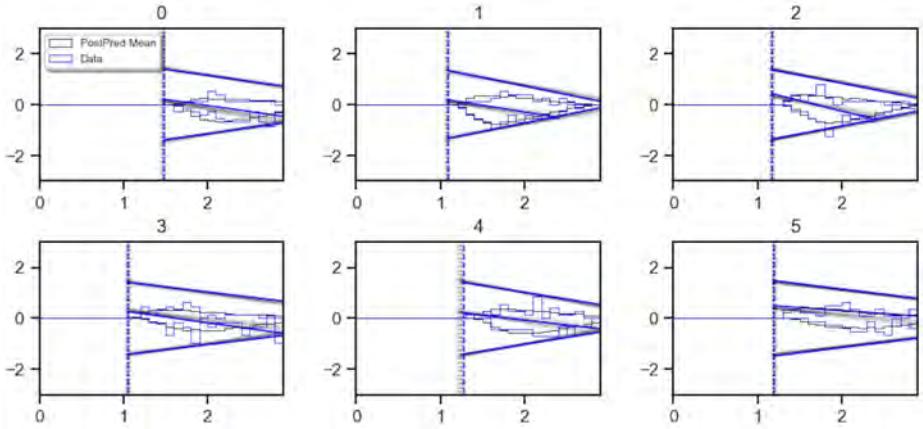

Figure 4.10. Example of a `model_plot`. This plot shows the underlying data in blue, choices and reaction times presented as a histogram (positive y-axis for choice option 1, negative y-axis or choice option 0 or −1). The black histograms show the reaction times and choices under the parameters corresponding to the posterior mean. In addition the plot shows a graphical depiction of the model corresponding to parameters drawn from the posterior distribution in black, as well as such a depiction for the *ground truth* parameters in blue, in case these were provided (e.g., if one is performing recovery from simulated data). Inclusion of the ground truth parameters distinguishes the present display form Figure 4.7. Various options exist to add and drop elements from this plot, the provided example corresponds to what we consider the most useful settings for purposes of illustration. Note that, in the interest of space, we only illustrate the first six subjects here.

```
from hddm.plotting import plot_posterior_pair
posterior_pair_plot(hddmnn_model_sim, parameter_recovery_mode = True,
                    samples=500, figsize=(6, 6))
```

Codeblock 4.20. `posterior_pair_plot` for fit to simulated data.

Adding to the bank of SSMs: User Supplied Custom Models

The new models immediately available for use with HDDM are just the beginning. HDDM allows users to define their own models via adjusting the `model_config` and the provision of custom likelihood

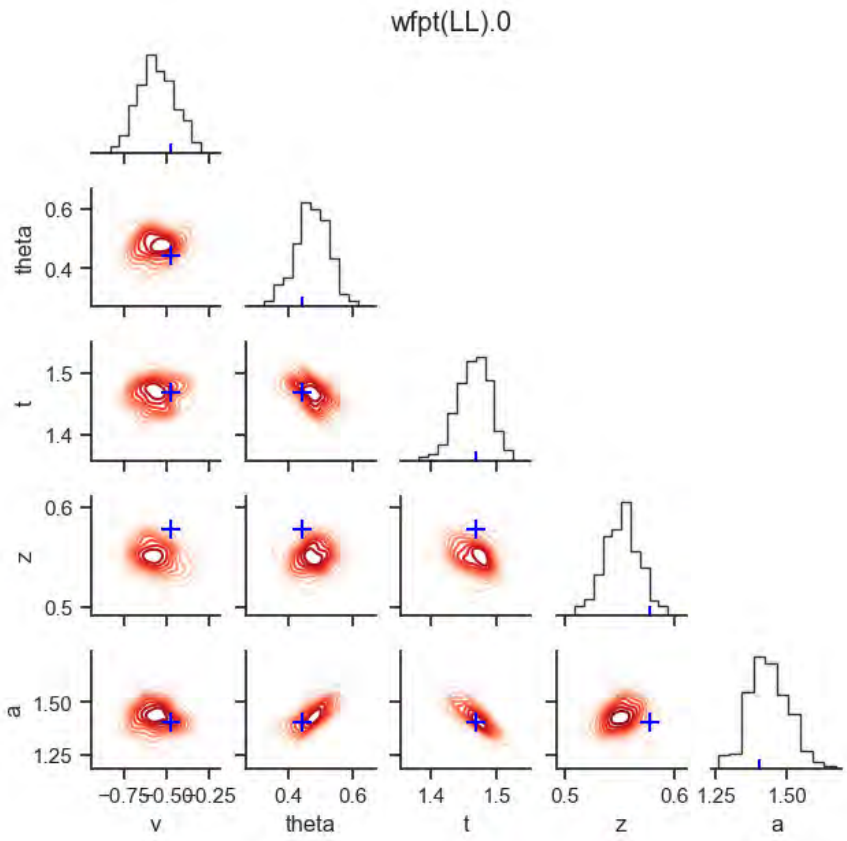

Figure 4.11. Example of a `posterior_pair_plot` in the context of parameter recovery. The plot is organized per stochastic node (here, grouped by the `'stim'` and `'subj_idx'` columns where in this example (`'stim' = 'LL'`, `'subj_idx' = '0'`). The diagonal show the *marginal posterior* of a given parameter as a histogram, adding the *ground truth* parameter as a blue tick-mark. The elements below the diagonal show pair-wise posteriors via (approximate) level curves, and add the respective *ground truths* as a blue cross.

functions. The goal of this functionality is two fold. First, we aim to make HDDM maximally flexible for advanced users, cutting down red-tape to allow creative usage. Second, we hope to motivate users to follow through with a two-step process of model integration. Step one involves easy testing of new likelihoods through HDDM, however with somewhat limited auxiliary functionality (one can generate plots based on the posterior traces, but other plots will not work because of the lack of a simulator). Step two involves sharing the model likelihood and a suitable simulator with the community to allow full integration with HDDM as well as other similar toolboxes which operate across programming languages and probabilistic programming frameworks. In future work we hope to flesh out a pipeline that allows users to follow a simple sequence of steps to full integration of their custom models with HDDM. Here, we show how to complete step one, defining a `HDDMnn` model with a custom likelihood to allow fitting a new model through HDDM. See the section on *future work* for some guidance on producing your own LAN, or contact the authors.

We start with configuring the `model_config` dictionary. We add a `"custom"` key and assign

the specifics of our new model. For illustration purposes we will add the **angle** model to HDDM (even though it is already provided with the LAN extension). Additionally we need to define a basic likelihood function that takes in a vector (or matrix / `2d numpy array`) of parameters, ordered according to the list in the `"params"` key above. As an example, we load our LAN for the **angle** model (as supplied by HDDM) as if it is a custom network. Finally we can fit our newly defined *custom model*. Codeblock 4.21 illustrates the whole process.

```
from hddm.torch.mlp_inference_class import load_torch_mlp
# Define our custom model config
my_model_config = {
            "params": ["v", "a", "z", "t", "theta"],
            "params_trans": [0, 0, 1, 0, 0],
            "params_std_upper": [1.5, 1.0, None, 1.0, 1.0],
            "param_bounds": [[-3.0, 0.3, 0.1, 1e-3, 0.0], [3.0, 2.5, 0.9, 2.0, 1.1]],
            "boundary": hddm.simulators.bf.constant,
            "params_default": [0.0, 1.0, 0.5, 1e-3],
            "hddm_include": ["z"],
            "choices": [-1, 1],
            "slice_widths": {"v": 1.5, "v_std": 1, "a": 1, "a_std": 1,
                             "z": 0.1, "z_trans": 0.2, "t": 0.01, "t_std": 0.15},
            }
# Load our custom network (here we load one supplied by HDDM)
custom_network = load_torch_mlp(model='angle')
# Define HDDM model
hddm_model_custom = hddm.HDDMnn(data=data, include=["z", "theta"],
                                model='custom', model_config=my_model_config,
                                network=custom_network)
# Sample from the HDDM model
hddm_model_custom.sample(1000, burn=500)
```

```
 [-----------------100%-----------------]
 1001 of 1000 complete in 365.3 sec
```

Codeblock 4.21. Construct and fit a HDDMnn model using a custom likelihood.

Note that the only difference to a normal call to the `hddm.HDDMnn` class is supplying appropriate model specifications for our custom likelihood. We supply the `model` argument as `"custom"`, alongside our own config dictionary to the `model_config` argument. In addition, we explicitly pass to the `network` argument, our `custom_network` defining the likelihood.

Moreover, we note that the supply of custom networks opens up multiple degrees of freedom to explore improved likelihood approximations. As an example, users may utilize LANs trained on the log-RT distributions instead of the original RT distributions of a SSM.

Combining SSMs with Reinforcement Learning

While the previous sections focused on employing SSMs in modeling stationary environments, a host of commonly applied experimental task paradigms involve some form of learning that results from the

agent's interactions with the environment. While SSMs can be used to model the decision processes, we need additional machinery to capture the learning dynamics that arise while subjects perform such tasks. Reinforcement learning (RL) (Sutton and Barto, 2018) is one computational framework which can allow us to account for such learning processes. In reinforcement learning, researchers typically assume a simple *softmax* choice rule, informed by some 'utility' (or 'goodness') measure of taking a particular action in a given state. Mathematically, the choice probabilities are expressed as,

$$p(action_i; t) = \frac{e^{q_{action,i}(t)}}{\sum_j e^{q_{action,j}(t)}}$$

While reinforcement learning models can account for learning dynamics in basic choice behavior, the choice functions commonly employed (e.g., softmax) cannot capture the reaction time. To combine the strengths of sequential sampling models and reinforcement learning models, recent studies have used the drift diffusion model to jointly model choice and response time distributions during learning (Mads Lund Pedersen, Frank, and Biele, 2017; Mads L Pedersen and Frank, 2020; Laura Fontanesi et al., 2019). Such an approach allows researchers to study not only the across-trial dynamics of learning but also the within-trial dynamics of choice processes, using a single model. The main idea behind these models is to allow a reinforcement learning process to drive the trial-by-trial parameters of a sequential sampling model (such as the basic drift diffusion model), which in turn is used to jointly capture reaction time and choice behavior for a given trial. This can be applied in complex tasks which involve learning from feedback (see Figure 4.12). This results in a much more broadly applicable class of models and naturally lends itself for use in computational modeling of numerous cognitive tasks where the 'learning process' informs the 'decision-making process'. Indeed, a recent study showed that the joint modeling of choice and RT data can improve parameter identifiability of RL models, by providing additional information about choice dynamics (Ballard and McClure, 2019). However, to date, such models have been limited by the form of the decision model. Many RL tasks involve more than two responses, making the DDM inapplicable. Similarly, the assumption of a fixed threshold may not be valid. For example, during the early learning phase, the differences in Q-values, and hence drift rates, will be close to zero and there is little value in accumulating evidence. A standard DDM model would predict that such choices are associated with very long tail RT distributions. A more appropriate assumption would be that learners use a collapsing bound so that when no evidence is present, the decision process can terminate.

Utilizing the power of LANs, we can further generalize the RL-DDM framework to include a much broader class of SSMs as the 'decision-making process'. The rest of this section provides some details and code examples for these new RL-SSMs.

Test-bed We test our method on a synthetic dataset of the two-armed bandit task with binary outcomes. However, our approach can be generalized to any n-armed bandit task given a pre-trained LAN that outputs likelihoods for the corresponding n-choice decision process (e.g. race models). The model employed a simple delta learning rule (Rescorla, 1972) to update the action values

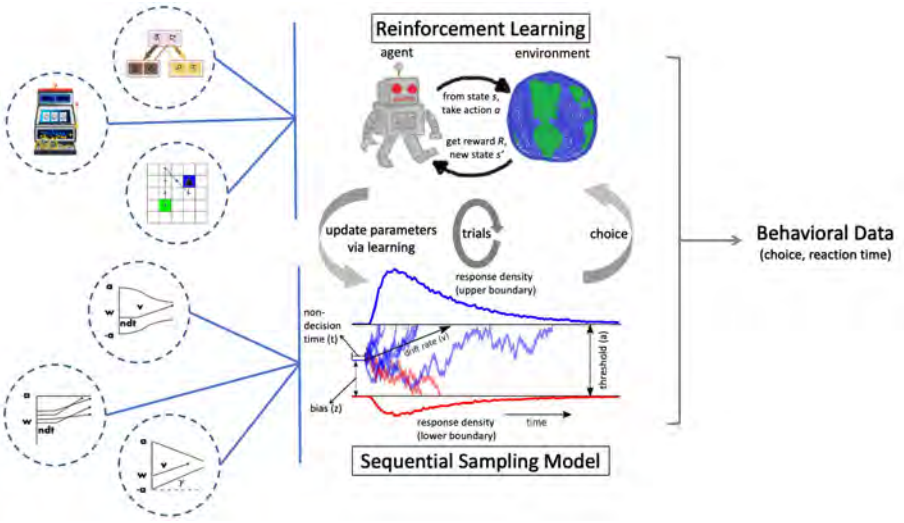

Figure 4.12. RLSSM - combining reinforcement learning and sequential sampling models.

$$q_{action,i}(t+1) = q_{action,i}(t) + \alpha * [r(t) - q_{action,i}(t)],$$

where $q_{action}(t)$ denotes expected reward (Q-value) for the chosen action at time t, $r(t)$ denotes reward obtained at time t and α (referred to as *rl_alpha* in the result plots) denotes the learning rate. The trial-by-trial drift rate depends on the expected reward value learned by the RL rule. The drift rate is therefore a function of Q-value updates, and is computed by the following linking function

$$v(t) = [q_{action,1}(t) - q_{action,2}(t)] * s,$$

where s is a scaling factor of the difference in Q-values. In other words, the scalar s is the drift rate when the difference between the Q-values of both the actions is exactly one (Note that we refer to the scalar s as v in the corresponding figure). We show an example parameter recovery plot for this Rescorla-Wagner learning model connected to a SSM with collapsing bound in Figure 4.13.

Model definitions for RL with `model_config_rl` Just like the `model_config`, the new HDDM version includes `model_config_rl`, which is the central database for the RL models used in the RLSSM settings. Below is an example for simple Rescorla-Wagner updates (Rescorla, 1972), a basic reinforcement learning rule. The learning rate (referred to as 'rl_alpha' in Figure 4.13 to avoid nomenclature conflicts with the 'alpha' parameter in some SSMs) is the only parameter in the update rule. We do not transform this parameter (`"params_trans"` is set to 0) and specify the parameter bounds for the sampler as $[0, 1]$. Note that for hierarchical sampling, the learning rate parameter α is transformed internally in the package. Therefore, the output trace for the learning rate parameter must be transformed by an inverse-logit function,

$$\frac{1}{(1 + exp(-\alpha))}$$

to get the learning rate values back in range $[0, 1]$. Codeblock 4.22 shows us an example of such a `model_config_rl` dictionary.

```
hddm.model_config_rl.model_config_rl["RWupdate"] =
    {
        "doc": "Rescorla-Wagner update rule.",
        "params": ["rl_alpha"],
        "params_trans": [0],
        "params_std_upper": [10],
        "param_bounds": [[0.0], [1.0]],
        "params_default": [0.5],
    }
```

Codeblock 4.22. `model_config` definition for RL-SSM models.

Analyzing instrumental learning data: The `HDDMnnRL` class Running `HDDMnnRL` presents only a few slight adjustments compared to the other `HDDM` classes. First, the data-frame containing the experimental data should be properly formatted. For every subject in each condition, the trials must be sorted in ascending order to ensure proper RL updates. The column `split_by` identifies each row with a specific task condition (as integer). The `feedback` column gives the reward feedback on the current trial and `q_init` denotes the initial q-values for the model. The rest of the data columns are the same as in other `HDDM` classes. Codeblock 4.23 provides an example.

```
import pandas as pd
data = pd.read_csv(hddm.__path__[0] + '/examples/demo_HDDMnnRL/rlssm_data.csv')
```

```
       response  rt         feedback   subj_idx split_by trial    q_init
0         0.0    2.729579   0.0        0        0        1        0.5
1         1.0    3.090593   1.0        0        0        2        0.5
2         1.0    3.892617   1.0        0        0        3        0.5
3         1.0    2.429583   1.0        0        0        4        0.5
4         1.0    2.566581   1.0        0        0        5        0.5
...       ...    ...        ...        ...      ...      ...      ...
29995     1.0    3.381547   1.0        19       2        496      0.5
29996     1.0    3.324544   0.0        19       2        497      0.5
29997     1.0    3.132535   0.0        19       2        498      0.5
29998     0.0    3.206539   0.0        19       2        499      0.5
29999     1.0    5.009474   0.0        19       2        500      0.5

[30000 rows × 7 columns]
```

Codeblock 4.23. Reading in RL-SSM example data.

We can fit the data loaded in Codeblock 4.23 using the `HDDMnnRL` class. We showcase such a

fit using the *weibull model* in conjunction with the classic Rescorla-Wagner learning rule (Rescorla, 1972). The `HDDMnnRL` class definition (shown in Codeblock 4.24) takes a few additional arguments compared to the `HDDMnn` class: `"rl_rule"` specifies the RL update rule to be used and `non_centered` flag denotes if the RL parameters should be re-parameterized to avoid troublesome sampling from the neck of the funnel of probability densities (Betancourt and Girolami, 2013; Papaspiliopoulos, Roberts, and Sköld, 2007).

```
rlssm_model = hddm.HDDMnnRL(data, model="weibull", rl_rule="RWupdate",
                            non_centered=True, p_outlier=0.0,
                            include=["z", "alpha", "beta", "rl_alpha"],
                           )
rlssm_model.sample(3000, burn=1500)
```

Codeblock 4.24. Constructing and sampling from a HDDMnnRL model.

Figure 4.13 shows a caterpillar plot to verify the LAN-based parameter recovery on a sample RLSSM model.

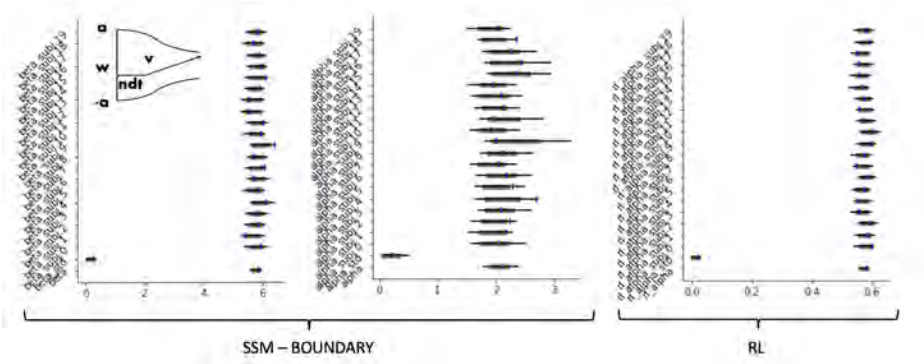

Figure 4.13. Parameter recovery on a sample synthetic dataset using RL+Weibull model. Posterior distributions for subject-level and group-level parameters are shown using caterpillar plots. The thick **black** lines correspond to 5-95 percentiles, thin black lines correspond to 1-99 percentiles. The blue tick-marks show the ground truth values of respective parameters. Note, in the interest of space, we show only a subset of the parameters of the model - the two boundary parameters **alpha** and **beta** and the reinforcement learning rate **rl_alpha**.

Neural Regressors for RLSSM with the `HDDMnnRLRegressor` class The new `HDDMnnRLRegressor` class, is aimed at capturing even richer (learning or choice) dynamics informed by neural activity, just like the `HDDMnnRegressor` class described above for basic SSMs. The extension works the same as the bespoke `HDDMnnRegressor` class, except that the model is now informed by a reinforcement learning process to account for the across-trial dynamics of learning. The method allows estimation of the parameters (coefficients and intercepts) linking the neural activity in a given region and time point to the RLSSM parameters.

The usage of `HDDMnnRLRegressor` class is the same as `HDDMnnRL` class except that our dataframe will now have additional column(s) for neural (or other, e.g. EEG, pupil dilation etc.) trial-by-trial

covariates. Just as with the `HDDMnnRegressor` class, the model definition will also include specifying regression formulas which link covariates to model parameters. For example, if the boundary threshold parameter a is dependent on some neural measure *neural_reg*, Codeblock 4.25 shows us how to specify a corresponding `HDDMnnRLRegressor` model.

```
rlssm_reg_model = hddm.HDDMnnRLRegressor(data, 'a ~ 1 + neural_reg', model="weibull",
                                         rl_rule="RWupdate",  p_outlier=0.0
                                         include=["z", "alpha", "beta", "rl_alpha"],
                                        )
rlssm_reg_model.sample(3000, burn=1500)
```

Codeblock 4.25. Constructing and sampling from a HDDMnnRLRegressors model.

Lastly, it is important to note that we are continually adding new functionalities to the `HDDMnnRL` and `HDDMnnRLRegressor` classes. Given the state of active development for these classes, we suggest that the users refer to the HDDM documentation for any updates to the usage syntax or other changes.

More Resources

The original HDDM (Wiecki, Sofer, and Frank, 2013) paper as well as the original HDDMrl paper (Mads L Pedersen and Frank, 2020) are good resources on the basics of HDDM. The documentation provides examples for many complex use cases, including a long tutorial specifically designed to illustrate the `HDDMnn` classes and another tutorial specifically designed to showcase the `HDDMnnRL` classes. Through the hddm user group, an active community of HDDM users, one can find support on many problems and use cases which may not come up in the official documentation or published work.

4.5 Concluding Thoughts

We hope this tutorial can help kick-start a more widespread application of SSMs in the analysis of experimental choice and reaction time data. We consider the initial implementation with focus on LANs (Fengler et al., 2021) as a starting point, which allows a significant generalization of the model space that can be considered by experimenters. The ultimate goal however is to lead towards community engagement, providing an easy interface for the addition of custom models as a start, which could greatly expand the space of models accessible to research groups across the world. We elaborate on a few possible directions for advancements in the next section.

4.6 Limitations and Future Work

The presented extension to HDDM greatly expands the capabilities of a tried and tested Python toolbox, popular in the cognitive modeling sphere. However, using HDDM as the vehicle of choice,

limitations endemic to the toolbox design remain and warrant a look ahead. First, HDDM is based on PyMC2 (Patil, Huard, and C. J. Fonnesbeck, 2010) a probabilistic modeling framework that has since been superseded by it's successor PyMC3 (Salvatier, Wiecki, and C. Fonnesbeck, 2016) (PyMC 4.0, a rebranded PyMC has just been released too). Since PyMC2 is not an evolving toolbox, HDDM is currently bound to fairly basic MCMC algorithms, specifically a coordinate-wise slice sampler (Neal, 2003). While we have confirmed adequate posterior sampling and estimation using our LANs, estimation may be rendered more efficient if one were to leverage more recent MCMC algorithms such as Hamiltonian Monte Carlo (Hoffman and Gelman, 2014). Moreover, new libraries have emerged that act as independent functionality providers for other probabilistic programming frameworks, e.g. the ArViz (Kumar et al., 2019) python library which provides a wide array of capabilities from posterior visualizations to the computation of model comparison metrics such as the WAIC (Watanabe, 2013). Custom scripts can be used currently to deploy ArViz within HDDM. We are moreover working on a successor to HDDM (we dub it HSSM) which will be built on top of one or more of these modern probabilistic programming libraries. Second, we realize that a major bottleneck in the wider adoption of LANs (and other likelihood approximators), lies in the supply of amortizers. While our extension comes batteries included, we focused on supplying a few SSM variants of proven interest in the literature, as well as some that we used for our or lab-adjacent research. It is not HDDM, but user friendly training pipelines for amortizers, which we believe to spur the quantum leap in activity in this space. Although we are working on the supply of such a pipeline for LANs (Fengler et al., 2021), our hope is that the community will provide many alternatives. Third, we caution against uninformed use of approximate likelihoods. Before basing results of empirical studies on inference performed with LANs or other approximate likelihoods (e.g. user supplied), it is essential to test for the quality of inference that may be expected. Inference can be unreliable in manifold ways (Gelman, Rubin, et al., 1992; Talts et al., 2018; Geweke, 1992). Parameter recovery studies and calibration tests, e.g. simulation based calibration (Talts et al., 2018) should form the backbone of trust in reported analysis on empirical (experimental) datasets. To help the application of a universal standard of rigor, we are working on a set of guidelines, such as a suggested battery of tests to pass before given user supplied likelihoods should be made available to the public. Other interesting work in this sphere is emerging (Lueckmann, Boelts, et al., 2021; Hermans et al., 2021).

Acknowledgments

This work was funded by NIMH grants P50 MH 119467-01 and R01 MH084840-08A1, and additionally supported by the Brainstorm Program at the Robert J. and Nancy D. Carney Institute for Brain Science. We thank Lakshmi N Govindarajan for useful tips concerning code quality. We also thank Thomas Wiecki for reviewing the additions to the HDDM package.

References

Ballard, Ian C. and Samuel M. McClure (2019). "Joint modeling of reaction times and choice improves parameter identifiability in reinforcement learning models". In: *Journal of Neuroscience Methods* 317, pp. 37–44. ISSN: 0165-0270. DOI: 10.1016/j.jneumeth.2019.01.006.

Behnel, Stefan et al. (2010). "Cython: The best of both worlds". In: *Computing in Science & Engineering* 13.2, pp. 31–39.

Betancourt, M. J. and Mark Girolami (2013). *Hamiltonian Monte Carlo for Hierarchical Models*. DOI: 10.48550/ARXIV.1312.0906. URL: https://arxiv.org/abs/1312.0906.

Boehm, Udo et al. (2021). "Fast solutions for the first-passage distribution of diffusion models with space-time-dependent drift functions and time-dependent boundaries". In: *Journal of Mathematical Psychology* 105, p. 102613. DOI: 10.1016/j.jmp.2021.102613.

Brooks, Stephen P and Andrew Gelman (1998). "General methods for monitoring convergence of iterative simulations". In: *Journal of computational and graphical statistics* 7.4, pp. 434–455. DOI: 10.1080/10618600.1998.10474787.

Cisek, Paul, Geneviève Aude Puskas, and Stephany El-Murr (2009). "Decisions in changing conditions: the urgency-gating model". In: *Journal of Neuroscience* 29.37, pp. 11560–11571.

Collins, Anne GE and Amitai Shenhav (2022). "Advances in modeling learning and decision-making in neuroscience". In: *Neuropsychopharmacology* 47.1, pp. 104–118. DOI: 10.1038/s41386-021-01126-y.

Doi, Takahiro et al. (2020). "The caudate nucleus contributes causally to decisions that balance reward and uncertain visual information". In: *ELife* 9, e56694.

Dowd, Erin C et al. (2016). "Probabilistic reinforcement learning in patients with schizophrenia: relationships to anhedonia and avolition". In: *Biological psychiatry: cognitive neuroscience and neuroimaging* 1.5, pp. 460–473. DOI: 10.1016/j.bpsc.2016.05.005.

Eckstein, Maria K, Linda Wilbrecht, and Anne GE Collins (2021). "What do reinforcement learning models measure? Interpreting model parameters in cognition and neuroscience". In: *Current opinion in behavioral sciences* 41, pp. 128–137. DOI: 10.1016/j.cobeha.2021.06.004.

Evans, Nathan J, Jennifer S Trueblood, and William R Holmes (2020). "A parameter recovery assessment of time-variant models of decision-making". In: *Behavior research methods* 52.1, pp. 193–206.

Fengler, Alexander et al. (2021). "Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience". In: *eLife*. DOI: 10.7554/eLife.65074. eprint: https://elifesciences.org/articles/65074.pdf. URL: https://elifesciences.org/articles/65074.

Fontanesi, L. (2022). *rlssm*. Version 0.1.1. DOI: https://zenodo.org/record/4562217. URL: https://github.com/laurafontanesi/rlssm.

Fontanesi, Laura et al. (2019). "A reinforcement learning diffusion decision model for value-based decisions". In: *Psychonomic bulletin & review* 26.4, pp. 1099–1121.

Forstmann, Birte U et al. (2010). "Cortico-striatal connections predict control over speed and accuracy in perceptual decision making". In: *Proceedings of the National Academy of Sciences* 107.36, pp. 15916–15920.

Frank, Michael J et al. (2015). "fMRI and EEG predictors of dynamic decision parameters during human reinforcement learning". In: *Journal of Neuroscience* 35.2, pp. 485–494.

Gelman, Andrew, Donald B Rubin, et al. (1992). "Inference from iterative simulation using multiple sequences". In: *Statistical science* 7.4, pp. 457–472.

Geweke, John (1992). "Evaluating the accuracy of sampling-based approaches to the calculations of posterior moments". In: *Bayesian statistics* 4, pp. 641–649.

Gold, Joshua I and Michael N Shadlen (2007). "The neural basis of decision making". In: *Annual review of neuroscience* 30.

Greenberg, David, Marcel Nonnenmacher, and Jakob Macke (2019). "Automatic posterior transformation for likelihood-free inference". In: *International Conference on Machine Learning*. PMLR, pp. 2404–2414.

Gutmann, Michael U et al. (2018). "Likelihood-free inference via classification". In: *Statistics and Computing* 28.2, pp. 411–425. DOI: 10.1007/s11222-017-9738-6.

Hawkins, Guy E et al. (2015). "Revisiting the evidence for collapsing boundaries and urgency signals in perceptual decision-making". In: *Journal of Neuroscience* 35.6, pp. 2476–2484.

Heathcote, Andrew, Yi-Shin Lin, et al. (2019). "Dynamic models of choice". In: *Behavior research methods* 51.2, pp. 961–985.

Heathcote, Andrew, Dora Matzke, and Andrew Heathcote (2022). "Winner takes all! What are race models, and why and how should psychologists use them?" In: *Current Directions in Psychological Science*.

Hermans, Joeri et al. (2021). "Averting a crisis in simulation-based inference". In: *arXiv preprint arXiv:2110.06581*.

Hoffman, Matthew D and Andrew Gelman (2014). "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623.

Holmes, William R and Jennifer S Trueblood (2018). "Bayesian analysis of the piecewise diffusion decision model". In: *Behavior research methods* 50.2, pp. 730–743. DOI: 10.3758/s13428-017-0901-y.

Krajbich, Ian, Dingchao Lu, et al. (2012). "The attentional drift-diffusion model extends to simple purchasing decisions". In: *Frontiers in psychology* 3, p. 193.

Krajbich, Ian and Antonio Rangel (2011). "Multialternative drift-diffusion model predicts the relationship between visual fixations and choice in value-based decisions". In: *Proceedings of the National Academy of Sciences* 108.33, pp. 13852–13857.

Kumar, Ravin et al. (2019). "ArviZ a unified library for exploratory analysis of Bayesian models in Python". In: *Journal of Open Source Software* 4.33, p. 1143. DOI: 10.21105/joss.01143. URL: https://doi.org/10.21105/joss.01143.

Lawlor, Victoria M et al. (2020). "Dissecting the impact of depression on decision-making". In: *Psychological medicine* 50.10, pp. 1613–1622. DOI: 10.1017/S0033291719001570.

Lueckmann, Jan-Matthis, Giacomo Bassetto, et al. (2019). "Likelihood-free inference with emulator networks". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 32–53.

Lueckmann, Jan-Matthis, Jan Boelts, et al. (2021). "Benchmarking Simulation-Based Inference". In: *arXiv preprint arXiv:2101.04653*.

McDougle, Samuel D and Anne GE Collins (2021). "Modeling the influence of working memory, reinforcement, and action uncertainty on reaction time and choice during instrumental learning". In: *Psychonomic bulletin & review* 28.1, pp. 20–39. DOI: 10.3758/s13423-020-01774-z.

McKinney, Wes (2010). "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman, pp. 51–56. DOI: 10.25080/majora-92bf1922-00a.

Neal, Radford M (1995). "Bayesian learning for neural networks". PhD thesis. University of Toronto.

— (2003). "Slice sampling". In: *Annals of statistics*, pp. 705–741.

Palestro, James J et al. (2019). "Likelihood-Free Methods for Cognitive Science". In: DOI: 10.1007/978-3-319-72425-6.

Papamakarios, George and Iain Murray (2016). "Fast ε-free inference of simulation models with bayesian conditional density estimation". In: *Advances in Neural Information Processing Systems*, pp. 1028–1036.

Papamakarios, George, Eric Nalisnick, et al. (2019). "Normalizing flows for probabilistic modeling and inference". In: *arXiv preprint arXiv:1912.02762*.

Papamakarios, George, David Sterratt, and Iain Murray (2019). "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 837–848.

Papaspiliopoulos, Omiros, Gareth O. Roberts, and Martin Sköld (2007). "A general framework for the parametrization of hierarchical models". In: *Statistical Science* 22 (1). DOI: 10.1214/088342307000000014.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Patil, Anand, David Huard, and Christopher J Fonnesbeck (2010). "PyMC: Bayesian stochastic modelling in Python". In: *Journal of statistical software* 35.4, p. 1. DOI: 10.18637/jss.v035.i04.

Pedersen, Mads L and Michael J Frank (2020). "Simultaneous Hierarchical Bayesian Parameter Estimation for Reinforcement Learning and Drift Diffusion Models: a Tutorial and Links to Neural Data". In: *Computational Brain & Behavior* 3, pp. 458–471.

Pedersen, Mads Lund, Michael J Frank, and Guido Biele (2017). "The drift diffusion model as the choice rule in reinforcement learning". In: *Psychonomic bulletin & review* 24.4, pp. 1234–1251.

Rangel, Antonio, Colin Camerer, and P Read Montague (2008). "A framework for studying the neurobiology of value-based decision making". In: *Nature reviews neuroscience* 9.7, pp. 545–556.

Ratcliff, Roger (1978). "A theory of memory retrieval." In: *Psychological review* 85.2, p. 59.

Ratcliff, Roger and Michael J Frank (2012). "Reinforcement-based decision making in corticostriatal circuits: mutual constraints by neurocomputational and diffusion models". In: *Neural computation* 24.5, pp. 1186–1229.

Ratcliff, Roger, Philip L Smith, et al. (2016). "Diffusion decision model: Current issues and history". In: *Trends in cognitive sciences* 20.4, pp. 260–281.

Ratcliff, Roger, Anjali Thapar, and Gail McKoon (2006). "Aging, practice, and perceptual tasks: a diffusion model analysis." In: *Psychology and aging* 21.2, p. 353. DOI: 10.1037/0882-7974.21.2.353.

Rescorla, Robert A (1972). "A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement". In: *Current research and theory*, pp. 64–99.

Salvatier, John, Thomas V Wiecki, and Christopher Fonnesbeck (2016). "Probabilistic programming in Python using PyMC3". In: *PeerJ Computer Science* 2, e55.

Shinn, Maxwell, Norman H Lam, and John D Murray (2020). "A flexible framework for simulating and fitting generalized drift-diffusion models". In: *Elife* 9, e56938.

Silverman, Bernard W (1986). *Density estimation for statistics and data analysis*. Vol. 26. CRC press.

Smith, Philip L, Roger Ratcliff, and David K Sewell (2014). "Modeling perceptual discrimination in dynamic noise: Time-changed diffusion and release from inhibition". In: *Journal of Mathematical Psychology* 59, pp. 95–113. DOI: 10.1016/j.jmp.2013.05.007.

Spiegelhalter, David J et al. (2014). "The deviance information criterion: 12 years on". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 76.3, pp. 485–493.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Talts, Sean et al. (2018). "Validating Bayesian inference algorithms with simulation-based calibration". In: *arXiv preprint arXiv:1804.06788*.

Tejero-Cantero, Alvaro et al. (2020). "sbi: A toolkit for simulation-based inference". In: *Journal of Open Source Software* 5.52, p. 2505.

Tillman, Gabriel, Trish Van Zandt, and Gordon D Logan (2020). "Sequential sampling models without random between-trial variability: The racing diffusion model of speeded decision making". In: *Psychonomic Bulletin & Review* 27.5, pp. 911–936.

Tran, N-Han et al. (2021). "Systematic parameter reviews in cognitive modeling: towards a robust and cumulative characterization of psychological processes in the diffusion decision model". In: *Frontiers in psychology* 11, p. 608287. DOI: 10.3389/fpsyg.2020.608287.

Trueblood, Jennifer S et al. (2021). "Urgency, leakage, and the relative nature of information processing in decision-making." In: *Psychological Review* 128.1, p. 160. DOI: 10.1037/rev0000255.

Turner, Brandon M (2019). "Toward a common representational framework for adaptation." In: *Psychological review* 126.5, p. 660. DOI: 10.1037/rev0000148.

Turner, Brandon M and Per B Sederberg (2014). "A generalized, likelihood-free method for posterior estimation". In: *Psychonomic bulletin & review* 21.2, pp. 227–250.

Turner, Brandon M and Trisha Van Zandt (2018). "Approximating Bayesian inference through model simulation". In: *Trends in Cognitive Sciences* 22.9, pp. 826–840.

Usher, Marius and James L McClelland (2001). "The time course of perceptual choice: the leaky, competing accumulator model." In: *Psychological review* 108.3, p. 550.

Van Zandt, Trisha, Hans Colonius, and Robert W Proctor (2000). "A comparison of two response time models applied to perceptual matching". In: *Psychonomic bulletin & review* 7.2, pp. 208–256.

Vandekerckhove, Joachim and Francis Tuerlinckx (2008). "Diffusion model analysis with MATLAB: A DMAT primer". In: *Behavior research methods* 40.1, pp. 61–72.

Voss, Andreas et al. (2019). "Sequential sampling models with variable boundaries and non-normal noise: A comparison of six models". In: *Psychonomic bulletin & review* 26.3, pp. 813–832.

Watanabe, Sumio (2013). "A widely applicable Bayesian information criterion". In: *Journal of Machine Learning Research* 14.27, pp. 867–897.

Wiecki, Thomas V, Imri Sofer, and Michael J Frank (2013). "HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python". In: *Frontiers in neuroinformatics* 7, p. 14.

Wieschen, Eva Marie, Andreas Voss, and Stefan Radev (2020). "Jumping to conclusion? a lévy flight model of decision making". In: *The Quantitative Methods for Psychology* 16.2, pp. 120–132.

Wilson, Robert C and Anne GE Collins (2019). "Ten simple rules for the computational modeling of behavioral data". In: *Elife* 8, e49547.

Yartsev, Michael M et al. (2018). "Causal contribution and dynamical encoding in the striatum during evidence accumulation". In: *Elife* 7, e34929.

Chapter 5

Marrying Likelihood Approximations with Modern Inference Algorithms

Previous work concerning likelihood approximations for simulation-based inference has demonstrated the capacity of this framework to successfully expand the horizon of computational models which may routinely be applied for the analysis of complex experimental data in computational cognitive science and neuroscience. Speedy inference is enabled by the use of simple, but effective neural network architectures, for which a forward pass may be executed on ordinary hardware in milliseconds, replacing otherwise costly simulation runs. However, speed of inference remains a concern when datasets and corresponding modeling choices are sufficiently complex. Examples include the decision to apply a hierarchical generative model structure to data concerning a large number of subjects, each of which may have run through a large number of trials and experimental conditions. This work aims at further optimization of inference speed when utilizing likelihood approximations. Two paths are explored. First, the choice of Markov Chain Monte Carlo sampler. We investigate the use of modern Hamiltonian Monte Carlo samplers for posterior exploration. These samplers utilize gradients of LANs with respect model (input) parameters, demanding an extra level of robustness at the level of derivatives, which previous approaches could ignore. Second, we explore the use of stochastic Variational Inference as an alternative route to approximate inference. Here the focus is on the quantification of the loss of precision in posterior inference when relying on specific variational distributions. Proofs of concepts and extensive numerical experiments are provided for both avenues of exploration.

5.1 Introduction

Simulation Based Inference (SBI) methods are transforming the process by which inference is conducted across disciplines. They allow researchers to apply sophisticated inference procedures to models, which are a priori defined only as simulators. Statistics curricula tend to teach methods with the aid of statistical (stochastic) models for which not only simulators, but also likelihood functions (analytical probability distributions over data given model parameters) are accessible: necessary for the mechanics of typically taught inference algorithms. This biases the perception of researchers towards and consequently leads to an overapplication of such models driven by analytical convenience. Standard Bayesian inference, for example, proceeds under the assumption that we can evaluate the right-hand side of the proportional Bayes' rule,

$$p(\theta|\mathbf{x}) \tilde{\propto} p_{\mathcal{M}}(\mathbf{x}|\theta)p(\theta)$$

where θ is a vector of parameters for our given model $\mathcal{M}$, $\mathbf{x}$ is our observed data, $p(\theta)$ is called a *prior distribution* for parameters θ and $p_{\mathcal{M}}(\mathbf{x}|\theta)$ is the likelihood of observing $\mathbf{x}$ under model $\mathcal{M}$ with parameters θ. Repeated evaluation of this right-hand side drives any common Bayesian inference algorithm, from Importance Sampling (IS) to Markov Chain Monte Carlo (MCMC) to Variational Inference (VI) (Brooks et al., 2011). While, as stated above, statistical curricula focus predominantly on generative models for which this can be computed easily, it is quite trivial to find examples for which this computation cannot be carried out without a severe computational cost (Lotka, 1925; M. A. Beaumont, 2010).

The history of Simulation Based Inference (SBI, previously called ABC for Approximate Bayesian Computation) as a field goes back at least 25 years (Tavaré et al., 1997; Pritchard et al., 1999; Sisson, Fan, and M. Beaumont, 2018), focusing exactly on scenarios where posterior inference is desired without the ability to directly compute the likelihood term $p_{\mathcal{M}}(\mathbf{x}|\theta)$. A journey from initial breakthrough algorithms (Tavaré et al., 1997; Pritchard et al., 1999) with obvious limitations has brought us to a second generation that slowly embraced machine learning approaches to the problem (Wood, 2010; Wilkinson, 2014; Meeds and Welling, 2014; Järvenpää et al., 2018; Järvenpää et al., 2021; Acerbi, 2020) (as mentioned in chapter 2 the timeline is not strictly sequential) to the current and third generation of SBI methods driven by deep learning (Papamakarios and Murray, 2016; Papamakarios, Pavlakou, and Murray, 2017; Papamakarios, Sterratt, and Murray, 2019a; Greenberg, Nonnenmacher, and Macke, 2019; Lueckmann et al., 2019; S. T. Radev et al., 2020; Fengler, Govindarajan, et al., 2021b; Boelts et al., 2022) (see chapter 2 for a thorough review). Without recounting the full landscape of available SBI approaches, it is worth mentioning that the broad taxonomy of contemporary deep learning inspired methods evolves along two fault lines: First, the choice of target of the SBI algorithm. Recent methods tend to target one of two objects as learnable (via deep neural networks of varying architectures). Either they target the posterior distribution directly, bypassing any explicit likelihood computations at all, allowing near instant posterior inference after learning, however at the cost of inflexibility of the learned posterior with

respect to changes in the generative model (S. T. Radev et al., 2020; Greenberg, Nonnenmacher, and Macke, 2019; Papamakarios and Murray, 2016). Or, second, they target some form of likelihood approximation (Papamakarios, Sterratt, and Murray, 2019a; Fengler, Govindarajan, et al., 2021b; Boelts et al., 2022), which once learned serves to facilitate the evaluation of an approximate right-hand side to Bayes rule (as above) in the context of standard Bayesian inference algorithms applied downstream. Methods targeting approximate likelihoods offer much greater flexibility as to the ability for downstream applications (see a detailed discussion in Fengler, Govindarajan, et al., 2021b and chapter 2), which we argued previously (Fengler, Govindarajan, et al., 2021b) to be desirable.

We proposed *likelihood approximation networks* (LANs) (Fengler, Govindarajan, et al., 2021b) a method for trial-wise likelihood approximation based on deep learning, with a strong focus on downstream speed of inference. As a natural test-bed for LANs, we focused on the class of sequential sampling models (SSMs) (Forstmann, Ratcliff, and Wagenmakers, 2016). We considered SSMs as ripe to benefit from an infusion of reliable targeted SBI methods: The basic SSM, the famous Ratcliff Drift Diffusion Model (DDM) for which likelihoods can be computed with relative ease (Navarro and Fuss, 2009; Foster and Singmann, 2021), is widely applied across a range experimental paradigms (Rangel, Camerer, and Montague, 2008; Forstmann, Anwander, et al., 2010; Frank et al., 2015; Yartsev et al., 2018; Doi et al., 2020; Cavanagh, Wiecki, et al., 2011; Herz et al., 2016; Mads Lund Pedersen, Frank, and Biele, 2017; Mads L Pedersen and Frank, 2020), while variations have generated broad theoretical interest (Cisek, Puskas, and El-Murr, 2009a; Cisek, Puskas, and El-Murr, 2009b; Usher and McClelland, 2001; Wieschen, Voss, and S. Radev, 2020; Krajbich et al., 2012; Holmes, Trueblood, and Heathcote, 2016) with hardly any applications to real data, due to a systematic lack of easy to compute likelihood functions. A textbook case of where SBI can be useful. Historical data analysis applications are broadly restricted to overly simple models due to analytical convenience. While evidence for the benefit of variations has accumulated (Cisek, Puskas, and El-Murr, 2009a; Cisek, Puskas, and El-Murr, 2009b; Usher and McClelland, 2001; Wieschen, Voss, and S. Radev, 2020; Krajbich et al., 2012; Holmes, Trueblood, and Heathcote, 2016), those which we can easily simulate data from, lack analytical likelihoods (see more concrete examples in 2).

Our previous work (Fengler, Govindarajan, et al., 2021b; Fengler, Bera, et al., 2022), chapter 3 and chapter 4 in this thesis, has shown LANs to perform well on a selection of SSMs for which likelihood computations are either costly or intractable. Comparing the speed of inference, we found LANs to be on average as fast as the fastest analytical approaches for the DDM (Navarro and Fuss, 2009). Depending on specific properties of the experimental datasets, LANs may outperform analytical methods or end up slightly slower. We moreover made LANs available to the community via an extension to the HDDM python toolbox, which now goes beyond performing hierarchical Bayesian inference for DDMs and instead incorporates a large selection of alternative SSMs out of the box.

While this previous research translates into a great expansion of the type of data analysis options available at the fingertips of experimentalists, experience shows that room for improvement remains on the front of inference speed. Posterior inference for large datasets with complicated generative

models (e.g. hierarchical inference with multiple experimental conditions for each of > 50 subjects) may still take up more than a day's worth of computation, limiting the amount of computational experimentation and therefore the speed of the research enterprise and potential for applications in general.

In this work I explore two avenues to further improve the speed of inference, while maintaining LANs as likelihood approximators. The goal is two-fold. First, I attempt to realize improvements in runtime while maintaining (or improving) sample quality. Second, the work reveals whether a toolbox around LANs based on modern probabilistic programming packages (Carpenter, Gelman, M. D. Hoffman, Lee, Goodrich, Betancourt, M. A. Brubaker, et al., 2017; Phan, Pradhan, and Jankowiak, 2019; Bingham et al., 2019; Salvatier, Wiecki, and C. Fonnesbeck, 2016a) is fundamentally feasible.

The first avenue I explore is whether a modernization of the MCMC sampler is feasible with LAN-based likelihoods. HDDM (Wiecki, Sofer, and Frank, 2013; Fengler, Bera, et al., 2022) currently uses coordinate-wise slice sampling (Neal, 2003), a robust but old-fashioned posterior sampling method. This stands in contrast with the types of samplers which are the workhorses for modern Bayesian inference. All major probabilistic programming software with emphasis on MCMC today (Carpenter, Gelman, M. D. Hoffman, Lee, Goodrich, Betancourt, M. Brubaker, et al., 2017; Phan, Pradhan, and Jankowiak, 2019; Bingham et al., 2019; Salvatier, Wiecki, and C. Fonnesbeck, 2016a) rely on some form of gradient-informed sampling (Besag and Green, 1993; Neal et al., 2011; Betancourt, 2017) which avoids several shortcomings of more traditional MCMC methods, in particular improving on mixing behavior for high dimensional and potentially highly correlated target distributions (Neal et al., 2011; Vrugt et al., 2009; Ter Braak, 2006a; Gentle, Härdle, and Mori, 2012). Specifically the landscape is dominated by a Hamiltonian Monte Carlo (HMC) sampler (Betancourt and Girolami, 2015; Neal et al., 2011) (explained in more detail in the next section), dubbed NUTS (M. D. Hoffman and Gelman, 2014) for No-U-Turn Sampler. That LANs are suitable for HMC is not assured a priori. While LANs are differentiable with respect to model parameters by design (since they are inputs to the network, which is a differentiable function), they are not trained (Fengler, Govindarajan, et al., 2021b) in a way that assures regular behavior of gradients, which in turn is essential for taking successful steps through the implicit dynamical systems solved in the process of generating candidate draws in HMC samplers. I will test the feasibility and evaluate the potential for runtime speedups over traditional methods in the rest of this chapter.

The second avenue I explore is stochastic Variational Inference (VI) (Ranganath, Gerrish, and Blei, 2013; Wingate and Weber, 2013; Wainwright, Jordan, et al., 2008; Blei, Kucukelbir, and McAuliffe, 2017)based on LANs. Variational Inference is an optimization-based approach to Bayesian inference which has been shown to massively outperform MCMC methods in appropriate settings (Blei, Kucukelbir, and McAuliffe, 2017), despite its own shortcomings. I explain the approach in more detail in section 5.2, but it is helpful to briefly discuss the main idea here. Instead of attempting to sample from the posterior, VI uses optimization to fit a parametric distribution, the so-called variational distribution, to the posterior.

While we can perform stochastic VI without reliance on the gradients of LANs (gradient computations are necessary only for the variational distribution), it is important to assess how well a given family of variational distributions approximates the true posterior. For very complex posterior distributions, the user should specify a complex (i.e., containing many parameters) variational family, which in turn complicates the optimization procedure and thus potentially reduces runtime gains. This may impact the feasibility of the optimization problem altogether. In this work, I test two types of variational families with an eye toward runtime, but also toward quality of posterior inference: the family of Isotropic Multivariate Normal Distributions (IMVNs) as well as the family of multivariate Normal distributions (MVNs) with arbitrary covariance structures.

Across both MCMC and VI paradigms, I use DDM as a main test-bed to establish a proofs of concept concerning feasibility of the respective methods in our context. Based on this, the intend is to asses the potential for further modernization of probabilistic programming infrastructure around LAN-reliant Bayesian inference. To foreshadow the results: the numerical experiments support the conclusion (section 5.4) that LANs offer a robust basis for both HMC and VI for the DDM, providing high quality inference with massive runtime savings as compared to the coordinate-wise slice sampler implemented through HDDM (Wiecki, Sofer, and Frank, 2013; Fengler, Bera, et al., 2022). Both methods thus warrant further numerical experiments exploring the broader space of SSMs.

This chapter is organized as follows. In section 5.2 I introduce (albeit briefly) all relevant conceptual background, including the basic construction of SSMs, utilized MCMC and VI algorithms as well as basic aspects of the numerical experiments conducted, such as the design of the synthetic datasets, specific settings for inference algorithms and the computing environments on which the numerical experiments were run. Section 5.3 then describes the results of the numerical experiments. The focus is on an analysis of relative runtimes along with calibration and parameter recovery performance of the tested inference algorithms. Section 5.4 summarizes and contextualizes the results. I finish with limitations and an overview of potential future research in section 5.5.

5.2 Methods

5.2.1 Sequential Sampling Models

The test-bed for numerical experiments, as in the basic papers on LANs (Fengler, Govindarajan, et al., 2021b) and the corresponding HDDM expansion (Wiecki, Sofer, and Frank, 2013; Fengler, Bera, et al., 2022), will be a selection of sequential sampling models (SSMs). In general SSMs tend to be describable as specific instances of the following general stochastic differential equation (SDE),

$$d\mathbf{X}_t = a(t,x)\,dt + b(t,x)\,d\mathbf{B}_t, \quad \mathbf{X}_0 = w$$

where we are concerned with the probabilistic behavior of the particle (or vector of particles) $\mathbf{X}$. The behavior of this particle is driven by the drift function $a(t,x)$, the noise transformation function $b(t,x)$, the incremental noise process B_t, and $X_0 = w$, the starting point. The above equation

defines a process for stochastic evolution of the position of particle $\mathbf{X}$ over time. It is commonly assumed that this particle represents an instantaneous state of relative evidence in favour of one of two choice options. For purposes of the analysis of reaction time and choice data, of interest is not the position of the particle itself, but instead a derived quantity: the probability (density), defined on the time-axis (t), that the evidence (particle) crosses one of two given thresholds first (criterion, potentially changing as a function of time), $c_i(t)$ (indicating a choice of option c_i) at a given time t (or rt for reaction time), which we finally write as $p(\{rt, c\}|\theta)$. This defines a probability density function over *reaction time* and *choice* data, which depends on a given set of SSM parameters θ, and is generally knows as a *First Passage Distribution* (FPD). But $p(\{rt, c\}|\theta)$ is also called a *likelihood function* and may instead canonically denoted as $\ell(\theta|\{rt, c\})$ (signifying it as a function of the parameters given data).

We will focus on two specific such SSMs in this chapter. First we begin with the vanilla **DDM** as a benchmark, given its popularity in the community. Although a closed-form analytical solution for the likelihood is available, I use the LAN here to evaluate how approximation errors induced by the LAN approach might impact inference; previous work showed that inference using LANs was nearly as good as the analytic solution (Fengler, Govindarajan, et al., 2021a), but the present idea is to investigate this in the presence of other modern inference methods. I also test these methods using the **ANGLE** model, which has no easy-to-compute closed form likelihood function to begin with. I now describe these models in some detail, following the general notation laid out above.

DDM

The DDM has four basic parameters: (v, a, z, t): A *drift rate* v, a constant *criterion* (otherwise known as boundary separation or decision threshold) a, a *non-decision time* t (added to the time of the actual decision process modeled via the stochastic differential equation) and an initial *starting point bias* z.

In our established notation this amounts to the following. We set the *drift function* $a(t, x) = v$, that is for a given trial the rate of accumulation is constant. Moreover we set the *noise function* to $b(t, x) = 1$ (the noise scale is fixed across time) and $\mathbf{X}_0 = z$ (the diffusion starts at point z). Finally the criterion for choice 1 is set to $c_1(t) = a$ and the criterion for choice 0 (or also denoted -1) is set to $c_0(t) = -a$. The non-decision time t will simply be added to the time of the process. It simply shifts the reaction time distribution to the right.

Hence we end up with the following SDE,

$$d\mathbf{X}_t = v\, dt + d\mathbf{B}_t, \quad \mathbf{X}_0 = z$$

and its corresponding FPD

$$p(\{rt, c\}|(v, a, z, t))$$

which specify the **DDM** model.

ANGLE

The ANGLE model add as parameter θ: the *criterion angle* θ.

In our established notation, the only difference between the ANGLE model and the DDM is that the criterion is now described by the function,

$$c_1(t) = a - (t * \frac{sin(\theta)}{cos(\theta)})$$

for choice 1, and $c_0(t) = -c_1(t)$, for choice 0. Hence we end up with the following SDE,

$$d\mathbf{X}_t = v\,dt + d\mathbf{B}_t, \quad \mathbf{X}_0 = z$$

and its corresponding FPD

$$p(\{rt, c\}|(v, a, z, t, \theta))$$

that specify the **DDM** model.

Simulation

Model simulations across all models were performed using a custom Python package developed for the purpose. The code is available under https://github.com/AlexanderFengler/ssm_simulators. The package uses the standard Euler-Maruyama method to simulate trajectories iteratively according to the following discretization of the underlying SDE:

$$X_{t+\Delta t} = X_t + a(t, x)\Delta t + b(t, x)\Delta\mathbf{B}$$

Δt was set to 0.001, meaning steps will be on the order of milliseconds.

5.2.2 LANs

A detailed discussion of LANs can be found in Fengler, Govindarajan, et al., 2021b (or chapter 3 in this thesis), but I provide a quick overview below. The fundamental idea behind LANs is to use function approximators. These are Multi-Layered Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) in the dedicated paper, but many choices of architecture and function approximators would be valid. The numerical experiments in this chapter relied on MLPs as they are supplied with HDDM (Wiecki, Sofer, and Frank, 2013; Fengler, Bera, et al., 2022). The parameters $\mathbf{w}$ are used to learn a mapping $f_{\mathbf{w}}(\theta, \mathbf{x})$ from model parameters θ and data $\mathbf{x}$ to the corresponding log-likelihood under model $\mathcal{M} : \ell_{\mathcal{M}}(\theta|\mathbf{x})$. Since we are interested in the application to likelihood-free models (models for which we have access only to a simulator), we do not assume direct access to an analytical expression for $\ell_{\mathcal{M}}(\mathbf{x}|\theta)$ and instead rely on model simulations to construct continuous empirical likelihood functions via kernel density estimation. We then use evaluations of such empirical likelihood functions $\hat{\ell}_{\mathcal{M}}(\theta|\mathbf{x})$ as training targets, finally training the models on tuples $(\{\mathbf{x}_i, \theta_i\}, \hat{\ell}_{\mathcal{M}}(\theta_i|\mathbf{x}_i))$, ultimately receiving

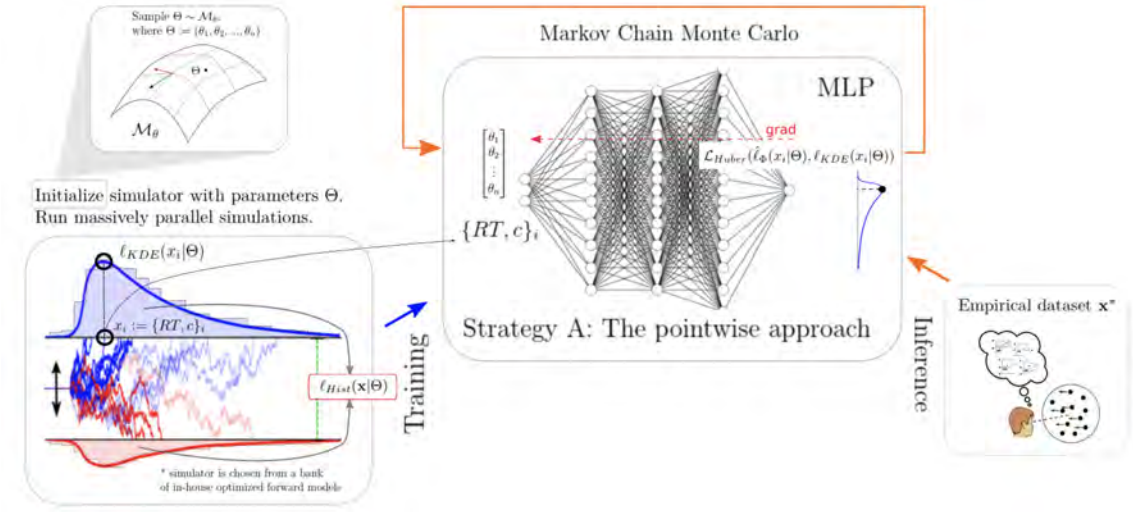

Figure 5.1. High level overview of the LAN framework. For a given model $\mathcal{M}$, we sample model parameters θ from a region of interest (left 1), then run simulations (upper left). We use those simulations to construct a KDE-based empirical likelihood. The combination of parameters and the respective likelihood is then used to train the likelihood approximation network (middle). Once trained we can use the MLP for posterior inference given an empirical / experimental dataset (right).

learned approximate log-likelihood function $f_{\mathbf{w}^*}(\theta, \mathbf{x})$ (where w^* signifies the *learned* parameters of the network), which we consequently dubbed a "LAN" or *likelihood approximation network*.

We can then use such a LAN for Bayesian statistical inference, simply substituting it for the otherwise-unknown likelihood function on the right-hand side of Bayes Rule to get

$$\hat{p}(\theta|\mathbf{x}) \propto f_{\mathbf{w}^*}(\theta, \mathbf{x})p(\theta)$$

where $\hat{p}(\theta|\mathbf{x})$ signifies the resulting (proportional) approximate posterior, and $p(\theta)$ stands for a given prior on the model parameters θ. I emphasize that, since the trial-wise likelihood functions are the learned objects, LANs can be reused flexibly, for instance when including trial-by-trial regressions with neural or other covariates, modeling arbitrarily complex experimental designs or evaluating a large number of subjects and trials (Frank et al., 2015; Cavanagh and Frank, 2014).

As shown with the examples of SSMs in Fengler, Govindarajan, et al., 2021b, for purposes of log-likelihood evaluation, the resulting LAN will outperform the process of constructing empirical likelihood functions by orders of magnitude: Savings which result in fast inference downstream once a LAN is trained. Moreover, important in the context of this chapter, LANs are differentiable with respect to their inputs, affording the application of MCMC methods that use posterior gradients to improve mixing behavior such as the Hamiltonian MCMC methods illustrated in the respective section below. Figure 5.1 illustrates the LAN framework. For a more detailed discussion of LANs please refer to chapter 3.

Since the numerical experiments in this chapter deal with SSMs, we often specialize the data $\mathbf{x}$ to tuples (c, rt) where c stands for *choice option* and rt refers to *reaction times*.

5.2.3 Variational Inference

Variational Inference (VI) is a framework for approximating Bayesian Inference which is based on optimization. This puts it in stark contrast to the otherwise-common inference approaches based on either analytical computations in conjugate problems or posterior sampling-based Markov Chain Monte Carlo (Gelman, Carlin, et al., 1995). Predominantly used in the context of Bayesian Statistical Modeling (Wainwright, Jordan, et al., 2008), it also has broad applications in the world of deep learning (Kingma, Salimans, and Welling, 2015; Kingma, Welling, et al., 2019; Doersch, 2016; Rezende and Mohamed, 2015; Rezende, Papamakarios, et al., 2020), where it can significantly speed up inference time at the cost of the exactness of the resulting posteriors.

The optimization objective of VI is in general described as

$$\arg\min_{q(\theta)\in\mathcal{Q}} \mathbf{KL}(q(\theta)||p(\theta|\mathbf{x}))$$

where $q(\theta)$ represents a candidate approximate posterior, which we select from a permissible function class $\mathcal{Q}$, and $p(\theta|\mathbf{x})$ is the actual posterior distribution. We call solutions to the arg min problem $q^*(\theta)$ a *variational posterior*. Increasing the complexity of the function class $\mathcal{Q}$ allows better approximations of $p(\theta|\mathbf{x})$ but simultaneously increases the cost of a given optimization routine in solving for the arg min.

A few steps of algebra reveal an important structural aspect of the optimization problem (Wingate and Weber, 2013):

$$\begin{aligned}
\mathbf{KL}(q(\theta)||p(\theta|\mathbf{x})) &= \mathop{\mathbb{E}}_{q}[\log(q(\theta))] - \mathop{\mathbb{E}}_{q}[\log(p(\theta|\mathbf{x})] && (5.1)\\
&= \mathop{\mathbb{E}}_{q}[\log(q(\theta))] - \mathop{\mathbb{E}}_{q}\left[\log\left(\frac{p(\mathbf{x}|\theta)p(\theta)}{p(\mathbf{x})}\right)\right] && (5.2)\\
&= \underbrace{\mathop{\mathbb{E}}_{q}[\log(q(\theta))] - \mathop{\mathbb{E}}_{q}[\log(p(\mathbf{x}|\theta)p(\theta))]}_{-\mathbf{ELBO}} + \log(p(\mathbf{x})) && (5.3)
\end{aligned}$$

where **ELBO** stands for *Evidence Lower BOund*. This is a lower bound since from

$$\mathbf{KL}(q(\theta)||p(\theta|\mathbf{x})) \geq 0$$

we conclude that

$$\mathbf{ELBO} \leq \log(p(\mathbf{x}))$$

where $\log(p(\mathbf{x}))$ is the *log-evidence* of our model, as per standard terminology used in Bayesian statistical methodology. Importantly, our optimization problem is independent of $\log(p(\mathbf{x}))$, and therefore VI boils down to *maximizing the* **ELBO**.

Early methodological papers concerning VI focused on analytical methods to derive closed form equations which can eventually be used in a *coordinate ascent* optimization routine to solve for the

variational posterior in a few special settings (Blei, Kucukelbir, and McAuliffe, 2017). These settings tended to be restrictive with respect to both the underlying *prior* and *likelihood model* as well as the function class $\mathcal{Q}$. Moreover, even if applicable, the need for model-specific analytical derivations of update equations made any application tedious (Ranganath, Gerrish, and Blei, 2013; Wingate and Weber, 2013; Blei, Kucukelbir, and McAuliffe, 2017). Worse, the usual mean-field assumption on posteriors (Murphy, 2012), which enables analytic computations in the first place, makes posterior calibration unlikely due to mode-seeking behavior of the VI approach in general (Murphy, 2012).

What allows consideration of VI for purposes of approximate inference in this context (SSMs), especially on top of a basis of arbitrarily defined approximate likelihoods (such as LANs), is the following crucial insight (Wingate and Weber, 2013): Instead of using the **ELBO** to start a sequence of analytical derivations leading to closed-form update equations, we ask only if we can take the **ELBO**'s derivative with respect to the parameters $\mathbf{w}$ of the variational distribution $q_{\mathbf{w}}(\theta)$. We can then perform stochastic gradient descent (Robbins and Monro, 1951). The following sequence of algebra (adapted from Wingate and Weber, 2013) demonstrates the feasibility of this approach. Considering the objective function $\mathbb{L}(\mathbf{w})$,

$$\begin{aligned}
\nabla_{\mathbf{w}}\mathbb{L}(\mathbf{w}) &= \left[\nabla_{\mathbf{w}} \mathop{\mathbb{E}}_{q_{\mathbf{w}}} \left[\log(q_{\mathbf{w}}(\theta))\right] - \mathop{\mathbb{E}}_{q_{\mathbf{w}}} \left[\log(p(\mathbf{x}|\theta)p(\theta))\right]\right] && (5.4)\\
&= \nabla_{\mathbf{w}} \mathop{\mathbb{E}}_{q_{\mathbf{w}}} \left[\log\left(\frac{q_{\mathbf{w}}(\theta)}{p(\mathbf{x}|\theta)p(\theta)}\right)\right] && (5.5)\\
&= \int \nabla_{\mathbf{w}} \left(q_{\mathbf{w}}(\theta) \log\left(\frac{q_{\mathbf{w}}(\theta))}{p(\mathbf{x}|\theta)p(\theta)}\right)\right) d\theta && (5.6)\\
&= \int \nabla_{\mathbf{w}} q_{\mathbf{w}}(\theta) \left(\log\left(\frac{q_{\mathbf{w}}(\theta)}{p(\mathbf{x}|\theta)p(\theta)}\right)\right) d\theta + \int q_{\mathbf{w}} \nabla_{\mathbf{w}} \log(q_{\mathbf{w}}(\theta))\, d\theta && (5.7)\\
&= \int \nabla_{\mathbf{w}} q_{\mathbf{w}}(\theta) \left(\log\left(\frac{q_{\mathbf{w}}(\theta)}{p(\mathbf{x}|\theta)p(\theta)}\right)\right) d\theta + 0 && (5.8)\\
&= \int q_{\mathbf{w}}(\theta) \left[\nabla_{\mathbf{w}} \log(q_{\mathbf{w}}(\theta))\right] \left(\log\left(\frac{q_{\mathbf{w}}(\theta)}{p(\mathbf{x}|\theta)p(\theta)}\right)\right) d\theta && (5.9)\\
&\approx \frac{1}{N} \sum_{i=1}^{N} \left[\nabla_{\mathbf{w}} \log(q_{\mathbf{w}}(\theta_i))\right] \left(\log\left(\frac{q_{\mathbf{w}}(\theta_i)}{p(\mathbf{x}|\theta_i)p(\theta_i)}\right)\right) && (5.10)
\end{aligned}$$

since $\nabla_{\mathbf{w}} q_{\mathbf{w}}(\theta) = q_{\mathbf{w}}(\theta) \nabla_{\mathbf{w}} \log(q_{\mathbf{w}})$ and therefore,

$$\int q_{\mathbf{w}} \nabla_{\mathbf{w}} \log(q_{\mathbf{w}}(\theta))\, d\theta = \int \nabla_{\mathbf{w}} q_{\mathbf{w}}(\theta)\, d\theta = \nabla_{\mathbf{w}} \int q_{\mathbf{w}}(\theta)\, d\theta = 0$$

We can now use the final right-hand side as an unbiased estimator of the **ELBO**'s gradient with respect to *variational parameters* $\mathbf{w}$ (Wingate and Weber, 2013), which can then be used to drive the optimization of the **ELBO**.

Note that this scheme imposes few restrictions on the form of $p(\mathbf{x}|\theta)$ (we just need the *log-likelihood*) or the function class $\mathcal{Q}$ (we need to be able to sample from a given $q_{\mathbf{w}}(.)$, and $\log(q_{\mathbf{w}})$ needs to be differentiable). The optimization problems itself may still be difficult for complex $\mathcal{Q}$, but the algorithm remains widely applicable.

Choosing the number N of *particles* (samples from the current variational distribution $q_{\mathbf{w}_i}(\theta)$) as a hyperparameter, which we later fix to $N \in \{1, 10\}$, the numerical experiments are based on the widely applied ADAM algorithm (Kingma and Ba, 2014) for stochastic gradient descent (Robbins and Monro, 1951) on the **ELBO** objective. The associated learning rate parameter Δ, was set to 0.02 for experiments with the DDM, and varied between 0.02 and 0.0001 for the experiments involving the ANGLE model. I discuss in section 5.2.5, which software packages were used for the implementation.

It is noteworthy that a variety of other optimization algorithms for VI have been proposed in recent years (Neal et al., 2011). These vary the optimization criteria (Li and Turner, 2016) and/or rely on different approaches to optimization (Liu and Wang, 2016). The numerical experiments serve only as a starting point for studying the many possible stochastic VI frameworks for inference with SSMs. Similarly, I explore only a small subset of the possible variational families (Rezende and Mohamed, 2015).

5.2.4 Hamiltonian Monte Carlo

The second general paradigm for posterior inference is known as Markov Chain Monte Carlo (MCMC). This approach, which has a longstanding history predating the inception of the VI framework by decades (Metropolis et al., 1953; Hastings, 1970; Robert, Casella, and Casella, 1999; Diaconis, 2009), relies on the generation of a sequence of auto-correlated samples (MCMC chain) from the posterior distribution. When drawing a large number N of such samples, this sequence of auto-correlated draws is then deemed *converged* and treated as a valid sample from the posterior distribution $p(\theta|\mathbf{x})$ (Brooks et al., 2011). Various checks have been developed to safeguard against failures of convergence (Geweke, 1992; Gelman and Rubin, 1992; Vehtari et al., 2021), so that practitioners are prevented from deriving strong conclusion based on untrustworthy inference procedures. While none of these methods are completely fool-proof, they are nevertheless successfully and widely applied.

In our previously published work Fengler, Govindarajan, et al., 2021a, appearing as chapter 3 in this thesis, we rely on MCMC for posterior sampling. Reported numerical experiments utilized the HDDM python toolbox Wiecki, Sofer, and Frank, 2013, which in turn relies on the pymc2 probabilistic programming framework (Patil, Huard, and C. J. Fonnesbeck, 2010). MCMC in HDDM is implemented via a coordinate-wise slice sampler Neal, 2003. While the slice-sampler via HDDM was sufficient for a proof of concept, and allowed the conclusion that LANs are effective for posterior sampling, a lot of room remains for testing more advanced samplers. For high-dimensional posteriors, which are common in settings where hierarchical modeling is applied, a class of MCMC samplers which utilize gradients (Rossky, Doll, and Friedman, 1978; Besag, 1994; M. D. Hoffman and Gelman, 2014; Neal et al., 2011; Betancourt and Girolami, 2015; Betancourt, 2017) of the posterior have become the dominant workhorse behind a range of probabilistic programming frameworks (Carpenter, Gelman, M. D. Hoffman, Lee, Goodrich, Betancourt, M. Brubaker, et al., 2017; Salvatier, Wiecki, and C. Fonnesbeck, 2016b; Phan, Pradhan, and Jankowiak, 2019). Specifically the No-U-Turn Sampler (or NUTS; (M. D. Hoffman and Gelman, 2014) can by now be considered the accepted standard backbone of applied Bayesian statistics.

While the theory of Hamiltonian Monte Carlo methods is involved (Betancourt, 2017), a summary of the basic idea follows. I tailor this summary to our application of interest and strip any unnecessary vocabulary (more comprehensive explanations of HMC can be found in Betancourt, 2017; Neal et al., 2011; Gelman, Carlin, et al., 1995)

Instead of our posterior $p(\theta|\mathbf{x})$, we instead sample from another distribution, which we define as

$$Q(\theta, \nu; \mathbf{x}) = \frac{1}{Z} \exp\left(-\mathbb{H}(\theta, \nu)\right)$$

where $\mathbb{H}(\theta, \nu) = -\log(p(\theta|\mathbf{x}) + \nu^T M^{-1}\nu/2$, and $dim(\theta) = dim(\nu)$. This makes HMC an *auxiliary variable method* (Besag and Green, 1993; Higdon, 1998; Gentle, Härdle, and Mori, 2012). Auxiliary variable methods generally use *extra dimensions* of random variables to achieve better mixing of the subset of random variables of interest in our target distribution (Green, 2003; Green and Hastie, 2009; Gentle, Härdle, and Mori, 2012). Note that the choice of $\mathbb{H}$ makes θ and ν independent in Q. Hence sampling from Q and dropping the draws of ν, will provide us with a valid sample from $p(\theta|\mathbf{x})$.

The key idea of HMC is to use the gradients of $\mathbb{H}$ with respect to θ and ν to facilitate better mixing of our MCMC algorithm than is possible with gradient-free schemes. Example of the latter include relying on local perturbations in random directions, as in the Metropolis family of algorithms (Metropolis et al., 1953), or coordinate-wise algorithms such as the slice-sampler (Neal, 2003) implemented in HDDM (Wiecki, Sofer, and Frank, 2013). These latter methods cannot effectively deal with strong correlations in the target distribution (e.g. parameter trade-offs in a posterior over model parameters given data, $p(\theta|\mathbf{x})$).

The algorithm follows a two-step process to produce MCMC samples. We *first* draw a sample from

$$\nu \sim \mathcal{N}(\mathbf{0}, M)$$

and then propagate through (θ, ν) space, via the following dynamical system (the Hamiltonian system):

$$\frac{d\theta}{dt} = \frac{\partial \mathbb{H}}{\partial \nu} \tag{5.11}$$

$$\frac{d\nu}{dt} = -\frac{\partial \mathbb{H}}{\partial \theta} \tag{5.12}$$

$$\tag{5.13}$$

The propagation is usually done numerically, via the *leap-frog* algorithm (Neal et al., 2011). We then accept or reject the proposed end-state (θ^*, ν^*) according to the following probability (this is also called a Metropolis-step):

$$p(accept) = \min[1, \exp(-\mathbb{H}(\theta^*, \nu^*) + \mathbb{H}(\theta, \nu))] \tag{5.14}$$

Note that, this is just a standard Metropolis-step, where $\exp(-\mathbb{H}(\theta^*, \nu^*) + \mathbb{H}(\theta, \nu))$ is essentially the *likelihood ratio*. This leaves us with a few important *tuning parameters* for the algorithm. The step size ϵ of the dynamical systems solver, the optimization of the covariance matrix M of the Gaussian distribution over ν, and the number of steps of integration of the dynamical system. NUTS (M. D. Hoffman and Gelman, 2014), introduces various schemes to automate the choice of these tuning parameters, to which HMC schemes are rather sensitive (Neal et al., 2011).

The development of more and more advanced HMC samplers is an active area of research, and a range of proposals are gaining traction (Betancourt, 2013; Betancourt, 2017; M. Hoffman, Sountsov, et al., 2019; Girolami and Calderhead, 2011). However, as mentioned above, the No-U-Turn Sampler is very widely applied (M. D. Hoffman and Gelman, 2014; Salvatier, Wiecki, and C. Fonnesbeck, 2016b; Carpenter, Gelman, M. D. Hoffman, Lee, Goodrich, Betancourt, M. A. Brubaker, et al., 2017; Nishio and Arakawa, 2019; Gelman, Carlin, et al., 1995). It is in fact the go-to sampler that is offered as a default in various probabilistic programming interfaces, such as STAN, PyMC, Pyro and NumPyro (Carpenter, Gelman, M. D. Hoffman, Lee, Goodrich, Betancourt, M. Brubaker, et al., 2017; Salvatier, Wiecki, and C. Fonnesbeck, 2016b; Bingham et al., 2019; Phan, Pradhan, and Jankowiak, 2019).

5.2.5 Software

For numerical experiments, I primarily use three python software libraries. First, as a benchmark from previous work, I use the HDDM python toolbox (Wiecki, Sofer, and Frank, 2013) for Bayesian inference with hierarchical DDMs, specifically its LAN extension (Fengler, Bera, et al., 2022). As mentioned above, this toolbox uses coordinate-wise slice sampling. HDDM served as the substrate to facilitate the numerical experiments reported in (Fengler, Govindarajan, et al., 2021b) as well as the further developments which aimed at making LANs available to the research community at large (Fengler, Bera, et al., 2022).

Second, for all experiments concerning VI,I use the Pyro software library (Bingham et al., 2019), which is built on top of PyTorch (Paszke et al., 2019). Pyro is a probabilistic programming framework with an explicit "VI-first" approach to inference. It has the largest variety of VI-algorithms available out-of-the-box, is extensible to custom likelihood functions (crucial in our case), and remains well-maintained and -documented. Third, for all experiments concerning HMC, I used the NumPyro library (Phan, Pradhan, and Jankowiak, 2019), which is built atop of the JAX (Frostig, Johnson, and Leary, 2018) library for differentiable programming. NumPyro is nearly analogous to Pyro in design and user interface, but has strong support for HMC methods, with an extremely efficient fully compiled NUTS serving as the main workhorse for MCMC. Both Pyro for VI and NumPyro for HMC are therefore representative of the currently achievable inference runtimes concerning numerical experiments such as the ones reported in this chapter.

5.2.6 Details of Numerical Experiments

Computing Setup

We used the computing resources offered via the compute cluster at Brown University, submitting the numerical experiments as sbatch jobs to the system. While the machines I accessed are all of similar quality, there was some variation in hardware setups across parameter recovery runs. This variation in hardware however was not systematic across inference algorithms.

For results reported as having run on CPU machines generally 4-cores and 8GB of RAM were used. The allocated machines distributed as follows. The jobs were allocated to either Intel(R) Xeon(R) Platinum 8268 CPUs clocked at 2.90GHz, Intel(R) Xeon(R) Gold 6126 CPUs clocked at 2.60Ghz or Intel(R) Xeon(R) Gold 6242 CPUs clocked at 2.80Ghz. The performance of these CPUs is similar enough that the main reported runtime findings are not significantly affected by slight differences in resource allocations. The reported differences in runtime trump any potential difference (especially average difference) in computing resources allocated across the tested algorithms.

Results reported as having run on GPU machines equally requested 4 CPU cores and 8 GB of RAM. CPU allocation for these machines varied quite widely. Nonetheless, the allocated GPUs are the main determinants of computation costs in these instance. Utilized GPUs were NVIDIA TITAN RTS, NVIDIA TITAN V, NVIDIA Quadro RTX 6000 and NVIDIA GeForce 3090.

Datasets

For each of the applied SSMs (DDM and ANGLE), I created *two basic datasets* from simulated data, one for *single subject* parameter inference and one for parameter inference with *hierarchical* models.

Datasets were chosen so as to generate the largest possible variance in parameters, while avoiding two defects:

First, I wish to avoid too many ground-truth model parameters hugging the bounds of the permissible parameter space. This will lead to predictable defects in both the MCMC and VI inference machines. The idea is to make sure that the posteriors themselves fall within the parameter training bounds of the LANs used for the numerical experiments, so that the parameter search in VI, as well as the sampling from the typical set (Betancourt, 2017; Cover and Thomas, 2006; MacKay, Mac Kay, et al., 2003) of the posterior following MCMC is not corrupted by reaching those training bounds where likelihood evaluations may become unreliable.

Second, I try to avoid ground truth parameters that lead to corresponding datasets with extremely lopsided choice probabilities. In the extreme case, a dataset might only contain one kind of choice (the other option was never chosen). Such datasets generally lead to problems with parameter identifiability (on top of such problems that may result from utilizing complex generative models to begin with). Parameter values close to the training bounds are generally responsible for such lopsided datasets. Hence, in addition to the restriction of parameters to a contracted hypercube of parameters, instead of sampling this hypercube uniformly, I apply a truncated Gaussian with standard deviation defined as $\frac{1}{6}$ of the allowed parameter range respectively for each of the model parameters θ_i.

The logic applies to the parameter vectors used for both the *Single Subject* and the *Hierarchical* datasets (where it refers to sampling of the group mean parameter).

Single Subject For the *single subject* dataset, I generated 300 parameter vectors (for each of the DDM and ANGLE models) and simulated 500 trials of choice/reaction time for each. The parameter vectors were constrained to lie within the training bounds of the respective LANs that were utilized. The bounds were as follows. The lower bounds for the DDM parameters (v, a, z, t) were set to $(-2.5, 0.6, 0.3, 0.25)$, while the upper bounds were set to $(2.5, 2.3, 0.7, 1.75)$. The lower bounds for the ANGLE parameters (v, a, z, t, θ) were set to $(-2.5, 0.6, 0.3, 0.25, 0.2)$, while the upper bounds were set to $(2.5, 2.3, 0.7, 1.75, 1.0)$.

Hierarchical For the *hierarchical* dataset I similarly generated 300 parameter vectors for each of the DDM and ANGLE models. The structure of parameters was in accordance with the hierarchical nature of the generative model. Subject parameters were drawn from a Gaussian Distribution, with *group* mean parameter μ_g sampled from a truncated normal as per the *Single Subject* section above. The standard deviation of the group distribution was chosen from a uniform distribution, where the lower limit was chosen as 0.05 and the upper limit was chosen as a fraction (0.1) of the range implied by LAN training bounds on the given parameter. All datasets were composed of 20 subjects for which 500 trials each were simulated.

MCMC

We chose to run 2000 burn-in and 3000 actual draws, both for the MCMC runs utilizing a coordinate-wise slice sampler (Neal, 2003) through HDDM (Wiecki, Sofer, and Frank, 2013; Fengler, Bera, et al., 2022), and for the MCMC runs utilizing NUTS (M. D. Hoffman and Gelman, 2014) through NumPyro (Phan, Pradhan, and Jankowiak, 2019). These numbers were chosen to make the runs and ultimately the presentation more uniform. 5000 samples are almost certainly more than needed for the *single subject* datasets, but may sometimes be too few for the *hierarchical datasets* (one reason for convergence metrics potentially going awry). The group-level parameters are often much harder to sample from (resulting in a lower effective number of samples). I generally used and based convergence metrics on two chains (the minimal number, to curtail computational cost of the numerical experiments). Uniform priors (group level priors) were applied for the *single subject* as well as the *hierarchical* runs. I use the $\hat{R}$ as our basic metric to judge convergence of MCMC chains.

The R-hat () statistic is defined as follows. For a given quantity ξ of interest (in our case $\xi = \theta_i$, one of the parameters of a SSM), a given number of chains m, and a number of MCMC samples (potentially after throwing some number of initial samples out as a burn-in sample) n, we start with the following two quantities, defined in direct analogy to the basic Analysis of Variance (ANOVA) framework. First, the between-sequence sum of squares,

$$B = \frac{n}{m-1}\sum_{j=1}^{m}(\bar{\xi}_{.j} - \bar{\xi}_{..})^2 \tag{5.15}$$

where, $\bar{\xi}_{.j} = \frac{1}{n}\sum_{i=1}^{n}\xi_{ij}$ and $\bar{\xi}_{..} = \frac{1}{m}\sum_{j=1}^{m}\bar{\xi}_{.j}$. Second we define the within-sequence sum of squares,

$$W = \frac{1}{m}\sum_{j=1}^{m}s_j^2 \tag{5.16}$$

where, $s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\xi_{ij} - \bar{\xi}_{.j})$.

In the context of MCMC we note that all chains are supposed to sample from exactly the same distribution and therefore both B and V are estimators of the same quantity (the within-sequence variance) and should converge in the limit of $n \to \infty$. However if the chains do not sample from the stationary distribution (yet, since e.g. the number of MCMC samples is still to low or because of other defects), then B will tend to *overestimate* this variance.

Now defining the statistic,

$$Var^{+}(\xi|\mathbf{x}) = \frac{n-1}{n}W + \frac{1}{n}B \tag{5.17}$$

which we can recognize as a weighted sum of W, the within chain variance estimator (unbiased) and B the between chains variance estimator (overestimate). Therefore $Var^{+}(\xi|\mathbf{x})$ is another overestimate of overall chain variance. The $(\hat{R})$ statistic is now defined as an *inflation factor*, between $Var^{+}(\xi|\mathbf{x})$ and W,

$$\hat{R} = \sqrt{\frac{Var^{+}(\xi|\mathbf{x}}{W}} \tag{5.18}$$

Commonly suggested (Gelman and Rubin, 1992; Gelman, Rubin, et al., 1992) is to accept chains for which $\hat{R} < 1.1$.

VI

The underlying probability models used for VI runs were equivalent to the ones used for the MCMC experiments, in which case they translate directly into the posterior shape. Two crucial choices need to be made when performing VI, however, discussed separately below.

Choice of Variational Posteriors A central component of any VI approach is the shape of the variational posterior. I used two types of distributions, Isotropic Multivariate Normal distributions (IMVNs) serving as the simpler option. These are examples of the popular class of variational posteriors known as *mean field posteriors*. Mean field approaches are characterized by the assumption of independence of parameters in the posterior. In addition, I used Multivariate Normal Distributions

(MVNs) with arbitrary covariances (adding more free parameters and allowing for linear dependencies between parameters in posteriors). A broad range of possibilities exist for the construction of variational posteriors and, as mentioned at the end of this chapter, the reported experiments should be treated as only a starting point in this regard.

Optimizer Settings In concurrence with general stochastic gradient descent approaches, one needs to choose settings for the optimizer of the ELBO objective. I chose the ADAM (Kingma and Ba, 2014) algorithm, which is widely applied in the deep learning literature as default. I moreover predetermined the number of optimizer steps to be 2000 across the board. I judged this to be a reasonable stopping criterion, as it is unlikely an underestimation of the number of steps necessary to reach an inescapable minimum. This evaluation was the result of initial pilot experiments. More sophisticated stopping criteria could be applied. However, the initial focus is put on the proof of concept, for which I a priori decided to avoid the otherwise useful philosophy of minimizing runtime. The runtime results below make an attempt at providing realistic relative runtimes that approximate the optimization behavior under reasonable stopping rules. Future experiments should explicitly incorporate an investigation of stopping rules and their effect on the resulting runtimes as well as how these choices may trade off computation with the quality of the variational posterior.

5.3 Results

5.3.1 Runtime

A major motivation for this work was to investigate the runtime behavior of NUTS and VI as they compare to what can currently be achieved with the HDDM python toolbox (Wiecki, Sofer, and Frank, 2013; Fengler, Bera, et al., 2022). In this section I present runtime-related results in two chunks. First, I consider global runtime performance across all tested methods. Second, I consider separately the VI and MCMC runtimes in slightly more detail, using respective appropriate metrics.

Global

As illustrated in Figure 5.2, some clear runtime patterns emerge across the tested algorithms. I preface the discussion by mentioning that I consider an overly detailed focus on these numbers futile, since they will partly be hardware-dependent. What matters more are the rough estimates of runtime coefficients, particularly orders of magnitude changes. Nevertheless, I report certain numbers in detail below, using this philosophy as a guideline as to which of these numbers to focus on.

For the *single subject* datasets consisting of 500 i.i.d trials each, I find that the total runtimes of NUTS via NumPyro Phan, Pradhan, and Jankowiak, 2019 (31s and 42s respectively for CPU and GPU) is roughly half of the runtimes I find for HDDM (87s and 69s respectively for CPU and GPU backends). VI inference performs similarly. This is likely an overestimation, however, given that I chose a predetermined number of optimizer steps. If instead an estimate of necessary optimizer steps

is used (i.e., time to the empirical loss minimum), the VI runtimes would drop by 1/2 to 1/3 to between 8s (VI-CPU - Normal ISO - 1 Particle) and 22s (VI - CPU - Normal ISO - 10 Particles).

For the *hierarchical* datasets, we can identify two broad patterns. First, the difference between GPU and CPU runtimes are pronounced for both VI and NUTS. For VI, GPU batching allows the 10 particle versions of inference to run essentially at the same speed as 1 particle versions (66s vs. 57s for the Normal - ISO version and 35s vs. 46s for the Multivariate Normal version respectively), allowing gradient stabilization at essentially no additional cost. With NUTS, in particular in contrast to HDDM, speed on GPU is roughly 5 times faster than CPU for *hierarchical* datsets (1326s on CPU vs 217s on the GPU), with the GPU version being roughly 8 times faster than the HDDM CPU version (1722s vs. 217s on average respectively). Overall, ignoring possible improvements with regards to optimization steps for VI, the VI algorithm ran roughly 30 times faster than the baseline formed by HDDM on a GPU machine, and VI ran roughly 4 times faster than NUTS on the same GPU machine.

A major contributor to the relatively slow speed of HDDM as compared to NUTS via NumPyro (Phan, Pradhan, and Jankowiak, 2019) is the chunking of computations imposed by HDDM. Batch-processing is impacted by the a priori decomposition of data into groups (e.g., by subject and/or experimental condition). LANs are internally called separately for each such group, therefore inflating the number of distinct forward passes on respectively small batch sizes for a given full sweep through the parameter space. NumPyro (Phan, Pradhan, and Jankowiak, 2019) instead allows consideration of all data in a single batch, which is then used for joint parameter updates, therefore tailoring the computations much more closely to optimal utilization of GPUs.

These results hint at the significant impact that modern posterior inference algorithms can have for the application of Bayesian inference to SSMs. A 30 fold speed increase will allow researchers to fit in a day what currently may take a month, severely affecting the possibilities for exploration in the modeling space.

Effective sample size

We can investigate some aspects of runtime in slightly more detail. As reported above, NUTS promises roughly a halfing of inference time on *single subject* data as well as providing an 8-fold speed increase for *hierarchical* data (point estimates based on the specific dataset settings). For MCMC methods, however, it is not just the time to draw a number N of samples which matters. Across samplers one may observe large discrepancies in mixing behavior (autocorrelation). Instead of reporting the number of samples drawn from a sampler directly, it is therefore common to instead focus on the number of *effective samples* embedded in a given number N of MCMC draws. The effective sample size represents the equivalent number of independent samples from the target distribution embedded in an autocorrelated MCMC sample. If a given algorithm is able to extract more effective samples per draw, we can therefore opt to run shorter chains with equivalent performance. The effective sample size is defined as

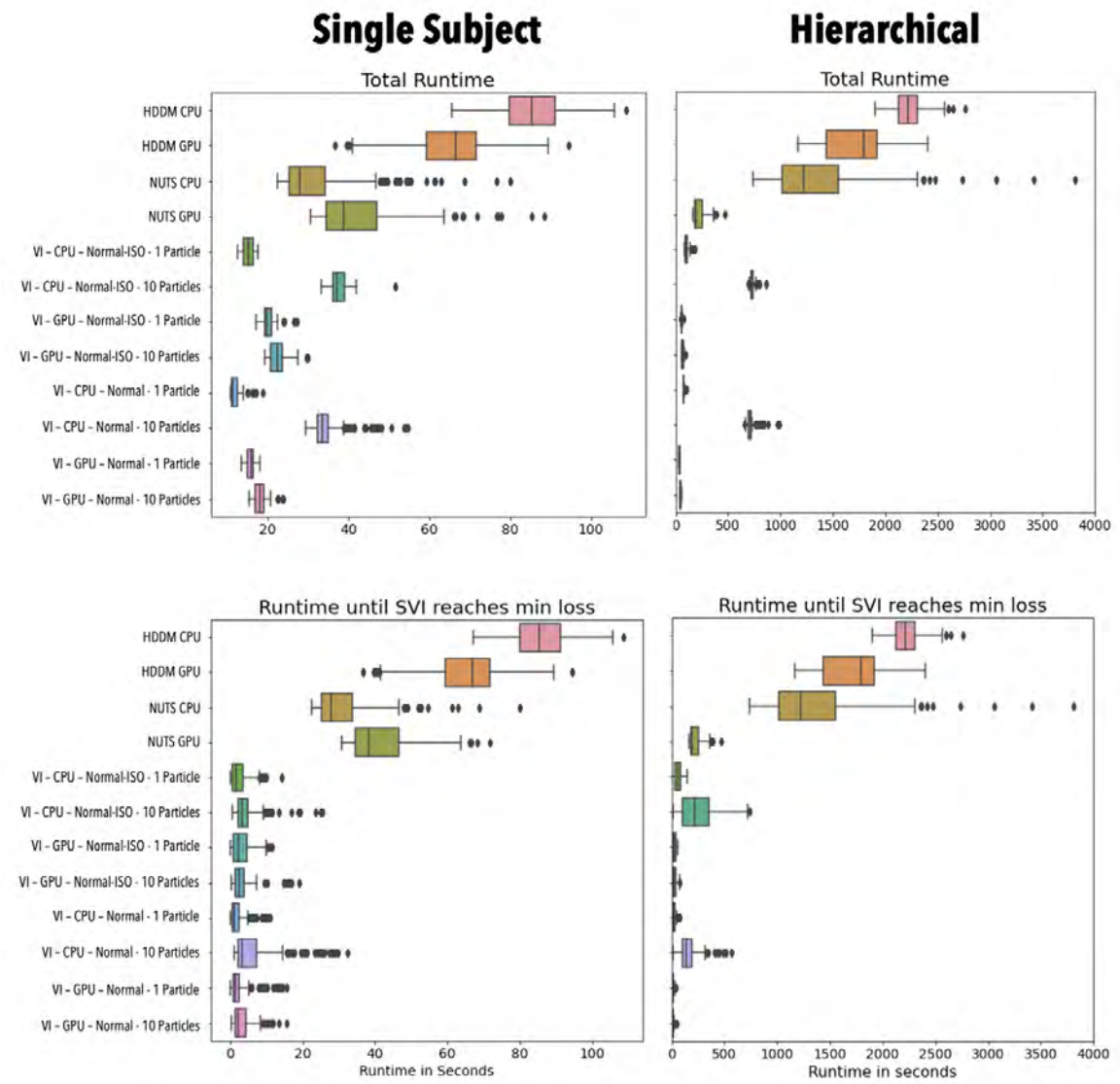

Figure 5.2. Illustration of runtime distributions across datasets for the two type of data structures considered. (*Left*) Single subject data of 500 trials each, and (*right*) hierarchical data which includes 20 subjects with 500 trials each. Two types of runtime metrics are reported, which are relevant for the VI algorithms under consideration. *Total Runtime* (*top*), which refers to the VI runtimes for the predetermined number of 2000 optimizer steps, and *Runtime until SVI reaches min loss* (*bottom*), which calculates runtime for VI algorithms based on how many optimizer steps were needed to find the actual minimum ELBO in the sequence of 2000 predetermined steps. Across the panels, *four* broad types of algorithms are shown with different settings (hardware and/or hyperparameters). *HDDM* refers to MCMC runs utilizing the HDDM python toolbox (Wiecki, Sofer, and Frank, 2013). *NUTS* refers to MCMC runs utilizing the No-U-Turn Sampler (M. D. Hoffman and Gelman, 2014) via the NumPyro python package (Phan, Pradhan, and Jankowiak, 2019), and *VI* via the Pyro python package (Bingham et al., 2019). VI uses two respective variational distributions, a Normal distribution with isotropic covariance structure (*Normal-ISO*) and a Normal distribution with full covariance structure (*Normal*).

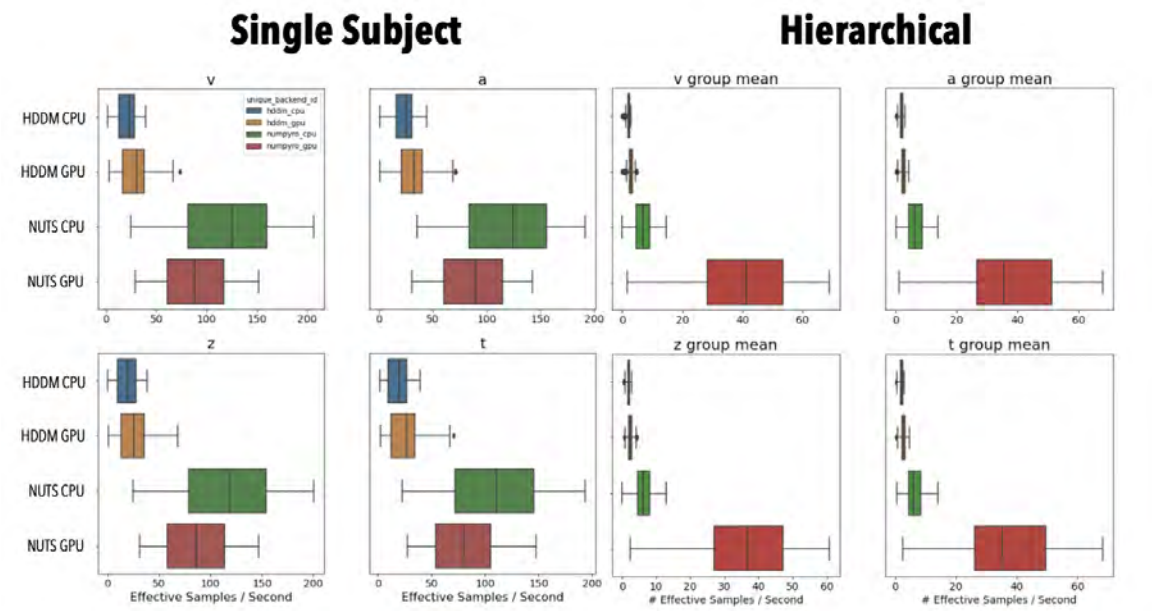

Figure 5.3. Runtime comparison for the MCMC algorithms under consideration. The metric is *Effective Samples per Second*. NUTS generally outperforms HDDM (and its slice sampler), with increasing benefits for hierarchical data structures compared to the single subject case. NUTS via NumPyro (Phan, Pradhan, and Jankowiak, 2019) can exploit batch-processing using the GPU across the whole dataset, which leads to massive speedups for large/hierarchical datasets.

$$n_{eff} = \frac{mn}{1 + 2\sum_{t=1}^{\infty} \rho_t}$$

where ρ_t refers to the autocorrelation with lag t of the given MCMC chain, m is the number of chains used and n is the number of samples per chain (used, without any potential burn-in period). The ρ_t's are estimated from the chains with some estimator of choice, $\hat{\rho}_t$, and the infinite series is truncated via some rule of thumb (e.g. the lowest t such that $\rho_t + \rho_{t+1} < 0$), so that functionally one works with,

$$\hat{n}_{eff} = \frac{mn}{1 + 2\sum_{t=1}^{T} \hat{\rho}_t}$$

for some T. Once we have the *effective sample size* we may then consider the a priori runtime and compute the *effective samples per second* of the HDDM and NUTS algorithms. These numbers are reported in Figure 5.3, where I find that the speed-up from HDDM to NUTS is roughly three-fold for *single subject* and 20-fold for the *hierarchical* data.

5.3.2 Parameter Recovery

While informative, the runtime results reported above are critically dependent on a comparable quality of posterior inference across methods. In this section I report the basic parameter recovery

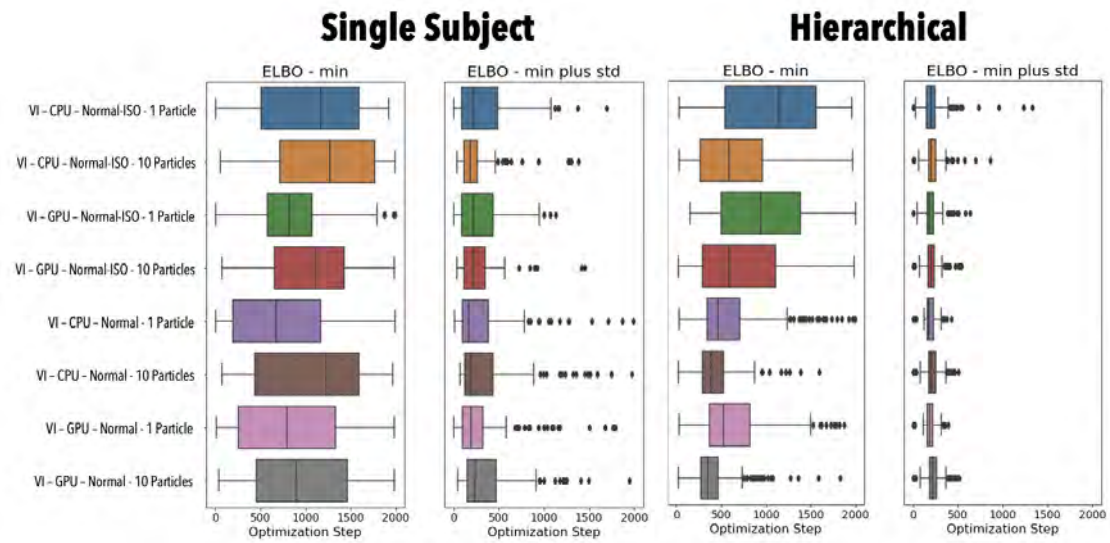

Figure 5.4. Illustration of runtime aspects for the VI algorithms under consideration. I show respectively for the *single subject* (*LEFT*) and *hierarchical* (*RIGHT*) cases, how many steps of the optimizer where necessary to reach the final reported ELBO minimum (the left of two subplots in each case, with *ELBO - min* title) and the ELBO minimum plus one standard deviation (computed from the tail of the loss trajectory, respectively the right of two subplots, with *ELBI - min plus std* title). The 2000 optimizer steps were chosen naively a-priori for all numerical experiments. Roughly 1000 optimizer steps would have been sufficient to find the reported ELBO minimum. Moreover, since most losses end in a long tail of minimal fluctuations, it is reasonable to not consider the ELBO minumum, but instead a representative 'average' for which I added 1 standard deviation of this tail loss trajectory to the ELBO minimum. This benchmark was achieved in less than 500 optimizer steps for the vast majority of cases, suggesting that the runtime for VI algorithms can be further reduced.

performance of the tested algorithms for a more detailed comparison.

Figures 5.5, 5.7 and 5.8 show that indeed, performance is very similar across all methods. For the *single subject* data, the R^2 values derived from regression recovery parameters on ground truths parameters are essentially identical. For the group mean parameters in hierarchical datasets, reported in 5.7 and 5.8 the same observation holds. I corroborate these findings by directly correlating the results across methods as shown in Figure 5.6 (for *single subject* datasets), Figure 5.9 (for *group means* in *hierarchical* datasets) and Figure 5.10 (for *group standard deviations* in *hierarchical* datasets). In all cases correlations very close to 1 are obtained. While currently limited to numerical experiments utilizing the basic DDM model, these results are encouraging as signs of the robustness of the VI and NUTS algorithms.

5.3.3 Calibration

While parameter recovery is crucial and the results in the previous section encourage the use of LANs for modern inference, evaluating the quality of posterior inference requires an additional step. In particular one would like to know how accurate the methods are in recovering the full posterior

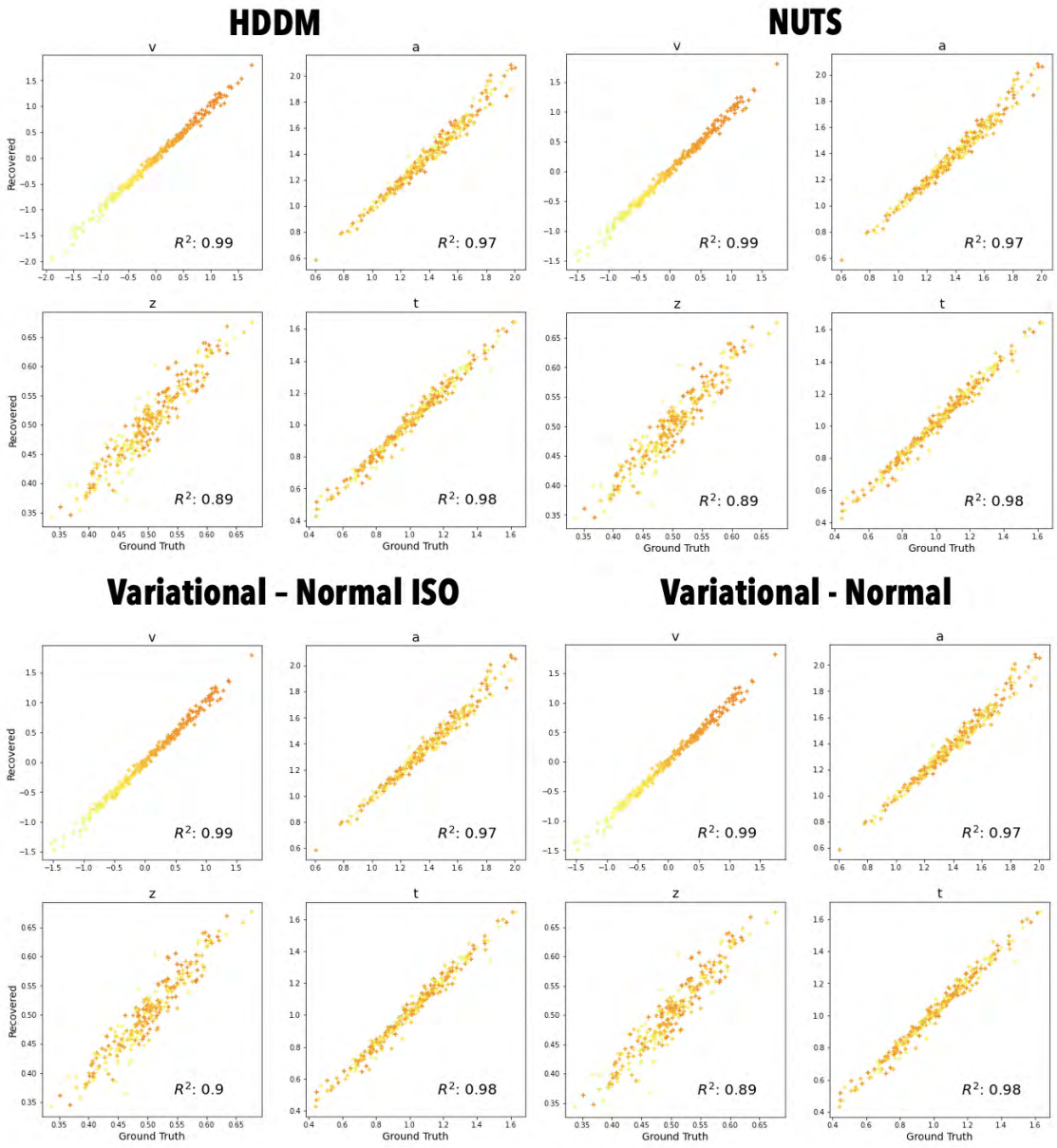

Figure 5.5. Depiction of parameter recovery performance for the DDM model, across the *four* general algorithms tested (HDDM, NUTS, VI via IMVNs, VI via MVNs). The R^2 statistics from a basic regression of *ground truth* on *recovered* parameters are reported for each parameter of the model, respectively for each algorithmic approach. The dots are colored according to the *choice probability* towards choice 1 in the underlying observed datasets. Yellow for choice probability 0 and red for choice probability 1. Parameter recovery is extremely similar across all methods, with R^2 values being nearly identical.

distribution, which is necessary for drawing conclusions about, e.g., whether a given parameter differs between two experimental conditions.

To test an inference algorithm for this purpose, Talts et al., 2018 introduced simulation-based

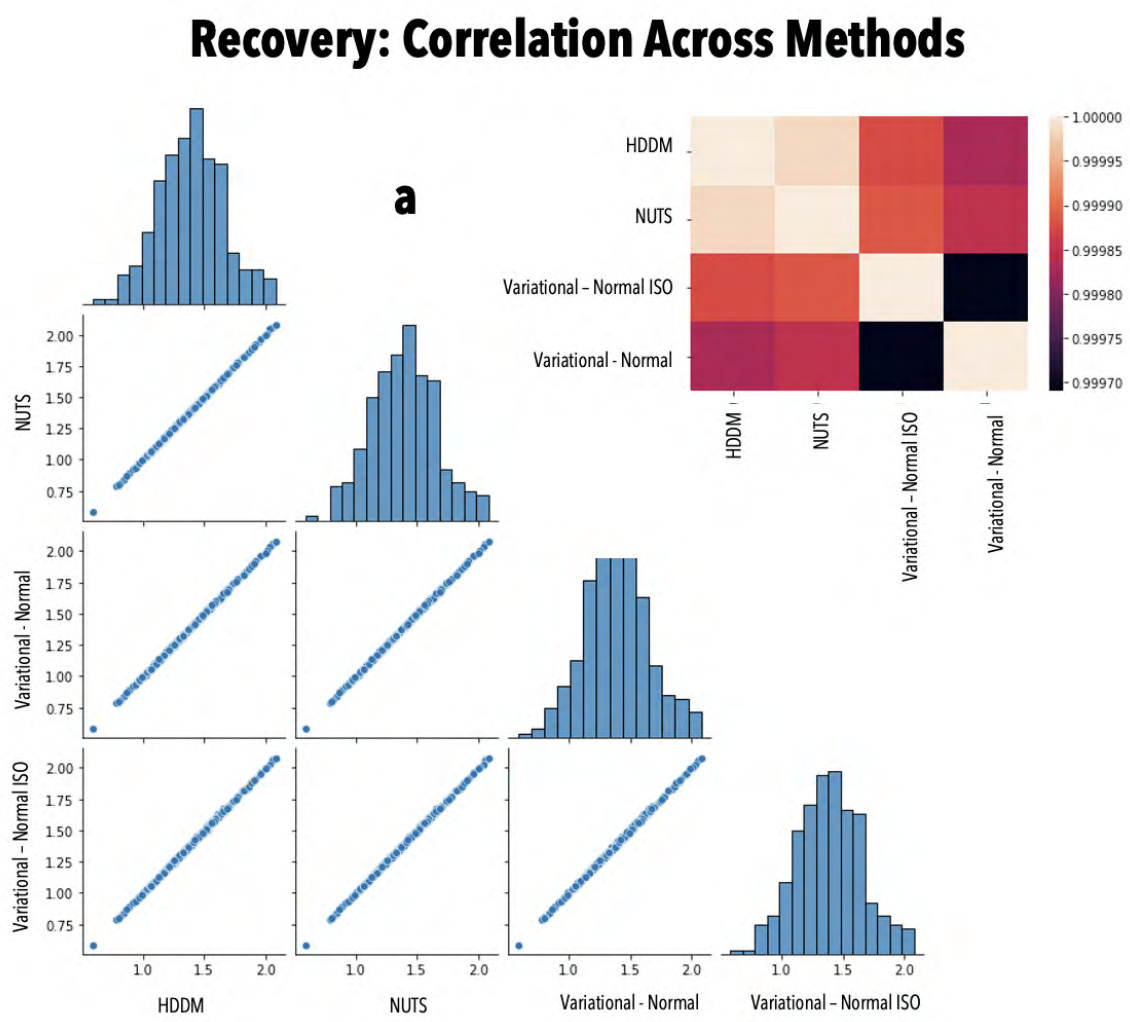

Figure 5.6. Another look at the similarities between the parameter recovery performance across the *four* tested algorithms. The example shows their respective similar values for the a parameter of the DDM. The performance on other parameters looks qualitatively identical. The heatmap (*right*) shows that the correlation between methods is very close to 1 across the board. The pairwise plot (*left*) similarly presents essentially unity lines of recovered parameter values across all methods (the parameter values are extremely close to identical across the board).

calibration, yielding the corresponding rank-plots I report in Figures 5.13, 5.15 and 5.16. The idea behind these rank plots is as follows. Under a consistent inference procedure, simulating data from the assumed generative model and running the inference algorithm will produce a distribution of *p-values* for ground truth parameters that is uniform in the respective posterior. The *p-value* is here defined as $\hat{P}_{(\theta|\mathbf{x})}(\theta_{groundtruth})$, the cumulative density of $\theta_{groundtruth}$ under the approximate posterior (derived either from HDDM, NUTS or VI). In the context of null-hypothesis significance testing, this is know as the *one-sided p-value*.

Most violations of this neatly fit into *four* major interpretable categories (Talts et al., 2018),

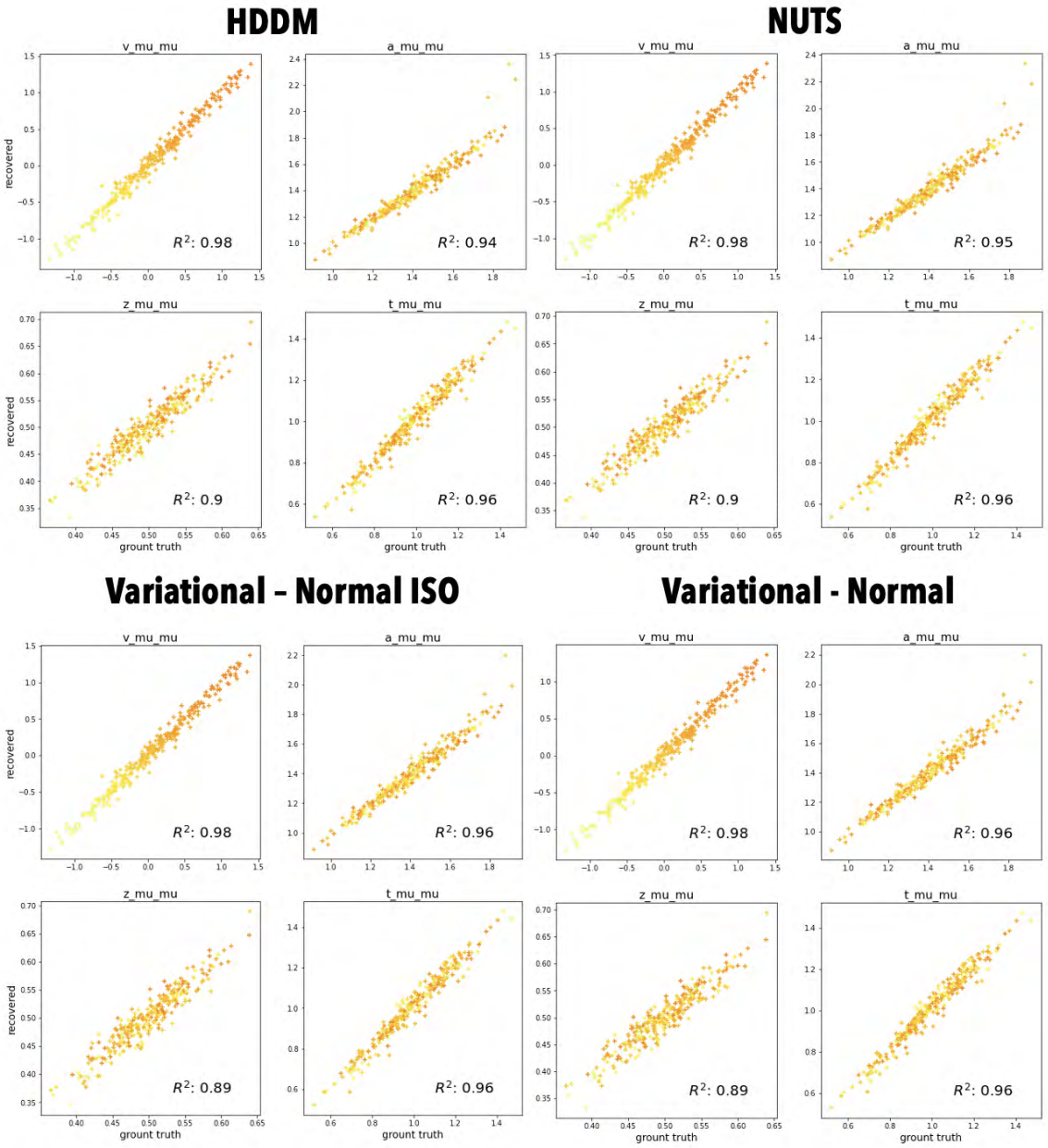

Figure 5.7. Depiction of parameter recovery performance for the DDM model, across the *four* general algorithms tested (HDDM, NUTS, VI via IMVNs, VI via MVNs). The R^2 statistics from a basic regression of *ground truth* on *recovered* parameters are reported for each parameter of the model, respectively for each algorithmic approach. Here I consider the hierarchical datasets, and *group level mean* (μ, *_mu_mu* in the titles) parameters are reported. The dots are colored according to the probability of choosing 1 in the underlying observed datasets. Yellow for choice probability 0 and red for choice probability 1. Parameter recovery is extremely similar across all methods, with nearly identical R^2 values.

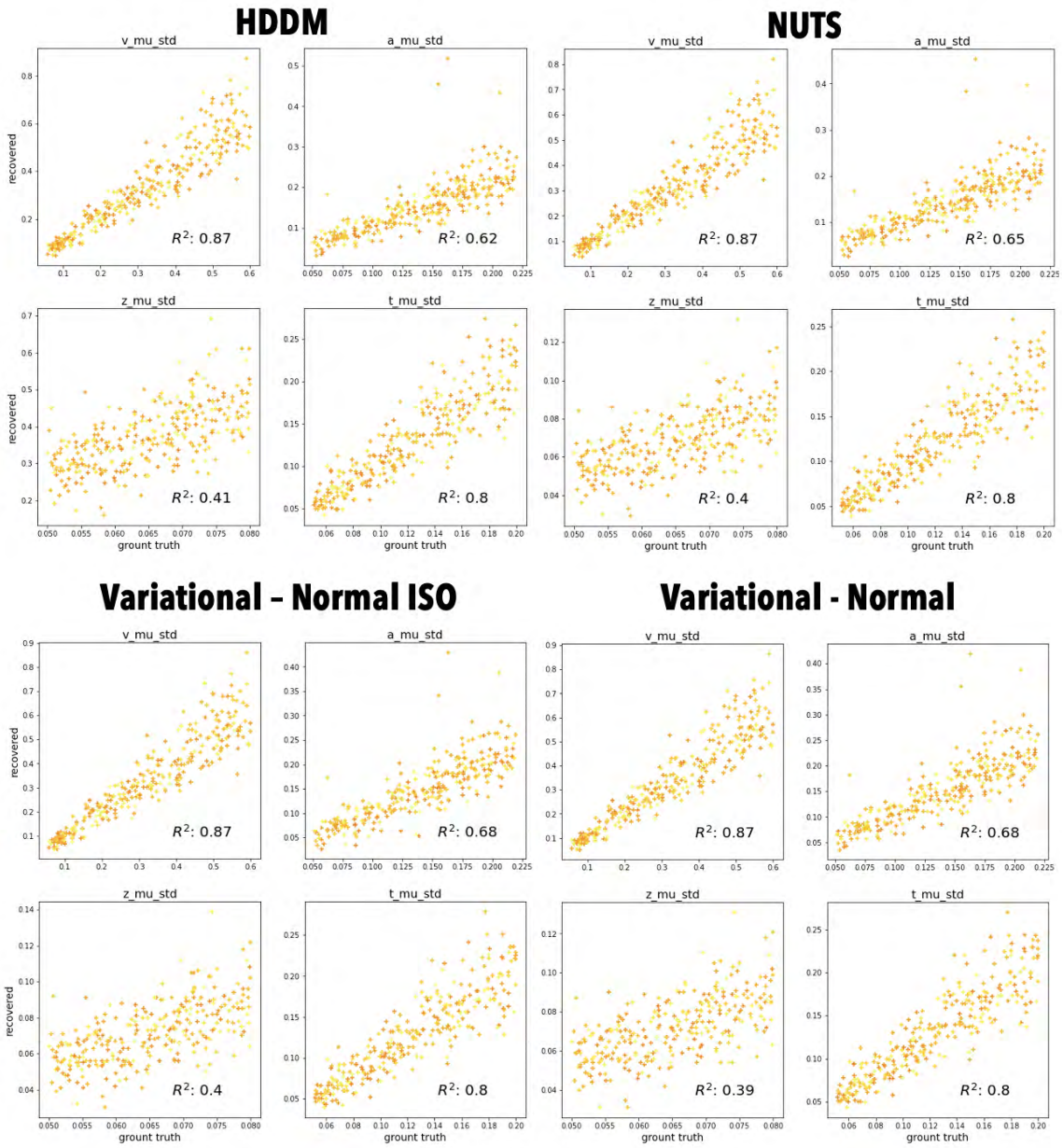

Figure 5.8. Depiction of parameter recovery performance for the DDM model, across the *four* general algorithms tested (HDDM, NUTS, VI via IMVNs, VI via MVNs). The R^2 statistics from a basic regression of *ground truth* on *recovered* parameters are reported for each parameter of the model, respectively for each algorithmic approach. Here I consider the hierarchical datasets, and *group-level standard deviation* (σ, *mu_std* in the titles) parameters are reported. The dots are colored according to the probability of choosing 1 in the underlying observed datasets. Yellow for choice probability 0 and red for choice probability 1. Parameter recovery is extremely similar across all methods, with R^2 values nearly identical.

illustrated in Figure 5.12. The rank-plots deriving from the work conducted in this chapter can now be assessed with with the aid of these failure mode patterns.

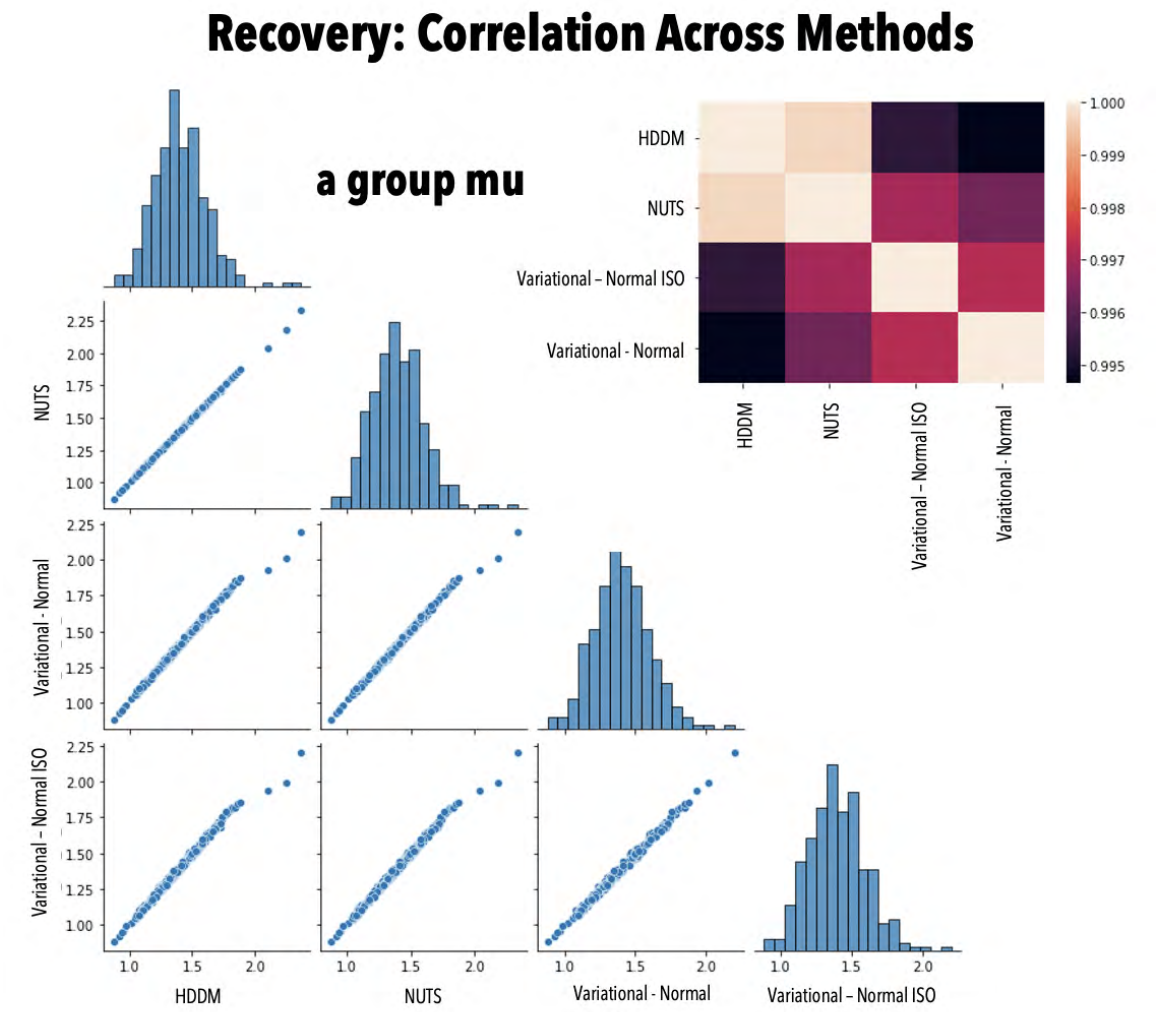

Figure 5.9. Another look at the similarities between the parameter recovery performance across the *four* tested algorithms, here for *group mean* parameters concerning hierarchical datasets. The example shows the respective similar values for the *a group mean* parameter of the DDM. The performance on other parameters looks qualitative identical. The heatmap (*right*) shows that the correlation between methods is very close to 1 across the board. The pairwise plot (*left*) paints a similar picture, presents essentially unity lines of recovered parameter values across all methods (the parameter values are extremely close to identical across the board).

Beginning with the *single subject* datasets, we can observe two types of problems in Figure 5.13. First, a general tendency to underestimate the *non-decision time* across all methods (more pronounced in the Variational - Normal ISO version). This tendency has been observed and pointed out in other recent research (Boelts et al., 2022). This bias is extremely small (this is hard to deduce from the rank plots, since these are not designed to show the actual size of a bias, only to show that a bias may exist), since posterior uncertainty on the non-decision time tends to be minimal

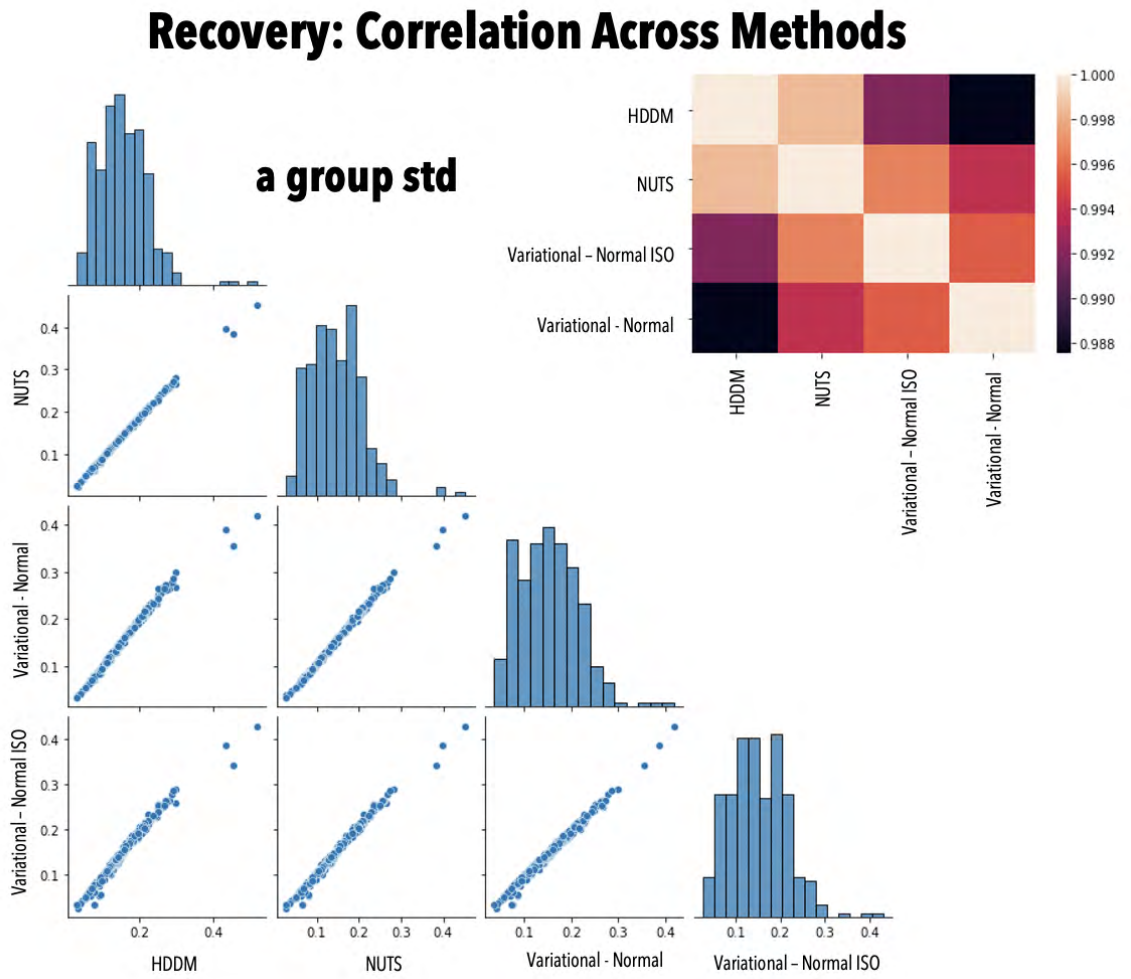

Figure 5.10. Another look at the similarities between the parameter recovery performance across the *four* tested algorithms, here for *group standard deviation* parameters concerning hierarchical datasets. The example shows the respective similarities for the *a group standard deviation* parameter of the DDM. The performance on other parameters looks qualitative identical. Via the heatmap one can observe (*right*) that the correlation between methods is very close to 1 across the board. Another via is afforded by the pairwise plot (*left*) which presents essentially unity lines of recovered parameter values across all methods (the parameter values are extremely close to identical across the board).

to begin with, and reflects a limitation in the current approach to training LANs, which can be easily remedied. [1] This is accompanied by a concomitant (but much less pronounced) tendency to overestimate the *boundary separation* parameter a (given that it is generally negatively correlated with the non-decision time parameter in the posterior). Both the v and z parameters seem well calibrated for the HDDM, NUTS and for VI with MVN variational distribution.

[1] LANs are built on top of kernel density estimates of reaction times, which allow a bleeding edge of positive density for essentially negative RTs under a given non-decision time parameter. This is a solvable problem but outside the scope of this work.

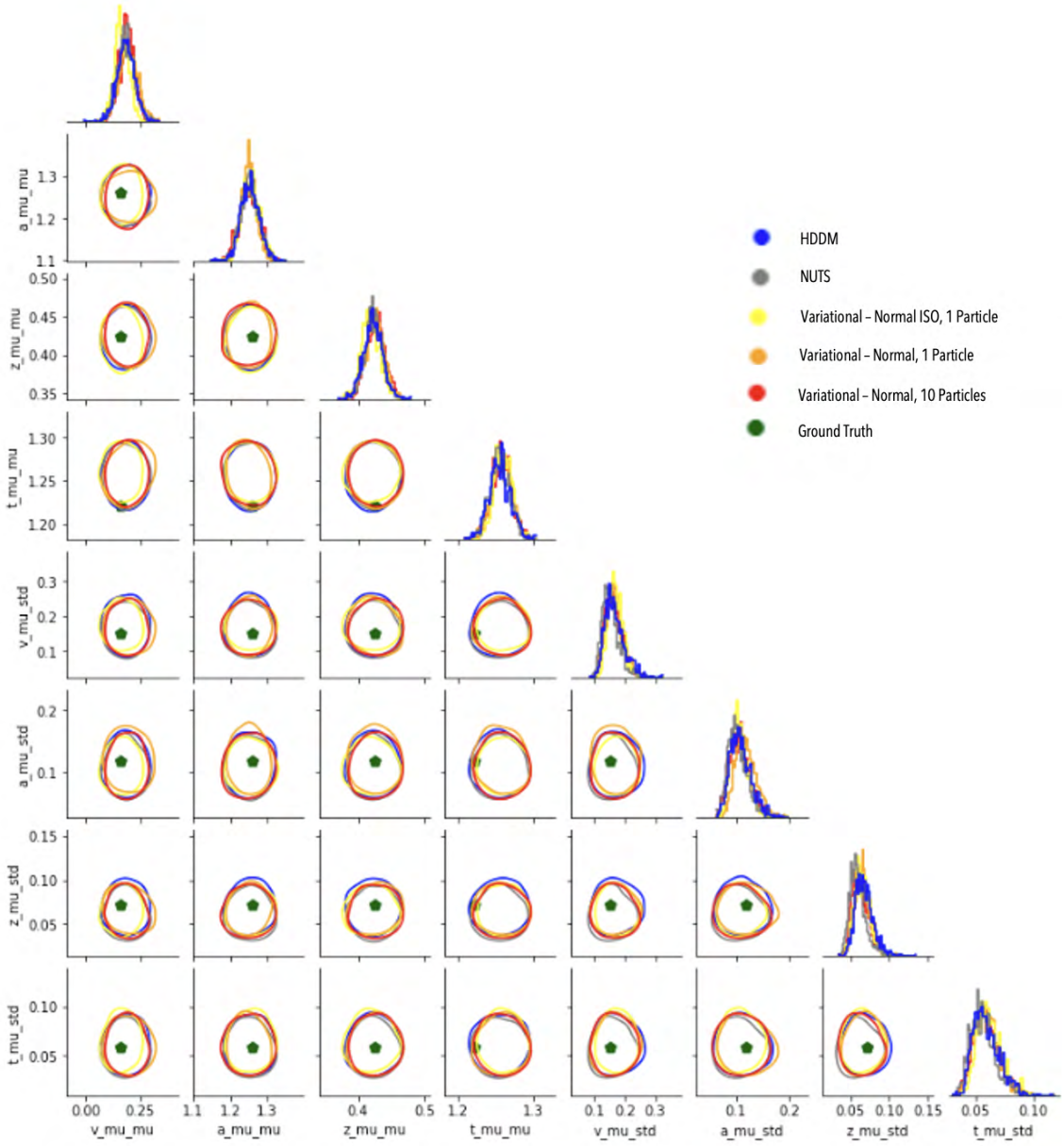

Figure 5.11. Example of actual (pairwise) posteriors across a selection of tested algorithms. One can see how the generated posteriors are nearly identical for HDDM (blue), NUTS (gray) and VI with full covariance matrices (orange and red). In contrast to the *single subject* datasets, VI using IMVNs performs essentially identically to the other algorithms, since at the group level one doesn't observe strong parameter correlations. Hypothesis testing based on posteriors from the IMVN-VI algorithm will therefore tend to inflate the Type I error probability (percentage of falsely rejected null hypotheses, or in other words wrongly detected effects).

The second observation is that the calibration of VI with IMVNs exhibits a tendency towards overconfidence (upper right in Figure 5.12). Figure 5.14 makes it clear that this is a function of the

mean-field assumption (Murphy, 2012) embedded in the IMVN variational distribution (assuming no dependencies in the posterior). Figure 5.14 illustrates the observed sizable posterior covariances (e.g. between the z and v parameters, and the z and t parameters). Mean-field VI (using IMVNs e.g.) will underestimate the posterior variances while capturing posterior modes in this case, while VI using MVNs allows capturing these variances and establish appropriate calibration. This phenomenon confirms that simply checking parameter recovery performance is insufficient in establishing the calibration of a Bayesian inference procedure. Posterior means may be perfectly estimated (as shown with VI based on IMVNs), while much of the (potentially equally important) structure of the true posterior is not modeled well.

Next I consider calibration performance for the *hierarchical* datasets. The rank-plots are split across Figures 5.15 and 5.16 which separately consider the *group mean* and *group standard deviation* parameters. The slight biases in *non decision time* t and *boundary separation* a are maintained as in the *single subject* datasets. The same reasoning applies. Two novel patterns, however can also be observed.

First, at the group level, VI using an IMVNs is well-calibrated, like HDDM, NUTS and VI using the MVN variational family. Figure 5.11 illustrates why. There seem to be no significant posterior correlations present at the group level and therefore choosing an IMVN variational distribution is already sufficient for calibration.

Second, at the level of *group standard deviations* (illustrated in Figure 5.16) all algorithms seem to perform very similarly and one (as one would expect) does not observe a bias specific to the *non-decision time* and *boundary separation* parameters.

Overall, I conclude that NUTS and VI using MVN variational distributions are both at least as well calibrated as HDDM. Therefore, given their superior runtime performance, both algorithms should be preferred in newer versions of HDDM over slice samplers for inferring DDM paraemters using LANs. All algorithms ultimately use LANs for likelihood evaluation and therefore inherit any flaw of LANs, such as the slight miscalibration of the *non-decision time* and *boundary separation* estimations. I emphasize again that this miscalibration is essentially negligible in terms of the actual parameter shifts, and that there is little to no indication that the variance of the respective posteriors is inaccurate. Hence, statistical tests concerning parameter comparisons across groups will not be affected negatively. We, however, should strive to improve the LAN framework to eradicate the remaining shortcomings.

5.3.4 ANGLE Model

Confirming that VI and NUTS are viable for inference in DDMs with LANs, I added numerical experiments using the ANGLE model (as described above). For these experiments, I used *single subject* datasets with 1000 trials, and *hierarchical* datasets with 100 subjects, 100 trials each.

We made the following general observations. First, as can be gleaned from our previous work (Fengler, Govindarajan, et al., 2021b), posteriors for the ANGLE model include strong parameter correlations, beyond even those observed for the DDM (illustrated in Figure 5.17. Hence, I focus the

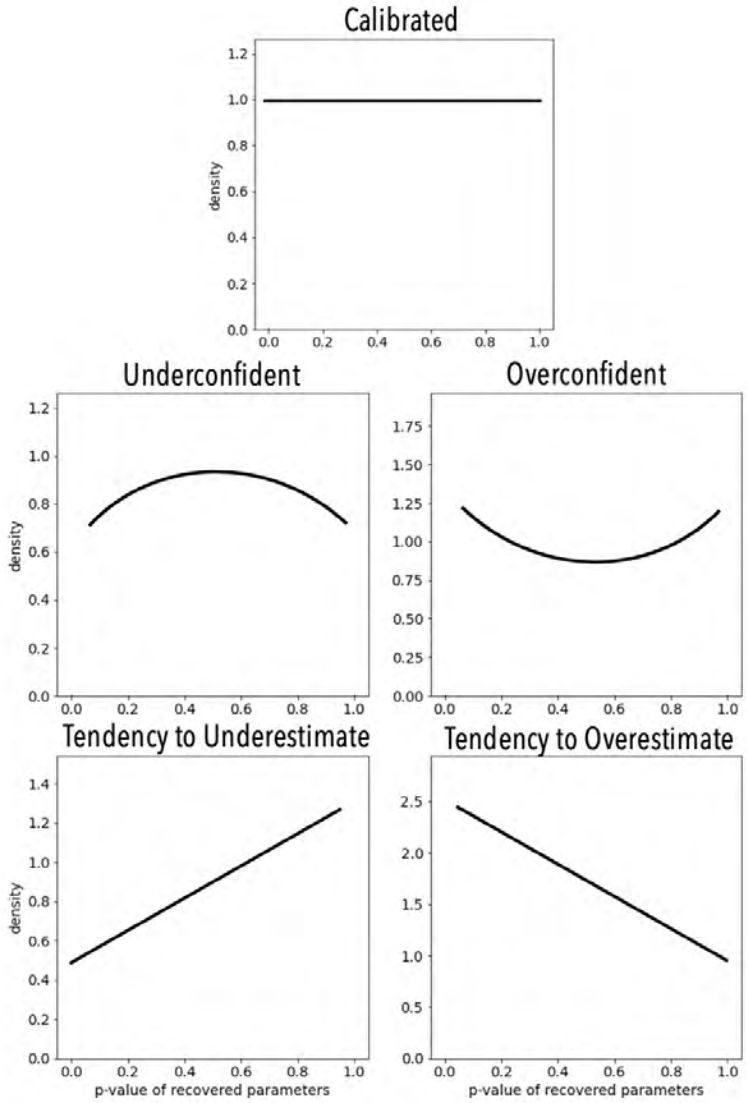

Figure 5.12. Failure modes of calibration as they appear in SBC rank plots. A uniform distribution of p-values suggests good calibration (top). An inverse-U shape suggests under-confident posteriors, i.e., posteriors that are too wide (upper left). A U shape suggests overconfident, or overly narrow, posteriors (upper right). A positive slope suggests a tendency to underestimate a parameter (lower left). A negative slope suggest a tendency to overestimate a parameter (lower right).

VI experiments on MVNs, dropping IMVNs since the IMVN family is trivially insufficient for the estimation problem at hand.

For both NUTS and VI based on MVNs one can observe a number of parameter runs that go awry, shooting specifically the *boundary separation* parameter a to the bounds of the allowed range. This problem is exacerbated for *hierarchical* datasets with NUTS, where 50% of runs had to be thrown out due to lack of convergence. Usually one of the two chains shoot to the boundaries, compromising

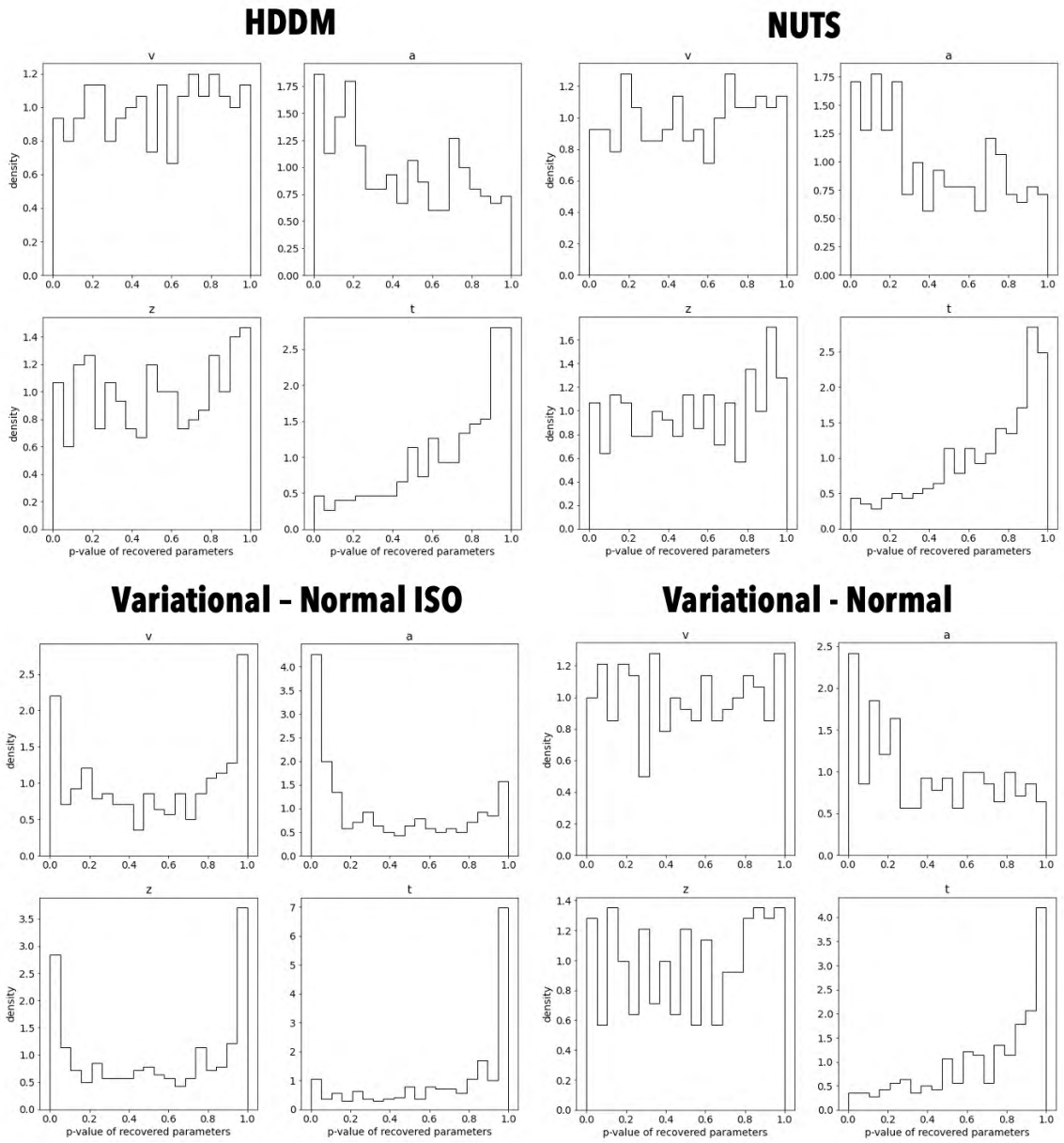

Figure 5.13. Simulation-based calibration (Talts et al., 2018) plots by DDM parameter for *single subject* datasets, split by the *four* inference algorithms under consideration. Note that HDDM, NUTS and VI using MVN variational posteriors are generally well calibrated, apart from a slight bias in the *non decision time* (it tends to be slightly underestimated) and an even smaller consequent bias in the *boundary parameter a*. VI with IMVN posteriors shows worse calibration. As a result of overly narrow posterior distributions, more *p-values* cluster at the extremes of 0 and 1.

the $\hat{R}$ statistic (Gelman, Rubin, et al., 1992).

VI is shown to be somewhat more robust to this phenomenon, especially for *hierarchical* datasets. However, note that estimation benefited form increasing the number of *particles* underlying gradient

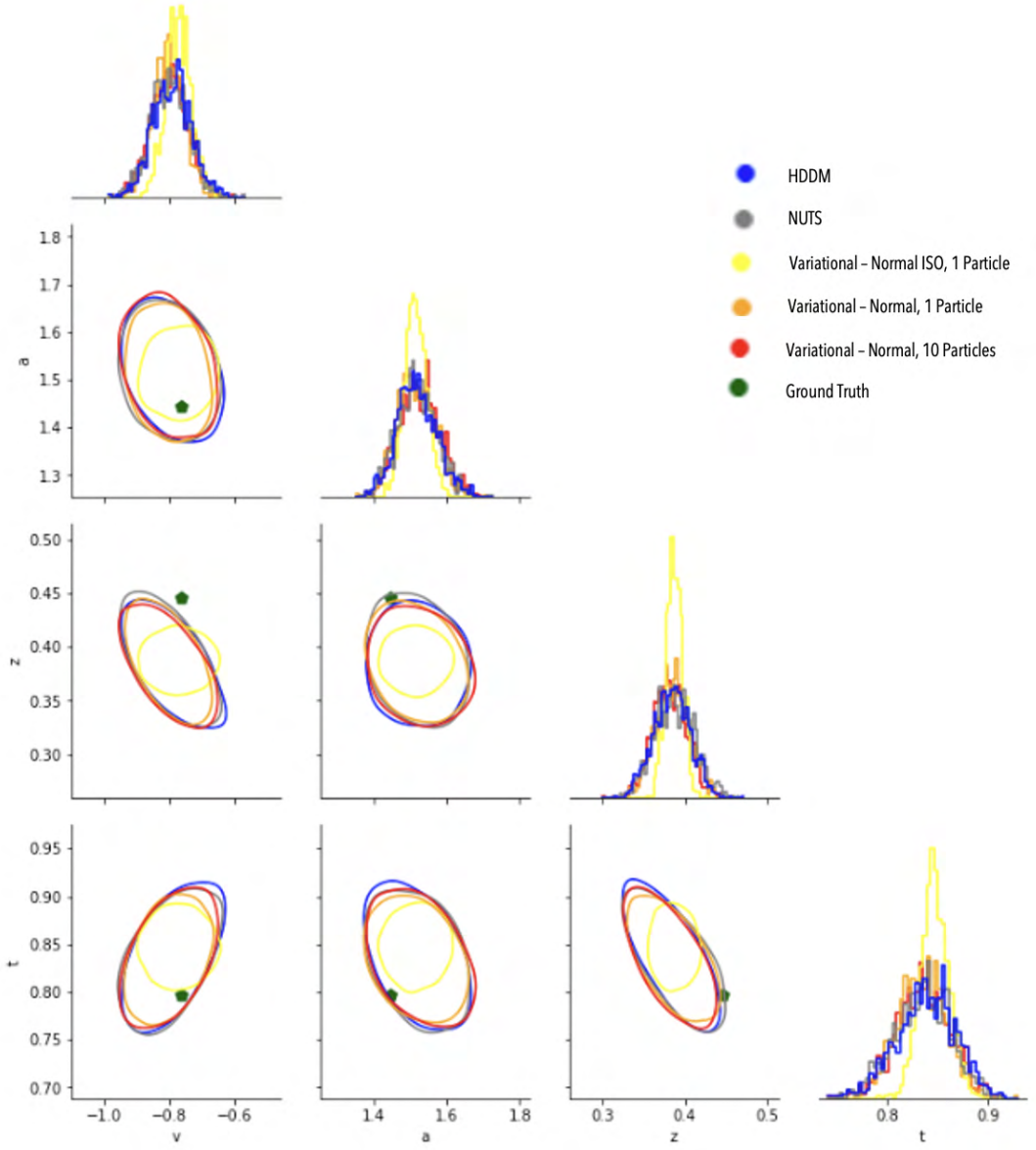

Figure 5.14. Example of actual (pairwise) posteriors across a selection of tested algorithms. The generated posteriors are nearly identical for HDDM (blue), NUTS (gray) and VI with full covariance matrices (orange and red). VI using IMVNs (essentially VI under the mean-field assumption with marginal Normal distributions), suffers from mode-seeking behavior. Since covariances are ignored, one finds a posterior which is correct in the mean estimate, but too narrow. Hypothesis testing based on posteriors from the IMVN-VI algorithm will therefore tend to inflate the Type I error probability (percentage of falsely rejected null hypotheses or wrongly detected effects).

computations (stabilizing the corresponding expected loss), which were increased from 10 to run numerical experiments with 50 and 100 particles. Moreover, the *learning rate hyperparameter* of ADAM (Kingma and Ba, 2014) did have an additional effect on the stability of the optimization

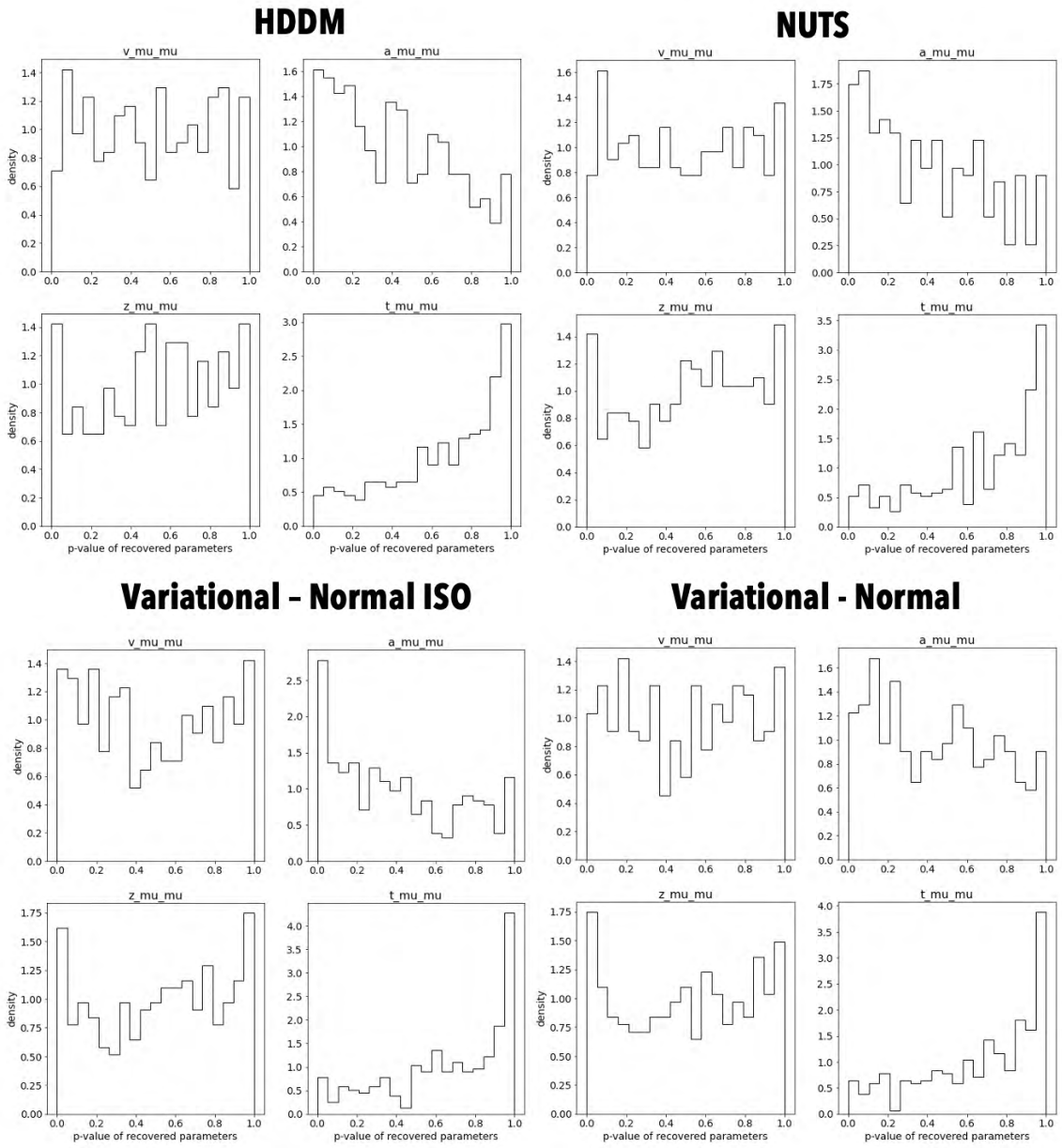

Figure 5.15. Simulation-based calibration (Talts et al., 2018) plots by DDM parameter for *single subject* datasets, split by the *four* inference algorithms under consideration. HDDM, NUTS and VI with MVN posteriors are generally well calibrated, apart from a slight bias in the *non-decision time* (it tends to be slightly underestimated) and a resulting even slighter bias in the *boundary parameter a*. VI with IMVN posterior shows worse calibration. As a result of overly narrow posterior distributions, more *p-values* cluster at the extremes of 0 and 1.

routines. I decreased it from 0.02, underlying the numerical experiments on the DDM model to 0.001 for the numerical experiments involving the ANGLE model, significantly improving the results. Running Pyro (Bingham et al., 2019) on a GPU machine, made the extra computational burden concerning #-particles negligible to runtime concerns, since #-particles are subject to batch

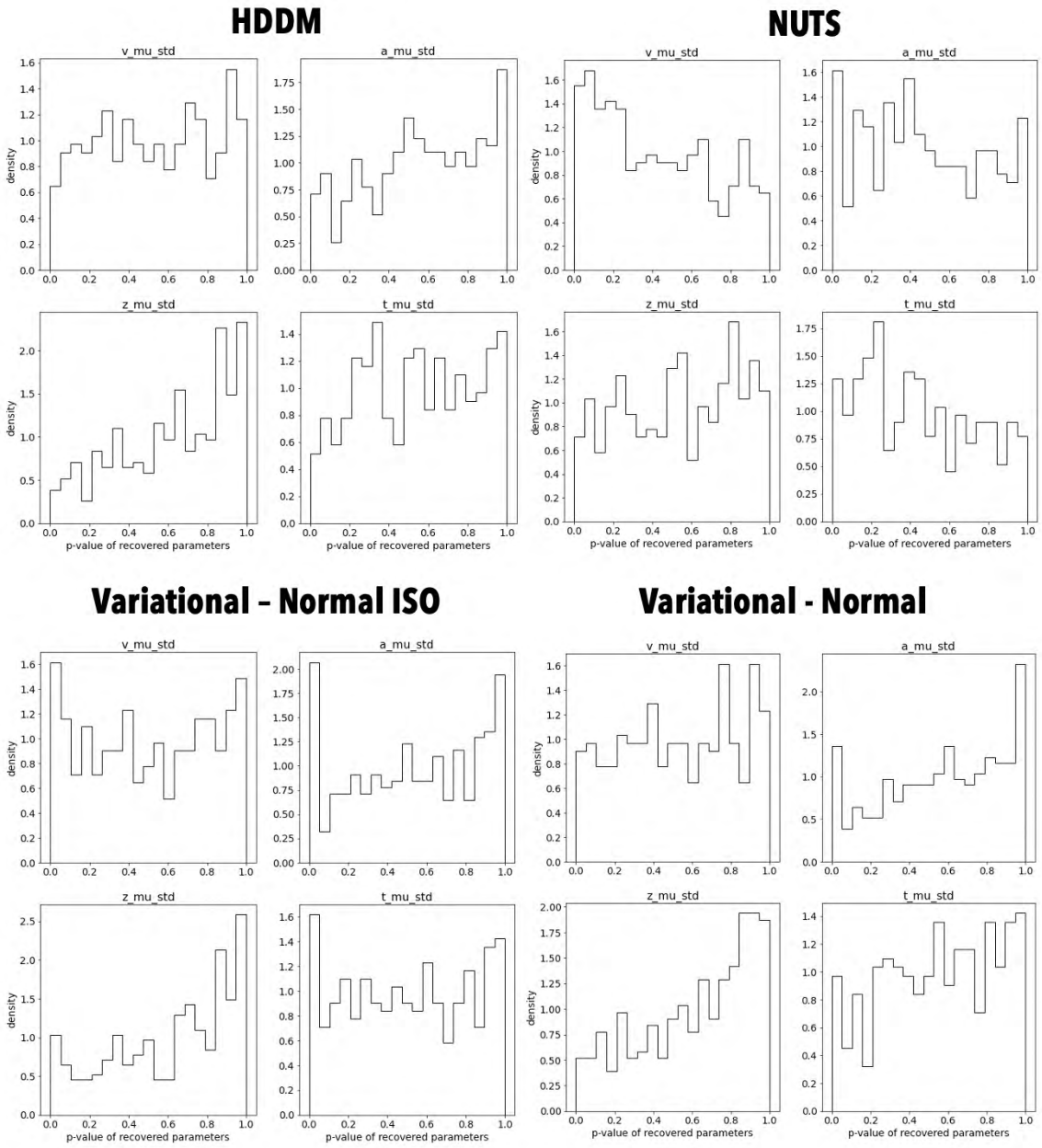

Figure 5.16. Simulation based calibration (Talts et al., 2018) plots by DDM parameter for *hierarchical* datasets, split by the *four* inference algorithms under consideration. The HDDM, NUTS and both VI approaches are generally well calibrated at the *group standard deviation level*, the bias concerning *non-decision times* does not translate to a bias in the *group standard deviation*. There seems to be a slight bias in the *bias parameter z* for HDDM and both VI algorithms, which does not persist under NUTS. This may result from unintionally informative priors.

computation.

Figures 5.18 and 5.19 report parameter recovery results respectively for the *single subject* and *hierarchical* datasets. Calibration for *single subject* data is included in Figure 5.18. For the *hierarchical*

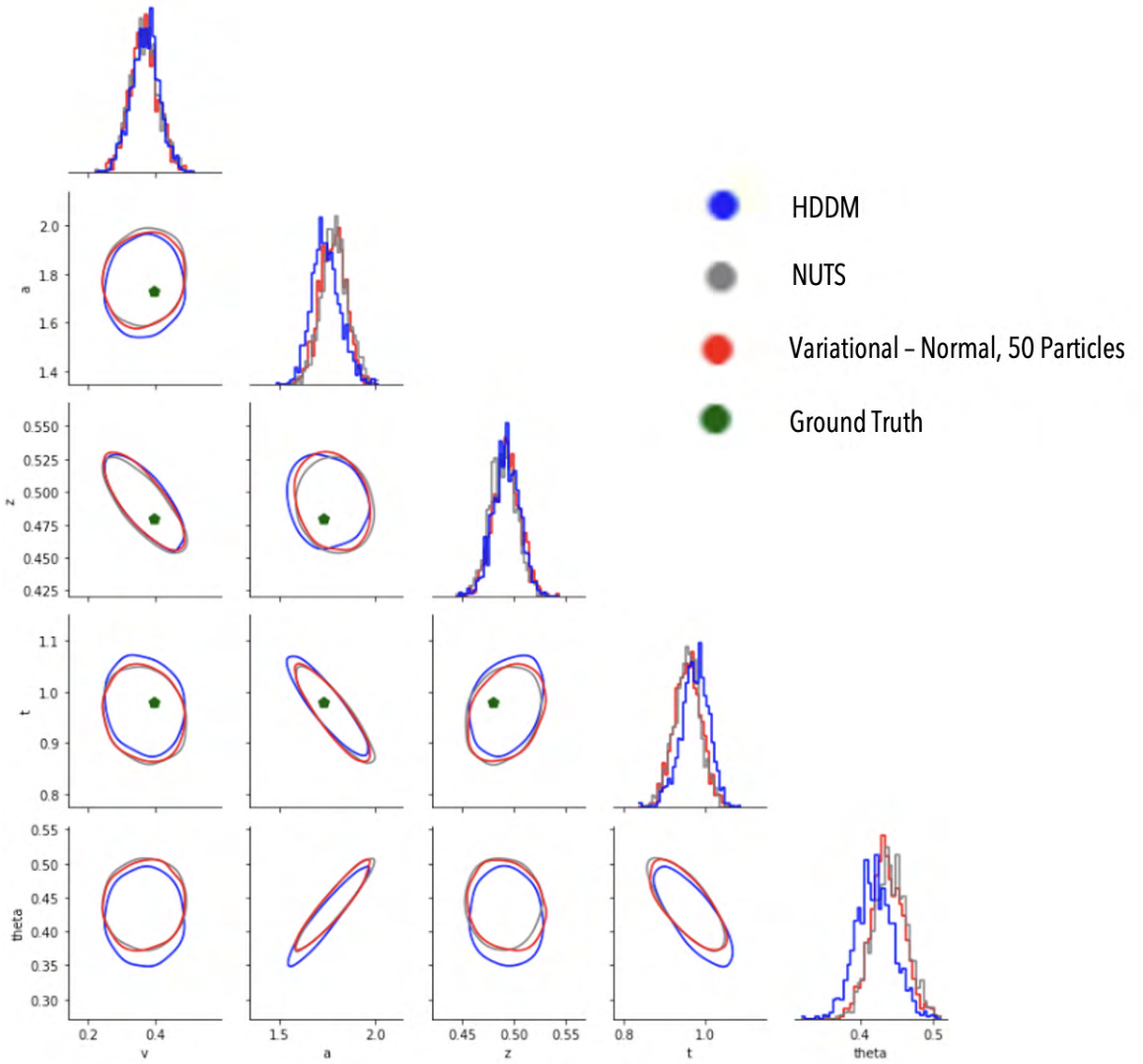

Figure 5.17. Example posterior from Bayesian parameter estimation with the ANGLE model. MVN variational posteriors (Variational - Normal in the Figure, red), track both NUTS (gray) and HDDM (blue) posterior shapes closely, successfully capturing extant posterior correlations. IMVN variational posteriors, as opposed to the equivalent plot in Figure 5.14 for the DDM, are ignored here, since the posterior correlations make IMVN application hopeless from the perspective of capturing posterior uncertainty accurately.

datasets, calibration (Talts et al., 2018) is reported in Figure 5.20.

We can make some general observations. In general, the ANGLE model presents both the NUTS and VI approaches with a number of complications as compared to the DDM. Higher parameter correlations and potentially mutliple posterior models, affect convergence of NUTS (a large fraction of runs, specifically for *hierarchical* data, had to be discarded) and are likely responsible for at least some of the badly recovered parameters across both methods. Furthermore, the multivariate

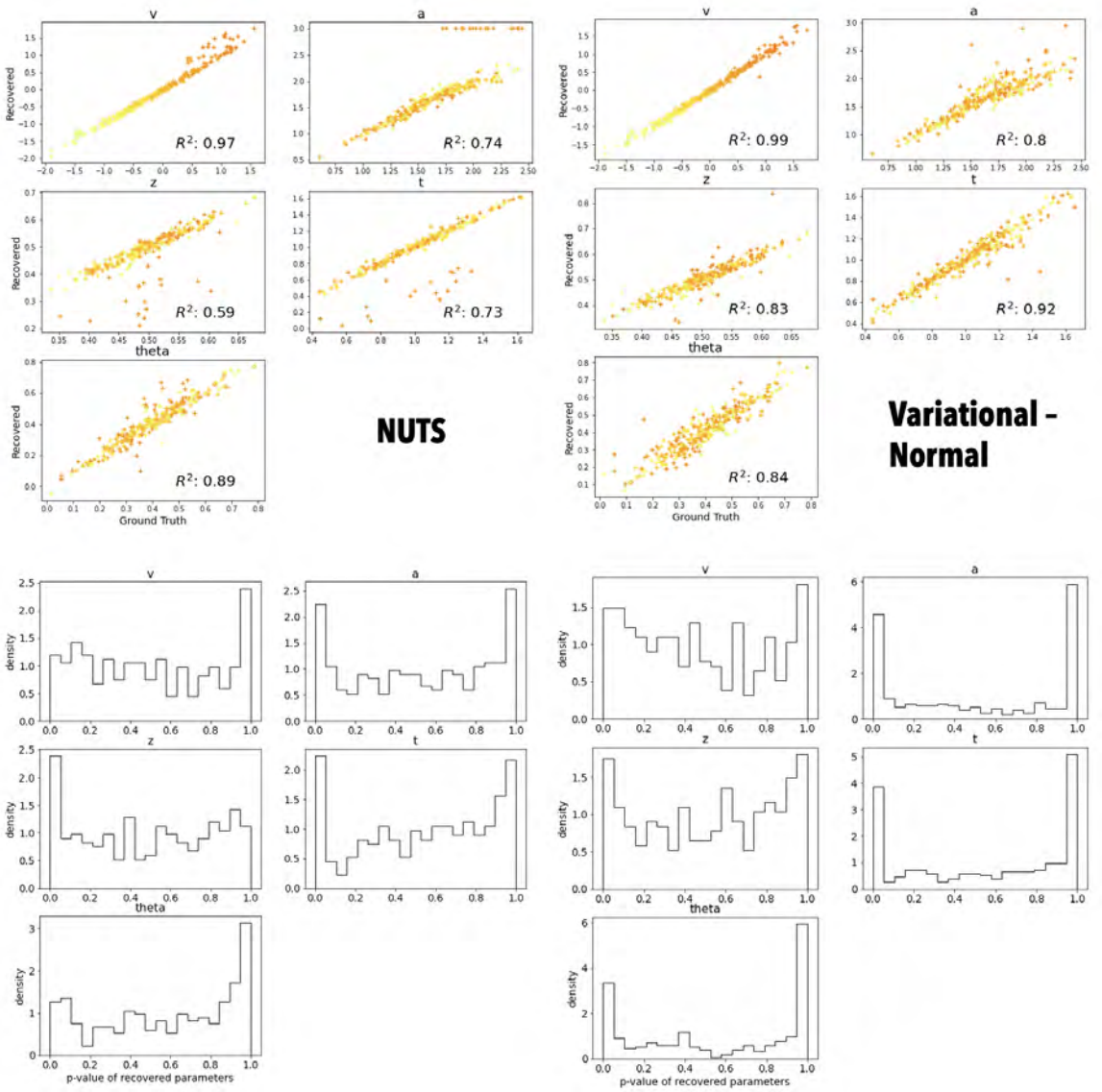

Figure 5.18. Parameter recovery and calibration plots for the ANGLE model using NUTS and VI through MVNs (Variational - Normal in the Figure). NUTS as well as VI suffer from a few outliers (top). Visual inspection of the calibration plots (bottom) suggests NUTS to be somewhat better calibrated than VI with MVNs for the ANGLE model, while mean parameter recovery is slightly superior using VI.

normal assumption, which led to nearly perfect calibration of VI under the DDM, seems to lead to overconfident posteriors using VI for the ANGLE model.

While naive application of the settings chosen for the DDM model did not yield equivalently successful results on the ANGLE model, much room for tractable improvement remains.

First, VI allows the setting of multiple hyperparameters which need investigation. The choice of optimizer, as well as corresponding learning rate settings remains to be explored in more detail. The

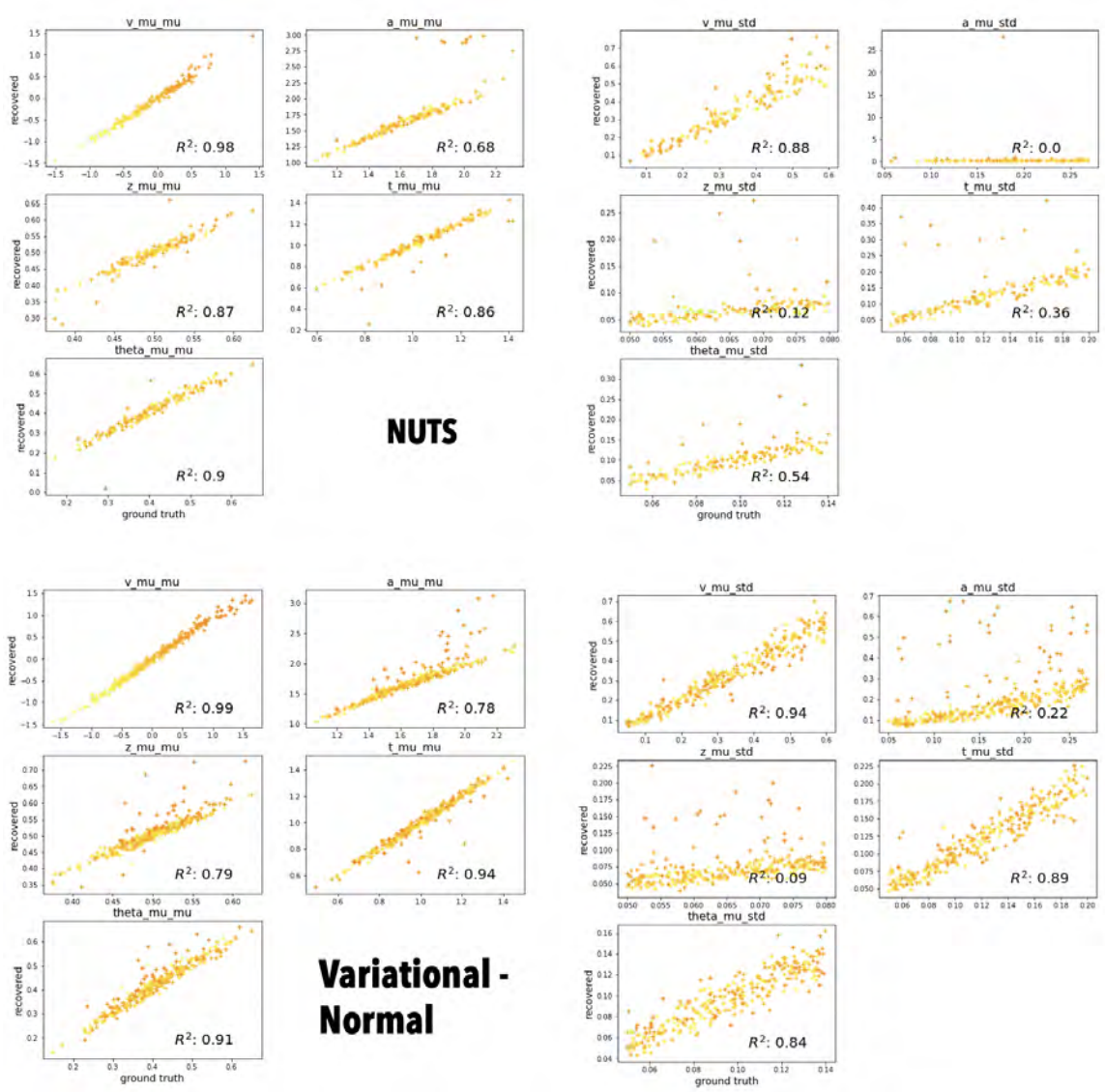

Figure 5.19. Parameter recovery for the ANGLE model using NUTS and VI (MVNs) on *hierarchical* data wit 100 subjects and 100 trials each. Both VI and NUTS show a few outliers, NUTS reporting a large fraction (50%) of failed runs. Estimation of *group standard deviation* (right, _mu_std in the Figure) is less successful than estimation of *group means* (left, _mu_mu in the Figure), with qualitatively consistent patterns across VI and NUTS. VI however is more successfull at estimation of *group standard deviation* parameters overall.

basic VI algorithm may be exchanged (e.g.; (Liu and Wang, 2016)) and a vast array of alternative variational families may still be explored to improve calibration.

Second, as a different layer of hyperparameters, setting slightly informative priors may help regularize both VI and NUTS to escape unfavorable posterior modes. Relatedly, one can include posterior annealing (Geyer and Thompson, 1995) into the warm-up phase for NUTS, to further help avoid the sampler being drawn into inescapable corner regions of the parameter space early on.

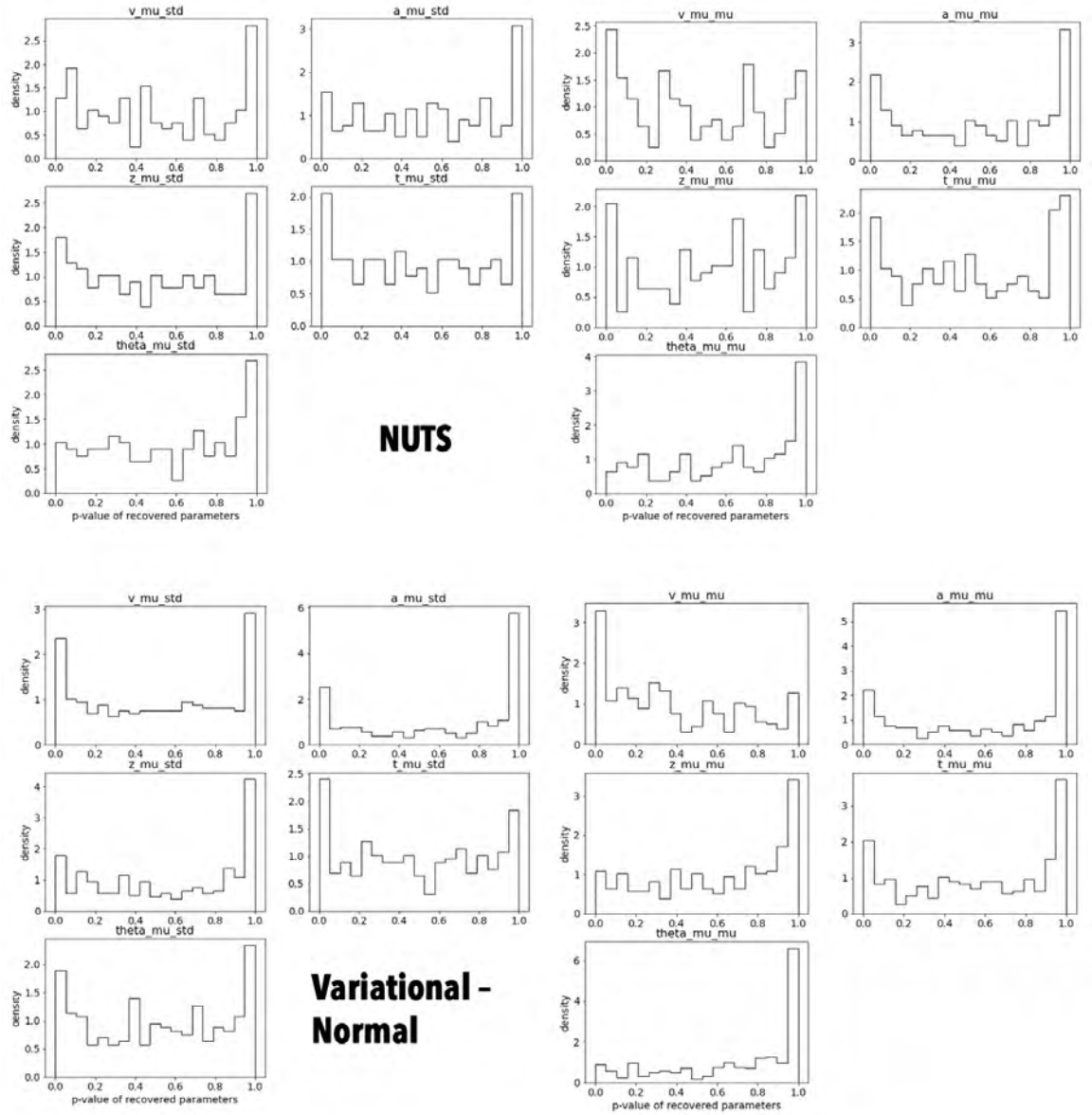

Figure 5.20. The plot shows the calibration performance (Talts et al., 2018) of NUTS and VI on the ANGLE model for *hierarchical* datasets of 100 subjects, 100 trials each. While calibration as reported here suggests NUTS to be superior, note that the results reflect only NUTS runs which resulted in $\hat{R} < 1.01$. Indeed 50% of NUTS runs needed to be deleted for insufficient convergence as per this metric.

Third, (equivalent) reformulations of the generative model via e.g. non-centered parameterizations (Bernardo et al., 2003; Papaspiliopoulos, Roberts, and Sköld, 2007), may additionally help convergence issues of NUTS.

Lastly, the LAN framework is subject to potential for improvement itself. E.g. to help NUTS one avenue is to investigate and improve the regularity of LAN-gradients.

5.4 Discussion

The goal of this chapter was to provide a proof of concept for the usefulness of LANs in conjunction with modern algorithms for (approximate) Bayesian Inference. The landscape of such algorithms is vast (Robert, Casella, and Casella, 1999; Gentle, Härdle, and Mori, 2012; Brooks et al., 2011) and ever expanding (Ter Braak, 2006b; Girolami and Calderhead, 2011; Betancourt and Girolami, 2015; M. Hoffman, Sountsov, et al., 2019; Lao et al., 2020; Lu et al., 2019; M. Hoffman, Radul, and Sountsov, 2021). However, three basic kinds of algorithms can be identified as the main workhorses of applied Bayesian statistics. First are MCMC algorithms which do not make use of likelihood gradients. These numerous algorithms include the traditional Metropolis Random Walk sampler (Metropolis et al., 1953), the Gibbs sampler, (S. Geman and D. Geman, 1984) various combinations thereof, and a large variety of competitors (Ter Braak, 2006b; Haario et al., 2006; Robert, Elvira, et al., 2018). Amongst this array of gradient-free samplers we find the famously robust slice sampler (Neal, 2003), which forms the backbone for posterior sampling in the HDDM software package, the latter serving as the basic comparison benchmark in this work. Second, the workhorse of modern software for Bayesian statistics (Carpenter, Gelman, M. D. Hoffman, Lee, Goodrich, Betancourt, M. A. Brubaker, et al., 2017; Salvatier, Wiecki, and C. Fonnesbeck, 2016a; Phan, Pradhan, and Jankowiak, 2019; Bingham et al., 2019) is the set of gradient-based MCMC samplers. Important examples are samplers based on Langevin dynamics (Rossky, Doll, and Friedman, 1978; Besag and Green, 1993; Mou et al., 2021; Girolami and Calderhead, 2011) and those derived from Hamiltonian systems (Neal et al., 2011; Betancourt and Girolami, 2015; Betancourt, 2017) (see the methods section). For the purposes of this work, I chose NUTS from this class, as implemented by the NumPyro (Phan, Pradhan, and Jankowiak, 2019) python software package, because it forms the backbone of other widely-applied modern inference libraries (Carpenter, Gelman, M. D. Hoffman, Lee, Goodrich, Betancourt, M. A. Brubaker, et al., 2017; Salvatier, Wiecki, and C. Fonnesbeck, 2016a; Phan, Pradhan, and Jankowiak, 2019; Bingham et al., 2019). VI algorithms make up the third branch of modern inference algorithms (Wainwright, Jordan, et al., 2008; Blei, Kucukelbir, and McAuliffe, 2017), representing an optimization-based route towards inference. I used Stochastic Variational Inference (M. D. Hoffman, Blei, et al., 2013; Ranganath, Gerrish, and Blei, 2013; Wingate and Weber, 2013) via the dedicated Pyro (Bingham et al., 2019) python package. VI promises orders of magnitude improvements in inference time over the other classes if applied in the right setting.

LANs are trained to provide good approximations of the trial by trial log-likelihoods for a given SSM and by construction (as Neural Networks) will be differentiable with respect to the input parameters (therefore parameters of the generative model/SSM). But moving from the slice sampler (Neal, 2003) implemented in HDDM Wiecki, Sofer, and Frank, 2013 towards NUTS when basing inference on LANs is not a trivial step. The training loss is independent of the accuracy of those gradients, which may be arbitrarily unstable. NUTS crucially depends on good behavior of these gradients to pass through the implicitly defined Hamiltonian dynamics upon which proposed sampler draws rely. I was therefore skeptical, but cautiously optimistic as to the promise of NUTS for approximate inference using LANs.

The proof of concept reported in the previous sections, using the DDM as an initial test-bed, exceeded my expectations about the potential of the LAN framework for use in modern gradient-based MCMC methods (and therefore being embedded within modern statistical computing ecosystems). I found that NUTS is as well-calibrated as HDDM, for both simple (*single subject*) and more complex datasets/generative models (*hierarchical*), although inheriting slight parameter estimation biases which I believe derive from improvable properties of LANs. Moreover, I provide clear evidence, as illustrated, e.g., in Figure 5.2, that NUTS can lead to massive computing speed improvements at the same time. I thus conclude from these initial findings that further work in this direction is warranted. This may include larger scale numerical experiments as well as further improvements to LANs and similar likelihood approximators (Papamakarios, Sterratt, and Murray, 2019a; Boelts et al., 2022).

Switching from MCMC approaches to VI imposes a different set of trade-offs. The stochastic VI algorithm utilized in the numerical experiments (Wingate and Weber, 2013), like the slice sampler in HDDM (Wiecki, Sofer, and Frank, 2013), demands only that the likelihood can be evaluated. Gradient computations are restricted to the parameters of the variational distributions. The complexity of the class of variational distributions, however, is an important hyperparameter. Complex variational distributions will enable better approximations of a given target distribution, while increasing the instability and computational costs of the optimization process. The experiments explored only a small sliver of the vast search space of possible configurations but nonetheless served two aims. First, they provide a proof of concept that Variational Inference is feasible within the context of approximated likelihoods via LANs. Second, they provide initial evidence regarding the accuracy/speed trade-off across different classes of variational distributions. To illustrate the accuracy/speed trade-off I used two types of variational distributions: IMVNs and MVNs (as described earlier).

Our results show clearly that VI is not only feasible in the context of LANs, but a highly successful method when used within the primary test-bed, the DDM (Ratcliff, 1978; Ratcliff et al., 2016). But I concluded that the mean-field approximation implicit in the IMVN variational distributions is inadequate for subject-wise parameter estimation, a predictable finding given the clear parameter correlations in the posterior exposed via the MCMC methods. IMVN posteriors successfully find the mode of the posterior distribution (and hence provide excellent parameter recovery performance as illustrated in Figures 5.5, 5.7, 5.5), but result in ill-calibrated inference due to their underestimation of posterior uncertainty (see Figure 5.13). On the other hand, I can conclude that MVN posteriors are a good approximation to true posteriors for the DDM and in turn lead to well calibrated inference, offering a performance identical to NUTS and HDDM, as reported in 5.13. I moreover did not observe significant differences in speed of convergence and runtime across both of the tested variational distributions and therefore judge MVNs to be the variational distribution of choice for this type of inference. The fact that VI was shown to be competitive with MCMC in the quality of the posterior approximation is important, not only because it provides an alternative route to posterior inference, but because it also offers two extra benefits. First, stochastic VI does not rely on likelihood gradients and therefore is expected to be slightly more robust than NUTS in the context of LANs (which in their current form do not include any type of gradient regularization and therefore no explicit

guarantees of gradient stability). Second, on top of the heavily optimized implementation of NUTS available in NumPyro (Phan, Pradhan, and Jankowiak, 2019), 4 to 5 fold speed improvements with VI can be reported (see the results section, e.g., Figure 5.2).

Our numerical experiments based on the ANGLE model were somewhat less successful however. Two main problems await resolution. First, the NUTS sampler turned out to be significantly more brittle when estimating *hierarchical* models from respective datasets. Second, VI using MVNs, shown sufficient for calibration in the DDM model, seemed insufficient for good calibration when applied in the context of *hierarchical* ANGLE model estimation (see Figures 5.20, 5.19 and 5.18). As described earlier, I however believe that these problems are addressable and I listed several routes for such investigations which are relayed to future work. My optimism regarding NUTS and VI for LAN-based SBI hence remains unaffected.

Overall, this work provides a promising starting point for the integration of LANs with modern probabilistic programming ecosystems. I showed that NUTS is viable with LAN-based likelihoods, therefore demonstrating that LANs provide not only a good first order approximation to log-likelihoods, but also stable enough gradients for reliable MCMC inference. I moreover showed that stochastic VI can also be performed using LANs. Importantly, I demonstrated how both methods can result in massive speed gains compared to the current HDDM interface for LANs.

5.5 Limitations and Directions for Future Research

The basic goal of this chapter was to advance towards a solid foundation for the development of a larger software toolbox which, backed by modern inference algorithms, provides a flexible framework for conducting Bayesian inference across simulator models. Since LANs (Fengler, Govindarajan, et al., 2021b), and competing methods for likelihood approximation (Boelts et al., 2022; Papamakarios, Sterratt, and Murray, 2019b) do not provide direct guarantees for the quality of likelihood approximations and the stability of their gradients, extensive numerical experiments are necessary to produce a set of building blocks that users can ultimately rely on.

While the numerical experiments strongly suggest that LANs can successfully serve as likelihood approximators for MCMC-based inference with NUTS, as well as basic approaches to stochastic Variational Inference, nevertheless certain limitations remain. I will focus on a few of these weaknesses below.

First, to confirm the runtime benchmarks and fully understand the relative speed of VI compared to MCMC, one should adopt a wider range of stopping metrics (Prechelt, 1998) for the VI algorithms. An attempt was made to provide *adjusted* runtimes for VI, taking into account the number of steps it took to come close to the eventual reported ELBO after a fixed 2000 steps (a number deemed sufficient for convergence after investigating some initial pilot runs). However, a wide range of choice for early stopping metrics are available in the machine learning literature. Building on this work, one should dedicate some experiments to the investigation of stopping rule choices and their effect on the performance of the resulting Variational Inference algorithms.

Second, in this chapter I chose to limit the VI numerical experiments to two numbers of particles: 1 and 10. The number of particles has an effect on the stability of the loss landscape across optimization steps, with higher number of particles, at the potential cost of step-wise runtime, leading to better estimation of the ELBO and resulting gradients, therefore ultimately to a more robust optimization routine. This trade-off warrants more detailed investigation. Moreover, room for code optimization likely remains, since contrary to a priori expectations, the step-wise runtime increased linearly with the number of particles in the numerical experiments. This may hint at sub-optimal batching of the computations. The fundamental trade-off between the number of particles and the number of steps necessary for robust VI remains a valid target for research, regardless of whether the code used for this work may not have optimally utilized computing resources.

Third, the present application of NUTS, especially to the case of hierarchical models, led to a few mishaps, including a large number of divergences and a fraction of cases which failed to converge according to the Gelman-Rubin $\hat{R}$ statistic (Gelman and Rubin, 1992). While on average we can consider the application of NUTS to be a success, further investigations concerning the stability of NUTS in the hierarchical case will be beneficial. Such investigations may involve re-parameterizations (Gelman, 2004; Gelman, Van Dyk, et al., 2008; Betancourt and Girolami, 2015; Bernardo et al., 2003; Papaspiliopoulos, Roberts, and Sköld, 2007), scrutinizing the effects of the warm-up routines of the samplers, settings for initial parameters and robustness to a variety of prior choices and dataset structures (instead of only examining 20 subjects with 500 trials for each).

Fourth, as a VI analogue to point three, a larger variety of VI algorithms and associated dataset structures should be tested. Does VI tend to break down if the parameter space gets inflated (e.g., as a result of including more subjects in the context of hierarchical modeling)? Are there performance trade-offs across VI approaches (Kucukelbir, Ranganath, et al., 2015; Kucukelbir, Tran, et al., 2017; Liu and Wang, 2016)?

Fifth, further benchmarks will be necessary to achieve the vision of general-purpose software. Probabilistic programming backends in this chapter are limited to Pyro (Bingham et al., 2019) for VI, NumPyro (Phan, Pradhan, and Jankowiak, 2019) for MCMC, and HDDM (Wiecki, Sofer, and Frank, 2013) as a basic reference. Two candidates to broaden the benchmark suite should be included in future work: the STAN library for probabilistic programming (Carpenter, Gelman, M. D. Hoffman, Lee, Goodrich, Betancourt, M. Brubaker, et al., 2017) and PyMC (Patil, Huard, and C. J. Fonnesbeck, 2010), both of which are staples of probablistic programming. Moreover, a wider range of likelihood approximators should be evaluated for their relative benefits and shortcomings. In addition to LANs (Fengler, Govindarajan, et al., 2021b), one should consider MNLEs (Boelts et al., 2022) and SNLEs (Papamakarios, Sterratt, and Murray, 2019a) as well as likelihood approximations based on Gaussian Processes (Acerbi, 2020).

The future work summarized in the preceding paragraphs will form a solid foundation from which one can derive broad guidelines for the application of Simulation Based Inference algorithms in computational cognitive and neuroscience. Equally, these efforts should support better choices in the development of software toolboxes designed to aid the research community with parameter inference.

Incorporating novel computational models will ultimately help apply the right tools to the right problems.

References

Acerbi, Luigi (2020). "Variational Bayesian Monte Carlo with Noisy Likelihoods". In: *arXiv preprint arXiv:2006.08655*.

Beaumont, Mark A (2010). "Approximate Bayesian computation in evolution and ecology". In: *Annual review of ecology, evolution, and systematics* 41, pp. 379–406.

Bernardo, JM et al. (2003). "Non-centered parameterisations for hierarchical models and data augmentation". In: *Bayesian Statistics 7: Proceedings of the Seventh Valencia International Meeting*. Vol. 307. Oxford University Press London.

Besag, Julian (1994). "Comments on "Representations of knowledge in complex systems" by U. Grenander and MI Miller". In: *J. Roy. Statist. Soc. Ser. B* 56.591-592, p. 4.

Besag, Julian and Peter J Green (1993). "Spatial statistics and Bayesian computation". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 55.1, pp. 25–37.

Betancourt, Michael (2013). "A general metric for Riemannian manifold Hamiltonian Monte Carlo". In: *International Conference on Geometric Science of Information*. Springer, pp. 327–334.

— (2017). *A Conceptual Introduction to Hamiltonian Monte Carlo*. DOI: 10.48550/ARXIV.1701.02434. URL: https://arxiv.org/abs/1701.02434.

Betancourt, Michael and Mark Girolami (2015). "Hamiltonian Monte Carlo for hierarchical models". In: *Current trends in Bayesian methodology with applications* 79.30, pp. 2–4.

Bingham, Eli et al. (2019). "Pyro: Deep universal probabilistic programming". In: *The Journal of Machine Learning Research* 20.1, pp. 973–978.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). "Variational inference: A review for statisticians". In: *Journal of the American statistical Association* 112.518, pp. 859–877.

Boelts, Jan et al. (2022). "Flexible and efficient simulation-based inference for models of decision-making". In: *Elife* 11, e77220.

Brooks, Steve et al. (2011). *Handbook of markov chain monte carlo*. CRC press.

Carpenter, Bob, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, et al. (2017). "Stan: A probabilistic programming language". In: *Journal of statistical software* 76.1.

Carpenter, Bob, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus A Brubaker, et al. (2017). "Stan: a probabilistic programming language." In: *Grantee Submission* 76.1, pp. 1–32.

Cavanagh, James F and Michael J Frank (2014). "Frontal theta as a mechanism for cognitive control". In: *Trends in cognitive sciences* 18.8, pp. 414–421.

Cavanagh, James F, Thomas V Wiecki, et al. (2011). "Subthalamic nucleus stimulation reverses mediofrontal influence over decision threshold". In: *Nature neuroscience* 14.11, pp. 1462–1467.

Cisek, Paul, Geneviève Aude Puskas, and Stephany El-Murr (2009a). "Decisions in changing conditions: the urgency-gating model". In: *Journal of Neuroscience* 29.37, pp. 11560–11571.

— (2009b). "Decisions in changing conditions: the urgency-gating model". In: *Journal of Neuroscience* 29.37, pp. 11560–11571.

Cover, Thomas M and Joy A Thomas (2006). "Elements of information theory 2nd edition (wiley series in telecommunications and signal processing)". In: *Acessado em*.

Diaconis, Persi (2009). "The markov chain monte carlo revolution". In: *Bulletin of the American Mathematical Society* 46.2, pp. 179–205.

Doersch, Carl (2016). "Tutorial on variational autoencoders". In: *arXiv preprint arXiv:1606.05908*.

Doi, Takahiro et al. (2020). "The caudate nucleus contributes causally to decisions that balance reward and uncertain visual information". In: *ELife* 9, e56694.

Fengler, Alexander, Krishn Bera, et al. (2022). "Beyond Drift Diffusion Models: Fitting a Broad Class of Decision and Reinforcement Learning Models with HDDM". In: *Journal of Cognitive Neuroscience* 34.10, pp. 1780–1805.

Fengler, Alexander, Lakshmi N Govindarajan, et al. (2021a). "Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience". In: *eLife*. DOI: 10.7554/eLife.65074. eprint: https://elifesciences.org/articles/65074.pdf. URL: https://elifesciences.org/articles/65074.

— (2021b). "Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience". In: *Elife* 10, e65074.

Forstmann, Birte U, Alfred Anwander, et al. (2010). "Cortico-striatal connections predict control over speed and accuracy in perceptual decision making". In: *Proceedings of the National Academy of Sciences* 107.36, pp. 15916–15920.

Forstmann, Birte U, Roger Ratcliff, and E-J Wagenmakers (2016). "Sequential sampling models in cognitive neuroscience: Advantages, applications, and extensions". In: *Annual review of psychology* 67, p. 641.

Foster, Kendal and Henrik Singmann (2021). *Another Approximation of the First-Passage Time Densities for the Ratcliff Diffusion Decision Model*. arXiv: 2104.01902 [stat.AP].

Frank, Michael J et al. (2015). "fMRI and EEG predictors of dynamic decision parameters during human reinforcement learning". In: *Journal of Neuroscience* 35.2, pp. 485–494.

Frostig, Roy, Matthew James Johnson, and Chris Leary (2018). "Compiling machine learning programs via high-level tracing". In: *Systems for Machine Learning* 4.9.

Gelman, Andrew (2004). "Parameterization and Bayesian modeling". In: *Journal of the American Statistical Association* 99.466, pp. 537–545.

Gelman, Andrew, John B Carlin, et al. (1995). *Bayesian data analysis*. Chapman and Hall/CRC.

Gelman, Andrew and Donald B Rubin (1992). "Inference from iterative simulation using multiple sequences". In: *Statistical science*, pp. 457–472.

Gelman, Andrew, Donald B Rubin, et al. (1992). "Inference from iterative simulation using multiple sequences". In: *Statistical science* 7.4, pp. 457–472.

Gelman, Andrew, David A Van Dyk, et al. (2008). "Using redundant parameterizations to fit hierarchical models". In: *Journal of Computational and Graphical Statistics* 17.1, pp. 95–122.

Geman, Stuart and Donald Geman (1984). "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images". In: *IEEE Transactions on pattern analysis and machine intelligence* 6, pp. 721–741.

Gentle, James E, Wolfgang K Härdle, and Yuichi Mori (2012). "Springer Handbooks of Computational Statistics". In.

Geweke, John (1992). "Evaluating the accuracy of sampling-based approaches to the calculations of posterior moments". In: *Bayesian statistics* 4, pp. 641–649.

Geyer, Charles J and Elizabeth A Thompson (1995). "Annealing Markov chain Monte Carlo with applications to ancestral inference". In: *Journal of the American Statistical Association* 90.431, pp. 909–920.

Girolami, Mark and Ben Calderhead (2011). "Riemann manifold langevin and hamiltonian monte carlo methods". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.2, pp. 123–214.

Green, Peter J (2003). "Trans-dimensional markov chain monte carlo". In: *Oxford Statistical Science Series*, pp. 179–198.

Green, Peter J and David I Hastie (2009). "Reversible jump MCMC". In: *Genetics* 155.3, pp. 1391–1403.

Greenberg, David, Marcel Nonnenmacher, and Jakob Macke (2019). "Automatic posterior transformation for likelihood-free inference". In: *International Conference on Machine Learning*. PMLR, pp. 2404–2414.

Haario, Heikki et al. (2006). "DRAM: efficient adaptive MCMC". In: *Statistics and computing* 16.4, pp. 339–354.

Hastings, W Keith (1970). "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57 (1).

Herz, Damian M et al. (2016). "Neural correlates of decision thresholds in the human subthalamic nucleus". In: *Current Biology* 26.7, pp. 916–920.

Higdon, David M (1998). "Auxiliary variable methods for Markov chain Monte Carlo with applications". In: *Journal of the American statistical Association* 93.442, pp. 585–595.

Hoffman, Matthew, Alexey Radul, and Pavel Sountsov (2021). "An adaptive-mcmc scheme for setting trajectory lengths in hamiltonian monte carlo". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3907–3915.

Hoffman, Matthew, Pavel Sountsov, et al. (2019). *NeuTra-lizing Bad Geometry in Hamiltonian Monte Carlo Using Neural Transport*. DOI: 10.48550/ARXIV.1903.03704. URL: https://arxiv.org/abs/1903.03704.

Hoffman, Matthew D, David M Blei, et al. (2013). "Stochastic variational inference". In: *The Journal of Machine Learning Research* 14.1, pp. 1303–1347.

Hoffman, Matthew D and Andrew Gelman (2014). "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623.

Holmes, William R, Jennifer S Trueblood, and Andrew Heathcote (2016). "A new framework for modeling decisions about changing information: The Piecewise Linear Ballistic Accumulator model". In: *Cognitive psychology* 85, pp. 1–29.

Järvenpää, Marko et al. (2018). "Gaussian process modelling in approximate Bayesian computation to estimate horizontal gene transfer in bacteria". In: *Annals of Applied Statistics* 12.4, pp. 2228–2251.

— (2021). "Parallel Gaussian process surrogate Bayesian inference with noisy likelihood evaluations". In: *Bayesian Analysis* 16.1, pp. 147–178.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kingma, Diederik P, Tim Salimans, and Max Welling (2015). "Variational dropout and the local reparameterization trick". In: *arXiv preprint arXiv:1506.02557*.

Kingma, Diederik P, Max Welling, et al. (2019). "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4, pp. 307–392.

Krajbich, Ian et al. (2012). "The attentional drift-diffusion model extends to simple purchasing decisions". In: *Frontiers in psychology* 3, p. 193.

Kucukelbir, Alp, Rajesh Ranganath, et al. (2015). "Automatic variational inference in Stan". In: *Advances in neural information processing systems* 28.

Kucukelbir, Alp, Dustin Tran, et al. (2017). "Automatic differentiation variational inference". In: *Journal of machine learning research*.

Lao, Junpeng et al. (2020). "tfp. mcmc: Modern Markov chain Monte Carlo tools built for modern hardware". In: *arXiv preprint arXiv:2002.01184*.

Li, Yingzhen and Richard E Turner (2016). "Rényi divergence variational inference". In: *Advances in neural information processing systems* 29.

Liu, Qiang and Dilin Wang (2016). "Stein variational gradient descent: A general purpose bayesian inference algorithm". In: *Advances in neural information processing systems* 29.

Lotka, Alfred James (1925). *Elements of physical biology*. Williams & Wilkins.

Lu, Han et al. (2019). "Parallel multiple-chain DRAM MCMC for large-scale geosteering inversion and uncertainty quantification". In: *Journal of Petroleum Science and Engineering* 174, pp. 189–200.

Lueckmann, Jan-Matthis et al. (2019). "Likelihood-free inference with emulator networks". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 32–53.

MacKay, David JC, David JC Mac Kay, et al. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Meeds, Edward and Max Welling (2014). "GPS-ABC: Gaussian process surrogate approximate Bayesian computation". In: *arXiv preprint arXiv:1401.2838*.

Metropolis, Nicholas et al. (1953). "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6, pp. 1087–1092.

Mou, Wenlong et al. (2021). "High-Order Langevin Diffusion Yields an Accelerated MCMC Algorithm." In: *J. Mach. Learn. Res.* 22, pp. 42–1.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. MIT press.

Navarro, Daniel J and Ian G Fuss (2009). "Fast and accurate calculations for first-passage times in Wiener diffusion models". In: *Journal of mathematical psychology* 53.4, pp. 222–230.

Neal, Radford M (2003). "Slice sampling". In: *Annals of statistics*, pp. 705–741.

Neal, Radford M et al. (2011). "MCMC using Hamiltonian dynamics". In: *Handbook of markov chain monte carlo* 2.11, p. 2.

Nishio, Motohide and Aisaku Arakawa (2019). "Performance of Hamiltonian Monte Carlo and No-U-Turn Sampler for estimating genetic parameters and breeding values". In: *Genetics Selection Evolution* 51.1, pp. 1–12.

Papamakarios, George and Iain Murray (2016). "Fast ε-free Inference of Simulation Models with Bayesian Conditional Density Estimation". In: *Advances in Neural Information Processing Systems* 29, pp. 1028–1036.

Papamakarios, George, Theo Pavlakou, and Iain Murray (2017). "Masked autoregressive flow for density estimation". In: *Advances in Neural Information Processing Systems*, pp. 2338–2347.

Papamakarios, George, David Sterratt, and Iain Murray (2019a). "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 837–848.

— (2019b). "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 837–848.

Papaspiliopoulos, Omiros, Gareth O Roberts, and Martin Sköld (2007). "A general framework for the parametrization of hierarchical models". In: *Statistical Science*, pp. 59–73.

Paszke, Adam et al. (2019). "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems*, pp. 8026–8037.

Patil, Anand, David Huard, and Christopher J Fonnesbeck (2010). "PyMC: Bayesian stochastic modelling in Python". In: *Journal of statistical software* 35.4, p. 1. DOI: 10.18637/jss.v035.i04.

Pedersen, Mads L and Michael J Frank (2020). "Simultaneous Hierarchical Bayesian Parameter Estimation for Reinforcement Learning and Drift Diffusion Models: a Tutorial and Links to Neural Data". In: *Computational Brain & Behavior* 3, pp. 458–471.

Pedersen, Mads Lund, Michael J Frank, and Guido Biele (2017). "The drift diffusion model as the choice rule in reinforcement learning". In: *Psychonomic bulletin & review* 24.4, pp. 1234–1251.

Phan, Du, Neeraj Pradhan, and Martin Jankowiak (2019). "Composable effects for flexible and accelerated probabilistic programming in NumPyro". In: *arXiv preprint arXiv:1912.11554*.

Prechelt, Lutz (1998). "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, pp. 55–69.

Pritchard, Jonathan K et al. (1999). "Population growth of human Y chromosomes: a study of Y chromosome microsatellites." In: *Molecular biology and evolution* 16.12, pp. 1791–1798.

Radev, Stefan T et al. (2020). "BayesFlow: Learning complex stochastic models with invertible neural networks". In: *arXiv preprint arXiv:2003.06281*.

Ranganath, Rajesh, Sean Gerrish, and David M Blei (2013). "Black box variational inference". In: *arXiv preprint arXiv:1401.0118*.

Rangel, Antonio, Colin Camerer, and P Read Montague (2008). "A framework for studying the neurobiology of value-based decision making". In: *Nature reviews neuroscience* 9.7, pp. 545–556.

Ratcliff, Roger (1978). "A theory of memory retrieval." In: *Psychological review* 85.2, p. 59.

Ratcliff, Roger et al. (2016). "Diffusion decision model: Current issues and history". In: *Trends in cognitive sciences* 20.4, pp. 260–281.

Rezende, Danilo Jimenez and Shakir Mohamed (2015). "Variational inference with normalizing flows". In: *arXiv preprint arXiv:1505.05770*.

Rezende, Danilo Jimenez, George Papamakarios, et al. (2020). "Normalizing flows on tori and spheres". In: *arXiv preprint arXiv:2002.02428*.

Robbins, Herbert and Sutton Monro (1951). "A stochastic approximation method". In: *The annals of mathematical statistics*, pp. 400–407.

Robert, Christian P, George Casella, and George Casella (1999). *Monte Carlo statistical methods*. Vol. 2. Springer.

Robert, Christian P, Vıctor Elvira, et al. (2018). "Accelerating MCMC algorithms". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 10.5, e1435.

Rossky, Peter J, Jimmie D Doll, and Harold L Friedman (1978). "Brownian dynamics as smart Monte Carlo simulation". In: *The Journal of Chemical Physics* 69.10, pp. 4628–4633.

Salvatier, John, Thomas V Wiecki, and Christopher Fonnesbeck (2016a). "Probabilistic programming in Python using PyMC3". In: *PeerJ Computer Science* 2, e55.

— (2016b). "Probabilistic programming in Python using PyMC3". In: *PeerJ Computer Science* 2, e55.

Sisson, Scott A, Yanan Fan, and Mark Beaumont (2018). *Handbook of approximate Bayesian computation*. CRC Press.

Talts, Sean et al. (2018). "Validating Bayesian inference algorithms with simulation-based calibration". In: *arXiv preprint arXiv:1804.06788*.

Tavaré, Simon et al. (1997). "Inferring coalescence times from DNA sequence data". In: *Genetics* 145.2, pp. 505–518.

Ter Braak, Cajo JF (2006a). "A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces". In: *Statistics and Computing* 16.3, pp. 239–249.

— (2006b). "A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces". In: *Statistics and Computing* 16.3, pp. 239–249.

Usher, Marius and James L McClelland (2001). "The time course of perceptual choice: the leaky, competing accumulator model." In: *Psychological review* 108.3, p. 550.

Vehtari, Aki et al. (2021). "Rank-normalization, folding, and localization: an improved R for assessing convergence of MCMC (with discussion)". In: *Bayesian analysis* 16.2, pp. 667–718.

Vrugt, JA et al. (2009). "Accelerating Markov chain Monte Carlo simulation by differential evolution with self-adaptive randomized subspace sampling". In: *International Journal of Nonlinear Sciences and Numerical Simulation* 10.3, pp. 273–290.

Wainwright, Martin J, Michael I Jordan, et al. (2008). "Graphical models, exponential families, and variational inference". In: *Foundations and Trends® in Machine Learning* 1.1–2, pp. 1–305.

Wiecki, Thomas V, Imri Sofer, and Michael J Frank (2013). "HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python". In: *Frontiers in neuroinformatics* 7, p. 14.

Wieschen, Eva Marie, Andreas Voss, and Stefan Radev (2020). "Jumping to conclusion? a lévy flight model of decision making". In: *TQMP* 16.2, pp. 120–132.

Wilkinson, Richard (2014). "Accelerating ABC methods using Gaussian processes". In: *Artificial Intelligence and Statistics*. PMLR, pp. 1015–1023.

Wingate, David and Theophane Weber (2013). "Automated variational inference in probabilistic programming". In: *arXiv preprint arXiv:1301.1299*.

Wood, Simon N (2010). "Statistical inference for noisy nonlinear ecological dynamic systems". In: *Nature* 466.7310, pp. 1102–1104.

Yartsev, Michael M et al. (2018). "Causal contribution and dynamical encoding in the striatum during evidence accumulation". In: *Elife* 7, e34929.

Chapter 6

General Discussion

Simulation Based Inference is quickly progressing from the hidden realms of specialist niche methodologies to being part of the repertoire in computationally minded experimental laboratories across cognitive science and neuroscience (but also other disciplines; Wood, 2010; Alsing et al., 2019; Cole et al., 2022). The great promise of these SBI methods is to bridge the distance between experts in computational methods and experimentalists interested in computational methods who lack specific expertise in mathematical statistics and related fields.

Historically, stochastic mechanistic models of brain and behavior have their beginnings as simulators that can serve as data generators (or forward models), and which may be rigorously defined in mathematical terms. However, the derivation of easy-to-compute analytical likelihood functions, if a tractable problem at all, can lag years (sometimes decades) behind the availability of such simulators, in part because the mathematical expertise needed for such derivations is distinct from the expertise necessary to propose new models or slight variations of existing ones. The implication is that experimentalists are impeded from advancing and testing novel models unless they have access to experts in mathematical statistics.

The DDM serves as a good example. Introduced as a cognitive process model by (Ratcliff, 1978; Ratcliff et al., 2016), it was not until the late 2000s when (Navarro and Fuss, 2009) developed a fast algorithm to compute semi-analytical likelihoods via intelligent series truncation. This is despite the fact that the infinite series representation of the so-called first passage distributions of a Wiener process was known at least since W. and V., 1968). Only after Navarro's work did software packages like HDDM become possible: These cheaper-to-compute likelihoods rendered hierarchical Bayesian inference tractable.

Modern SBI methods (Gutmann, Dutta, et al., 2018; Gutmann and Corander, 2016; Alsing et al., 2019; Lueckmann, Goncalves, et al., 2017; Lueckmann, Bassetto, et al., 2019; Greenberg, Nonnenmacher, and Macke, 2019; Boelts et al., 2022; Fengler, Govindarajan, et al., 2021; Papamakarios and Murray, 2016; Papamakarios, Nalisnick, et al., 2019; Papamakarios, Sterratt, and Murray, 2019a; Durkan, Murray, and Papamakarios, 2020; Durkan, Bekasov, et al., 2019; Radev et al., 2020) have brought sophisticated statistical machinery to generative models which, as described above, previously only afforded access to rudimentary statistical approaches. Emerging software toolboxes (Tejero-Cantero et al., 2020; Fengler, Bera, et al., 2022) substantially reduce the entry barriers for usage. Hence, researchers may now perform computationally tractable statistical inference for a broad array of models without requiring laborious effort from mathematical experts. While analytical derivations may further improve the speed of inference and remain an important avenue for research in the computational modeling sphere, Simulation Based Inference promises to grant modelers and experimentalists a much greater degree of statistical autonomy at the moment of model conception. The mathematician may still improve computational work via analytical approaches, but no strict dependency between bayesian inference and such analytical leg-work remains.

A significant increase in research activity concerning SBI methods is currently underway. Indeed, as this thesis was being written, new alternative approaches emerged (Boelts et al., 2022; Acerbi, 2020; Brehmer et al., 2020; Radev et al., 2020; Cole et al., 2022; Thomas et al., 2022). However,

while the current zeitgeist slowly starts to question the continued validity of Moore's Law, the no free lunch theorem (Wolpert, 1996), unperturbed, applies again. No single approach to SBI dominates all others for every application. Researchers still need to investigate the full space of available methods in order to fruitfully optimize their models.

To help navigate this maze of candidate algorithms, chapter 2 provided a history of SBI methods, tailored towards a detailed differentiation of paradigms and corresponding recent Neural Network-based approaches.

I distinguished between three overarching collections of methods, which were developed somewhat sequentially. From the initial ABC algorithms based on rejection sampling and their descendents (LFI-ABC) (Tavaré et al., 1997; Pritchard et al., 1999; M. A. Beaumont, Zhang, and Balding, 2002; M. A. Beaumont, 2010; Sisson, Fan, and M. Beaumont, 2018), to the application of more traditional machine learning paradigms such as Gaussian Processes and Logistic Regression (LFI-ML) (Acerbi, 2020; Sisson, Fan, and M. Beaumont, 2018), towards the plethora of recent methods which fully embrace a Neural Network-based learning approach to SBI (LFI-NN) (Papamakarios and Murray, 2016; Papamakarios, Pavlakou, and Murray, 2017; Papamakarios, Sterratt, and Murray, 2019a; Lueckmann, Bassetto, et al., 2019; Greenberg, Nonnenmacher, and Macke, 2019; Radev et al., 2020; Fengler, Govindarajan, et al., 2021; Boelts et al., 2022). I then discussed the main classification of these algorithms along two axes. First, which aspect of inference does the Neural Network target (likelihood functions or posteriors, local or global) and second, which class of learning problem and associated architecture is used (e.g., density estimators like flows, ratio estimators or is the setup a basic regression?). I put in focus the relationship between these algorithms, strengths and weaknesses and potential applications tailored towards cognitive process models.

Emphasis was placed on the relative strengths and weaknesses of posterior- versus likelihood-targeting algorithms. Algorithms that target the posterior directly (Greenberg, Nonnenmacher, and Macke, 2019; Papamakarios and Murray, 2016; Papamakarios, Sterratt, and Murray, 2019b; Radev et al., 2020) can provide instant posterior sampling on new data, but are specialized to single inference scenario. Global amortization of posteriors is useful to investigate structural parameter trade-offs in mechanistic models, and for data analysis in simple experimental setups or models with few parameters (e.g., parameter inference for single subject data, potentially with a few conditions). Networks, however, need to be retrained when any assumption in the model is changed (for instance, allowing parameter θ_1 to vary across experimental conditions but not parameter θ_{-1}), requiring the initial amortization costs again. No obvious application of transfer learning may ameliorate those retraining costs either. This is because the resulting networks across inference scenarios usually define distributions over output space of different dimensions for which network architectures cannot be directly recycled. Hence, while extremely effective in the domain to which they are tailored, the application of posterior-targeting methods to complex empirical data, e.g., from laboratory experiments in psychology or the cognitive sciences, is hampered severely.

In contrast, likelihood-targeting methods (Lueckmann, Bassetto, et al., 2019; Papamakarios, Sterratt, and Murray, 2019a; Fengler, Govindarajan, et al., 2021; Boelts et al., 2022) do not afford

instant posterior inference but provide us with encapsulated approximate likelihoods which can either work on summary statistics of data-sets (Lueckmann, Bassetto, et al., 2019; Papamakarios, Sterratt, and Murray, 2019a) or in the last consequence be defined trial-wise (Fengler, Govindarajan, et al., 2021; Boelts et al., 2022). Once given access to trial-wise likelihoods, these can a posteriori be reused for any inference scenario, including arbitrary numbers of subjects, trials, and theoretical assumptions on the structure of parameters like hierarchies. I highlight how this approach is key for widespread reuse of computational models amortized once throughout a research community, even if the original training is costly. Hence, likelihood targeting SBI methods decouple the considerations of experimental design from the SBI algorithm itself, a major barrier for efficient reuse across laboratories, datasets and experiments.

Motivated by the potential of likelihood-targeting approaches, my own contribution to this collection of algorithms is described in chapter 3 (Fengler, Govindarajan, et al., 2021). chapter 3 describes Likelihood Approximation Networks (LANs), which formulate the task of learning log-likelihoods as a regression problem and successfully apply simple Multi-Layered Perceptrons (MLPs) as well as Convolutional Neural Networks (CNNs) for speedy inference in data analysis problems. Applications to Sequential Sampling Models (SSMs) are highlighted and I showed, especially on GPU hardware, the speed of inference for simple DDMs is comparable to the fastest semi-analytical approaches that are based on truncated series (Navarro and Fuss, 2009; Foster and Singmann, 2021; W. and V., 1968).

LANs can therefore be used to test empirical data (old and new) against a much larger variety of cognitive process models, where access to approximate likelihoods naturally facilitates the computation of common model comparison metrics such as the DIC (Spiegelhalter et al., 2014) or WAIC (Watanabe, 2013). I contend that this helps facilitate the scientific process, which in the computational sciences can be characterized as an evolutionary path through candidate models, with model comparison metrics and statistical methods defining the governing evolutionary principles. LANs therefore serve as a stepping stone towards freeing experimentalist's modeling choices from the dictates of analytical convenience. The latter is epitomized by the ubiquitous application of the basic DDM, despite the exitence of equally (if not more) applicable, but likelihood-free, versions of sequential sampling models.

Working towards the realization of this promise, at first in the domain of sequential sampling models, chapter 4 discussed an extension to the HDDM python software package (Wiecki, Sofer, and Frank, 2013; Fengler, Bera, et al., 2022). The basic version of HDDM facilitates parameter estimation for a flexible class of hierarchical Bayesian models based on DDM likelihoods. The HDDM-LAN extensions, now an integral part of the HDDM package, provides access to LANs for a bank of SSMs, therefore allowing users with basic familiarity of HDDM to utilize a much larger selection of likelihood models. Since LANs provide trial-wise likelihoods, any trial-wise process driving model parameters, such as a reinforcement learning agent operating across trials, can be incorporated. As explained in chapter 4, I included a basic set of reinforcement learning algorithms that allows the specification of a variety of RL-SSM models (precedence for the use of this label, albeit without a SBI

connotation, can be found in another recent software package of the same name; Fontanesi, 2022).It's name notwithstanding, the HDDM-LAN extension is conceptualized a priori to not commit solely to LANs as the underlying likelihood approximator, but rather to enable the use of any likelihood approximator and allow inference for user-defined custom models. The spirit here is to provide with HDDM a convenient scaffolding for complex Bayesian hierarchical models, while increasing the user's flexibility at the level of likelihood functions.

LANs allow fast inference: three orders of magnitude faster than methods demanding simulations during inference (Turner and Sederberg, 2014; Turner and Van Zandt, 2012; Turner and Van Zandt, 2018) and one order of magnitude faster than even directly comparable recent flow-based likelihood approximators (Boelts et al., 2022). Nonetheless, inference for large, complex data-sets can still require hours and sometimes days.

In chapter chapter 5 I showcase two avenues towards improving this situation. First, I make use of LAN gradients with respect to their inputs (model parameters), available by construction. The tests in chapter 3 were based on slice sampling (Neal, 2003), the underlying MCMC algorithm of the HDDM package, which does not take advantage of likelihood gradients and instead only relies on repeated likelihood evaluations. While it was by no means guaranteed that the gradients of LANs would be stable enough to support modern gradient-based MCMC methods such as the NUTS algorithm (Hoffman and Gelman, 2014), I provide a proof of concept regarding the feasibility of this application and showcase the potential for significant inference speed gains. Specifically for complex datasets that demand hierarchical modeling, I observe an order of magnitude (or more) increase in the number of *effective samples per second* (Gentle, Härdle, and Mori, 2012).

As a second avenue of exploration I considered Variational Inference on the basis of LANs (Blei, Kucukelbir, and McAuliffe, 2017; Ranganath, Gerrish, and Blei, 2013; Wingate and Weber, 2013), for which again, the work in chapter 5 constitutes a proof of concept. Variational Inference adds another layer of approximation to inference, since the method treats posterior inference as an optimization problem on an a-priori parameterized distribution. Nonetheless, the observed speed improvements, using the DDM model as an initial test-bed, suggest that the method can be useful in practice. The approach tends to outperform even NUTS by roughly 5 times, consequently outpacing inference with HDDM at least 50 fold on hierarchical inference problems. I moreover found that the multivariate Normal variational family is sufficient to generate well-calibrated (Talts et al., 2018) posteriors (see chapter 5).

I hope that the work presented in this thesis will serve as a stepping stone towards better SBI methods, ones that even are faster, more elegant and more data efficient. Recent advances inspired by LANs already point in this direction (Boelts et al., 2022), and I plan to continue my research based on the foundation laid by LANs. I consider multiple pathways for future research worth exploring, marked also in the respective sections within chapter 2, chapter 3, chapter 4 and chapter 5. First, a direct improvement to LANs themselves. Small tweaks to the definition of the empirical likelihoods (KDEs) may allow us to overcome the slight estimation biases of LANs with regards to the non-decision time and boundary parameters, pointed out in subsequent research advances (Boelts

et al., 2022). Moreover, a fusion between elegant flow-based likelihood approximators and MLPs may prove fruitful. Flows tend to be slower than MLPs for likelihood evaluation but allow training directly on simulator output without a detour via empirical likelihood construction. They moreover enable fast sampling from an implicitly-trained surrogate simulator. One may first train a flow, then distill log-likelihood evaluations into an MLP to harmonize the LAN and MNLE paradigms and end with the best of both worlds via synthesis. Similarly, it may be fruitful to distill pre-trained LANs themselves into minimal architectures that maintain approximation performance, thereby maximizing inference speed even further.

My experiments with modern samplers and Variational Inference also encouraged the development of a new HDDM, with HSSM as a working title, modernizing the basic infrastructure of HDDM towards newer probabilistic programming backends. This will improve sampling performance and tighten the integration of likelihood-targeting SBI methods with modern workflows for Bayesian statistical analysis (Gelman et al., 2020).

I moreover see great potential in the development of infrastructure which can allow researchers to create and share their own likelihood approximators as well as integrate them into toolboxes that facilitate inference on empirically relevant data. The HDDM extension described in chapter 4 is a first step in this direction and allows users to access a common bank of models enabled by pretrained LANs as well as utilize novel likelihood functions via a *custom models* interface (as explained in chapter 4.

However, more formal modes of sharing likelihood approximators will be helpful in normalizing such workflows. Partially facilitating this, my current work strives to develop user-friendly pipelines for the training of LANs to help users fully bridge the gap between access to simulators and statistical inference. The initial work can be found under `https://github.com/AlexanderFengler/LANfactory`. The existence of some such pipelines (Radev et al., 2020; Tejero-Cantero et al., 2020), with emphasis on posterior-targeting SBI algorithms have already had a significant impact on adoption of said methods and I firmly believe that, specifically in the cognitive and behavioral sciences, similar if not more success in this regard is possible with likelihood-targeting methods.

Lastly, while SSMs have been the focal point of applications in this thesis, the presented methods can be applied far more generally, signified not least by the plethora of disciplines with interest in SBI (Turner and Van Zandt, 2012; Wood, 2010; Cranmer, Brehmer, and Louppe, 2020; Papamakarios, Nalisnick, et al., 2019; Alsing et al., 2019). I hope to explore a wider scope of applications in the future.

References

Acerbi, Luigi (2020). "Variational Bayesian Monte Carlo with Noisy Likelihoods". In: *arXiv preprint arXiv:2006.08655*.

Alsing, Justin et al. (2019). "Fast likelihood-free cosmology with neural density estimators and active learning". In: *Monthly Notices of the Royal Astronomical Society* 488.3, pp. 4440–4458.

Beaumont, Mark A (2010). "Approximate Bayesian computation in evolution and ecology". In: *Annual review of ecology, evolution, and systematics* 41, pp. 379–406.

Beaumont, Mark A, Wenyang Zhang, and David J Balding (2002). "Approximate Bayesian computation in population genetics". In: *Genetics* 162.4, pp. 2025–2035.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). "Variational inference: A review for statisticians". In: *Journal of the American statistical Association* 112.518, pp. 859–877.

Boelts, Jan et al. (2022). "Flexible and efficient simulation-based inference for models of decision-making". In: *Elife* 11, e77220.

Brehmer, Johann et al. (2020). "Mining gold from implicit models to improve likelihood-free inference". In: *Proceedings of the National Academy of Sciences* 117.10, pp. 5242–5249.

Cole, Alex et al. (2022). "Fast and credible likelihood-free cosmology with truncated marginal neural ratio estimation". In: *Journal of Cosmology and Astroparticle Physics* 2022.09, p. 004.

Cranmer, Kyle, Johann Brehmer, and Gilles Louppe (2020). "The frontier of simulation-based inference". In: *Proceedings of the National Academy of Sciences* 117.48, pp. 30055–30062.

Durkan, Conor, Artur Bekasov, et al. (2019). "Neural spline flows". In: *Advances in Neural Information Processing Systems*, pp. 7511–7522.

Durkan, Conor, Iain Murray, and George Papamakarios (2020). *On Contrastive Learning for Likelihood-free Inference*. arXiv: `2002.03712 [stat.ML]`.

Fengler, Alexander, Krishn Bera, et al. (2022). "Beyond Drift Diffusion Models: Fitting a Broad Class of Decision and Reinforcement Learning Models with HDDM". In: *Journal of Cognitive Neuroscience* 34.10, pp. 1780–1805.

Fengler, Alexander, Lakshmi N Govindarajan, et al. (2021). "Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience". In: *Elife* 10, e65074.

Fontanesi, L (2022). *Rlssm (Version 0.1. 1)*.

Foster, Kendal and Henrik Singmann (2021). *Another Approximation of the First-Passage Time Densities for the Ratcliff Diffusion Decision Model*. arXiv: `2104.01902 [stat.AP]`.

Gelman, Andrew et al. (2020). "Bayesian workflow". In: *arXiv preprint arXiv:2011.01808*.

Gentle, James E, Wolfgang K Härdle, and Yuichi Mori (2012). "Springer Handbooks of Computational Statistics". In.

Greenberg, David, Marcel Nonnenmacher, and Jakob Macke (2019). "Automatic posterior transformation for likelihood-free inference". In: *International Conference on Machine Learning*. PMLR, pp. 2404–2414.

Gutmann, Michael U and Jukka Corander (2016). "Bayesian optimization for likelihood-free inference of simulator-based statistical models". In: *The Journal of Machine Learning Research* 17.1, pp. 4256–4302.

Gutmann, Michael U, Ritabrata Dutta, et al. (2018). "Likelihood-free inference via classification". In: *Statistics and Computing* 28.2, pp. 411–425. DOI: 10.1007/s11222-017-9738-6.

Hoffman, Matthew D and Andrew Gelman (2014). "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623.

Lueckmann, Jan-Matthis, Giacomo Bassetto, et al. (2019). "Likelihood-free inference with emulator networks". In: *Symposium on Advances in Approximate Bayesian Inference*. PMLR, pp. 32–53.

Lueckmann, Jan-Matthis, Pedro J Goncalves, et al. (2017). "Flexible statistical inference for mechanistic models of neural dynamics". In: *arXiv preprint arXiv:1711.01861*.

Navarro, Daniel J and Ian G Fuss (2009). "Fast and accurate calculations for first-passage times in Wiener diffusion models". In: *Journal of mathematical psychology* 53.4, pp. 222–230.

Neal, Radford M (2003). "Slice sampling". In: *Annals of statistics*, pp. 705–741.

Papamakarios, George and Iain Murray (2016). "Fast ε-free Inference of Simulation Models with Bayesian Conditional Density Estimation". In: *Advances in Neural Information Processing Systems* 29, pp. 1028–1036.

Papamakarios, George, Eric Nalisnick, et al. (2019). "Normalizing flows for probabilistic modeling and inference". In: *arXiv preprint arXiv:1912.02762*.

Papamakarios, George, Theo Pavlakou, and Iain Murray (2017). "Masked autoregressive flow for density estimation". In: *Advances in Neural Information Processing Systems*, pp. 2338–2347.

Papamakarios, George, David Sterratt, and Iain Murray (2019a). "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 837–848.

— (2019b). "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 837–848.

Pritchard, Jonathan K et al. (1999). "Population growth of human Y chromosomes: a study of Y chromosome microsatellites." In: *Molecular biology and evolution* 16.12, pp. 1791–1798.

Radev, Stefan T et al. (2020). "BayesFlow: Learning complex stochastic models with invertible neural networks". In: *arXiv preprint arXiv:2003.06281*.

Ranganath, Rajesh, Sean Gerrish, and David M Blei (2013). "Black box variational inference". In: *arXiv preprint arXiv:1401.0118*.

Ratcliff, Roger (1978). "A theory of memory retrieval." In: *Psychological review* 85.2, p. 59.

Ratcliff, Roger et al. (2016). "Diffusion decision model: Current issues and history". In: *Trends in cognitive sciences* 20.4, pp. 260–281.

Sisson, Scott A, Yanan Fan, and Mark Beaumont (2018). *Handbook of approximate Bayesian computation*. CRC Press.

Spiegelhalter, David J et al. (2014). "The deviance information criterion: 12 years on". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 76.3, pp. 485–493.

Talts, Sean et al. (2018). "Validating Bayesian inference algorithms with simulation-based calibration". In: *arXiv preprint arXiv:1804.06788*.

Tavaré, Simon et al. (1997). "Inferring coalescence times from DNA sequence data". In: *Genetics* 145.2, pp. 505–518.

Tejero-Cantero, Alvaro et al. (2020). "Sbi: a toolkit for simulation-based inference". In: *Journal of Open Source Software* 5.52, p. 2505.

Thomas, Owen et al. (2022). "Likelihood-free inference by ratio estimation". In: *Bayesian Analysis* 17.1, pp. 1–31.

Turner, Brandon M and Per B Sederberg (2014). "A generalized, likelihood-free method for posterior estimation". In: *Psychonomic bulletin & review* 21.2, pp. 227–250.

Turner, Brandon M and Trisha Van Zandt (2012). "A tutorial on approximate Bayesian computation". In: *Journal of Mathematical Psychology* 56.2, pp. 69–85.

— (2018). "Approximating Bayesian inference through model simulation". In: *Trends in Cognitive Sciences* 22.9, pp. 826–840.

W., Feller and Feller V. (1968). *An Introduction to Probability Theory and its Applications Vol 1*. Vol. 1. Wiley.

Watanabe, Sumio (2013). "A widely applicable Bayesian information criterion". In: *Journal of Machine Learning Research* 14.27, pp. 867–897.

Wiecki, Thomas V, Imri Sofer, and Michael J Frank (2013). "HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python". In: *Frontiers in neuroinformatics* 7, p. 14.

Wingate, David and Theophane Weber (2013). "Automated variational inference in probabilistic programming". In: *arXiv preprint arXiv:1301.1299*.

Wolpert, David H (1996). "The lack of a priori distinctions between learning algorithms". In: *Neural computation* 8.7, pp. 1341–1390.

Wood, Simon N (2010). "Statistical inference for noisy nonlinear ecological dynamic systems". In: *Nature* 466.7310, pp. 1102–1104.