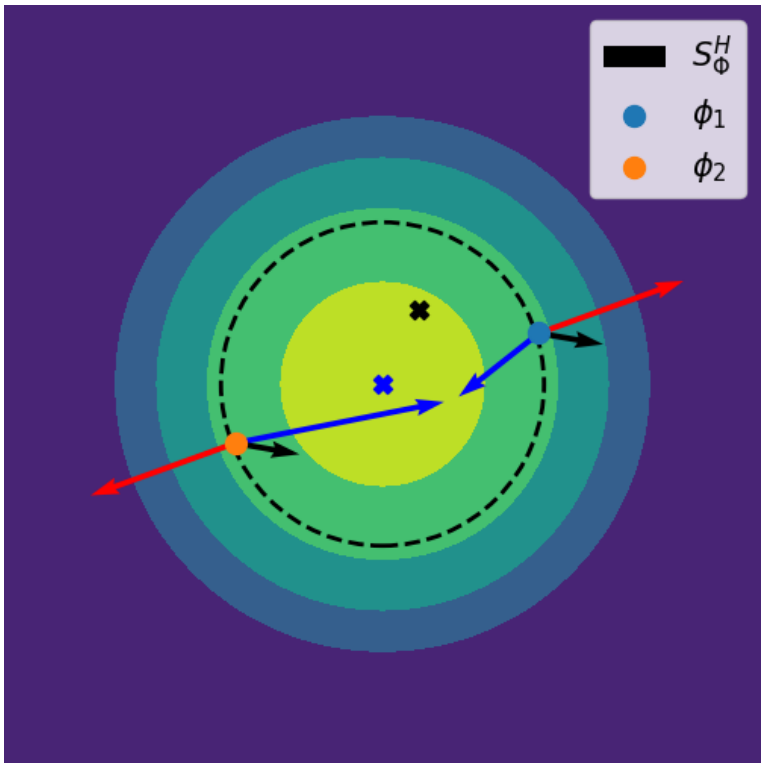
UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE



Ph.D. dissertation

A Probabilistic Approach to the Protein Folding Problem

Using Stein-based Variational Inference

Ola Rønning

Advisors: Thomas Hamelryck and Christophe Ley

Submitted: March 21, 2023

This thesis has been submitted to the PhD School of The Faculty of Science, University of Copenhagen

Abstract

Particle-based variational inference (ParVI) methods are a powerful class of Bayesian inferences algorithms due to their flexible and accurate approximations. ParVI can interpolate between Markov chain Monte Carlo (MCMC) and variational inference (VI) methods, and, as such, ParVI is a suitable candidate for a universal black box inference engine. Bayesian statistical inference is a cornerstone of the empirical sciences. However, correctly realizing the algorithms for Bayesian inference is notoriously tricky and should be left to experts.

Universal black box inference engines (and their associated probabilistic programming language (PPL)) enable the separation of model specification from statistical inference (data conditioning). This separation allows scientists to formulate statistical hypotheses (probabilistic graphical models) as stochastic (probabilistic) computer programs without implementing the inference algorithm. The black-box nature of the inference algorithm ensures the PPL can condition the probabilistic program on experimental observations automatically.

Modern PPLs are generally designed for either MCMC or VI inference, requiring translation to another PPL for an unbiased comparison. Tranlation introduces unnecessary overhead and the potential for code drift–something we can wholly avoid with a generalizing framework like ParVI.

An auspicious ParVI method is Stein variational gradient descent (Liu and Wang, 2016a) (SVGD) due to its direct connection with Stein's method, gradient flows and reproducing kernels. These connections make SVGD versatile and well-suited for theoretical analysis, yielding significant results regarding SVGDs convergence, kernel choice, and convergence rates. However, the adoption of SVGD by practitioners could be more extensive. This is partly due to insufficient tooling and a lack of a mature set of best practices for hyper-parameter choice.

In the first part of this thesis, we extend Stein mixture (SM) (an SVGD variate) to a whole class of approximate inference algorithms indexed by a scalar. We recommend the best choice of indexing scalar and demonstrate why by analyzing the gradient noise. We also present a ready-to-use library for inference with SM as an extension to the NumPyro PPL. We call the library EinStein, which includes the black box SM inference engine, automatic guide generation, many studied kernels, and copiable examples of Bayesian neural network (Neal, 2012) (BNN)s and Deep Markov model (Wu et al., 2018) (DMM)s.

In the second part of the thesis, we study the protein structure prediction problem as a showcase for applying PPLs in the natural sciences. The protein prediction problem aims to predict the (ensemble of) conformation(s) a particular protein may adopt(s) given its sequence of amino acids (and potentially known protein homologs). A high-fidelity solution to the problem could have a massive impact on treatment for misfolding

diseases such as cancer, Alzheimer's, Huntington's, and Parkinson's. A canonical representation of a protein conformation is its internal (toroidal) coordinates. Internal coordinates allow efficient updates to the protein's three-dimensional structure without violating physiochemical properties. To infer statistical models over internal coordinate representations, we introduce a variate of the bivariate von Mises distribution (a 2-torus distribution) in the (Num)Pyro PPLs. The distribution (known as the sine distribution) enables us to specify a hierarchical model over the two high-variance backbone torsion angles. Our model captures probable angle pairs for each amino acid order of magnitude faster than preexisting methods.

Finally, we present our preliminary results on inferring a distribution over protein folding forcefields. Current technologies for protein structure prediction are excellent at the single-structure forecast. However, these methods are black box deep models and yield no insights into physiochemical properties–sometimes even violating them. Our formulation of the folding force as a probabilistic program allows us to automate the other tedious process of tuning protein folding forcefields using our SM inference engine. We can incorporate the existing (known) hyperparameters by choice of prior. SM ability to capture rich correlations in the parameter space makes it a suitable statistical inference algorithm for these forcefields, which tend to be highly sensitive to parameterization. Presuming our forcefield can fold (small to medium size proteins) proteins, the parameter distributions will yield insights into the importance (and sensitivity) of the different (potential) energy terms of the forcefield and a high-resolution view of the aggregate folding trajectory of the protein.

Abstrakt

Partikelbaserede variationsinferens (ParVI) metoder er en kraftfuld klasse af Bayesianske inferensalgoritmer på grund af deres fleksible og nøjagtige tilnærmelser. ParVI kan interpolere mellem MCMC og VI metoder, og som sådan er ParVI en passende kandidat til en universel sort boks-inferensmotor. Bayesiansk statistisk inferens er en hjørnesten i de empiriske videnskaber. Det er imidlertid notorisk vanskeligt at realisere algoritmerne for Bayesiansk inferens korrekt og bør overlades til eksperter. Universal black box-inferensmotorer (og deres tilhørende PPL) muliggør adskillelse af modelspecifikation fra statistisk inferens (datakonditionering). Denne adskillelse gør det muligt for videnskabsmænd at formulere en stor klasse af statistiske hypoteser (sandsynlighed grafiske modeller) som stokastiske (sandsynlighedsmæssige) computerprogrammer, hvilket er væsentligt mere ligetil. Konditionering af modellen på eksperimentelle observationer udføres automatisk af inferensmotoren. Moderne PPL'er er generelt designet til enten MCMC eller VI inferens, hvilket kræver oversættelse til en anden PPL for en upartisk sammenligning. Translation introducerer unødvendig overhead og potentialet for kodedrift - noget vi helt kan undgå med en generaliserende ramme som ParVI.

En lovende ParVI-metode er en variant af SVGD på grund af dens direkte forbindelse med Steins metode, gradientflow og reproducerende kerner. Disse forbindelser gør SVGD alsidig og velegnet til teoretisk analyse, hvilket giver betydelige resultater vedrørende SVGDs konvergens, kernevalg og konvergenshastigheder. Imidlertid kunne vedtagelsen af SVGD af praktiserende læger være mere omfattende. Dette skyldes delvist et behov for mere nyttigt værktøj og et modent sæt af bedste praksis for valg af hyperparametre. I den første del af denne afhandling udvider vi SM (en SVGD-variat) til en hel klasse af tilnærmede inferensalgoritmer indekseret af en skalar. Vi anbefaler det bedste valg af indekseringsskalar og demonstrer hvorfor ved at analysere gradientstøjen. Vi kapitler også et klar-til-brug-bibliotek til inferens med SM som en udvidelse til NumPyro PPL. Vi kalder biblioteket EinStein, som inkluderer den sorte boks SM-inferensmotor, automatisk guidegenerering, mange studerede kerner og kopierbare eksempler på BNN'er og DMM'er.

I anden del af afhandlingen studerer vi problemet med forudsigelse af proteinstruktur som et udstillingsvindue for anvendelse af PPLs i naturvidenskaberne. Proteinforudsigelsesproblemet har til formål at forudsige (ensemblet) af konformation(er), et bestemt protein kan antage givet dets sekvens af aminosyrer (og potentielt kendte homologer). En high-fidelity-løsning på problemet kan have en massiv indvirkning på behandlingen af fejlfoldende sygdomme som kræft, Alzheimers, Huntingtons og Parkinsons. En kanonisk repræsentation af en proteinkonformation er dens indre (toroidale) koordinater. Interne koordinater tillader effektive opdateringer af proteinets tredimensionelle struktur uden at krænke de fysisk-kemiske egenskaber. For at udlede statistiske modeller over interne koordinatrepræsentationer introducerer vi en variant af den bivariate von

Mises-fordeling (en 2-torus-fordeling) i (Num)Pyro PPL'erne. Fordelingen (kendt som sinusfordelingen) gør det muligt for os at specificere en hierarkisk model over de to torsionsvinkler i rygraden med høj varians. Vores model fanger sandsynlige vinkelpar for hver aminosyrestørrelsesorden hurtigere end allerede eksisterende metoder. Til sidst præsenterer vi vores foreløbige resultater om at udlede en fordeling over proteinfoldningskraftfelter. Nuværende teknologier til forudsigelse af proteinstruktur er fremragende til at producere en enkeltstrukturprognose. Disse metoder er dog sorte boks-dybe modeller og giver ingen indsigt i fysisk-kemiske egenskaber - nogle gange endda krænker dem. Vores formulering af foldningskraften som et probabilistisk program giver os mulighed for at automatisere den anden kedelige proces med tuning af proteinfoldningskraftfelter ved hjælp af vores SM-inferensmotor. Vi kan inkorporere de eksisterende (kendte) hyperparametre ved valg af tidligere. SM evne til at fange rige korrelationer i parameterrummet gør det til en passende statistisk inferensalgoritme for disse kraftfelter, som har tendens til at være meget følsomme over for parameterisering. Forudsat at vores kraftfelt kan folde (små til mellemstore proteiner) proteiner, vil parameterfordelingerne give indsigt i vigtigheden (og følsomheden) af kraftfeltets forskellige (potentielle) energitermer og et højopløsningsbillede af den samlede foldningsbane af proteinet.

Statement of intend

The following dissertation is a monograph written during my Ph.D. fellowship (2020-2023) at the Department of Computer Science, University of Copenhagen. For coherence, the dissertation focuses on probabilistic programming with applications in the protein structure prediction problem. In doing so, the monograph will center only on a selection of the manuscripts I have (co-)authored during my fellowship.

The following pages contain unaltered text from the selected manuscripts I originally authored.

Page 9-10 Text from Rønning et al., 2021b and Ola Rønning, Christophe Ley, Ahmad Salim Al-Sibahi and Thomas Hamelryck, 2023,

Page 19-20 Text from Ola Rønning, Christophe Ley, Ahmad Salim Al-Sibahi and Thomas Hamelryck, 2023,

Page 43 Text from Rønning et al., 2021a,

Page 59-62 Text from Rønning et al., 2021a,

Page 56 Text from Rønning et al., 2021a,

Page 63-80 Text from Rønning et al., 2021b,

Page 93 Text from Ola Rønning, Christophe Ley, Ahmad Salim Al-Sibahi and Thomas Hamelryck, 2023 and Rønning et al., 2021b,

Page 95 Text from Ola Rønning, Christophe Ley, Ahmad Salim Al-Sibahi and Thomas Hamelryck, 2023 and Rønning et al., 2021b.

Acknowledgements

I have enjoyed my time a KU and look forward to continuing my research over the next year. Many people deserve a special mention for their contribution to my authoring this thesis. First, I'd like to thank my supervisor Thomas Hamelryck for guidance and sparring. Second, my co-supervisor Christophe Ley whose keen eye (and outstanding skill) has fixed several manuscripts. I would also like to acknowledge the Danish National Research Foundation-FTP (2020) project: Deep probabilistic programming for protein structure prediction, for funding my research.

A special mention goes to my loving family, my wife, Marie, whose advice and support are paramount to any enterprise I endeavor. My children, Sonja and Konrad–both of whom are growing up faster than I would allow. To my parents, Kristin and Kjell Bjørn, for letting their youngest son back in their house once a week and Anne-Dorte Johansen for hosting me and watching Sonja so I could finish manuscripts.

To my senior colleagues, Fritz Henglein, for saga advice and perspective, and Eugene Cosmin, whose energy and drive can lift one from the drudgery that is often modern research. Finally, my office mates, Robert Schenck, Mikkel Mathiesen, Lys Mareta, and Christian Thygesen, for walking their parallel paths and sharing their insights.

List of Abbreviations and Symbols

BNN Bayesian neural network (Neal, 2012). ii, iv, 11, 29, 54, 55

CPU central processing unit. 10, 68, 70, 71

$\mathcal{D}$ The finite set of observations $\mathcal{D} = \{\mathbf{x}_i\}_{i \in I \subseteq \mathbb{N}}$ such that all pairs of distinct elements $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{X}$ are IID according to (potentially unknown) generating process $p(\mathbb{X})$. xi, 15–17, 19, 21–23, 25, 26, 35–37, 39–41, 43, 45–47, 83

DMM Deep Markov model (Wu et al., 2018). ii, iv, 56, 58

DPP deep probabilistic programming. 4, 81, 82, 96

EL Expected likelihood kernel. 22

ELBO For distributions p and q such $q \ll p$ we have that $\log p(\mathcal{D}) \geq \mathbb{E}_{\log q(\mathbf{z}|\mathcal{D})}\left[\left(\frac{p(\mathbf{z},\mathcal{D})}{q(\mathbf{z}|\mathcal{D})}\right)\right]$. The inequality is known as the evidence lower bound. 16, 17, 19, 21, 23, 27, 41, 44, 47, 48, 58

EwS ELBO-within-Stein. 10, 11, 23, 26, 29, 31, 33, 34, 44, 95

FNN feedforward neural network. 56

GPU graphical processing unit. 10, 68, 70

IID Two random variables $\mathbf{x}$ and $\mathbf{y}$ are independently and identically distributed if $p(\mathbf{x}) = p(\mathbf{z})$ a.e. and $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}, \mathbf{y})$. xi

D_{KL} Let p and q be probability measures on a measure space $\mathbb{X}$ such that $p \ll q$ (p is absolutely continuous wrt. q) then the Kullback-Leibler (D_{KL}) divergence is given by $D_{\mathrm{KL}}[p \parallel q] = \int_{\mathbb{X}} dx \log(p/q)\, p$. 8, 19, 21, 22

KSD kernelized Stein discrepancy (Anastasiou et al., 2021; Liu, Lee, and Jordan, 2016). 5

MCMC Markov chain Monte Carlo. ii, iv, 4, 5, 8, 9, 16, 43, 95

PDF probability density function. 65, 66

PPL probabilistic programming language. ii–v, 4, 6, 9, 10, 44, 64, 93, 95

RBF Radial basis function kernel $k : \boldsymbol{x}, \boldsymbol{y} \mapsto \exp\left(-\frac{1}{h} \parallel \boldsymbol{x} - \boldsymbol{y} \parallel_2^2\right)$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ with $d \in \mathbb{N}$ and $h \in \mathbb{R}$.. 20, 22, 29, 45, 55, 59, 60

RNN recurrent neural network. 7

z We use **z** as a pronoun for the collection latent (meaning unobserved), the random variable of a given non-hierarchical statistical model p. The space of **z** is given by context. By construction, **z** flattens all latent parameters to a tuple.. xi, 8, 15–17, 19–23, 25, 26, 30, 31, 35–37, 39–41, 43, 45–48, 51, 52

SM Stein mixture. ii–v, 11, 16, 17, 21–23, 25, 33, 41, 54, 55

SteinVI Stein based variational inference. 4–6, 8, 9, 11, 43, 44, 49, 61, 93

SVGD Stein variational gradient descent (Liu and Wang, 2016a). ii, iv, 4, 5, 9, 11, 15–17, 19, 20, 22, 29, 43, 45–47, 54–57, 93, 95

SVI stochastic variational inference (Hoffman et al., 2013). 41, 54, 55

TPU tensor processing unit. 10

VAE variational autoencoder (Kingma and Welling, 2013). 11, 29, 30, 33, 41

VI variational inference. ii, iv, 4, 5, 8, 10, 43

Author's Contributions

[1] Lys Sanz Moreta, Ola Rønning, Ahmad Salim Al-Sibahi, Jotun Hein, Douglas Theobald, and Thomas Hamelryck. "Ancestral Protein Sequence Reconstruction using a Tree-Structured Ornstein-Uhlenbeck Variational Autoencoder". *International Conference on Learning Representations*. 2021.

[2] Johan-Frederik Nielsen, Thomas Hamelryck, and Ola Rønning. "Generalising Stein Variational Gradient Descent Through Rényi's α-divergence". *Unpublished* (2023).

[3] Ola Røinning, Christophe Ley and Thomas Hamelryck. "Probabilistic Differential Molecular Simluation". *Unpublished* (2023).

[4] Ola Rønning, Christophe Ley, Ahmad Salim Al-Sibahi and Thomas Hamelryck. "ELBO-ing Stein Mixtures". *Unpublished* (2023).

[5] Ola Rønning, Ahmad Salim Al-Sibahi, Christophe Ley, and Thomas Hamelryck. "EinSteinVI: General and Integrated Stein Variational Inference". *Unplublished* (2021).

[6] Ola Rønning and Thomas Hamelryck. "Variational Energy Conserving Subsampling". *ProbProg Workshop* (2021).

[7] Ola Rønning, Christophe Ley, Kanti V. Mardia, and Thomas Hamelryck. "Time-efficient Bayesian Inference for a (Skewed) Von Mises Distribution on the Torus in a Deep Probabilistic Programming Language". *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. 2021, pp. 1–8. DOI: 10.1109/MFI52462.2021.9591184.

[8] Ola Rønning, Yijie Zhang, and Eric Nalisnick. "On the Inconsistency of Bayesian Inference with Misspecified Bayesian Neural Networks". *Unpublished* (2023).

[9] Robert Schenck, Ola Rønning, Troels Henriksen, and Cosmin E. Oancea. "AD for an Array Language with Nested Parallelism". *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '22. Dallas, Texas: IEEE Press, 2022. ISBN: 9784665454445.

[10] Christian B. Thygesen, Ola Rønning, Christian Skjødt Steenmans, Anders Bundgård Sørensen, Kanti V. Mardia, John T. Kent, and Thomas Hamelryck. "A Multiscale Deep Generative Model of Protein Structure Using a Directional and a Procrustes Likelihood". *Unpublished* (2023).

Contents

[1]Department of Computer Science, University of Copenhagen, Denmark
[2]Département Mathématiques, Université du Luxembourg, Luxembourg
[3]Department of Biology, University of Copenhagen, Denmark

[4]Department of Computer Science, University of Copenhagen, Denmark
[5]Département Mathématiques, Université du Luxembourg, Luxembourg
[6]Department of Biology, University of Copenhagen, Denmark
[7]School of Mathematics, University of Leeds, United Kingdom
[8]Department of Statistics, University of Oxford, United Kingdom

[9]Department of Computer Science, University of Copenhagen, Denmark
[10]Département Mathématiques, Université du Luxembourg, Luxembourg
[11]Department of Biology, University of Copenhagen, Denmark

Part I

Foundations

Chapter 1

Introduction

One of the great current scientific challenges is the problem of protein structure prediction. The problem calls for a robust methodology to determine the 3-dimensional structures of proteins. The challenge exists is many variates–some of which were resolved within the last few years. The scientific interest stems from falsifying the following hypothesis (known as Anfinsen's dogma (Anfinsen, 1973)): the native conformation, the structure of the protein adopts in a standard environment–is solely determined by the sequence of amino acids for small globular proteins. Using available tools, efficient methods exist to compute single structures' native conformations (Jumper et al., 2021). Yet, the problem is not solved. Distributions of conformations are the new frontier (future) of protein structure prediction (Lane, 2023). To logically reason about uncertainty (quantified as distributions) we must use probability theory and if we want to be constructive (hence applied), probabilistic programs are the suitable formalism. Knowing these distributions could lead to tremendous breakthroughs with regard to understanding and healing diseases such as cancer, Alzheimer's, Huntington's, and Parkinson's. It could also help more efficient iteration and development of vaccines, which is timely given the recent COVID-19 pandemic.

Recent breakthroughs in single structure prediction should be attributed to advances in deep neural networks (Krizhevsky, Sutskever, and Hinton, 2017; Hochreiter and Schmidhuber, 1997; Goodfellow et al., 2020; Vaswani et al., 2017). However, these techniques do not provide insights into the folding dynamics. Further, like many state-of-the-art prediction methods in machine learning (ML) and artificial intelligence (AI), they are overly confident in their predictions (Nguyen, Yosinski, and Clune, 2015; Szegedy et al., 2013) and thus should not be trusted.

In protein science, deep methods such as AlphaFold(2) fail to account for the separation of uncertainty due to protein dynamics and uncertainty due to experimental noise (Thygesen et al., 2021) of observed structures. So while deep methods, such as AlphaFold (Jumper et al., 2021; Tunyasuvunakool et al., 2021)

and OmegaFold (Wu et al., 2022), have massively improved structure prediction of proteins, they remain challenged for highly disordered regions (Wilson, Choy, and Karttunen, 2022) and proteins with multimodal stable conformation states (Lane, 2023). The deep techniques further do not adequately account for chemophysical constraints. To overcome this lack simulation using experimental force fields (like the Amber force field (Wang et al., 2004)) are aPPLied post-hoc of prediction.

Deep probabilistic programming (DPP) is a new paradigm in machine learning, addressing the problem of uncertainty representation in deep neural models. DPP is usually embedded domain-specific languages that provide operators for computing conditional probability distributions (Devroye, 2006; Ackerman, Freer, and Roy, 2011; Kolmogoroff, 1933). The central construction of a probabilistic programming language is conditional probability. The challenge in developing DPPs is twofold. First, the Kolmogorov construction of conditional distributions (Kolmogoroff, 1933) describes properties that the conditional probability distribution satisfies rather than a recipe for its computation. Second, we now know that no general algorithm exists to compute all conditional distributions (Ackerman, Freer, and Roy, 2011). The upshot of a DPP is that it combines the scope of deep learning with a statistically sound treatment of uncertainty (Klenke, 2013).

Given a scientific hypothesis formalized as a statistical model and realized as a (computable) stochastic program, we generally have two methods for deriving conclusions (inferring) from data. Variational inference is computationally efficient but approximate. MCMC is exact, but only the asymptotic regime (Brooks et al., 2011), so if the chain mixes poorly, it will be burdensome to explore the entire state space of the model and data. The nomenclature clashes with that of randomized algorithms (another class of programs well suited for probabilistic programming) as both VI and MCMC are Monte Carlo algorithms. That is, they are incorrect with a (sometimes known) probability. In this thesis, we will explore Stein based variational inference (SteinVI) methods, which interpolate between VI and MCMC by extending the known SVGD to restricted mixtures. We will use this novel inference algorithm to capture distributions of conformations. This will require distributions over protein structures, a computational framework for inference, and statistical models of proteins–the three contributions of this thesis.

1.1 Objectives

Machine learning and uncertainty Statistical inference is a cornerstone of empirical science (Fisher, 1992; Gelman et al., 1995) and machine learning (Bishop and Nasrabadi, 2006). Bayesian inference is a general and powerful formulation of statistical inference but is often computationally burdensome for complex (e.g., deep) models. Generally speaking, there are two contrasting approaches to inference. First, *variational inference (VI) is an approximate variant of Bayesian inference* (Jordan et al., 1999; Blei, Kucukelbir, and McAuliffe, 2017) that scales to tall (i.e., many datapoints) and wide (i.e., high-dimensional) models. VI uses a simple model (called a **guide**) to approximate the desired model. SteinVI (**Objective 1**) realizes the particle method Stein Variational Gradient Descent (SVGD) (Liu and Wang, 2018a) as a VI method. Second, *Markov chain Monte Carlo (MCMC) is an exact variant of Bayesian inference* (Brooks et al., 2011) with theoretical guarantees about (asymptotic) convergence, detailed balance (Green, 1995), and ergodicity (Brooks et al., 2011). However, inference with tall and wide datasets for complex models requires Google-scale data-centers (Izmailov et al., 2021). Fundamentally Stein VI operators on the kernel Stein discrepancy kernelized Stein discrepancy (Anastasiou et al., 2021; Liu, Lee, and Jordan, 2016) (KSD). Minimizing the discrepancy with respect to a guide for a single particle recovers VI. Minimizing KSD with respect to a sequence of particles (Chen et al., 2019) is an MCMC method with properties akin to quasirandom number sequences (see O'neill, 2014 for an overview). This unifying property of SteinVI makes it an attractive candidate as a best-of-both-worlds universal inference engine for Bayesian inference.

Generalizing Stein Variational Inference Even though the theory of SVGD is well-developed, and the implementation in general frameworks like Pyro (Bingham et al., 2019) are still rudimentary. It is not possible to use a custom guide (inference program, including deep neural networks) and so the inference is limited to particles over point masses. We generalize the implementation in NumPyro (Phan, Pradhan, and Jankowiak, 2019a) to work with custom guides and integrate techniques such as message passing Stein VI and higher-order Stein VI for efficiency. This requires proving the interaction between Stein mixtures and traditional parametric VI methods. We prove this relation by the Rényi divergence. We use the algorithm to demonstrate we can model with fewer particles to uncertainty than SVGD while allowing much more flexible models than now.

Statistical model of protein folding The approach we like to take on protein folding is close to Nemo (Ingraham et al., 2019), where we rely on a combination of deep probabilistic programming and differentiable simulation. We will, however, separate the simulation (model) from the inference of param-

eters (heuristic) to provide an interpretable model that can guide the scientific understanding of the protein folding process. Our model will be realized as a probabilistic program that infers parameterized forces using experimentally obtained structures and our SteinVI framework. We believe the flexibility of the SteinVI framework will allow us to conduct efficient data exploration and give a high-fidelity approximation of the probable energy surfaces capable of folding proteins.

To realize a new frontier of protein structure prediction, we investigate an underdeveloped form of variational inference. We develop a computational framework for inference proving the connection to traditional variational inference (**Objective 1**), introduce distributions over torus to PPLs, demonstrating the inference and the massive computational advantages of modern frameworks (**Objective 2**), and develop a physically interpretable hypothesis of protein folding (**Objective 3**).

1.2 Description of the research project in the context of international state-of-the-art

The classical approach to protein folding is based on minimizing physical energy functions (Simons et al., 1997), which model the thermodynamics of the water-protein system. The search space is large and complex, so a naive search would likely not find structures of the target protein. The traditional software for physical protein folding, Rosetta (Simons et al., 1997) therefore relies on a method that assembles proteins from known fragments and then uses simulated annealing (Henderson, Jacobson, and Johnson, 2006) to randomly replace fragments and refold until an optimal solution is found. The general technique often produces excellent results, but the search is computationally inefficient.

The recent deep-learning revolution has brought renewed interest to solving the protein folding problem using these recently developed techniques. DeepMind–a daughter company of Alphabet and world-leading in deep learning–has developed AlphaFold (Senior et al., 2019; Jumper et al., 2021), a tool for folding proteins using a deep residual convolutional neural network. Their network beat existing approaches in the latest CASP13 competition (Kryshtafovych et al., 2019). Their neural network relies on co-evolutionary techniques where a database of sequences of existing similar proteins provides additional information about the input protein and predicts discrete distributions over the pairwise distances between amino acids and the torsion angles. These predicted distributions are then combined with a van der Waals smoothing term to construct a potential function over torsion angles, which is then minimized using an algorithm that combines evolutionary search and Newton's method for predicting the final structure. AlphaFold is inspired engineering and performs well in practice, but it is not the solution. The main backbone of the model, the neural network, is deep and impossible to interpret, and the folding process afterward is not physically relatable. The model only works for proteins where co-evolutionary information is available. Experimentally obtaining the structure of proteins is expensive and time-consuming. The chosen features seem ad-hoc and driven by empirical performance rather than understanding. Furthermore, no path specifies how to generalize their architecture beyond protein structure prediction to other similarly complex problems exist. An alternative approach to deep learning for protein structure prediction is recurrent geometric networks (AlQuraishi, 2019). The approach relies on a recurrent neural network (RNN) that predicts torsion angles and constructs the 3-dimensional structure on the fly. The architecture is more general than AlphaFold, since it does not require co-evolutionary information but does not seem to come close in empirical performance. The RGN architecture is fully black-box but simpler than AlphaFold.

A hybrid approach to protein folding is Nemo (Ingraham et al., 2019) which

performs folding using differentiable simulation with an unknown (and learned) energy function. The simulator acts in physical simulators where an energy function is minimized to fold the protein. Still, the energy is a deep neural network rather than chemo-physical properties, and the discretization of the dynamic is crude. Typically molecular dynamic simulations run for billions or trillions of steps (Irbäck and Mohanty, 2006), whereas Nemo runs for one hundred (Ingraham et al., 2019). In particular, the neural network as a vector field–the change to the structure is computed as a function of the current structure and its amino acid sequence. A loss function is backpropagated through the simulation to update the network's parameterization. The architecture is fully differentiable, which allows learning and performs well on a large family of proteins without requiring co-evolutionary information. The folding process, however, still involves the black-box neural energy function, and so the process is not necessarily representative of a realistic physically-understandable process.

For complex problems like protein folding, it is, in general, not possible to perform exact Bayesian inference due to the need to calculate a normalization constant. There are, however, several techniques to approximate the posterior distribution. Sampling-based techniques like MCMC (Chib and Greenberg, 1995; Hoffman, Gelman, et al., 2014; Neal et al., 2011; Duane et al., 1987) approximate the posterior distribution using a set of particles that are drawn by moving around the parameter space in a way that is driven towards high-probability regions of the conditioned probabilistic model (representing prior and likelihood). The approach can capture powerful correlations amongst parameter values and is guaranteed to converge to the target posterior distribution is given enough samples. However, MCMC techniques are computationally expensive and hard to use with mini-batches, making them infeasible for large and/or complex data sets. An alternative technique to MCMC techniques is VI (Hoffman et al., 2013; Ranganath, Gerrish, and Blei, 2014). The core idea of variational inference is to approximate the posterior $p(\mathbf{z}|\mathbf{x}) \propto p(\mathbf{x}, \mathbf{z})$ using a tractable distribution $q(\mathbf{z})$ drawn from a family of distributions $\mathbb{Q}$ in a way that minimizes the Kullback-Leibler divergence

$$\begin{aligned} q^* &= \arg\min_{q^* \in \mathbb{Q}} D_{\mathrm{KL}}\left[q \parallel p\right] \\ &\propto \mathbb{E}_{\mathbf{z}\sim q}\left[\log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z})\right]. \end{aligned}$$

Variational inference scales to massive data sets using mini-batch training, and can be integrated with deep neural networks for richer models (Kingma and Welling, 2013; Ritchie, Horsfall, and Goodman, 2016). Traditional parametric techniques to variational inference, however, assume conditional independencies for tractability, which result in a lack of correlations between parameters and over- confidence with regard to uncertainty. Recently, SteinVI (Han and Liu, 2018; Liu and Wang, 2018a; Liu, Lee, and Jordan, 2016; Liu and Wang, 2016a; Zhuo et al., 2018) has been developed as a technique that combines the flexibility of non-parametric (particle-based) methods with the

scalability of mini-batch-based training available with variation inference. The technique relies on Stein's discrepancy (Anastasiou et al., 2021) to guide particles toward high-probability values while building a repulsive force that avoids modal collapse. It has been shown to have better mode covering properties than parametric variational inference, capture correlations, and MCMC methods, and has excellent per-particle efficiency. It has well-explained mathematical interpretations as particles following the gradient flow of the differential geometry and as a technique for matching moments (mean, variance) of the target posterior. However, SVGD underestimates variance for high dimensional distributions (Ba et al., 2021) requiring a growing number of particles to adequately represent the target distribution.

Recently, Stein variational techniques relying on Newton's method (Detommaso et al., 2018) have been shown to solve complex problems with faster convergence than SVGD accurately. In particular, Stein variational newton descent works well with inferring parameters of the problem based on the Langevin equations (Gillespie, 2000), the same fundamental dynamic driving the simulation we use for protein folding. The protein folding simulator works with high-dimensional data and sparsity-based dimensionality–reduction techniques may be needed to explore the parameter space effectively. One could use several approaches to employ SteinVI to learn the simulator's parameters. The most straightforward approach would be to divide the protein dataset into related families, learn a posterior distribution for each family and combine the result using Bayesian meta-analysis (Blomstedt et al., 2019). The setup is straightforward and can be a good starting point for inference. More refined approaches would rely on including deep neural networks in the inference. The idea is to use amortized Stein inference (Feng, Wang, and Liu, 2017) or rely on fully Bayesian neural networks with Stein particles representing the weights.

PPL is at the interface between statistics and the theory of programming languages. PPLs formulate statistical models as stochastic programs that enable automatic inference algorithms and optimization. Pyro Bingham et al., 2019 and its sibling NumPyro Phan, Pradhan, and Jankowiak, 2019a are PPLs built on top of the deep learning frameworks PyTorch Paszke et al., 2019, and Jax Bradbury et al., 2018, respectively. Both PPLs provide simple, highly similar interfaces for inference using efficient implementations of Hamiltonian Monte Carlo (HMC), the No-U-Turn Sampler (NUTS), and stochastic variational inference. They generate variational distributions from a model, automatically enumerate discrete variables, and support formulating deep probabilistic models such as variational autoencoders and deep Markov models.

The Sine von Mises distribution and its skewed variant are toroidal distributions relevant to protein bioinformatics. They provide a natural way to model the dihedral angles of protein structures, which is important in protein structure prediction, simulation, and analysis. We present efficient implementations of the Sine von Mises distribution and its skewing in Pyro and NumPyro and devise a simulation method that increases efficiency by several orders of

magnitude when using parallel hardware (i.e., modern central processing unit (CPU)s, graphical processing unit (GPU)s, and tensor processing unit (TPU)s). We demonstrate the use of the skewed Sine von Mises distribution by modeling dihedral angles of proteins using a Bayesian mixture model inferred using NUTS, exploiting NumPyro's facilities for automatic enumeration (Obermeyer et al., 2020).

Stein variational inference is a technique for approximate Bayesian inference that has recently gained popularity because it combines the scalability of VI with the flexibility of non-parametric inference methods. While there has been considerable progress in developing algorithms for Stein VI, integration in existing PPLs with an easy-to-use interface is currently lacking. EinStein is a lightweight composable library that integrates the latest Stein VI methods with the PPL NumPyro (Phan, Pradhan, and Jankowiak, 2019b). EinStein also provides our novel algorithm ELBO-within-Stein (EwS) to support the use of custom inference programs (guides), in addition to implementations of a wide range of kernels, non-linear scaling of the repulsion force (Wang and Liu, 2019a) and second-order gradient updates using matrix-valued kernels (Wang et al., 2019a). We illustrate EinStein using toy examples and show results on par with or better than existing state-of-the-art methods for real-world problems. These include Bayesian neural networks for regression and a Stein-mixture deep Markov model, which also shows EinStein scales to large models with more than 125,000 parameters.

1.3 Scientific Contributions

In Chapter 2 we make the following concrete contributions;

- We demonstrate that inaccurate gradient estimates can lead to issues with convergence for SMs.
- We introduce a new family of inference algorithms for SMs indexed by the parameter α. The family results from connecting inference with SMs to the Rényi α-divergence and includes the inference algorithm by Nalisnick and Smyth, 2017 as a special case for $\alpha = 0$.
- Unlike previous work, our algorithm allows for investigating a range of values for α for a model of interest. This allows us to investigate the convergence stability for different α's by measuring the SN-ratio. We find that $\alpha = 1$ is optimal for models with a latent variable for each data point (local latent variables), resulting in better SN ratios than all other α values. For models where all datapoints share a latent variable (global latent variables), using $\alpha = 0.5$ (corresponding to the Hellinger distance) is on par with Nalisnick and Smyth, 2017's algorithm (which corresponds to $\alpha = 0$). Other values for α result in worse SN ratios.
- We evaluate our inference algorithm for different values of α on BNN and variational autoencoder (Kingma and Welling, 2013) (VAE), showing that the α that results in the highest performance varies depending on both model and data set.

Chapter 3 is dedicate to the software library EwS detailing its design and implementation. The contributions of the chapter are;

- We introduce a novel family of algorithms, called EwS, for inferring SMs based on connecting inference with Stein mixtures to the Rényi divergence. EwS includes the inference algorithm by Nalisnick and Smyth, 2017 as a particular case.
- We analyze the signal-to-noise ratio (SNR) of the gradient estimator in the SVGD update of SM, demonstrating the order of the Rényi divergence affects SNR.
- We detail our black-box inference library of EwS in NumPyro, which we call the library SteinVI. SteinVI allows SMs to work with custom guide programs based on EwS optimization. The library is compositional with NumPyro features, including support for deep learning, loss functions (ELBO, Rényi ELBO (Li and Turner, 2016)), and optimization methods, thus making it possible for SteinVI to grow organically with NumPyro development.

In Chapter 4, we look at statistical models of inner coordinate representations of protein structure. In this chapter, the following are the contributions;

- An efficient implementation of the (2-torus) sine distribution (Singh, Hnizdo, and Demchuk, 2002) in Pyro and NumPyro.
- An efficient implementation of the sine-skewing procedure (Ameijeiras-Alonso and Ley, 2021) in Pyro and NumPyro.
- The sine-skewing procedure (Ameijeiras-Alonso and Ley, 2021)
- A Bayesian mixture of sine skewed sine distributions for modeling dihedral angles in amino acids.

Finally, in Chapter 5, we outline the status of our differential simulator. We do not have results to share yet. However, we believe results are eminent and the setting constitutes a largely underdeveloped approach to the folding problem.

Part II

Description of Select Research Carried Out

Chapter 2

ELBOing Stein Mixtures

Ola Rønning[1], Christophe Ley[2], Ahmad Salim Al-Sibah[1] and Thomas Hamelryck[13]

2.1 Introduction

The ability of Bayesian deep learning to quantify the uncertainty of predictions by deep models is causing a surge of interest in using these techniques (Izmailov et al., 2021). Bayesian inference aims to describe i.i.d. data $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$ using a model with latent a variable $\mathbf{z}$. Bayesian inference does this by computing a posterior distribution $p(\mathbf{z}|\mathcal{D})$ over the latent variable given a model describing the joint distribution $p(\mathbf{z}, \mathcal{D}) = p(\mathcal{D}|\mathbf{z})p(\mathbf{z})$. We obtain the posterior by following Bayes' theorem,

$$p(\mathbf{z}|\mathcal{D}) = \prod_{i=1}^n p(\mathbf{x}_i|\mathbf{z})p(\mathbf{z}) / p(\mathcal{D}),$$

where $p(\mathcal{D}) = \int_{\mathbb{Z}} \prod_{i=1}^n p(\mathbf{x}_i|d\mathbf{z})p(d\mathbf{z})$ is the normalization constant. For most practical models, the normalization constant lacks an analytic solution or poses a computability problem, complicating the Bayesian inference problem.

SVGD is a recent technique for Bayesian inference that uses a set of particles $\mathcal{Z} = \{\mathbf{z}_i\}_{i=1}^N$ to approximate the posterior $p(\mathbf{z}|\mathcal{D})$. The idea behind SVGD is to iteratively transport $\mathcal{Z}$ according to a force field $S_{\mathcal{Z}}$, called the Stein force. The Stein force is given by

$$S_{\mathcal{Z}}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j \sim q_{\mathcal{Z}}}[k(\mathbf{z}_i, \mathbf{z}_j)\nabla_{\mathbf{z}_j} \log p(\mathbf{z}_j|\mathcal{D}) + \nabla_{\mathbf{z}_j} k(\mathbf{z}_i, \mathbf{z}_j)], \tag{2.1}$$

where $k(\cdot, \cdot)$ is a reproducing kernel (Berlinet and Thomas-Agnan, 2011), $q_{\mathcal{Z}} = N^{-1} \sum_i \delta_{\mathbf{z}_i}$ is the empirical measure on the set of particles $\mathcal{Z}$, $\delta_{\mathbf{x}}(\mathbf{y})$ represents the Dirac delta measure, which is equal to 1 if $\mathbf{x} = \mathbf{y}$ and 0 otherwise, and $\nabla_{\mathbf{z}_j} \log p(\mathbf{z}_j|\mathcal{D})$ is the gradient of the posterior with respect to the j-th

[1]Department of Computer Science, University of Copenhagen, Denmark
[2]Département Mathématiques, Université du Luxembourg, Luxembourg
[3]Department of Biology, University of Copenhagen, Denmark

particle. The technique is scalable to tall data (i.e. datasets with many data points) and offers the flexibility and scope of techniques such as MCMC. SVGD is good at capturing multi-modality (Liu and Wang, 2016b; Wang and Liu, 2019b), and has theoretical interpretations such as a set of particles following a gradient flow (Liu, 2017) or in terms of the properties of kernels (Liu and Wang, 2018b).

The main problem is that SVGD suffers from the curse of dimensionality: variance estimation scales inversely with dimensionality (Ba et al., 2021). Nalisnick and Smyth, 2017 suggest resolving this by using a Stein mixture (SM). SMs lift each particle to the parameters of a variational distribution q, also called a guide. The idea is that each guide in the Stein mixture represents the density of multiple particles in SVGD, thereby reducing the number of particles needed to represent a posterior. The Nalisnick and Smyth algorithm introduces guides by replacing each posterior gradient $\nabla_{\mathbf{z}_j} \log p(\mathbf{z}_j|\mathcal{D})$ in Equation (2.1) with the corresponding gradient of the marginal log-variational likelihood given by

$$\log p(\mathcal{D}|\boldsymbol{\phi}_j) = \log \mathbb{E}_{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi}_j)} \left[\frac{p(\mathcal{D},\mathbf{z})}{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi}_j)} \right]. \tag{2.2}$$

Here, we denote the particles by $\Phi = \{\boldsymbol{\phi}_j\}_{i=1}^N$ instead of $\mathcal{Z} = \{\mathbf{z}_i\}_{i=1}^N$ to emphasize they parameterize guide components $q(\mathbf{z}|\boldsymbol{\phi}_i, \mathcal{D})$. The change in gradient corresponds to minimizing $D_{\mathrm{KL}}[q_\Phi(\boldsymbol{\phi}) \parallel p(\boldsymbol{\phi}|\mathcal{D})]$ rather than $D_{\mathrm{KL}}[q_\mathcal{Z}(\mathbf{z}) \parallel p(\mathbf{z}|\mathcal{D})]$, as in SVGD. Note that the line between the model p and guide q becomes blurred, as $p(\mathcal{D}|\boldsymbol{\phi})$ is random in both data ($\mathcal{D}$), as is usually the case, but also in the guide hyper-parameters $\boldsymbol{\phi}$ (Ranganath, Tran, and Blei, 2016; Nalisnick and Smyth, 2017). To distinguish the two we subsequently refer to $p(\mathcal{D})$ as the evidence and $p(\mathcal{D}|\boldsymbol{\phi})$ as the hierarchical likelihood.

The Stein force using the log hierarchical likelihood, which we call the hierarchical Stein force S_Φ^{H}, becomes

$$S_\Phi^{\mathrm{H}}(\boldsymbol{\phi}_i) = \mathbb{E}_{\boldsymbol{\phi}_j \sim q_\Phi} \left[k(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j) \nabla_{\boldsymbol{\phi}_j} \log \mathbb{E}_{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi}_j)} \left[\frac{p(\mathcal{D},\mathbf{z})}{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi}_j)} \right] + \nabla_{\boldsymbol{\phi}_j} k(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j) \right], \tag{2.3}$$

where q_Φ is an empirical measure analogous to $q_\mathcal{Z}$.

Inference converges (i.e. reaches a fixed point) when $S_\Phi^{\mathrm{H}}(\boldsymbol{\phi}_i) = 0$ for all particles, meaning all gradients in S_Φ^{H} must cancel (i.e. sum to zero). However, computing the gradient of the log-variational likelihood requires numerical estimation as analytical solutions do not exist for most models. Hence, we cannot expect the inference converges with noisy gradient estimations as the Stein force will compensate for the error in the gradient by a counterforce in the next iteration. Therefore, SM require good (i.e. low relative variance) gradient approximations; otherwise, the particles will fluctuate around a fixed point without reaching it. We demonstrate that replacing the log hierarchical likelihood with the ELBO can provide better (lower relative variance) gradient approximations. We call the new algorithm ELBO-within-Stein (EoS). We

connect EoS with the algorithm proposed by Nalisnick and Smyth, 2017 in terms of computing the gradient of different orders of the variational Rényi (VR) bound (Van Erven and Harremos, 2014). Similarly to the ELBO, the VR bound is a lower bound of the evidence, $p(\mathcal{D})$, also called the normalization constant, and is given by

$$p(\mathcal{D}) \geq \frac{1}{1-\alpha} \log \mathbb{E}_{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})} \left[\left(\frac{p(\mathcal{D}, \mathbf{z})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})} \right)^{1-\alpha} \right], \tag{2.4}$$

where $\alpha \geq 0$ is known as its order[4]. Understanding the inference of SM in terms of the VR bound yields insight into the behavior of the two algorithms, as we can now understand α as controlling the variance of each component of the guide. Furthermore, presuming accurate gradient approximation for all (viable) values of α, the connection leads to a family of inference algorithms indexed by the VR bound order.

After reviewing SVGD, the Rényi divergence, and the signal-to-noise ratio (SN-ratio) that is used to estimate the relative variance in Section 2.2, we make the following contributions:

- We demonstrate that inaccurate gradient estimates can lead to issues with convergence for SM.

- We introduce a new family of inference algorithms for SM indexed by the parameter α. The family results from connecting inference with SM to the Rényi α-divergence and includes the inference algorithm by Nalisnick and Smyth, 2017 as a special case for $\alpha = 0$.

- Unlike previous work, our algorithm allows for investigating a range of values for α for a model of interest. This allows us to investigate the convergence stability for different α's by measuring the SN-ratio. We find that $\alpha = 1$ is optimal for models with a latent variable for each data point (local latent variables), resulting in better SN-ratios than all other α values. For models where all datapoints share a latent variable (global latent variables), using $\alpha = 0.5$ (corresponding to the Hellinger distance) is on par with Nalisnick and Smyth, 2017's algorithm (which corresponds to $\alpha = 0$). Other values for α result in worse SN-ratios.

- We evaluate our inference algorithm for different values of α on Bayesian neural networks (BNNs) and variational autoencoders (VAEs), showing that the α that results in the highest performance varies depending on both model and data set.

[4]The VR bound can be extended to $\alpha \in \mathbb{R}$. We presume α is finite, but we allow α to be less than or equal to zero (Van Erven and Harremos, 2014)

- We describe a black-box inference algorithm for our proposed family of inference algorithms and provide a software library, called `EinSteinVI`, in NumPyro.

In Section 2.4 we discuss related work. We benchmark our algorithm in Section 2.5. Finally, we summarize our results in Section 2.6.

2.2 Background

Variational inference

Let $\mathbf{z}$ be a latent variable of interest-taking values in a space $\mathcal{Z} \subseteq \mathbb{R}^d$ (up to a diffeomorphism) and $\mathcal{D} = \{\mathbf{x}_i\}_{n \in I \subseteq \mathbb{N}}$ be a set of i.i.d. observations. For many models, exact Bayesian inference is computationally impracticable due to the cost of evaluating the evidence $p(\mathcal{D})$. Therefore, practitioners turn to tractable approximate variational inference (VI).

VI aims to bring a computationally cheap variational distribution $q(\mathbf{z}|\mathcal{D})$ close to the model posterior. Typically, we measure closeness by the Kullback-Leibler divergence (D_{KL}), i.e. $D_{\mathrm{KL}}\left[q(\mathbf{z}|\mathcal{D}) \parallel p(\mathbf{z}|\mathcal{D})\right]$. However, we generally avoid directly evaluating $D_{\mathrm{KL}}\left[q(\mathbf{z}|\mathcal{D}) \parallel p(\mathbf{z}|\mathcal{D})\right]$ as this requires evaluating the evidence, $p(\mathcal{D})$. We will concern ourselves with two types of VI.

The first type of VI searches for a parameterization $\boldsymbol{\psi}^*$ of q in a family of distributions $\mathcal{Q}$ that minimizes the divergence to the posterior. When the divergence is measured by D_{KL}, this type of VI is made tractable by maximizing the evidence lower bound (ELBO), that is

$$\begin{aligned}\boldsymbol{\psi}^* &= \arg\max_{\boldsymbol{\psi}} \left(\log p(\mathcal{D}) - D_{\mathrm{KL}} q(\mathbf{z}|\mathcal{D};\boldsymbol{\psi}) \parallel p(\mathbf{z}|\mathcal{D})\right) \\ &= \arg\max_{\boldsymbol{\psi}} \mathbb{E}_{q(\mathbf{z}|\mathcal{D})}\left[\log \frac{p(\mathcal{D},\mathbf{z})}{q(\mathbf{z}|\mathcal{D};\boldsymbol{\psi})}\right].\end{aligned}$$

The second type of VI we consider is particle-based methods and is the focus of this article. This type of VI relies on transporting a finite set of particles such that their empirical measure is close to the posterior. We will discuss this method in detail below.

Stein variational gradient descent (SVGD) The core idea of SVGD is to perform inference by approximating the target posterior distribution $p(\mathbf{z}|\mathcal{D})$ by an empirical distribution $q_{\mathcal{Z}}(\mathbf{z}) = N^{-1}\sum_i \delta_{\mathbf{z}_i}(\mathbf{z})$ based on a set of particles $\mathcal{Z}$, where $\mathcal{Z} = \{\mathbf{z}_i\}_{i=1}^N$. One could thus see the approximating distribution $q_{\mathcal{Z}}(\mathbf{z})$ as a (uniform) mixture of point estimates, each represented by a particle $\mathbf{z}_i \in \mathcal{Z}$. The SVGD algorithm minimizes the Kullback-Leibler divergence $D_{\mathrm{KL}} q_{\mathcal{Z}}(\mathbf{z}) \parallel p(\mathbf{z}|\mathcal{D})$ between the approximated and the true posterior by iteratively updating the particles using the following expression:

$$\mathbf{z}_{i+1} \leftarrow \mathbf{z}_i + \epsilon S_{\mathcal{Z}}(\mathbf{z}_i)$$

where ϵ is the learning rate and $S_{\mathcal{Z}}$ denotes the Stein force.

The two forces of SVGD The Stein force $S_{\mathcal{Z}}$ consists of two underlying forces that work additively, with $S_{\mathcal{Z}} = S^+_{\mathcal{Z}} + S^-_{\mathcal{Z}}$. The attractive force is given by

$$S^+_{\mathcal{Z}}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j \sim q_{\mathcal{Z}}}[k(\mathbf{z}_i,\mathbf{z}_j)\nabla_{\mathbf{z}_j} \log p(\mathbf{z}_j|\mathcal{D})]$$

and the repulsive force by

$$S_{\mathcal{Z}}^{-}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j \sim q_{\mathcal{Z}}}[\nabla_{\mathbf{z}_j} k(\mathbf{z}_i, \mathbf{z}_j)]. \tag{2.5}$$

Here $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel. The attractive force can be seen as pushing the particles towards the modes of the true posterior distribution, smoothed by some kernel. The repulsive force stops particles with high kernel values from collapsing onto each other.

For an example of a kernel, consider the radial basis function kernel (RBF) $k(\mathbf{z}_i, \mathbf{z}_j) = \exp\left(-\frac{1}{h} \parallel \mathbf{z}_i - \mathbf{z}_j \parallel_2^2\right)$ with bandwidth parameter h, chosen as $\frac{1}{\log N}\text{med}(\mathcal{Z})$, where med is the median operator.

The repulsive force moves particles away from each other, ensuring that they do not collapse onto the same mode. For the RBF kernel, the repulsive force becomes

$$\mathbb{E}_{\mathbf{z}_j \sim q_{\mathcal{Z}}}\left[\nabla_{\mathbf{z}_j} k(\mathbf{z}_i, \mathbf{z}_j)\right] = \sum_j -\frac{2}{h} k(\mathbf{z}_i, \mathbf{z}_j)\,(\mathbf{z}_i - \mathbf{z}_j) \cdot \mathbf{1}_d,$$

where $\cdot$ is the (euclidean) inner product and $\mathbf{1}_d$ is a d-dimensional one vector. It follows that $\mathbf{z}_i$ is pushed away from $\mathbf{z}_j$ when $k(\mathbf{z}_i, \mathbf{z}_j)$ is large. The computational cost of $S_{\mathcal{Z}}$ is quadratic in the size of $\mathcal{Z}$, i.e. $\mathcal{O}(N^2)$, which makes SVGD computationally burdensome for high-dimensional posteriors. For a particle method such as SVGD, the number of particles required to represent a posterior distribution adequately is inversely proportional to the dimensionality.

SVGD suffers from the curse of dimensionality (Ba et al., 2021), which results in variance collapse (i.e. variance is underestimated). Wang, Zeng, and Liu, 2018a demonstrates the problem with a simple factorized Gaussian, suggesting the (RBF) kernel introduces global statistical dependence driving the need for particles up for accurate representation. Ba et al., 2021 demonstrate that the collapse is due to the deterministic update of the attractive force. They do this by showing that re-sampling the particles at each iteration eliminates the underestimation of variance. Note that their particle re-sampling scheme by Ba et al., 2021 is not generally tractable; hence, it is not a practical solution.

Rényi divergence and the variational Rényi bound

The Rényi divergence (Rényi, 1961) is a family of divergences between distributions p and q indexed by the order parameter $\{\alpha | \alpha \in \mathbb{R}^+ / \{0, 1\}, |D_\alpha| < \infty\}$. The divergence is given by

$$D_\alpha\left[p \parallel q\right] = \frac{1}{\alpha - 1} \log \int p(\mathbf{z})^\alpha q(\mathbf{z})^{1-\alpha} d\mathbf{z}.$$

The Rényi divergence can be extended to $\alpha \in \{0, 1, \infty\}$ by continuity. In addition, if we allow for $D_\alpha[p \parallel q] \leq 0$, the order can be further extended to $\alpha \in \mathbb{R}$ (Van Erven and Harremos, 2014). Several orders correspond to known

divergences (see (Van Erven and Harremos, 2014) and (Li and Turner, 2016) for an overview). In particular, $\alpha = 1$ corresponds to D_{KL}.

Analogous to the use of the D_{KL} in the ELBO, D_α leads to a variational Rényi bound (Li and Turner, 2016) which, when formulated as used with SMs, is given by

$$\log p(\mathcal{D}) - D_\alpha \left[q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}) \parallel p(\mathbf{z}|\mathcal{D}) \right] = \tag{2.6}$$

$$\frac{1}{1-\alpha} \log \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi})} \left[\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})} \right)^{1-\alpha} \right]. \tag{2.7}$$

Note that model hyper-parameters ($\boldsymbol{\phi}$) in the variational posterior, $q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})$, are lifted to random variables when doing inference with SMs. See Section 2.7 for the derivation of Equation (2.6). Assuming reparameterization of $\mathbf{z}$ is possible, we can approximate the gradient $\Lambda(\boldsymbol{\phi})$ of Equation (2.6) using Monte Carlo integration by

$$\Lambda_K(\boldsymbol{\phi}) = \sum_{k=1}^{K} \boldsymbol{\omega}_k^\alpha(\mathbf{Z}, \mathcal{D}) \nabla_{\boldsymbol{\phi}} \log \left(\frac{p(\mathbf{Z}_k, \mathcal{D})}{q(\mathbf{Z}_k|\mathcal{D}, \boldsymbol{\phi})} \right), \quad \text{with } \mathbf{Z}_k \sim q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}), \tag{2.8}$$

where $K \in \mathbb{N}$ number of draws used to compute the VR bound and

$$\boldsymbol{\omega}_k^\alpha(\mathbf{z}, \mathcal{D}) = \frac{1}{C} \left(\frac{p(\mathcal{D}, \mathbf{z}_k)}{q(\mathbf{z}_k|\mathcal{D}, \boldsymbol{\phi})} \right)^{1-\alpha}, \text{ with } C = \sum_{i=1}^{K} \left(\frac{p(\mathcal{D}, \mathbf{z}_i)}{q(\mathbf{z}_i|\mathcal{D}, \boldsymbol{\phi})} \right)^{1-\alpha}. \tag{2.9}$$

We provide the derivation in Section 2.7.

The signal-to-noise ratio

The signal-to-noise (SN) ratio was introduced by Rainforth et al., 2018 to study the effect of tighter variational bounds on gradient estimation. The SNR is given by

$$\mathrm{SNR}_{M,K}(\boldsymbol{\phi}) = \left| \frac{\mathbb{E}\left[\Delta_{M,K}^\alpha(\boldsymbol{\phi}) \right]}{\sigma\left[\Delta_{M,K}^\alpha(\boldsymbol{\phi}) \right]} \right|, \tag{2.10}$$

where $\sigma[\cdot]$ is the standard deviation, $M, K \in \mathbb{N}$ are the number of Monte Carlo draws, and $\Delta_{M,K}^\alpha(\boldsymbol{\phi})$ derives from rewriting Equation (2.8) in the form

$$\Delta_{M,K}^\alpha(\boldsymbol{\phi}) = \frac{1}{1-\alpha} \frac{1}{M} \sum_{m}^{M} \nabla_{\boldsymbol{\phi}} \log \left[\frac{1}{K} \sum_{k=1}^{K} \left(\frac{p(\mathbf{Z}_{m,k}, \mathcal{D})}{q(\mathbf{Z}_{m,k}|\mathcal{D}, \boldsymbol{\phi})} \right)^{1-\alpha} \right]. \tag{2.11}$$

Here, we separate tightening the bound (by increasing K) from reducing the noise in the gradient estimation (by increasing M). If the rate at which the expected gradient decreases is faster than the rate of decrease of the variance, the gradient estimates worsen as K increases. The counter-intuitive implication is that a tighter bound can worsen the gradient estimation.

The Stein mixture

Variational inference with SMs (Nalisnick and Smyth, 2017) approximates the target posterior distribution $p(\mathbf{z}|\mathcal{D})$ by letting the Stein particles $\Phi = \{\boldsymbol{\phi}_i\}_{i=1}^N$ parameterize guide programs, $q(\mathbf{z}|\boldsymbol{\phi}_i, \mathcal{D})$. A SM yields a mixture marginal variational posterior, $p(\mathbf{z}|\mathcal{D}) \approx {}^1/_{|\Phi|} \sum_{\boldsymbol{\phi}\in\Phi} q(\mathbf{z}|\boldsymbol{\phi}, \mathcal{D})$, from which it takes its name. Formally, SM is a hierarchical variational model (HVM) (Ranganath, Tran, and Blei, 2016) with an empirical measure of particles q_Φ (defined in the same way as $q_\mathcal{Z}$) as its variational posterior, a uniform variational prior, and variational likelihood $\mathbb{E}_{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})}\left[{}^{p(\mathcal{D},\mathbf{z}|\boldsymbol{\phi})}/_{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})}\right]$. Similarly to SVGD, SM minimizes $D_{\mathrm{KL}}(q(\boldsymbol{\phi}) \parallel p(\boldsymbol{\phi}|\mathcal{D}))$ by iteratively transporting the particles according to the following expression

$$\boldsymbol{\phi}_{i+1} \leftarrow \boldsymbol{\phi}_i + \epsilon S_\Phi^{\mathrm{H}}(\boldsymbol{\phi}_i)$$

where $\epsilon \geq 0$ is the learning rate and S_Φ^{H} is the hierarchical Stein force.

The attractive force of SM Like SVGD, SM also makes use of two additive forces, $S_\Phi^{\mathrm{H}} = S_\Phi^{\mathrm{H}+} + S_\Phi^-$. The repulsive force S_Φ^- is the same as in SVGD, given by Equation (2.5). The attractive force is given by

$$S_\Phi^{\mathrm{H}+}(\boldsymbol{\phi}_i) = \mathbb{E}_{\boldsymbol{\phi}\sim q_\Phi}\left[k(\boldsymbol{\phi}_i, \boldsymbol{\phi})\nabla_{\boldsymbol{\phi}} \log \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi})}\left[\frac{p(\mathcal{D}, \mathbf{z})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right]\right],$$

where $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel. From the construction of SVGD, we require that the kernel has the reproducing property, so the kernel is dense in the space of continuous functions. If we choose Gaussian guides, the expected likelihood (EL) kernel (Jebara, Kondor, and Howard, 2004) is a natural choice because it accounts for the geometry of $q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}_j)$ and reduces to the RBF kernel for fixed variance, which is a reproducing kernel. The EL kernel is given by

$$k(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j) = \int q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}_i) q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}_j) d\mathbf{z} = \langle q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}_i), q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}_j)\rangle_{L_2},$$

where L_2 is an inner product and k is a positive definite kernel.

2.3 α-indexed SM inference and EwS

To see the connection between the hierarchical Stein force given in Equation (2.3) and the Rényi divergence, consider the gradient of the log hierarchical likelihood (that occurs in $S_\Phi^{\mathrm{H}+}$) and the VR bound given in Equation (2.6) for $\alpha = 0$. Presuming the support of the variational likelihood $q(\mathbf{z}|\boldsymbol{\phi})$ is a subset of the support of the prior of p, $\mathrm{supp}(q(\mathbf{z}|\boldsymbol{\phi})) \subseteq \mathrm{supp}(p(\mathbf{z}))$, the gradient of the log hierarchical likelihood is given by

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} \log p(\mathcal{D}|\boldsymbol{\phi}) &= \nabla_{\boldsymbol{\phi}} \log \mathbb{E}_q \left[\frac{p(\mathcal{D}, \mathbf{z})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right] && (\alpha = 0, \text{Equation (2.6)}) \\ &= \nabla_{\boldsymbol{\phi}} \left(\log p(\mathcal{D}) - D_{\alpha=0}[q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}) \parallel p(\mathbf{z}|\mathcal{D})]\right) \\ &= -\nabla_{\boldsymbol{\phi}} D_{\alpha=0}[q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}) \parallel p(\mathbf{z}|\mathcal{D})]. && (2.12)\end{aligned}$$

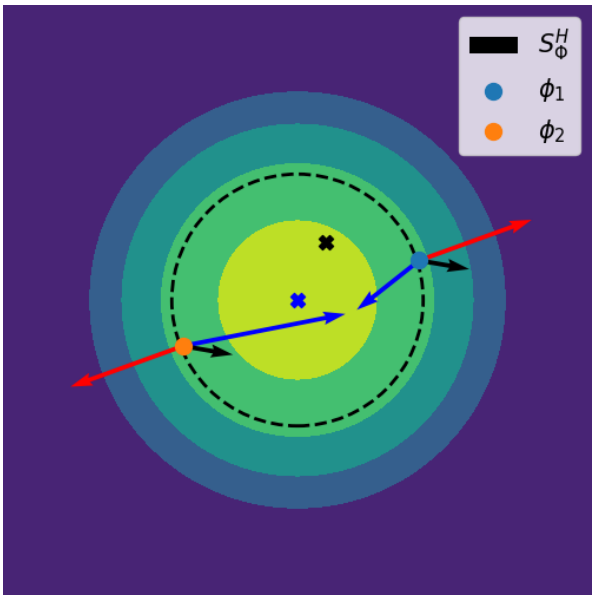
From Equation (2.12), we see that the gradient of the log marginal likelihood is exactly the gradient of the difference between the score $\log p(\mathcal{D})$, on the one hand, and the Rényi divergence (at $\alpha = 0$) between the variational posterior $q(\mathbf{z}|\boldsymbol{\phi})$ and the model posterior $p(\mathbf{z}\|\mathcal{D})$, on the other hand. Thus, Equation (2.12) shows that the attractive hierarchical force ($S_\Phi^{\mathrm{H}+}$) pushes the components of the variational posterior, $q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})$, towards the model posterior, $p(\mathbf{z}|\mathcal{D})$, see Section 2.9 for details. The equivalence in Equation (2.12) suggests a whole class of hierarchical attractive forces indexed by the order α of the VR bound. Note that choosing $\alpha \neq 0$ means we lose the interpretation of the attractive force as moving the particles towards the nearest peak of the conditional evidence. Assuming our marginal variational posterior $q(\mathbf{z}|\boldsymbol{\phi})$ is reparameterizable, we can approximate the attractive force for any $\alpha \geq 0$ as

$$S_\Phi^{\alpha+}(\boldsymbol{\phi}_i) = \mathbb{E}_{\boldsymbol{\phi} \sim q_\Phi} \left[k(\boldsymbol{\phi}_i, \boldsymbol{\phi}) \Lambda_K(\boldsymbol{\phi})\right], \tag{2.13}$$

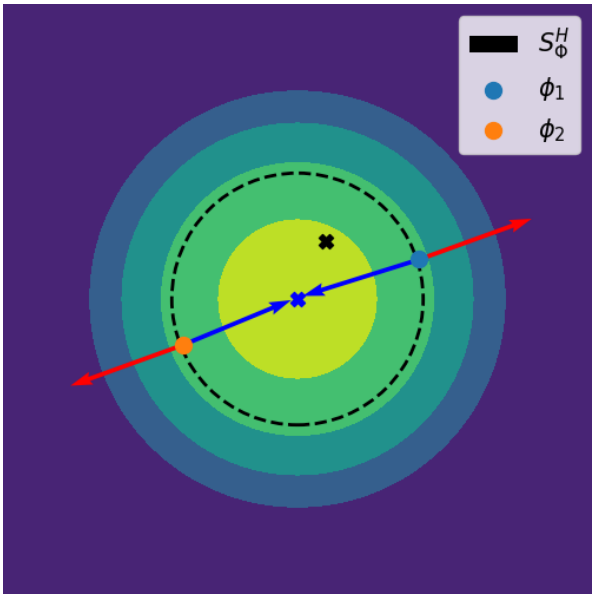
where $\Lambda_K(\phi)$ is given by Equation (2.8). We call inference with Equation (2.13) α-indexed SM inference. There are two special cases of α that are worth highlighting. The first is $\alpha = 1/2$, for which the Rényi divergence corresponds to the Hellinger divergence (Van Erven and Harremos, 2014; Li and Turner, 2016). The second is $\alpha = 1$, corresponding to the D_{KL}-divergence. The VR bound recovers the ELBO in this case. We call this instance of our α-indexed SM inference algorithm EwS. In Section 2.10, we show that we can recover the $\alpha = 1$ case directly by applying Jensen's inequality to the conditional evidence.

Investigating the signal-to-noise ratio

Estimation of a SM converges when $S_\Phi^H = 0$, meaning that the repulsive and attractive forces must be equal and opposite to cancel. Hence, convergence requires $\Delta_{M,K}^\alpha(\boldsymbol{\phi})$ and $\nabla_{\boldsymbol{\phi}_1} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2)$ to be accurate. In Figure 2.1, we demonstrate the effect of inaccurate gradient approximations. To study the sensitivity of gradient approximations to the choice of α, we measure the SN-ratio (see



(a) Low accuracy gradient approximation



(b) High accuracy gradient approximation

Figure 2.1: Two particle system at a theoretical fixed point. The blue arrows indicate the magnitude and direction of the attractive force, the red arrows show the repulsive force, and the black arrows the Stein force. Note that Figure 2.1b has no Stein force as expected for a converged system.

Equation (2.10)) of the VR bound gradients (see Equation (2.6)). We simulate data $\{\mathbf{x}_i\}_{i=1}^n$ from a simple latent variable model given by $\mathcal{N}(\mathcal{D}|\mathbf{z}, I_d)\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, I_d)$, where $\boldsymbol{\mu} \in \mathbb{R}^d$ is unknown and I_d is the d-dimensional identity matrix. To approximate its posterior, we use a SM of the form

$$\tfrac{1}{2}\left(\mathcal{N}(\boldsymbol{\phi}_1, \tfrac{3}{2}I_d) + \mathcal{N}(\boldsymbol{\phi}_2, \tfrac{3}{2}I_d)\right),$$

and an expected likelihood kernel. We choose a (computationally convenient) fixed variance such that the SM cannot exactly recover the posterior. We can see this as the posterior is unimodal, which is only the case for the SM if $|\boldsymbol{\phi}_1 - \boldsymbol{\phi}_2| < 3$ (Behboodian, 1970), but in this interval, the variance of the SM will be greater than or equal to $\tfrac{3}{2}$. With the expected likelihood kernel, we can analytically characterize all fixed points for the Stein particles as

$$-\nabla_{\boldsymbol{\phi}_1} \frac{1}{1-\alpha} \log \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi}_1)}\left[\left(\frac{p(\mathbf{z},\mathcal{D})}{q(\mathbf{z}|\boldsymbol{\phi}_1)}\right)^{1-\alpha}\right] = \nabla_{\boldsymbol{\phi}_2} \frac{1}{1-\alpha} \log \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi}_2)}\left[\left(\frac{p(\mathbf{z},\mathcal{D})}{q(\mathbf{z}|\boldsymbol{\phi}_2)}\right)^{1-\alpha}\right].$$

See Section 2.8 for the derivation. To measure the effect of gradient approximation on the system, we use Equation (2.11) to estimate the gradients.

To conduct our experiment, we sample the location $\boldsymbol{\mu}$ from a 20-dimensional standard Gaussian and use this $\boldsymbol{\mu}$ to simulate $n = 64$ data points $\mathcal{D}$. We then approximate the gradients at a random point close to a fixed point

$$(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) = \left(\frac{\boldsymbol{\mu} + n\overline{\mathcal{D}}}{n+1} + \nabla_{\boldsymbol{\phi}_1}\Delta^{\alpha}_{M,K}(\boldsymbol{\phi}_1) + \epsilon, \frac{\mu + n\overline{\mathcal{D}}}{n+1} + \nabla_{\boldsymbol{\phi}_2}\Delta^{\alpha}_{M,K}(\boldsymbol{\phi}_2)\right),$$

where $\overline{\mathcal{D}}$ is the data average, $\mu+n\overline{\mathcal{D}}/n+1$ is the posterior mean and ϵ offsets each dimension by a Gaussian with mean zero and variance 0.01.

Figure 2.2a illustrates the experimental setup in two dimensions. For legibility in Figure 2.2a, we do not include the perturbation (i.e., added ϵ noise) on $\boldsymbol{\phi}_1$ to the visualization. The contours correspond to the exact posterior. As the particles are placed equidistant from the posterior mean (marked with a blue cross), the Stein forces are zero in this setting. As we would expect (see blue arrows in Figure 2.2a), the gradient estimations of $\nabla_{\boldsymbol{\phi}}\Delta^{\alpha}_{M,K}$ are equal and opposite for the two particles.

For $\alpha \neq 1$ we fix $M = 1$ and vary K, while for $\alpha = 1$ we fix $K = 1$ and vary M. We do not need to consider $\alpha \neq 1$ when $K = 1$ as the associated gradient scaling $(1 - \alpha)$ cancels in the SN-ratio. We empirically estimate the SN-ratio by estimating the expectation and standard deviation from $10,000$ gradient samples. In Figure 2.2b we show a local variate of the model where there is a latent variable $\mathbf{z}_i$ for each $\mathbf{x}_i \in \mathcal{D}$. We see that for $\alpha \neq 1$ the SN-ratio does not depend on the particular choice of α and that that of $\alpha = 1$ supersedes

the growth in SN-ratio. This means there is little to no benefit in increasing K beyond $K = 1$, for which we recover the ELBO gradient when the guide is reparameterizable (see Section 2.2). In Figure 2.2c we evaluate a global latent variable variate of the model so that there are one $\mathbf{z}$ for all datapoints in $\mathcal{D}$. As with the local version, we fix either M or K. For this model, we see that $\alpha = 0$ and $\alpha = 1/2$ achieve the highest SN-ratio. The result aligns with the BNN example, where $\alpha = 1$ is not dominating in performance over the $\alpha \in \{0, 1/2\}$ on all datasets.

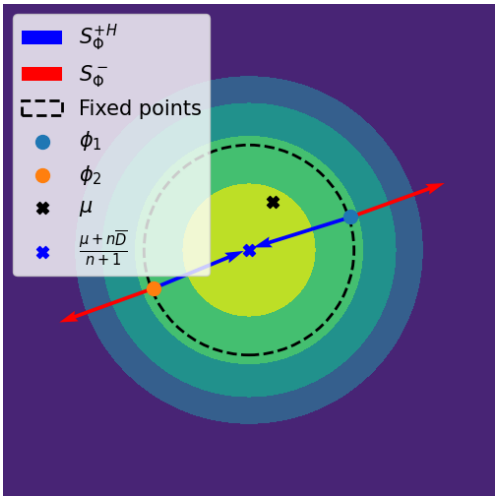
High precision is desirable to avoid fluctuation at convergence. The above results show that $\alpha = 0$ is not necessarily the best choice for precise (high SN-ratio) gradient estimation. In particular, for local latent variable models, $\alpha = 1$ is a better choice, and for that, for global latent models, $\alpha = 1/2$ is on par with $\alpha = 0$ in our experiment.

Black-box inference for EwS

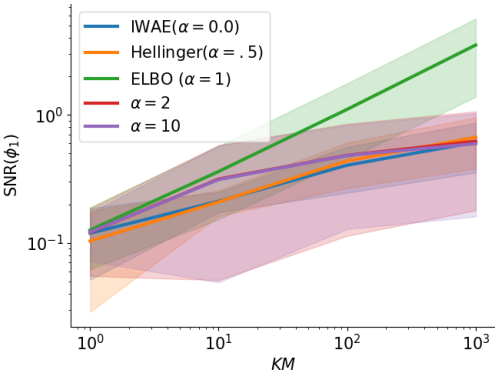
We provide a mini-batch version of EwS, called EinStein, in NumPyro. Computing the VR bound requires all the data points; that is, we cannot represent the bound as a point-wise expectation, except for $\alpha = 1$. Therefore, to make EinStein scalable to tall data, we provide an approximation of the VR bound which replaces the likelihood by $p_{\mathcal{I}}(\mathcal{D}|\mathbf{z}, \boldsymbol{\phi}) = \prod_{i \in \mathcal{I}} p(x_i|\mathbf{z}, \boldsymbol{\phi})^{|\mathcal{D}|/|\mathcal{I}|}$, where $\mathcal{I}$ is a subset of a permutation of the data indices. The approximate attractive force $S_\Phi^{\alpha+}(\boldsymbol{\phi})$ used in EinStein is given by

$$S_\Phi^{\alpha+}(\boldsymbol{\phi}_i) = \mathbb{E}_{\boldsymbol{\phi} \sim q_\Phi}\left[k(\boldsymbol{\phi}, \boldsymbol{\phi}_i)\nabla_{\boldsymbol{\phi}} \frac{1}{1-\alpha} \log \mathbb{E}_{q_{\mathcal{I}}(\mathbf{z}|\mathcal{D})}\left[\left(\frac{p_{\mathcal{I}}(\mathcal{D}|\mathbf{z}, \boldsymbol{\phi})p(\mathbf{z})}{q_{\mathcal{I}}(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right)^{1-\alpha}\right]\right], \quad (2.14)$$

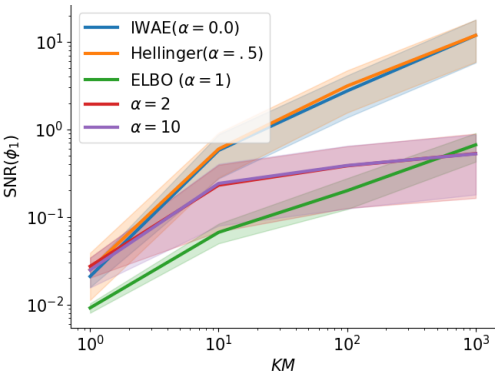
which recovers the exact VR bound when $|\mathcal{I}| = |\mathcal{D}|$. We describe the NumPyro integration and provide example programs in Chapter 3.



(a) Experimental setup.



(b) SN-ratio convergence with a local latent variable.



(c) SN-ratio convergence with a global latent variable.

Figure 2.2b and Figure 2.2c show the convergence of the SN-ratio (see Equation (2.10)) as we tighten the VR bound by increasing either K or M. We only show the SN-ratio for the first particle ($\boldsymbol{\phi}_1$) as the behaviour for the second particle is the same. For the ELBO ($\alpha = 1$), we fix $K = 1$ and increase M to reduce the gradient approximation variance, while for the rest ($\alpha \in \{0, 0.5, 2, 10\}$), we fix $M = 1$ and increase K to tighten the VR bound. In Figure 2.2b, there is a latent variable for each data point. Note how the SN-ratio only improves with tightness for $\alpha = 1$ (green line). In Figure 2.2c all data points share a latent variable.

2.4 Related Work

Nalisnick and Smyth, 2017 first suggested Stein mixtures as an alternative to HVMs (Ranganath, Tran, and Blei, 2016). Using SVGD allows Stein mixtures to side-step the need of HVMs for an auxiliary distribution to keep the bound (learning objective) tight. This is an improvement as the effect of the auxiliary distribution on the approximation is implicit and therefore hard to understand, whereas with Stein mixtures the choice of kernel controls the tightness and we have theoretical understanding of kernels (Wang et al., 2019b; Gorham and Mackey, 2017; Liu and Wang, 2018b).

Mixture approximations have a long history of work (Jaakkola and Jordan, 1998; Bishop et al., 1997; Gershman, Hoffman, and Blei, 2012; Miller, Foti, and Adams, 2017) focusing on approximating or lower-bounding the intractable mixture ELBO.

Van Erven and Harremos, 2014 unifies a number of variational techniques by considering them as optimizing different orders of the VR bound. They further demonstrate that two different variants of mini-batch training with the VR bound recover Stochastic EP (Li, Hernández-Lobato, and Turner, 2015) and Black-box α (Hernandez-Lobato et al., 2016), respectively.

The Rényi divergence has been studied in other forms under the name α-divergence (Amari, 2012; Tsallis, 1988). Hernandez-Lobato et al., 2016 introduced a black-box algorithm for variational inference based on the α-divergence using automatic differentiation. Unlike our algorithm, their algorithm is not for HVMs. Rainforth et al., 2018 demonstrated that for VAEs the gradient estimation degrades for multi-sample approximations when using the importance weighted variational autoencoder (IWAE) bound (Burda, Grosse, and Salakhutdinov, 2015). Furthermore, Rainforth et al., 2018 showed that this is not the case when using the ELBO. Rainforth et al., 2018 differs from our work in that the VAEs estimated are with a point mass guide, as their inference algorithm is not for HVMs.

Le et al., 2020 investigates the deterioration experimentally, providing evidence for it on several real world tasks. Tucker et al., 2018 show that by double reparameterizing the gradient estimator, they can eliminate the degrading SN-ratio for multi-sample estimation of the IWAE gradient, among others.

Table 2.1: Average RMSE (lower is better) test results for BNNs on UCI benchmark. EoS (ours) corresponds $\alpha = 1$, Hell (ours) to $\alpha = 0.5$, and SM (Nalisnick and Smyth, 2017) to $\alpha = 0$. Parentheses mean we use Dirac delta guides. EoS and Hell gives the best results.

	Average Root Mean Squared Deviation (RMSE)					
Dataset	EoS	Hell	SM	SVGD	MFVI	Laplace
Boston	3.5 ± 0.82 (2.8 ± 0.4)	3.92 ± 1.3 $(\mathbf{2.71 \pm 0.26})$	3.9 ± 1.29 (2.715 ± 0.263)	2.86 ± 0.23	3.28 ± 0.1	3.68 ± 0.33
Concrete	5.76 ± 0.55 $(\mathbf{4.61 \pm 0.34})$	6.55 ± 0.63 (5.2 ± 0.3)	6.51 ± 0.59 (5.23 ± 0.32)	5.54 ± 0.33	5.6 ± 0.3	5.22 ± 0.43
Energy	0.53 ± 0.05 $(\mathbf{0.45 \pm 0.03})$	0.94 ± 0.15 (0.67 ± 0.03)	0.81 ± 0.16 (0.74 ± 0.05)	1.30 ± 0.08	1.75 ± 0.15	0.46 ± 0.03
Naval	0.04 ± 0.04 $(\mathbf{0.00 \pm 0.00})$	0.004 ± 0.001 (0.001 ± 0.00)	0.004 ± 0.002 0.001 ± 0.000	0.007 ± 0.000	$\mathbf{0.000 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
Wine	0.6 ± 0.038 $(\mathbf{0.07 \pm 0.00})$	0.61 ± 0.03 (0.08 ± 0.00)	0.61 ± 0.03 0.08 ± 0.00	0.62 ± 0.04	0.59 ± 0.04	0.61 ± 0.01
Yacht	1.76 ± 0.41 $(\mathbf{0.45 \pm 0.03})$	1.66 ± 0.65 (0.67 ± 0.03)	1.61 ± 0.5 (0.74 ± 0.05)	1.11 ± 0.3	4.09 ± 0.34	2.16 ± 0.37
Power	4.04 ± 0.16 $(\mathbf{3.91 \pm 0.18})$	4.15 ± 0.21 (3.98 ± 0.19)	4.16 ± 0.21 (3.97 ± 0.2)	4.06 ± 0.17	3.94 ± 0.18	3.99 ± 0.17

Table 2.2: Summary statistics of datasets from the UCI regression benchmark.

Dataset	Data points	Feature count
Boston (Harrison Jr and Rubinfeld, 1978)	506	13
Concrete (Yeh, 1998)	1030	8
Energy (Tsanas and Xifara, 2012)	768	8
Power (Tüfekci, 2014)	9568	4
Protein (Rana, 2013)	45730	9
Year (Bertin-Mahieux et al., 2011)	515345	90

2.5 Examples

We evaluate α-indexed Stein mixture inference by inferring BNNs and VAE on standard datasets. We use the BNNs for regression on the UCI regression benchmark (the same as Hernández-Lobato and Adams, 2015) and VAE for unsupervised learning on MNIST (Salakhutdinov and Murray, 2008; LeCun et al., 1998) and OMNIGLOT (Lake, Salakhutdinov, and Tenenbaum, 2013).

Bayesian neural networks For brevity, we present BNNs for the subset of the UCI regression benchmark

We compare ELBO-within-Stein for $\alpha \in \{0, 0.5, 1\}$ on BNNs regression point mass (Dirac delta) guide and a RBF kernel. With EwS we recover a variant of SVGD with a VR gradient rather than the score function. Like Liu and Wang, 2016b, we use a BNN with one hidden layer of size fifty and a `RELU` activation. We put a Gamma$(1, 0.1)$ prior on the precision of the neurons and the likelihood. For both versions we use 5 particles and update Year for 40

epochs, Protein for 100 epochs and 500 epochs for the rest. We use Adagrad (Duchi, Hazan, and Singer, 2011) with a step size of 0.05 and a subsample size of 100. All measurements are repeated five times and obtained on a GPU[5].

All datasets use real-valued features. We use a 90-10 split for training and test datasets. We compare α-indexed SM inference for $\alpha \in \{0, 0.5, 1\}$ on BNN regression. We test with two guides: factorized Gaussian guides with an EL kernel and point mass (Dirac delta) guides with an RBF kernel. Like Liu and Wang, 2016b, we use a BNN with one hidden layer of size fifty and a `RELU` activation. We put a Gamma$(1, 0.1)$ prior on the precision of the neurons and the likelihood. We use five particles for all experiments. We run all datasets for 35,000 epochs with a subsample size of 32, the Adam optimizer (Kingma and Ba, 2014) and a step size of 0.002. All measurements are repeated three times and obtained on a GPU[6].

We compare against the SVGD implantation from Liu, Lee, and Jordan, 2016 with 20 particles, mean field variational Bayes (MFVI) with a factorized Gaussian guide (Hoffman et al., 2013) and Laplace approximation. For the latter two we inference engines from NumPyro (Phan, Pradhan, and Jankowiak, 2019b).

Table 2.1 shows the root mean squared error (RMSE) on test sets. EoS with delta Guides out performance baselines and other α-orders, except on Boston. SM and Hell perform similarly with factorized Gaussian guides, which aligns with our SN-ratio experiment that shows the gradient approximations are similar for these two cases. Note the Stein mixtures use only five particles, whereas SVGD uses twenty. Table 2.3 gives the log-likelihood on the same test sets. EoS achieves better average log-likelihood for all datasets with factorized Gaussian guides than other α-orders. We see that $\alpha = 0$ and $\alpha = .5$ performs similarly with a factorized Gaussian prior, which aligns with our SN-ratio experiment in that the quality of gradient approximations is similar.

Variational autoencoder We evaluate Stein mixtures and SVGD for VAEs on two datasets with $\alpha \in \{0, 0.5, 1\}$. We use binarized MNIST (Salakhutdinov and Murray, 2008; LeCun et al., 1998), a dataset of 28×28 pixel images of handwritten single digit numbers, and a variate of OMNIGLOT (Lake, Salakhutdinov, and Tenenbaum, 2013), which contains 28×28 pixel images of characters from fifty different alphabets. We use the same VAE architecture as Burda, Grosse, and Salakhutdinov, 2015.

Following Li and Turner, 2016; Burda, Grosse, and Salakhutdinov, 2015; Rainforth et al., 2018, we use VAEs with multiple stochastic layers. The idea is to define the model through ancestral sampling as

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{\mathbf{z}_1,\ldots,\mathbf{z}_L} p(\mathbf{z}_L)p(\mathbf{z}_{L-1}|f_{\boldsymbol{\theta}_{L-1}}(\mathbf{z}_L))\ldots p(\mathbf{x}|f_{\boldsymbol{\theta}_0}(\mathbf{z}_1)),$$

[5]Quadro RTX 6000 with Cuda V11.4.120
[6]Quadro RTX 6000 with Cuda V11.4.120

where $\mathbf{x}$ is a data-point (which we will also denote $\mathbf{z}_0$), $\mathbf{z}_1, \ldots, \mathbf{z}_L$ are the L stochastic layers, and $\boldsymbol{\theta}_l$ parameterizes a neural network f_l which takes $\mathbf{z}_{l+1}$ to the parameters of the distribution p_l, i.e. $p(\mathbf{z}_l|f_l(\mathbf{z}_{l+1})$. We then let the guide factor in the opposite direction, resulting in

$$q(\mathbf{z}|\boldsymbol{\phi}, \mathbf{x}) = q(\mathbf{z}_1|f_{\boldsymbol{\phi}_1}(\mathbf{x}))q(\mathbf{z}_2|f_{\boldsymbol{\phi}_2}(\mathbf{z}_1)) \ldots p(\mathbf{z}_L|f_{\boldsymbol{\phi}_L}(\mathbf{z}_{L-1})).$$

We use the same network architecture as Rainforth et al., 2018 (summarized in Table 2.4). In Table 2.4, $\mathbf{s}$ denotes a stochastic layer and $\mathbf{d}$ denotes a deterministic layer (affine transforms). We use `tanh` as the activation functions on deterministic layers. Stochastic layers distribute according to a factorized Gaussian distribution, and for the likelihood, we use the Bernoulli distribution (hence the binarization of the datasets).

For both datasets we optimize using the Adam optimizer and learning rate of $5 \cdot 10^{-4}$. We optimize with a batch size of 20 and use 20 draws to approximate the gradients. For OMNIGLOT we use 20 epochs and for MNIST we use 50. Table 2.5 show the performance of EwS for $\alpha \in \{0.0, 0.5, 1.0\}$. We find that EwS with $\alpha = 1$ achieves better log-likelihoods on MNIST datasets. On OMNIGLOT $\alpha = 0.5$ and $\alpha = 0$ achieve comparable log-likelihood with $\alpha 0$ the slightly outperforming $\alpha = 0.5$.

Table 2.3: Average log likelihood (higher is better) test results for BNNs on UCI benchmark. EoS (ours) corresponds $\alpha = 1$, Hell (ours) to $\alpha = 0.5$, and SM (Nalisnick and Smyth, 2017) to $\alpha = 0$. Parentheses mean we use Dirac delta guides. EoS generally outperforms Hell and SM.

	Average log-likelihood					
Dataset	EoS	Hell	SM	SVGD	MFVI	Laplace
Boston	-0.66 ± 0.34	-0.79 ± 0.32	-0.79 ± 0.33	-2.55 ± 0.08	-0.74 ± 0.04	$\mathbf{-0.56 \pm 0.03}$
	(-0.64 ± 0.27)	(-1.50 ± 1.34)	(-1.48 ± 1.32)			
Concrete	-0.58 ± 0.25	-1.02 ± 0.37	-1.01 ± 0.37	-3.18 ± 0.06	-0.54 ± 0.11	-0.96 ± 0.49
	$(\mathbf{-0.5 \pm 0.09})$	(-1 ± 0.27)	(-1 ± 0.29)			
Energy	0.02 ± 0.76	-0.10 ± 0.63	-0.13 ± 0.70	-1.76 ± 0.03	0.06 ± 0.12	$\mathbf{0.31 \pm 0.70}$
	(-0.25 ± 0.81)	(-3.02 ± 1.64)	(-1.61 ± 1.50)			
Naval	1.72 ± 0.84	-0.54 ± 0.49	-0.99 ± 0.78	$\mathbf{3.46 \pm 0.04}$	1.48 ± 1.37	2.05 ± 0.93
	(-21.40 ± 19.71)	(-5.32 ± 0.96)	(-4.80 ± 1.04)			
Wine	-1.38 ± 0.01	-1.35 ± 0.04	-1.35 ± 0.04	$\mathbf{-0.96 \pm 0.5}$	-1.26 ± 0.07	-1.52 ± 0.08
	(-1.23 ± 0.05)	(-1.43 ± 0.11)	(-1.43 ± 0.10)			
Yacht	$\mathbf{0.08 \pm 0.46}$	-0.26 ± 0.53	-0.27 ± 0.51	-2.11 ± 0.63	-0.83 ± 0.05	-0.12 ± 0.10
	(-0.20 ± 0.58)	(-2.83 ± 2.19)	(-2.89 ± 2.17)			
Power	$\mathbf{0.04 \pm 0.06}$	-0.08 ± 0.10	-0.08 ± 0.11	-2.83 ± 0.03	0.03 ± 0.06	0.02 ± 0.07
	(-0.20 ± 0.58)	(-0.23 ± 0.18)	(-0.23 ± 0.15)			

Table 2.4: Variational autoencoder architecture for MNIST and OMNIGLOT. **s** denotes a stochastic layer and **d** denotes a deterministic layer. Read the networks left-to-right for guide description and right-to-left for model description.

Dataset	Architecture	Activation
MNIST	**d**200-**d**200-**s**50	`tanh`
OMNIGLOT	**d**200-**d**200-**s**100-**d**100-**d**100-**s**50	`tanh`

Table 2.5: Log likelihood (higher is better) test results for VAE. EwS (ours) corresponds $\alpha = 1$, Hell (ours) to $\alpha = 0.5$, and SM (Nalisnick and Smyth, 2017) to $\alpha = 0$.

Dataset	SM	Hell	EwS
MNIST	-101.874	-100.541	-77.400
OMNIGLOT	-146.241	-146.257	-148.428

2.6 Summary

We introduce a new algorithm called EwS based on a new connection between the inference of Stein mixtures and the Rényi variational bound. We demonstrate that EwS provides better gradient approximations than alternative algorithms, which results in better performance for standard benchmark problems. EwS is integrated as a black box library in the NumPyro PPL which is distributed freely. We detail this library in Chapter 3.

2.7 Variational Rényi bound

For convenience, we derive the variational Rényi bound (Li and Turner, 2016) in the context of inference with our algorithm below. Recall that Stein mixtures lift the guide hyper-parameters $\boldsymbol{\phi}$ (optimized in VI) to a random variable $\boldsymbol{\phi}$. Let $\mathcal{D}$ be a finite set of observations, $\mathbf{z} \in \mathbb{R}^d$ be a latent variable, $D_\alpha\left[q||p\right]$ the Rényi α-divergence (Rényi, 1961) between distributions p and q, and $\boldsymbol{\phi} \in \mathbb{R}^d$ a set of guide hyper-parameters lifted to a random variable. Then we have

$$\log p(\mathcal{D}) - D_\alpha\left[q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})\|p(\mathbf{z}|\mathcal{D})\right] = \frac{1}{1-\alpha}\log \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi})}\left[\left(\frac{p(\mathbf{z},\mathcal{D})}{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})}\right)^{1-\alpha}\right]. \tag{2.15}$$

To see this, consider,

$$\begin{aligned}
D_\alpha\left[q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})\|p(\mathbf{z}|\mathcal{D})\right] &= \frac{1}{\alpha-1}\log\int q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})^\alpha p(\mathbf{z}|\mathcal{D})^{1-\alpha}d\mathbf{z} \\
&= \frac{1}{\alpha-1}\log\int q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})^\alpha \left(\frac{p(\mathbf{z},\mathcal{D})}{p(\mathcal{D})}\right)^{1-\alpha} d\mathbf{z} \\
&= \frac{1}{\alpha-1}\log\int q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})^\alpha p(\mathbf{z},\mathcal{D})^{1-\alpha}d\mathbf{z}\cdot p(\mathcal{D})^{\alpha-1} \\
&= \frac{1}{\alpha-1}\log\int q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})^\alpha p(\mathbf{z},\mathcal{D})^{1-\alpha}d\mathbf{z} + \frac{\alpha-1}{\alpha-1}\log p(\mathcal{D}) \\
&= \frac{1}{\alpha-1}\log\int q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})^{-(1-\alpha)} p(\mathbf{z},\mathcal{D})^{1-\alpha} + \log p(\mathcal{D}) \\
&= \frac{1}{\alpha-1}\log \mathbb{E}_{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})}\left[\left(\frac{p(\mathbf{z},\mathcal{D})}{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})}\right)^{1-\alpha}\right] + \log p(\mathcal{D}),
\end{aligned}$$

from which we recover the desired equality by rearranging and multiplying both sides by negative one,

$$\log p(\mathcal{D}) - D_\alpha(q(\mathbf{z}|\boldsymbol{\phi})\|p(\mathbf{z}|\mathcal{D})) = \frac{1}{1-\alpha}\log \mathbb{E}_{q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})}\left[\left(\frac{p(\mathbf{z},\mathcal{D})}{q(\mathbf{z}|\boldsymbol{\phi})}\right)^{1-\alpha}\right].$$

To shorten notation, we let $C^\alpha(\mathcal{D},\boldsymbol{\phi}) = \mathbb{E}\left[\left(p(\mathbf{z},\mathcal{D})/q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})\right)^{1-\alpha}\right]$ where the expectation is with respect to $q(\mathbf{z}|\mathcal{D},\boldsymbol{\phi})$. The gradient of the variational Rényi

bound with respect to $\boldsymbol{\phi}$ is

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \frac{1}{1-\alpha} \log \mathbb{E}\left[\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right)^{1-\alpha}\right] &= \frac{1}{1-\alpha} C^{\alpha}(\mathcal{D}, \boldsymbol{\phi})^{-1} \mathbb{E}\left[\nabla_{\boldsymbol{\phi}}\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right)^{1-\alpha}\right] \\
&= C^{\alpha}(\mathcal{D}, \boldsymbol{\phi})^{-1} \mathbb{E}\left[\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right)^{1-\alpha} \nabla_{\boldsymbol{\phi}} \log\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right)\right] \\
&= \mathbb{E}\left[\frac{\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right)^{1-\alpha}}{C^{\alpha}(\mathcal{D}, \boldsymbol{\phi})^{-1}} \nabla_{\boldsymbol{\phi}} \log\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right)\right] \\
&= \mathbb{E}\left[\omega^{\alpha}(\mathbf{z}, \mathcal{D}) \nabla_{\boldsymbol{\phi}} \log\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})}\right)\right],
\end{aligned}$$

where

$$\omega^{\alpha}(\mathbf{z}, \mathcal{D}) = \frac{\left(p(\mathbf{z}, \mathcal{D}) / q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})\right)^{1-\alpha}}{\mathbb{E}\left[\left(p(\mathbf{z}, \mathcal{D}) / q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})\right)^{1-\alpha}\right]}.$$

2.8 Characterizing two particle fixed points

We give the full derivation of stationary points for the Stein mixture in Section 2.3. Recall that Section 2.3 investigated the SN-ratio for a Stein mixture given by

$$\frac{1}{2}\left(\mathcal{N}(\boldsymbol{\phi}_1, {}^3\!/_2 I_d) + \mathcal{N}(\boldsymbol{\phi}_2, {}^3\!/_2 I_d)\right),$$

where $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2 \in \mathbb{R}^d$ are two d-dimensional particles. We use the kernel given by

$$k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) = \exp\left(-\frac{1}{h}\|\boldsymbol{\phi}_1 - \boldsymbol{\phi}_2\|_2^2\right), \tag{2.16}$$

where $h \in \mathbb{R}^+$ is the bandwidth. The kernel has the following properties:

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}_1} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) &= -\nabla_{\boldsymbol{\phi}_2} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2), \\
k(\cdot, \cdot) &= 1, \\
k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) &= k(\boldsymbol{\phi}_2, \boldsymbol{\phi}_1), \\
\nabla_{\boldsymbol{\phi}} k(\cdot, \cdot) &= 0,
\end{aligned}$$

which we will use in the derivation. Finally, we introduce

$$\xi_\alpha(\boldsymbol{\phi}) = \frac{1}{1-\alpha} \log \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi})}\left[\left(\frac{p(\mathbf{z}, \mathcal{D})}{q(\mathbf{z}|\boldsymbol{\phi})}\right)^{1-\alpha}\right]$$

as notation short-hand.

Our two-particle configuration reaches a fixed point when

$$(\boldsymbol{\phi}_1 + \epsilon S_\Phi^H(\boldsymbol{\phi}_1), \boldsymbol{\phi}_2 + \epsilon S_\Phi^H(\boldsymbol{\phi}_2)) = (\boldsymbol{\phi}_1, \boldsymbol{\phi}_2),$$

where $\epsilon \geq 0$ is the step size. Therefore, $S_\Phi^H(\boldsymbol{\phi}_1) = 0$ and $S_\Phi^H(\boldsymbol{\phi}_2) = 0$ at any fixed point. $S_\Phi^H(\boldsymbol{\phi}_1)$ is given by

$$\begin{aligned}
S_\Phi^H(\boldsymbol{\phi}_1) &= \overbrace{k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_1)}^{1} \nabla_{\boldsymbol{\phi}_1} \xi_\alpha(\boldsymbol{\phi}_1) + \overbrace{\nabla_{\boldsymbol{\phi}_1} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_1)}^{0} + k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \nabla_{\boldsymbol{\phi}_2} \xi_\alpha(\boldsymbol{\phi}_2) + \nabla_{\boldsymbol{\phi}_2} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \\
&= \nabla_{\boldsymbol{\phi}_1} \xi_\alpha(\boldsymbol{\phi}_1) + k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \nabla_{\boldsymbol{\phi}_2} \xi_\alpha(\boldsymbol{\phi}_2) + \nabla_{\boldsymbol{\phi}_2} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) = 0.
\end{aligned}$$

Therefore,

$$-\nabla_{\boldsymbol{\phi}_2} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) = \nabla_{\boldsymbol{\phi}_1} \xi_\alpha(\boldsymbol{\phi}_1) + k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \nabla_{\boldsymbol{\phi}_2} \xi_\alpha(\boldsymbol{\phi}_2) \tag{2.17}$$

at a fixed point. By a similar argument for $S_\Phi^H(\boldsymbol{\phi}_2)$, we have

$$\nabla_{\boldsymbol{\phi}_1} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) = -(\nabla_{\boldsymbol{\phi}_2} \xi_\alpha(\boldsymbol{\phi}_2) + k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \nabla_{\boldsymbol{\phi}_1} \xi_\alpha(\boldsymbol{\phi}_2)) \tag{2.18}$$

at a fixed point. As $\nabla_{\boldsymbol{\phi}_1} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) = -\nabla_{\boldsymbol{\phi}_2} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2)$, it follows from Equations (2.17) and (2.18) that

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}_1} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) &= -\nabla_{\boldsymbol{\phi}_2} k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \\
-(\nabla_{\boldsymbol{\phi}_2} \xi_\alpha(\boldsymbol{\phi}_2) + k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \nabla_{\boldsymbol{\phi}_1} \xi_\alpha(\boldsymbol{\phi}_2)) &= \nabla_{\boldsymbol{\phi}_1} \xi_\alpha(\boldsymbol{\phi}_1) + k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \nabla_{\boldsymbol{\phi}_2} \xi_\alpha(\boldsymbol{\phi}_2) \\
-\nabla_{\boldsymbol{\phi}_2} \xi_\alpha(\boldsymbol{\phi}_2)(1 + k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2)) &= \nabla_{\boldsymbol{\phi}_1} \xi_\alpha(\boldsymbol{\phi}_1)(1 + k(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2)) \\
-\nabla_{\boldsymbol{\phi}_2} \xi_\alpha(\boldsymbol{\phi}_2) &= \nabla_{\boldsymbol{\phi}_1} \xi_\alpha(\boldsymbol{\phi}_1).
\end{aligned}$$

Hence, we see that at any fixed point for our two particle configuration, the gradients of the VR-bound are equal and opposite.

2.9 Conditional evidence as Rényi divergence between posteriors

That Equation (2.12) pushes posteriors towards each other follows from properties of Rényi divergence with $\alpha \in [0, 1]$ and the negative direction of the gradient on the Rényi divergence. In particular, we have (i) that the divergence is a similarity measure of distributions for $\alpha \geq 0$ so $D_{\alpha=0}[q||p] = 0 \implies q = p$, (ii) that $D_\alpha[q \parallel p]$ is everywhere positive, and (iii) the divergence is jointly convex (i.e. convex in both distributions) (Van Erven and Harremos, 2014). Putting it all together, we see from (ii) and (iii) that the extremum at $D_{\alpha=0}[q||p] = 0$ is global and from the (negative) gradient we are minimizing $D_{\alpha=0}[q \parallel p]$. So, we have that $-\nabla_{\boldsymbol{\phi}} D_{\alpha=0}[q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}) \parallel p(\mathbf{z}|\mathcal{D})] = 0 \implies D_{\alpha=0}[q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}) \parallel p(\mathbf{z}|\mathcal{D})] = 0$ which from (i) means $q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}) = p(\mathbf{z}|\mathcal{D})$.

2.10 Alternative ELBO-within-Stein derivation

For $\alpha = 1$ we can derive the attractive force of S_Φ^H directly by applying Jensen's inequality to the log conditional evidence, resulting in

$$\nabla_{\boldsymbol{\phi}} \log \mathbb{E}_q \left[\frac{p(\mathcal{D}, \mathbf{z}|\boldsymbol{\phi})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})} \right] \geq \nabla_{\boldsymbol{\phi}} \mathbb{E}_q \left[\log \frac{p(\mathcal{D}, \mathbf{z}|\boldsymbol{\phi})}{q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi})} \right]. \tag{2.19}$$

In ELBO-within-Stein, the attractive force takes the simple form

$$S_\Phi^{\mathrm{ELBO}+}(\boldsymbol{\phi}_i) = \\ \mathbb{E}_{\boldsymbol{\phi} \sim q_\Phi} \left[k(\boldsymbol{\phi}_i, \boldsymbol{\phi}) \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi})} \left[\log p(\mathcal{D}, \mathbf{z}|\boldsymbol{\phi}) \right] - k(\boldsymbol{\phi}_i, \boldsymbol{\phi}) \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q(\mathbf{z}|\boldsymbol{\phi})} \left[q(\mathbf{z}|\mathcal{D}, \boldsymbol{\phi}) \right] \right], \tag{2.20}$$

and the repulsive force is given by Equation (2.5).

2.11 Illustrating stein mixtures

To illustrate the use of an SM, consider the VAE. The VAE simultaneously trains a generative model $p(\mathcal{D}|g_{\boldsymbol{\theta}}(\mathbf{z}))p(\mathbf{z})$ and a variational approximation $q(\mathbf{z}|f_{\boldsymbol{\psi}}(\mathcal{D}))$ of the posterior $p(\mathbf{z}|\mathcal{D})$. Here, $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$ are parameters of the generative neural network $g_{\boldsymbol{\theta}}(\cdot)$ and the inference network $f_{\boldsymbol{\psi}}(\cdot)$, respectively. VAE training is typically done by stochastic variational inference (Hoffman et al., 2013) (SVI) which optimizes $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$ to minimize the ELBO. With a SM, the generative model remains the same, that is, we obtain a point estimate of $\boldsymbol{\theta}$. However, the marginal posterior approximation changes to $1/|\Phi| \sum_{\boldsymbol{\phi}\in\Phi} q(\mathbf{z}|f_{\boldsymbol{\phi}}(\mathcal{D}))$. So with a Stein mixture, each particle $\boldsymbol{\phi}$ parameterizes a separate inference network, i.e. $f_{\boldsymbol{\phi}}(\cdot)$, meaning the guide becomes amortized similar to Shu et al., 2018.

Chapter 3

EinStein: General and Integrated SteinVI

Ola Rønning[1], Christophe Ley[2], Ahmad Salim Al-Sibah[1] and Thomas Hamelryck[13]

3.1 Introduction

Interest in Bayesian deep learning has surged due to the need for quantifying the uncertainty of predictions obtained from machine learning algorithms (Wilson and Izmailov, 2020; Wilson, 2020). Bayesian inference aims to describe observed data $\mathcal{D}$ using a model with latent variables $\mathbf{z}$. The goal is to infer a posterior distribution $p(\mathbf{z}|\mathcal{D})$ over the latent variables given a model describing the joint distribution $p(\mathbf{z}, \mathcal{D}) = p(\mathcal{D}|\mathbf{z})p(\mathbf{z})$. We obtain the posterior by following the rules of Bayesian inference:

$$p(\mathcal{D}|\mathbf{x}) = Z^{-1}p(\mathcal{D}|\mathbf{z})p(\mathbf{z})$$

where $Z = \int_{\mathbb{Z}} p(\mathcal{D}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}$ is the normalization constant. For most practical models, the normalization constant lacks an analytic solution or requires an infeasible number of computations, complicating the Bayesian inference problem.

SVGD is a recent SteinVI technique that uses a set of particles $\{\mathbf{z}_i\}_{i=1}^N$ as the approximating distribution $q(\mathbf{z})$. SVGD is well suited for capturing correlations between latent variables as a particle-based method. The technique preserves the scalability of traditional VI approaches while offering the flexibility and modeling scope of techniques such as MCMC. SVGD is good at capturing multi-modality (Liu and Wang, 2016b; Wang and Liu, 2019b), and has useful

[1]Department of Computer Science, University of Copenhagen, Denmark
[2]Département Mathématiques, Université du Luxembourg, Luxembourg
[3]Department of Biology, University of Copenhagen, Denmark

theoretical interpretations such as a set of particles following a gradient flow (Liu, 2017) or in terms of the properties of kernels (Liu and Wang, 2018b).

This article details the SteinVI library called EinStein in the PPL `NumPyro` (Bingham et al., 2019; Phan, Pradhan, and Jankowiak, 2019b). SteinVI uses EwS described in Chapter 2 as its core inference algorithm.

NumPyro is a universal PPL (Meent et al., 2018) embedded in Python. NumPyro provides specialized constructs for expressing probabilistic models as python programs and allows executing arbitrary code in its model and guide. The computational backend of NumPyro is `Jax` (Frostig, Johnson, and Leary, 2018), which gives access to an accelerated linear algebra (Sabne, 2020) program transformations and automatic differentiation through the `Jax` compiler in Python. As SteinVI works with arbitrary guides, NumPyro is a well-suited language for embedding SteinVI. Further, we chose NumPyro because:

- `NumPyro` is embedded in Python, the de-facto programming language for data science;
- `NumPyro` includes the necessary data structures for tracking random variables in both model and guide;
- `NumPyro` features stochastic variational inference with an API that is highly suitable for SteinVI; and
- `NumPyro` benefits computationally from `Jax`.

After reviewing the SteinVI methods included in EinStein, we make the following contributions:

- We detail our black-box inference library of ELBO-within-stein in NumPyro, which we call the library SteinVI. SteinVI allows SMs to work with custom guide programs based on EwS optimization. The library is compositional with NumPyro features, including support for deep learning, loss functions (ELBO, Rényi ELBO (Li and Turner, 2016)), and optimization methods, thus making it possible for SteinVI to grow organically with NumPyro development.
- In Section 3.6 we provide example programs for inference with regression and time series data, and investigate SteinVI as a sampling method.

In Section 3.5, we discuss related work. Finally, we summarize our results and discuss future work in Section 3.7.

3.2 Stein Variational Gradient Descent

The core idea of SVGD is to perform inference by approximating the target posterior distribution $p(\mathbf{z}|\mathcal{D})$ by an empirical distribution $q_{\mathcal{Z}}(\mathbf{z}) = N^{-1}\sum_i \delta_{\mathbf{z}_i}(\mathbf{z})$ based on a set of particles $\mathcal{Z} = \{\mathbf{z}_i\}_{i=1}^N$. Here, $\delta_{\mathbf{x}}(\mathbf{y})$ represents the Dirac delta measure, which is equal to 1 if $\mathbf{x} = \mathbf{y}$ and 0 otherwise. One could thus see the approximating distribution $q_{\mathcal{Z}}(\mathbf{z})$ as a mixture of point estimates, each represented by a particle $\mathbf{z}_i \in \mathcal{Z}$. The idea is to minimize the Kullback-Leibler divergence $D_{\mathrm{KL}}(q_{\mathcal{Z}}(\mathbf{z}) \parallel p(\mathbf{z}|\mathcal{D}))$ between the approximation and the true posterior by iteratively updating the particles using the Stein forces:

$$\mathbf{z}_i \leftarrow \mathbf{z}_i + \epsilon S_{\mathcal{Z}}(\mathbf{z}_i)$$

where ϵ is the learning rate and $S_{\mathcal{Z}}$ denotes the Stein forces.

The Two Forces of SVGD Stein VI consists of two forces that work additively under the form $S_{\mathcal{Z}} = S^+_{\mathcal{Z}} + S^-_{\mathcal{Z}}$, where the attractive force is given by

$$S^+_{\mathcal{Z}}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j \sim q_{\mathcal{Z}}(\mathbf{z})}[k(\mathbf{z}_i, \mathbf{z}_j)\nabla_{\mathbf{z}_j} \log p(\mathbf{z}_j|\mathcal{D})]$$

and the repulsive force by

$$S^-_{\mathcal{Z}}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j \sim q_{\mathcal{Z}}(\mathbf{z})}[\nabla_{\mathbf{z}_j} k(\mathbf{z}_i, \mathbf{z}_j)].$$

Here $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel. The attractive force can be seen as pushing the particles towards the modes of the true posterior distribution, smoothed by some kernel. For an example of a kernel, consider the radial basis function (RBF) kernel $k(\mathbf{z}_i, \mathbf{z}_j) = \exp\left(-\frac{1}{h} \parallel \mathbf{z}_i - \mathbf{z}_j \parallel_2^2\right)$ with bandwidth parameter h, chosen as $\frac{1}{\log n}\mathrm{med}(\mathbf{z})$.

The repulsive force moves particles away from each other, ensuring that they do not collapse to the same mode. For example, with the RBF kernel, the repulsive force becomes $\mathbb{E}_{\mathbf{z}_j \sim q_{\mathcal{Z}}(\mathbf{z})}[-\exp\left(-\frac{1}{h} \parallel \mathbf{z}_i - \mathbf{z}_j \parallel_2^2\right) \frac{2}{h}\sum_\ell (\mathbf{z}_{i\ell} - \mathbf{z}_{j\ell})]$, which has high kernel values for particles far from each other, causing them to stay apart.

SVGD works with unnormalized distributions p as the normalization constant becomes additive in the log-posterior $\log p(\mathbf{z}_i|\mathcal{D}) = -\log Z + \log p(\mathcal{D}|\mathbf{z}) + \log p(\mathbf{z})$ and is thus not required for the calculation of the gradient. This property is desirable as normalizing p is often computationally expensive.

Non-linear Stein In non-linear Stein (Wang and Liu, 2019b), the repulsive force can be scaled by a factor λ, resulting in $S_{\mathcal{Z}} = S^+_{\mathcal{Z}} + \lambda S^-_{\mathcal{Z}}$. This approach is useful when dealing with multi-modal distributions. It is also useful in cases where the repulsive force vanishes compared to the likelihood, which happens for large datasets ($\mathcal{X}$). Scaling the repulsive force by a constant $\lambda = c(|\mathcal{X}|)$ proportional (e.g. $c = 0.1$ or $c = 0.01$) to the size of the dataset $|\mathcal{X}|$ addresses this issue and can be chosen by cross-validation on a subset of the data.

Matrix-valued kernels The choice of kernels can be extended to matrix-valued ones (Wang et al., 2019b), $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$, in which case the Stein forces become

$$S^+_{\mathcal{Z}}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j\sim q_{\mathcal{Z}}(\mathbf{z})}[K(\mathbf{z}_i, \mathbf{z}_j)\nabla_{\mathbf{z}_j} \log p(\mathbf{z}_j|\mathcal{D})]$$

and

$$S^-_{\mathcal{Z}}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j\sim q_{\mathcal{Z}}(\mathbf{z})}[K(\mathbf{z}_i, \mathbf{z}_j)\nabla_{\mathbf{z}_j}]$$

where the standalone del $\nabla_{\mathbf{z}_j}$ in the repulsive force represents the vector $\left(\frac{\partial}{\partial z_{j,1}}, \ldots, \frac{\partial}{\partial z_{j,d}}\right)$. This results in

$$(K(\mathbf{z}_i, \mathbf{z}_j)\nabla_{\mathbf{z}_j})_\ell = \sum_k \nabla_k K_{\ell,k}(\mathbf{z}_i, \mathbf{z}_j).$$

The advantage of matrix-valued kernels is that they allow preconditioning[4] using the Hessian or Fisher Information matrix, which can capture local curvature and thus achieve better optima and convergence rate than standard SVGD. Furthermore, it is easy to represent graphical kernels (Wang, Zeng, and Liu, 2018b) using matrix kernels, e.g. $K = \text{diag}(\{K^{(\ell)}\}_\ell)$ where the set of variables are partitioned with each their own local kernel $K^{(\ell)}$.

[4]Using preconditioner matrix Q, such that $Q^{-1}M$ has a lower condition number than M.

3.3 Stein Mixtures

The Stein-mixture was proposed by Nalisnick and Smyth, 2017 to resolve the representation issue for SVGD when the target distribution has high dimensionality. For SVGD, the number of particles needed to represent a distribution adequately grows exponentially with its dimensionality. As the computational complexity for the update rule in SVGD is quadratic in the number of particles, the exponential growth quickly becomes computationally intractable.

Stein-mixtures are hierarchical variational models (Ranganath, Tran, and Blei, 2016) such that the posterior of the second tier variational distribution is $q(\mathbf{z}|\mathcal{D}) = \frac{1}{N}\sum_{k=1}^{N}\delta(\mathbf{z}_k)$. The guide is the joint model $q(\boldsymbol{\theta}, \mathbf{z}) = q(\boldsymbol{\theta}|\mathbf{z})q(\mathbf{z})$, with $q(\boldsymbol{\theta}|\mathbf{z})$ the first tier distribution and $q(\mathbf{z})$ its prior. The second tier posterior $q(\mathbf{z}|\mathcal{D})$ is optimized using SVGD, so that the marginal posterior is $q(\boldsymbol{\theta}; \mathbf{z}) = \frac{1}{N}\sum_{k=1}^{N} q(\boldsymbol{\theta}|\mathbf{z})$, a restricted mixture.

Nalisnick and Smyth, 2017 showed that the Stein force for a mixture approximation of the posterior $p(\mathbf{z}|\mathcal{D})$ is given by

$$\begin{aligned} S_{\mathcal{Z}}(\mathbf{z}_i) = &\mathbb{E}_{\mathbf{z}_j\sim q_{\mathcal{Z}}(\mathbf{z})}\left[k(\mathbf{z}_i, \mathbf{z}_j)\mathbb{E}_{\boldsymbol{\theta}_l\sim q(\theta|\mathbf{z}_j)}\left[p(\mathcal{D}, \boldsymbol{\theta}_l)/q(\boldsymbol{\theta}_l|\mathbf{z}_j)\right]\right] + \\ &+ \mathbb{E}_{\mathbf{z}_j\sim q_{\mathcal{Z}}(\mathbf{z})}\left[k(\mathbf{z}_i, \mathbf{z}_j)\nabla_{\mathbf{z}_j}\log q(\mathbf{z}_j)\right] + \\ &+ S^{-}_{\mathcal{Z}}(\mathbf{z}_i), \end{aligned}$$

where $\log \nabla_{\mathbf{z}_j} q(\mathbf{z}_j)$ acts as a regularizer on the variational parameters which they assume is sufficiently small to be dropped. To evaluate $\mathbb{E}_{\mathbf{z}_j\sim q_{\mathcal{Z}}(\mathbf{z})}\left[p(\mathcal{D}, \theta)/q(\theta|\mathbf{z}_j)\right]$, Nalisnick and Smyth, 2017 use DNCP and importance weighted Monte Carlo gradients to obtain the black-box update,

$$S_{\mathcal{Z}}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j\sim q_{\mathcal{Z}}(\mathbf{z})}\left[k(\mathbf{z}_i, \mathbf{z}_j)\sum_{s=1}^{S}\tilde{w}_s\nabla_{\mathbf{z}_j}\log\left(\frac{p(\mathcal{D}, \hat{\boldsymbol{\theta}}_s)}{q(\hat{\boldsymbol{\theta}}_s|\mathbf{z}_j)}\right)\right] + S^{-}_{\mathcal{Z}}(\mathbf{z}_i), \tag{3.1}$$

where $\tilde{w}_s$ is an importance weight for sample $\hat{\boldsymbol{\theta}}_s = q(\mathbf{z}_j, \boldsymbol{\xi})$, $\boldsymbol{\xi} \sim p_0$ (i.e. the guide).

ELBO-within-Stein In ELBO-within-Stein, we replace the weighted average over S samples in Equation (3.1) by a single loss ($\mathcal{L}$). If we choose $\mathcal{L}$ to be an f-divergence, such as the ELBO, we can reduce the loss bias by averaging multiple samples. However, ELBO-within-Stein only computes one gradient of $\mathcal{L}$ per particle regardless of the number of samples we use to estimate $\mathcal{L}$. This is different from Nalisnick and Smyth, 2017 who estimate $\mathbb{E}_{\boldsymbol{\theta}_l\sim q(\theta|\mathbf{z}_j)}\left[p(\mathcal{D}, \boldsymbol{\theta}_l)/q(\boldsymbol{\theta}_l|\mathbf{z}_j)\right]$ by averaging over gradients, therefore computes a gradient and an importance weighting for each DNCP sample rather than per particle. The difference makes ELBO-within-Stein computationally cheaper

as the complexity of adding a gradient is $\mathcal{O}(n)$ whereas adding a sample to estimate $\mathcal{L}$ is $\mathcal{O}(1)$. The Stein force in ELBO-within-Stein is given by

$$S_{\mathcal{Z}}(\mathbf{z}_i) = \mathbb{E}_{\mathbf{z}_j \sim q_{\mathcal{Z}}(\mathbf{z})} \left[k(\mathbf{z}_i, \mathbf{z}_j) \nabla_{\mathbf{z}_j} \mathcal{L}(z_j) \right] + S^{-}_{\mathcal{Z}}(\mathbf{z}_i) \tag{3.2}$$

where $\mathcal{L}$ is the ELBO.

3.4 Compositional Implementation using NumPyro

EinStein integrates with the existing NumPyro API by adding the Stein VI interface, which closely mimicks NumPyro's SVI interface. Mimicking the SVI interface makes programs that use SVI in NumPyro trivial to convert to EinStein (see Figure 3.1).

Below, we discuss the key features of EinStein, which include re-initializable guides, EinStein's core algorithm, and the new kernel interface.

Re-initializable Guides

The Stein VI interface requires that the initialization is different for each parameter in an inference program. The reason is that different Stein particles need to be initialized to different values in order for optimization to work correctly and to avoid all particles collapsing into the posterior mode.

To support re-initializable guides, we provide the `ReinitGuide` interface, which requires implementing a function `find_params` that accepts a list of random number generator (RNG) keys in addition to the arguments for the guide and returns a set of freshly initialized parameters for each RNG key.

The `WrappedGuide` class provides a guide written as a function. `WrappedGuide` makes a callable guide re-initializable. It works by running the provided guide multiple times and reinitializing the parameters using NumPyro's interface as follows:

- `WrappedGuide` runs the guide transforming each parameter to unconstrained space.
- It replaces the values of the parameters with values provided by a NumPyro initialization strategy, e.g., `init_to_uniform`(r), which initializes each parameter with a uniform random value in the range $[-r; r]$.
- It saves the parameter values for each particle and the required inverse transformations to constrained space to run the model correctly.

We also allow parameters without reinitialization in order to support neural network libraries like `stax` [5] that have their own initializers.

The Stein VI interface will correctly wrap the guide during initialization, so from a user perspective the syntax for guides follows the API of SVI.

SteinVI in NumPyro

The integration of SteinVI into NumPyro requires handling transformations between the parameter representation of NumPyro[6] and the vectorized Stein

[5] https://jax.readthedocs.io/en/latest/jax.experimental.stax.html

[6] A dictionary mapping parameters to their values, which can be arbitrary Python type

particles that EinStein operates on. For this, we rely on `Jax PyTrees`[7] which converts back and forth between Python collections and a flattened vectorized representation.

Algorithm 1 shows the core algorithm of EinStein. EinStein updates the standard variational model parameters $\boldsymbol{\phi}$ and guide parameters $\boldsymbol{\psi}$ by averaging the loss over the Stein particles. For the Stein parameters, the process is more elaborate. First, we convert the set of individual parameters to a monolithic vector-encoded particle using `Jax PyTrees`. The monolithic particle represents the particles as a flattened and stacked `Jax` array. Then we compute a kernel value based on the vector-encoded Stein particle. Kernel computation is delegated to the kernel interface (see Section 3.4 as the type is kernel-dependent.

We apply `Jax`'s `vmap` operator (Frostig, Johnson, and Leary, 2018; Phan, Pradhan, and Jankowiak, 2019b) to compute the Stein forces for each particle in a vectorized manner. This is done in unconstrained space so the Stein force must the corrected by the Jacobian of the bijection between constrained and unconstrained space. Doing this directly on the `Jax` on the monolithic particle incurs a massive memory overhead in the adjoint. However, as NumPyro registers a bijection for each distribution parameter we can eliminate the overhead by computing the Jacobian on the `Jax` representations of the individual parameters. The operation is embarrassingly parallel and so we again use a `vmap` operator with a nested `tree_map` to compute the desired Jacobians. Note that Algorithm 1 presents how EinStein works with scalar kernels and does not account for the different features presented in Section 3.2.

Finally, we convert the monolithic Stein particle to their non-vectorized dictionary-based form and return the expected changes for standard- and Stein-parameters.

Kernel interface

The Kernel interface is straightforward. To extend the interface, users must implement the `compute` function, which accepts as input the current set of particles, the mapping between model parameters and particles, and the loss function $\mathcal{L}$ and returns a differentiable kernel k. All kernels are currently static, but the interface could be extended to stateful kernels, allowing conjugate gradients or quasi-Newton optimization. Section 3.4 gives the complete list of kernels in EinStein. We plan to extend EinStein with probability product kernels Nalisnick and Smyth, 2017; Jebara, Kondor, and Howard, 2004, which take into account the information geometry of the first-tier variational distributions in Stein mixtures.

[7] https://jax.readthedocs.io/en/latest/pytrees.html

Procedure 1 EinStein

Input: Classical VI parameters $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$, Stein parameters $\{\boldsymbol{\theta}_i\}_i$, model $p_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{x})$, guide $q_{\boldsymbol{\theta},\boldsymbol{\psi}}(\mathbf{z})$, loss $\mathcal{L}$, kernel interface KI.

Output: Parameter changes based on classical VI ($\Delta\boldsymbol{\phi}$, $\Delta\boldsymbol{\psi}$) and Stein VI forces ($\{\Delta\boldsymbol{\theta}_i\}_i$).

procedure EINSTEIN($\boldsymbol{\phi}$, $\boldsymbol{\psi}$, $\{\boldsymbol{\theta}_i\}_i$, $p_{\boldsymbol{\phi}}$, $q_{\boldsymbol{\theta},\boldsymbol{\psi}}$)

$\Delta\boldsymbol{\phi} \leftarrow \mathbb{E}_{\boldsymbol{\theta}}[\nabla_{\boldsymbol{\phi}} \mathcal{L}(p_{\boldsymbol{\phi}}, q_{\boldsymbol{\theta},\boldsymbol{\psi}})]$

$\Delta\boldsymbol{\psi} \leftarrow \mathbb{E}_{\boldsymbol{\theta}}[\nabla_{\boldsymbol{\psi}} \mathcal{L}(p_{\boldsymbol{\phi}}, q_{\boldsymbol{\theta},\boldsymbol{\psi}})]$

$\{\mathbf{a}_i\}_i \leftarrow \text{PyTreeFlatten}(\{\boldsymbol{\theta}_i\}_i)$

$k \leftarrow \text{KI}(\{\mathbf{a}_i\}_i)$

procedure EINSTEINFORCES($\mathbf{a}_i$) ▷

Calculate forces per particle for higher-order vmap function.

$\boldsymbol{\theta}_i \leftarrow \text{PYTREERESTORE}(\mathbf{a}_i)$

$\Delta\mathbf{a}_i \leftarrow \sum_{\mathbf{a}_j} k(\mathbf{a}_j, \mathbf{a}_i) \nabla_{\mathbf{a}_i} \mathcal{L}(p_{\boldsymbol{\phi}}, q_{\boldsymbol{\theta}_i,\boldsymbol{\psi}}) + \nabla_{\mathbf{a}_i} k(\mathbf{a}_j, \mathbf{a}_i)$

return $\Delta\mathbf{a}_i$

end procedure

$\{\Delta\mathbf{a}_i\}_i \leftarrow \text{VMap}(\{\mathbf{a}_i\}_i, \text{EINSTEINFORCES})$

$\{\Delta\boldsymbol{\theta}_i\}_i \leftarrow \text{PYTREERESTORE}(\{\Delta\mathbf{a}_i\}_i)$

return $\Delta\boldsymbol{\phi}$, $\Delta\boldsymbol{\psi}$, $\{\Delta\boldsymbol{\theta}_i\}_i$

end procedure

Table 3.1: Kernel included in the EinStein NumPyro module.

Kernel	Definition	Detail	Type	Reference
Radial Basis Function (RBF)	$\exp(\frac{1}{h} \parallel \mathbf{x}-\mathbf{y} \parallel_2^2)$		scalar	Liu and Wang, 2016b
	$\exp(\frac{1}{h}(\mathbf{x}-\mathbf{y}))$		vector	Pyro[8]
Inverse Multi-Quadratic (IMQ)	$(c^2+ \parallel \mathbf{x}-\mathbf{y} \parallel_2^2)^{\beta}$	$\beta \in (-1,0)$ and $c>0$	scalar	Gorham and Mackey, 2017
Random Feature Expansion	$\mathbb{E}_{\mathbf{w}}[\phi(\mathbf{x},\mathbf{w})\phi(\mathbf{y},\mathbf{w})]$	$\phi(\mathbf{x},\mathbf{w}) = \sqrt{2}\cos(\frac{1}{h}\mathbf{w}_1^\top \mathbf{x} + \mathrm{w}_0)$ where $\mathrm{w}_0 \sim \mathrm{Unif}(0,2\pi)$ and $\mathbf{w}_1 \sim \mathcal{N}(0,1)$	scalar	Liu and Wang, 2018b
Linear	$\mathbf{x}^\top \mathbf{y} + 1$		scalar	Liu and Wang, 2018b
Mixture	$\sum_i w_i k_i(\mathbf{x},\mathbf{y})$	$\{k_i\}_i$ individual kernels, weights w_i	scalar, vector, matrix	Liu and Wang, 2018b
Scalar-based Matrix	$k(\mathbf{x},\mathbf{y})\boldsymbol{I}$	k scalar-valued kernel	matrix	Wang et al., 2019b
Vector-based Matrix	$\mathrm{diag}(k(\mathbf{x},\mathbf{y}))$	k vector-valued kernel	matrix	Wang et al., 2019b
Graphical	$\mathrm{diag}(\{K^{(\ell)}(\mathbf{x},\mathbf{y})\}_\ell)$	$\{K^{(\ell)}\}_\ell$ matrix-valued kernels, each for a unique partition of latent variables	matrix	Wang et al., 2019b
Constant Preconditioned	$\boldsymbol{Q}^{-\frac{1}{2}} K(\boldsymbol{Q}^{\frac{1}{2}}\mathbf{x}, \boldsymbol{Q}^{\frac{1}{2}}\mathbf{y})\boldsymbol{Q}^{-\frac{1}{2}}$	K is an inner matrix-valued kernel and $\boldsymbol{Q}$ is a preconditioning matrix like the Hessian $-\nabla^2_{\bar{\mathbf{z}}} \log p(\bar{\mathbf{z}}\|\mathbf{x})$ or Fischer information $-\mathbb{E}_{\mathbf{z}\sim q_{\mathcal{Z}}(\mathbf{z})}[\nabla^2_{\mathbf{z}} \log p(\mathbf{z}\|\mathbf{x})]$ matrices	matrix	Wang et al., 2019b
Anchor Point Preconditioned	$\sum_{\ell=1}^m K_{\boldsymbol{Q}_\ell}(\mathbf{x},\mathbf{y}) w_\ell(\mathbf{x}) w_\ell(\mathbf{y})$	$\{\mathbf{a}_\ell\}_{\ell=1}^m$ is a set of anchor points, $\boldsymbol{Q}_\ell = \boldsymbol{Q}(\mathbf{a}_\ell)$ is a preconditioning matrix for each anchor point, $K_{\boldsymbol{Q}_\ell}$ is an inner kernel conditioned using $\boldsymbol{Q}_\ell$, and $w_\ell(\mathbf{x}) = \mathrm{softmax}_\ell(\{\mathcal{N}(\mathbf{x}\|\mathbf{a}_{\ell'}, \boldsymbol{Q}_{\ell'}^{-1})\}_{\ell'})$	matrix	Wang et al., 2019b

3.5 Related Work

There has been a proliferation of deep probabilistic programming languages (Ge, Xu, and Ghahramani, 2018; Bingham et al., 2019; Salvatier, Wiecki, and Fonnesbeck, 2016; Tran et al., 2016; Cusumano-Towner et al., 2019; Dillon et al., 2017) based on tensor frameworks featuring automatic differentiation and supporting various inference techniques. However, only `PyMC3` (Salvatier, Wiecki, and Fonnesbeck, 2016) includes inference using SteinVI.

In `PyMC3` the inference techniques that use Stein's method include SVGD with a scalar RBF-kernel, amortized SVGD (Wang, Feng, and Liu, 2016; Feng, Wang, and Liu, 2017), and Operator Variational Inference (OVI) (Ranganath et al., 2016).

SVGD in `PyMC3` manipulates particles to directly capture the target distribution (Liu and Wang, 2016b). In EinSteinVI, SVGD is a special case of Stein mixtures where the guides are point mass distributions. Because we have guide programs, EinSteinVI allows arbitrary computations to transform random variables in the guide. This is not possible with SVGD in `PyMC3`.

Amortized SVGD (Feng, Wang, and Liu, 2017) trains a stochastic network to draw samples from a target distribution. The network is iteratively adjusted so that the output changes in the direction of the Stein variational gradient (the same gradient as used in SVGD). In comparison, EinSteinVI transports a fixed set of particles (each parameterizing a guide) to the target distribution. Amortized SVGD is not in EinSteinVI as its extension to arbitrary guides is an open problem.

OVI optimizes operator objectives, which take functions of functions to a non-negative number. Ranganath et al., 2016 include an operator objective based on the Langevin-Stein operator (Anastasiou et al., 2021). This is the same operator used for the kernelized Stein discrepancy (Liu, Lee, and Jordan, 2016) which also underlies SVGD. Unlike EinSteinVI, Amortized SVDG and SVGD, OPV is not a particle-based method.

```
def model():
  sample('x', NormalMixture(jnp.array([1 / 3, 2 / 3]),
                            jnp.array([-2.0, 2.0]),
                            jnp.array([1.0, 1.0])))
```

(a) 1D Gaussian mixture model

```
svi = SVI(
        model,
        AutoNormal(model),
        Adagrad(step_size=1.0),
        Trace_ELBO()
    )
results = svi.run(rng_key,
                  num_iterations)
```

(b) SVI

```
stein = SteinVI(
        model,
        AutDelta(model),
        Adagrad(step_size=1.0),
        Trace_ELBO(),
        RBFKernel(),
    )
results = stein.run(rng_key,
                    num_iterations)
```

(c) SVGD with EinStein

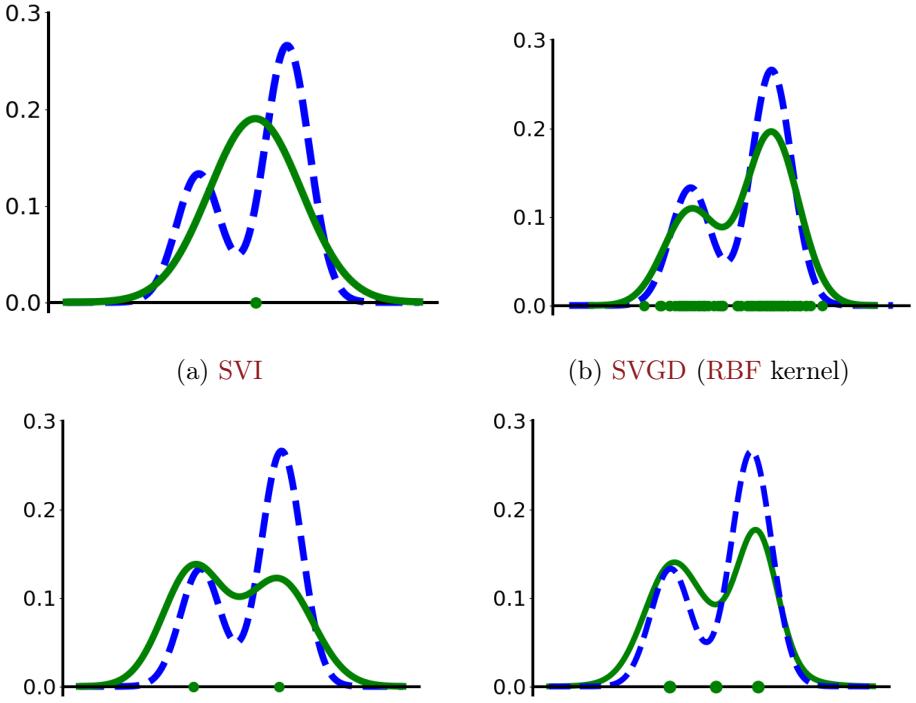
Figure 3.1: 1D Gaussian mixture model in NumPyro.

3.6 Examples

We illustrate the features of EinStein on two toy examples, namely a 1D mixture of Gaussians below, and 2D mixtures of Gaussian. We demonstrate EinStein on real-world examples and show that EinStein tends to outperform alternative methods. These examples include regression with a BNN and deep Markov models. All example programs and notebooks are in the `NumPyro` python module.

1D Gaussian mixture To demonstrate the two modes of VI with EinStein, we consider the 1D Gaussian mixture $1/3\mathcal{N}(-2,1)+2/3\mathcal{N}(2,1)$ (see Figure 3.1 and Figure 3.2). The Gaussian target mixture is bi-modal and well-suited for the non-parametric nature of SVGD and SMs. Figure 3.2 shows that both SVGD and the SM naturally capture the bi-modality of the target distribution, compared to SVI with a simple Gaussian guide. Note the reduction in particles required to estimate the target when using SMs compared to SVGD. Also, note that the SM overestimates the variance and slightly perpetuates the locations.

The error seen at the right mode for the SM with two particles is due to the uniform weighting of the particles in SVGD (the target posterior is approximated with an empirical distribution, see Section 3.2), and as such is algorithmic. The SM will therefore not be able to exactly capture the mixing components of a target mixture model with one particle per component. However, with more particles, the mixture can be approximated better as demonstrated using three particles.



(a) SVI (b) SVGD (RBF kernel)

(c) Two particle SM (linear kernel) (d) Three particle SM (RBF kernel)

Figure 3.2: The blue dashed line is the target pdf, while the solid green line is the density of the particles. We estimate the particle density for SVGD with Gaussian kernel density estimation. We use 100 particles for SVGD, and two or three particles for the SM. SVI uses a Gaussian guide.

Bayesian Neural Networks We compare SVGD and non-linear Stein in EinStein with the implementation of SVGD by Liu and Wang, 2016b[9] (without model selection), and amortized SVGD (using the Theano backend[10], Al-Rfou et al., 2016) on BNN for regression. The `PyMC3` documentation clearly states that amortized SVGD (Feng, Wang, and Liu, 2017) is experimental and is not suggested to be used. We include its experimental results for completeness and note that our findings reflect the warning in the `PyMC3` documentation. We use an RBF kernel in EinStein for a fair comparison with Liu and Wang, 2016b and `PyMC3`. For non-linear Stein we determined the best repulsion factor λ^* by a grid search for $\lambda \in \{10^{-2}, 10^{-1}, 10^{1}, 10^{2}\}$. For SVGD `PyMC3` we use a temperature[11] of ten. Like Liu and Wang, 2016b we use a BNN with one hidden layer of size fifty and `RELU` activation. We put a Gamma$(1, 0.1)$ prior

[9] https://github.com/dilinwang820/Stein-Variational-Gradient-Descent/blob/master/python/bayesian_nn.pyy

[10] `PyMC4`, which uses `Jax` as its backend, omits SVGD at the time of writing.

[11] We found the performance to be very sensitive to the choice of temperature.

on the precision of the neurons and the likelihood. For all versions, we use 100 particles and 2000 iterations. We use a mini-batch of 1000 for Year and 100 for the rest. All measurements are repeated ten times and obtained on a GPU[12] for EinStein and `PyMC3`, except for Year with `PyMC3` which was only run once. We do not report times for Liu and Wang, 2016b because only the CPU version of their code could be executed without irresolvable issues.

Table 3.2 shows the performance in terms of the average root mean squared error (RMSE) on the test set. We find that EinStein achieves significantly better RMSE than Liu and Wang, 2016b and that both systems outperform SVGD in `PyMC3`. The times in Table 3.2 measure how long it takes to infer parameters. Table 3.2 excludes the first run for two reasons: i) that run will fill the GPU caches, and ii) `Jax` will trace the programs. As a result, this run is more costly than subsequent ones. The times for the first run are given in Table 3.3.

By running EinStein for more iterations, we can amortize the initial cost.

The amortized SVGD in `PyMC3` is an experimental implementation and the documentation clearly discourages its use. For completeness we include the performance here. The temperature and learning rate were determined by a grid search on the Boston data set. All measurements are repeated five times. In Table 3.4 we report test RMSE and time the the UCI regression benchmark. We did not include the Year data set as it takes over 21 hours to run. The RMSE for amortized SVGD is poor and highly variable across all data sets, except Naval. However, even for Naval the RMSE is two orders of magnitude greater than SVGD in EinSteinVI and Liu and Wang, 2016b.

Stein Mixture Deep Markov Model Music generation requires a model to learn complex temporal dependencies to achieve local consistency between notes. The Stein-mixture deep Markov model (SM-DMM) is a DMM that uses a mixture of Stein particles to estimate distributions over model parameters. We consider a vectorized version of the DMM (Jankowiak and Karaletsos, 2019) to generate polyphonic music using the JSB chorales data set.

The SM-DMM model consists of two feedforward neural network (FNN) networks. The *Transition* network handles the conditional dependencies between subsequent latent states in the Markov chain. It consists of two layers with hidden dimension 200 and ReLU activation on the first layer and sigmoid on the second. The *Emitter* network is a three layers FNN which produces a likelihood at each time step using the current latent state. The layers have hidden dimensions 100, 100, and 88, respectively. The variational distribution is of the form $\prod_{n=1}^{N} q(z^n_{1:T_n} | f(X_{1:T_n}))$ where the parametrized feature function $f_{1:T_n}$ is a one layered gated recurrent unit (Chung et al., 2014), with hidden dimension 150. The total number of parameters in the DMM is 128,654, so

[12] Quadro RTX 6000 with Cuda V11.4.120

Table 3.2: Average test RMSE and time for inference for the UCI regression benchmarks.

	Test RMSE				Time	
	EinStein		**PyMC3**	**Liu and Wang, 2016b**	**EinStein**	**PyMC3**
Dataset	SVGD	NL-Stein	SVGD	SVGD	SVGD	SVGD
Boston	2.610 ± 0.044	2.492 ± 0.028	6.059 ± 0.244	2.990 ± 0.013	$5.758s \pm 0.048s$	1m6.396s ± 0.301s
Concrete	4.175 ± 0.025	4.782 ± 0.047	6.255 ± 0.390	5.927 ± 0.021	$5.602s \pm 0.048s$	1m5.845s ± 0.278s
Energy	0.321 ± 0.004	0.375 ± 0.007	3.494 ± 1.500	1.251 ± 0.017	$5.562s \pm 0.053s$	1m6.175s ± 0.41s
Kin8nm	0.073 ± 0.000	0.076 ± 0.000	0.164 ± 0.040	0.115 ± 0.001	$6.141s \pm 0.04s$	2m40.106 ± 0.298s
Naval	0.001 ± 0.000	0.001 ± 0.000	0.029 ± 0.015	0.008 ± 0.001	$6.51s \pm 0.048s$	$3m51.05s \pm 0.385s$
Power	3.952 ± 0.004	3.907 ± 0.006	5.389 ± 2.219	4.215 ± 0.002	$6.314s \pm 0.054s$	2m51.654s ± 0.566s
Protein	4.509 ± 0.006	4.654 ± 0.006	5.921 ± 2.559	4.846 ± 0.002	$9.436s \pm 0.032s$	11m11.9s ± 0.399s
Wine	0.599 ± 0.006	0.596 ± 0.005	0.769 ± 0.212	0.604 ± 0.001	$5.614s \pm 0.052s$	1m7.767s ± 0.286s
Yacht	0.531 ± 0.009	0.246 ± 0.013	8.341 ± 3.744	0.94 ± 0.03	$5.519s \pm 0.07s$	1m4.748s ± 0.392s
Year	8.662 ± 0.002	8.701 ± 0.002	135.316	8.895 ± 0.001	1m37.569s ± 0.07s	4h19m36s

Table 3.3: Time for first repetition with EinStein for UCI regression benchmarks.

Dataset	Boston	Concrete	Energy	Kin8nm	Naval	Power	Protein	Wine	Yacht	Year
Time	41.665s	41.642s	41.591s	43.592s	44.2570s	44.058s	47.87s	41.9s	41.409s	2m18.19s

Table 3.4: Average test RMSE and time for inference for the UCI regression benchmarks with amortized SVGD.

	RMSE	Time
Boston	87230.469 ± 54219.534	$5\text{m}46\text{s} \pm 3\text{s}$
Concrete	2458.250 ± 754.653	$6\text{m}3\text{s} \pm 2\text{s}$
Energy	2458.250 ± 754.653	$5\text{m}45\text{s} \pm 3\text{s}$
Kin8nm	302.218 ± 43.818	$19\text{m}54\text{s} \pm 5s$
Naval	0.386 ± 0.128	$28\text{m}39\text{s} \pm 3s$
Power	3333.796 ± 517.241	$21\text{m}38\text{s} \pm 6s$
Protein	64.791 ± 12.181	$1\text{h}32\text{m}28\text{s} \pm 1\text{m}1\text{s}$
Wine	12.929 ± 6.202	$7\text{m}21\text{s} \pm 3\text{s}$
Yacht	2822.983 ± 944.449	$5\text{m}35\text{s} \pm 4\text{s}$

Table 3.5: Test negative log- likelihood (lower is better) on Polyphonic Music Generation (JSB) dataset. Baseline results from Krishnan, Shalit, and Sontag, 2016. ISN-DMM and ISN-DMM-Aug (Krishnan, Shalit, and Sontag, 2016), TSBN and HMSBN (Gan et al., 2015)

	ISN-DMM	ISN-DMM-Aug	HMSBN	TSBN	SM-DMM
NLL (a)	6.926	6.773	8.0473	-	-45.983
NLL (b)	6.856	6.692	7.9970	7.48	-46.066

with five particles, EinStein is optimizing 643,270 parameters for the SM-DMM model.

We train the SM-DMM using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-5}, using an RBF kernel and five Stein particles for four thousand epochs on the polyphonic music generation dataset JSB chorals (Boulanger-Lewandowski, Bengio, and Vincent, 2012). We follow Krishnan, Shalit, and Sontag, 2016 and report two version NLL of a) $\frac{\sum_{i=1}^{N} -p(\mathbf{x}_i|\boldsymbol{\theta})}{\sum_{i=1}^{N} T_i}$ and b) $\frac{1}{N}\sum_{i=1}^{N} \frac{-p(\mathbf{x}_i|\boldsymbol{\theta})}{T_i}$, where T_i is the length of the ith sequence. In Table 3.5, we report NLL(a) and NLL(b) on a held-out test set of JSB. Compared to baseline methods, SM-DMM achieves a significant improvement using Stein mixtures. We see similar improvements in the test ELBO, Jankowiak and Karaletsos, 2019 reports a test ELBO of -6.82 nats on the JSB dataset for their approach, SM-DMM with EinStein achieves a test ELBO of *45.10* (higher is better).

Star Distribution To illustrate the kernels included in EinStein, we approximate the star distribution, following Wang et al., 2019a (see Figure 3.3). The star distribution is constructed as a 2D Gaussian mixture,

$$
\begin{aligned}
p(x) &= \frac{1}{K}\sum_{k=1}^{K}\mathcal{N}(x|\mu_k, \Sigma_k) \\
\mu_1 &= [0, 1.5], \mu_k = U_k\mu_1 \\
\Sigma_1 &= \mathrm{diag}([1, 1/100]), \Sigma_k = U_k\Sigma_1 U_k
\end{aligned}
$$

where U_k is a rotation matrix given by

$$
U_k = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad (k \in [1, .., K]) \quad \theta = 2k\pi/K.
$$

We use fifty particles and Adagrad with the learning rate that yields the best maximum mean discrepancy (MMD) (Gretton et al., 2012) and the RBF kernel. To compute the MMD, we use 10,000 samples from the 2D Gaussian mixture. Keeping these choices fixed, we vary the type of kernel, using a point mass distribution as guide.

We consider the scalar ($\mathbb{R}^d \to \mathbb{R}$) linear kernel, the inverse multi-quadric (IMQ) kernel, the RBF kernel, and the random feature kernel; details are given in section 3.4. In addition, we use a mixture kernel, which is a uniform mixture of the linear and the random feature kernels. We also use the matrix version of the RBF kernel with constant preconditioning, using the Hessian matrix for preconditioning. We see that including curvature information by means of the Hessian leads to faster convergence in EinStein and that the matrix RBF with preconditioning yields the lowest MMD.

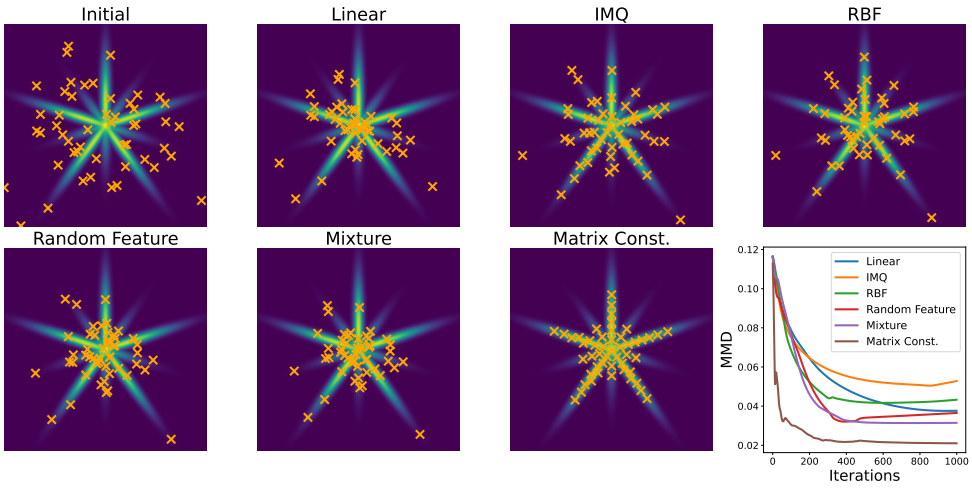


Figure 3.3: Particle positions for EinStein with different kernels after 1000 iterations with a point mass guide, starting from particle positions given in the upper left frame labeled Initial. The MMD, bottom right frame, is evaluated using the RBF kernel.

3.7 Summary

EinStein provides the latest techniques for glsSteinVI as an extension to Numpyro. Our results indicate that the library is substantially faster and more expressive than other available libraries for SteinVI. EinStein provides a familiar and efficient interface for practitioners working with the Pyro/NumPyro PPL and provides a unified code base to researchers for benchmarking new developments in SteinVI.

Chapter 4

The (Skewed) Sine Distribution in NumPyro

Ola Rønning[1], Christophe Ley[2], Kanti V. Mardia[34] and Thomas Hamelryck[15]

4.1 Introduction

Until recently, protein structure prediction posed a long-standing challenge in structural biology. At the fourteenth Critical Assessment of Techniques for Protein Structure Prediction (CASP), (Moult et al., 1995) the deep learning-based method AlphaFold2 (AlQuraishi, 2021; Senior et al., 2020) achieved levels of accuracy that enticed the CASP committee to declare the protein structure prediction problem solved. Despite this progress, contemporary methods for protein structure prediction generally do not systematically account for uncertainty. In particular, they fail to model aleatory uncertainty (due to protein dynamics or disorder (Shea, Best, and Mittal, 2021)) and epistemic uncertainty (caused by experimental error, limitations of the probabilistic model, and limited amounts of data). As proteins are often parameterized in terms of dihedral angles, directional statistics potentially provide a robust framework for factoring in such uncertainties (Hamelryck, Mardia, and Ferkinghoff-Borg, 2012).

Distributions on the 2-torus are attractive for protein structure modeling because a dihedral angle pair corresponding to a point on the 2-torus is often used to represent the conformation of the backbone of a single amino acid. (Hamelryck, Mardia, and Ferkinghoff-Borg, 2012). Despite their potential,

[1]Department of Computer Science, University of Copenhagen, Denmark
[2]Département Mathématiques, Université du Luxembourg, Luxembourg
[3]School of Mathematics, University of Leeds, United Kingdom
[4]Department of Statistics, University of Oxford, United Kingdom
[5]Department of Biology, University of Copenhagen, Denmark

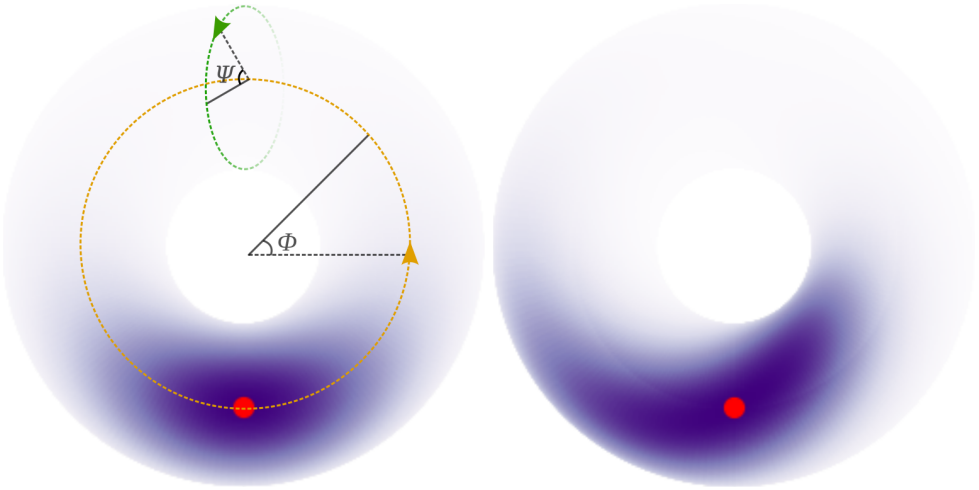
2-torus distributions are available in a few contemporary inference frameworks. To address this, we introduce efficient implementations of a 2-torus distribution in the PPLs Pyro (Bingham et al., 2019) and Numpyro (Phan, Pradhan, and Jankowiak, 2019a). The distribution is a submodel of the Bivariate von Mises (BvM) distribution (Mardia, 1975), known as the sine distribution (Singh, Hnizdo, and Demchuk, 2002).

The sine distribution is pointwise symmetric. However, it is not immediately evident that the probability distributions of dihedral angles observed in proteins display symmetry. Ameijeiras-Alonso and Ley, 2021 suggests an efficient procedure for skewing a toroidal distribution, which we call sine skewing. Sine skewing works for any dimensionality gives rise to simple simulation procedures, and is computationally efficient. In addition to the sine distribution, we also provide implementations of the sine skewing procedure in Pyro and NumPyro.

Pyro and NumPyro are modern deep PPLs with nearly identical user interfaces. However, the two PPLs differ in their computation backend and main inference engine. Pyro uses PyTorch (Paszke et al., 2019) as its backend and is mainly designed for inference with Stochastic Variational Inference (Kingma and Welling, 2019). NumPyro is a daughter framework of Pyro that uses Jax (Bradbury et al., 2018) as the backend. The main inference engine of NumPyro is Hamiltonian Monte Carlo with Iterative NUTS (Phan, Pradhan, and Jankowiak, 2019a), a variant of the No-U-Turn sampler (NUTS) (Hoffman, Gelman, et al., 2014). Iterative NUTS leverages Jax's tracing capabilities for considerable computational gains. Both frameworks provide methods to generate variational distributions called guides automatically, enumerate (i.e., sum out) discrete variables, and formulate deep probabilistic models. Tailoring the simulation of the sine distribution (Kent, Ganeiber, and Mardia, 2018) to use parallelism, we observe a 6000-fold increase in sampling time using a GPU in Pyro over the R-based implementation provided by (Kent, Ganeiber, and Mardia, 2018). These optimizations are impossible in NumPyro; our implementation provides a more modest 3000-fold increase in this framework over (Kent, Ganeiber, and Mardia, 2018).

To illustrate the sine distribution and sine skewing, we use both to formulate mixture models of dihedral angles in proteins. We implement and infer the mixture models using NumPyro, but the provided code snippets translate to Pyro with minimal effort. We use iterative NUTS for inference and apply NumPyro's automatic enumeration to obtain differentiable models.

The paper proceeds as follows. In Section 4.2, we define the sine distribution and the sine skewing procedure and highlight their most prominent features. We provide implementation details in Section 4.3. In Section 4.4, we demonstrate the sine skewed sine distribution in mixture models. Finally, in Section 4.5, we summarize and point to future work.



(a) $\rho = 0$ (b) $\rho = 1.7$

Figure 4.1: (Sine) distributions over angle pairs (ϕ, ψ) given by probability density function (PDF) $f(\phi, \psi | \boldsymbol{\mu}, \boldsymbol{\kappa}_\phi = 2, \boldsymbol{\kappa}_\psi = 2, \rho)$ projected onto a torus. Here $\boldsymbol{\kappa}_\phi, \boldsymbol{\kappa}_\psi > 0$ denote the concentration (anti-variance), and $\rho \geq 0$ is the correlation between the angles. The density is shaded in purple. The red dot marks the location $\boldsymbol{\mu}$.

4.2 Toroidal distribution

The Bivariate von Mises (BvM) distribution (Mardia, 1975) is defined on the unit torus and is point-wise symmetric around its location $\boldsymbol{\mu} = (\mu_\phi, \mu_\psi)$. That is, the density of angle pairs (ϕ, ψ) given by $f(\phi, \psi; \boldsymbol{\theta})$ satisfies $f(\mu_\phi - \phi, \psi; \boldsymbol{\theta}) = f(\mu_\phi + \phi, \psi; \boldsymbol{\theta})$ and $f(\phi, \mu_\psi - \psi; \boldsymbol{\theta}) = f(\phi, \mu_\psi + \psi; \boldsymbol{\theta})$. We have collected all the parameters of f in $\boldsymbol{\theta}$ for conciseness. Figure 4.1 illustrates this type of distribution. Because of its redundant number of parameters, various more parsimonious submodels have been proposed in the literature. Here, we focus on the sine distribution of (Singh, Hnizdo, and Demchuk, 2002), which we shall describe below. Sine skewing is a general approach on a d-torus to skew a given symmetric distribution (such as the BvM), and we have implemented it as such in both Pyro and NumPyro. However, we are particularly interested in the 2-D version (i.e., as a distribution over a toroid or 2-torus) for application purposes.

Sine Distribution

Mardia, 1975 introduced a circular analog to the bivariate normal distribution with log PDF

$$\begin{aligned} \log f(\phi, \psi | \boldsymbol{\mu}, \boldsymbol{\kappa}, \boldsymbol{A}) &\propto \kappa_\phi \cos(\phi - \boldsymbol{\mu}_\phi) + \boldsymbol{\kappa}_\psi \cos(\psi - \boldsymbol{\mu}_\psi) \\ &+ [\cos(\phi - \boldsymbol{\mu}_\phi), \sin(\phi - \boldsymbol{\mu}_\phi)] \boldsymbol{A} [\cos(\phi - \boldsymbol{\mu}_\phi), \sin(\psi - \boldsymbol{\mu}_\psi)]^T, \end{aligned} \tag{4.1}$$

where $\kappa_\phi, \kappa_\psi \in \mathbb{R}^+$ are concentration (anti-variation) parameters, $\boldsymbol{\mu}_\phi, \boldsymbol{\mu}_\psi \in [-\pi, \pi)$ are location parameters and $\boldsymbol{A}$ is a real 2-by-2 matrix that captures the correlation between ϕ and ψ. (Rivest, 1988) introduces a submodel of the above distribution with two fewer parameters and log PDF

$$\begin{aligned}\log f(\phi, \psi|\boldsymbol{\mu}, \boldsymbol{\kappa}, \alpha, \beta) \propto\ & \kappa_\phi \cos(\phi - \boldsymbol{\mu}_\phi) \\ & + \boldsymbol{\kappa}_\psi \cos(\psi - \boldsymbol{\mu}_\psi) + \alpha \cos(\phi - \boldsymbol{\mu}_\phi)\cos(\psi - \mu_p si) \\ & + \beta \sin(\phi - \boldsymbol{\mu}_\phi)\sin(\psi - \boldsymbol{\mu}_\psi),\end{aligned} \tag{4.2}$$

where $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$. (Rivest, 1988) formulated the normalization constant as a doubly infinite series. (Singh, Hnizdo, and Demchuk, 2002) resolved the issue with over-parameterization by setting $\alpha = 0$ and $\beta = \rho \in \mathbb{R}$ in Equation (4.2), yielding the PDF

$$\begin{aligned}f(\phi, \psi|\boldsymbol{\mu}, \boldsymbol{\kappa}, \rho) = C^{-1} \exp(&\boldsymbol{\kappa}_\phi \cos(\phi - \boldsymbol{\mu}_\phi) + \boldsymbol{\kappa}_\psi \cos(\psi - \boldsymbol{\mu}_\psi) \\ &+ \rho \sin(\phi - \boldsymbol{\mu}_\phi)\sin(\psi - \boldsymbol{\mu}_\psi)).\end{aligned} \tag{4.3}$$

The ρ parameters control statistical dependence between the angles ϕ and ψ, with $\rho = 0$ meaning ϕ and ψ are independent and Equation (4.3) reducing too two von Mises distributions. (Singh, Hnizdo, and Demchuk, 2002) also formulated the calculation of the normalization constant C as the infinite series

$$\frac{1}{C} = (2\pi)^2 \sum_{i=0}^{\infty} \binom{2i}{i} \left(\frac{\rho^2}{4\kappa_1\kappa_2}\right)^2 I_i(\kappa_1) I_i(\kappa_2), \tag{4.4}$$

where I_i is the ith order modified Bessel function of the first kind. (Mardia, Taylor, and Subramaniam, 2007) considered another subclass of Equation (4.2) by setting $\alpha = \beta = -\boldsymbol{\kappa}_3$. To distinguish the two subclasses, (Mardia, Taylor, and Subramaniam, 2007) named Equation (4.3) the *sine model* and their own model, the cosine model. (Mardia, Taylor, and Subramaniam, 2007) further proved that Equation (4.3) is unimodal if $\boldsymbol{\kappa}_\phi \boldsymbol{\kappa}_\psi > \rho^2$ and bimodal if $\kappa_\phi \kappa_\psi < \rho^2$, with $\kappa_\phi, \kappa_\psi > 0$ and $\rho \in \mathbb{R}$.

Sine Skewing Procedure

The sine and cosine distributions are symmetric. However, for applications in bioinformatics, symmetry is not necessarily desirable. To address the issue of asymmetry, (Ameijeiras-Alonso and Ley, 2021) introduced a general way to turn any symmetric density $f(\phi, \psi; \boldsymbol{\mu}, \boldsymbol{\theta})$ into a skewed version

$$\begin{aligned}g(\phi, \psi; \boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{\theta}) = & f(\phi, \psi; \boldsymbol{\mu}, \boldsymbol{\theta}) \\ & \times (1 + \boldsymbol{\lambda}_\phi \sin(\phi - \boldsymbol{\mu}_\phi) + \boldsymbol{\lambda}_\psi \sin(\psi - \boldsymbol{\mu}_\psi)),\end{aligned} \tag{4.5}$$

where $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_\phi, \lambda_\psi)^T$ with $|\boldsymbol{\lambda}_i| \leq 1$ and $|\boldsymbol{\lambda}_\phi| + |\boldsymbol{\lambda}_\psi| \leq 1$.

The resulting asymmetric distribution is prefixed with sine-skewed, so skewing the sine distribution yields the sine-skewed sine distribution. (Ameijeiras-Alonso and Ley, 2021) shows that this results in several desirable properties. Most notable is that the normalization constant remains the same. They also show that the skewing procedure preserves the computability of any trigonometric moments of the underlying toroidal distribution. Finally, (Ameijeiras-Alonso and Ley, 2021) provides a simple and efficient simulation scheme for the skewed distributions.

4.3 Implementation

In Pyro and NumPyro, the implementation of a distribution needs to provide four attributes and two methods. The attributes are

1. the batch shape, which denotes the shape of a batch of conditionally independent random variables;
2. the event shape, which denotes the shape of a single random variable;
3. the support (e.g., reals, positive, interval) and its dimensionality; and
4. the parameter types and their dimensionality.

The two methods implement i) calculating the log probability of a batch of events and ii) simulation, which given a desired sample shape must adhere to both the batch shape and the event shape of the distribution.

We can expand an instantiated distribution to an arbitrary batch shape and declare a given number of these dimensions part of an event. We call a distribution instantiated once its parameters are fixed. Computationally, an instantiated distribution is represented as a single object on a hardware device (i.e., GPU or CPU). It is common when running inference to use both devices when a GPU is available. Pyro's computational backend, PyTorch, provides simple methods for checking the current device of an object and moving it to another device. NumPyro's backend, Jax, handles moving objects between devices automatically, but provides manual intervention methods.

Batch Sampling the Sine Distribution

The support of the sine distribution is the 2-torus, and thus an event always comprises two dependent angles. In Pyro and NumPyro, a Python tuple is used to specify the event shape. Thus, the event shape of the sine distribution is always (2,), using the syntax of a singleton Python tuple.

To sample from the distribution in Pyro, we extend the work of (Kent, Ganeiber, and Mardia, 2018) to parallel sampling of the arbitrary batch- and sample shapes. (Kent, Ganeiber, and Mardia, 2018) provides a bivariate rejection-sampling for (among others) the sine distribution using an Angular Central Gaussian (ACG) distribution (Tyler, 1987) as the envelope.

Our extension samples the marginal distribution over the first angle. This requires the sampling method to track and sample missing samples in parallel while avoiding branching and overusing pseudo-random numbers. To track the missing samples, we flatten[6] the sample- and batch-shape and allocate an empty piece of memory $\boldsymbol{M}$ for storing the samples with the flattened shape. Flattening has the advantages of simplifying indexing and providing a coalesced memory layout for GPU computation, which is needed for high utilization.

[6] $(2, 3, 4)$ becomes $(2 * 3 * 4) = (24,)$

To minimize the number of pseudo-random numbers, we sample the smallest number of missing samples from the batches that are not filled, which yields proposals $\boldsymbol{P}$. In order to place the accepted proposals correctly into $\boldsymbol{M}$ (where samples are accumulated) we apply a $n \times m$ Boolean mask matrix $\boldsymbol{B}$, where n is the number of samplers and m total number of desired samples. The entries of $\boldsymbol{B}$ are given by

$$\boldsymbol{B}_{(i,j)} = \left[0 \leq j - \sum_{j'=1}^{j} 1_{\boldsymbol{M}_{(i,j')}} < \sum_{k=1}^{m} 1_{\boldsymbol{M}_{(i,k)}} \right]_{(i,j)}, \tag{4.6}$$

where $1_{\boldsymbol{A}_{(i,j)}}$ is the indicator function for an accepted samples in $\boldsymbol{M}$ at i, j and similarly $1_{\mathbf{P}_{(i,j)}}$ is the indicator for accepted proposals at i, j. Using a Boolean mask to select where accepted proposals are placed, we avoid copying already accepted samples and branching on the Boolean values in the mask. Pyro avoids branching due to the tensor abstraction deployed in PyTorch, which provides primitives for using sub-tensors as references.

Because in NumPyro all tensors are immutable with concrete shapes, the shape of tensors cannot be a function of the state of execution. This means that the optimizations possible in Pyro are not possible in NumPyro. Therefore, the implementation in NumPyro is a straightforward translation of the R method provided in (Kent, Ganeiber, and Mardia, 2018).

Sine Skewing

Sine skewing (Ameijeiras-Alonso and Ley, 2021) is a procedure for breaking the symmetry of a given toroidal distribution. Treating sine skewing as a higher-order method, which takes a distribution and yields its skewed variate, allows for a simple implementation that works for any toroidal distribution with a simulation method.

The simulation method for the skewed distribution follows three steps outlined in (Ameijeiras-Alonso and Ley, 2021). For some k-dimensional base density f with location $\boldsymbol{\mu}$, parameters $\boldsymbol{\theta}$, and skewness $\boldsymbol{\lambda}$ their sampling scheme is

1. Sample $\boldsymbol{y}$ from base density f;
2. Sample u from the uniform density over $[0, 1]$;
3. Generate skewed samples $\boldsymbol{x}$ by

$$\boldsymbol{x} = \begin{cases} \boldsymbol{y} & \text{if } u \leq \frac{1}{2}\left(1 + \sum_{i=1}^{k} \sin(\boldsymbol{y}_i - \boldsymbol{\mu}_i)\right) \\ -\boldsymbol{y} + 2\boldsymbol{\mu} & \text{otherwise,} \end{cases}$$

where $\boldsymbol{y} \in [-\pi, \pi)^k$ is a draw from f and $\boldsymbol{x} \in [-\pi, \pi)^k$ is the corresponding skewed sample.

To compute the log probability density, we simply take the log of Equation (4.5), which yields

$$\log f(\phi, \psi; \boldsymbol{\mu}, \boldsymbol{\theta}) + \log(1 + \boldsymbol{\lambda}_\phi \sin(\phi - \boldsymbol{\mu}_\phi) + \boldsymbol{\lambda}_\psi \sin(\psi - \boldsymbol{\mu}_\psi)),$$

for a density f on the 2-torus.

Benchmarking

We benchmark the sampling time for varying sampling sizes on CPU[7] and GPU[8]. Dispatching to the computation device (i.e., GPU or CPU) is declared in logic and handled by the backend frameworks, so the same high-level code (our implementations) is executed regardless of the device. We compare our implementation of the sine distribution to the implementation provided as a supplement to (Kent, Ganeiber, and Mardia, 2018) in R. We are unaware of publicly available implementations of the sine skewed sine distribution with simulation methods and so did not benchmark our implementation of sine skewing against any reference. The timings are given in log (base ten) scale in Figure 4.2.

For Pyro, the average time of ten repetitions is presented along with the associated standard deviation; only the average timings are given for the reference R implementation. For NumPyro we run eleven repetitions and dismiss the first call which takes between two and seven seconds. We then report the average and deviation for the remaining ten repetitions. We do this because Jax can amortize the high cost of the first iteration (where the python program is traced) upon repeated computation. As expected, we observe that for Pyro, sampling is faster on CPU for a low number of samples (i.e., less than around three thousand); see the solid green lines in the left and middle plots of Figure 4.2. Note that the simulation with skewing adds negligible overhead. The faster simulation for small sample sizes on the CPU is because i) the CPU does not incur overhead from transferring memory between devices and ii) small sample sizes provide limited parallelism for the GPU to exploit. The same plots for NumPyro (cross markers) show the effect of the different computational backend (i.e. Jax). Once the sampling scheme is traced, we observe close to constant computation time for exponentially growing sample sizes on GPU and only a single order of magnitude increase for CPU. The Pyro implementation is faster due to its use of the extended sampling scheme.

For large sample sizes, we see a significant 3114-fold speedup for the GPU in the Pyro framework; and a 6785-fold speedup compared to the reference R implementation. NumPyro provides a more modest 3000-fold speedup compared to the R implementation.

[7] Intel® Xeon® Silver 4114 Processor (13.75M Cache 2.20 GHz).
[8] NVIDIA Quadro RTX 6000.

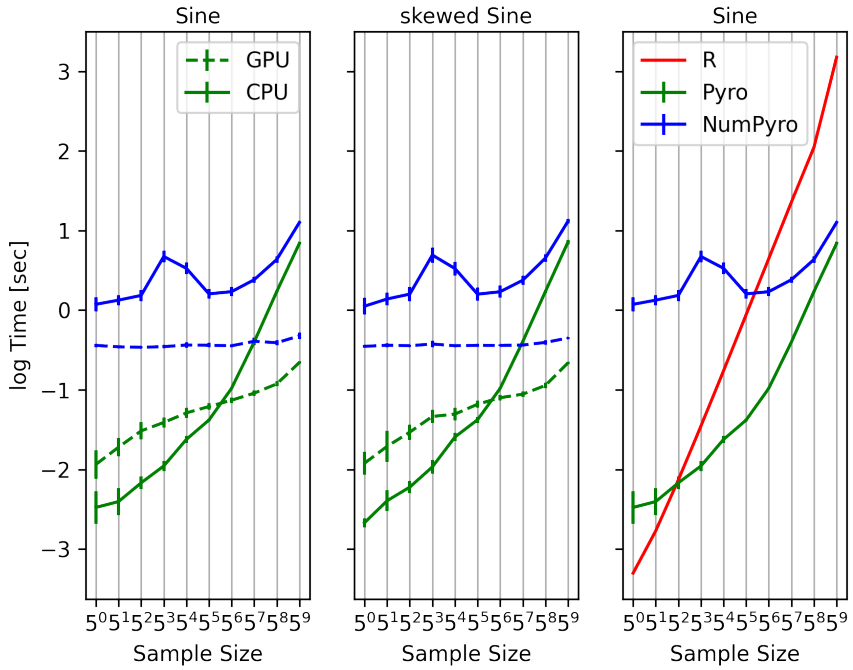


Figure 4.2: Average timings in seconds for differing sample sizes. We compare sampling in R (Kent, Ganeiber, and Mardia, 2018), with our sampling implementations in Pyro and NumPyro.

The right plot of Figure 4.2 shows that the reference implementation of the sine distribution is faster in R for sample sizes smaller than 25, which is due to the overhead associated with Pyro's PyTorch backend and its use of parallel subroutines. This also causes the improved performance of Pyro for larger sample sizes. NumPyro on CPU needs substantially larger sample sizes before outperforming the R implementation.

4.4 Application

To illustrate our implementations, we model the dihedral angles pairs, see Figure 4.3, of serine (S), glycine (G), and proline (P). The models are Bayesian mixture models with sine skewed sine distributions as likelihoods. We infer model parameters with Iterative NUTS in NumPyro. Following the analysis of (Bystroff, Thorsson, and Baker, 2000) (Table 5), we use ten mixture components for glycine, nine for serine, and seven for proline. The angle pairs are taken from the 9mer dataset described in (Thygesen et al., 2021). We restrict our attention to these three amino acids as they are representative of all amino acids.

Data

We obtained our dihedral angle data set from a previously described protein fragment data set (Thygesen et al., 2021). The data set consists of protein fragments of length 9 (9mers) derived from the 3733 proteins selected with cullpdb (Rohl et al., 2004). All included proteins have i) resolution less than 1.6 ångström, ii) R-factor less than 0.25, iii) sequence identity below twenty percent, and iv) sequence identity below 20 percent with proteins used in CASP13. All fragments consist of angles from the allowed region on the Ramachandran plot (Ramachandran, Ramakrishnan, and Sasisekharan, 1963), with outliers removed using the PHENIX software (Liebschner et al., 2019). We use the 60/20/20 split provided in the dataset and pool the angles by amino acid type. There are 47130 angles for proline, 74830 for glycine, and 59304 for serine. However, we only use a random subsample of one thousand angle pairs for each amino acid. We do this to avoid running long chains with enumeration, which scales linearly with the number of mixture components. Further, this alleviates the density peaks present in the data, which are challenging for both HMC and NUTS.

Mixture Model

To model the dihedral angles we consider mixture models of the form

$$\sum_{i=1}^{n} w_i \, \mathrm{g}_{\mathrm{SS}}(\phi, \psi | \boldsymbol{\theta}_i),$$

where $0 < w_i < 1, \sum_i w_i = 1$ are mixture weights and $\boldsymbol{\theta}_i$ denoted all parameters of sine skewed sine distribution g_{SS} with pdf given by Equation (4.5) and antecedent pdf given by Equation (4.3). We refer to the mixture as the Skewed model.

Our approach is close to (Mardia, 2013) who considers a mixture of sine distributions for modeling the entire Ramachandran plot. Also, like our approach, (Ameijeiras-Alonso and Ley, 2021) modeled angle pairs for individual amino

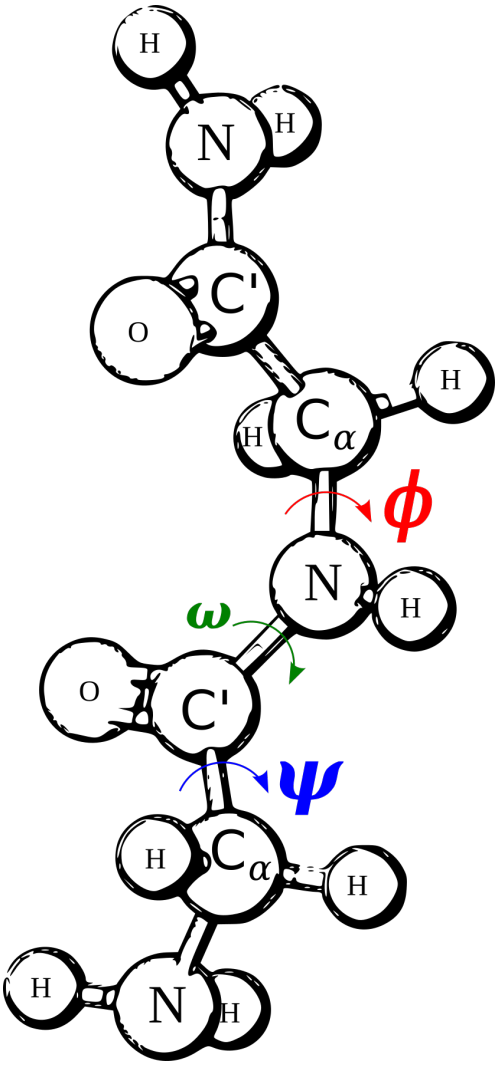


Figure 4.3: A protein is a polymer of amino acids, whose geometry can be quantified by their dihedral or torsion angles. The ϕ, ψ, and ω triplet denotes the torsion angles associated with the $C' - N - C_\alpha - C'$ bonds, the $N - C_\alpha - C' - N$ bonds, and the $C_\alpha - C' - N - C_\alpha$ bonds, respectively. Assuming ideal bond angles and bond lengths, these angles suffice to describe the geometry of the protein's backbone. The ω angle tends to be very close to either *pi* or (in rare cases) 0. Not all triplet values can occur in amino acids, mainly because of steric clashes between atoms. The top row of Figure 4.4 (known as Ramachandran plots) shows empirically measured angles pairs for serine, glycine, and proline. The angle pairs of serine are representative for the rest of the amino acids.

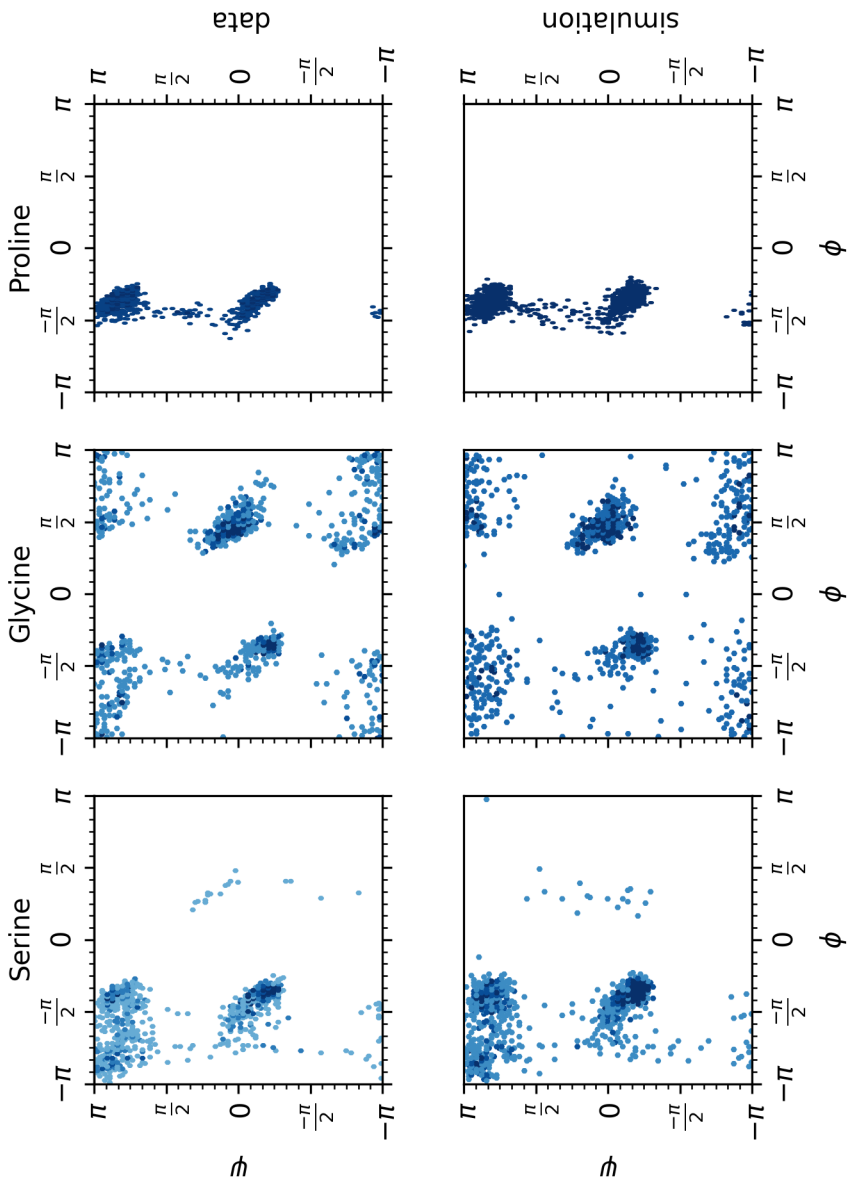


Figure 4.4: The top row shows the Ramachandran plots (see Figure 4.3 for details) of the data used for inference. The bottom row shows the Ramachandran plots of the posterior predictive samples from the Skewed mixture model, see Figure 4.5.

acids. However, they use only a two-component mixture model for comparing a range of toroidal distributions to their skewed variants. Furthermore, unlike (Mardia, 2013; Ameijeiras-Alonso and Ley, 2021) our approach is Bayesian, which allows quantifying the uncertainty of the posterior.

For the concentration of the sine distribution we use a constant scaled Beta prior. As it is not obvious whether to favour high or low concentrations we introduce a hyper prior on the Beta prior. The hyper prior controls the mean $\beta_\mu \sim \texttt{Uni}(0, 1)$, where $\texttt{Uni}$ denotes the uniform distribution, and virtual count $\beta_c \sim \texttt{Gam}(1, 1/20)$, where $\texttt{Gam}$ is the gamma distribution.

The Beta prior is given by $\texttt{Beta}(\beta_\mu\beta_c, (1-\beta_\mu)\beta_c)$. To see this, consider a random variable $X \sim \texttt{Beta}(A, B)$ whose expectation is $\beta_\mu = \mathbb{E}[X] = A/\beta_c = A/(A+B)$. Rearranging yields $\texttt{Beta}(\beta_\mu\beta_c, (1-\beta_\mu)\beta_c)$. The concentration scaling is fixed to seventy, and thus $0 \leq \phi, \psi \leq 70$.

Recall that the location of the sine distribution denote is $\boldsymbol{\mu} = (\mu_\phi, \mu_\psi) \in [-\pi, \pi)^2$. $\boldsymbol{\kappa} = (\kappa_\phi, \kappa_\psi) \in [0, \infty)^2$ denotes its concentration. For the correlation we scale $\sqrt{\kappa_\phi\kappa_\psi}$ by $\rho \in [0, 1]$, which ensures we obtain mixtures of uni-modal sine distributions.

We use uni-variate von Mises distributions on the locations, denoted as $\texttt{vM}(\mu, \kappa)$. Like the sine distribution, the von Mises distribution is parameterized by a location $\mu \in [-\pi, \pi)$ and concentration $\kappa \in [0, \infty)$. We use the prior $\texttt{vM}(\pi, 2)$ on μ_ψ to avoid the forbidden region of the Ramachandran plot. For μ_ψ, we choose a flat von Mises prior.

For skewness we use a uniform prior. Recall the skewness is constrained to $|\lambda_\phi| + |\lambda_\psi| \leq 1$. We accommodate this constraint by first sampling λ_ϕ uniformly from $[-1, 1]$ and then using the sampled λ_ϕ to decide the support interval (i.e., $[-1 + |\lambda_\phi|, 1 - |\lambda_\phi|]$) for the uniform sample of λ_ψ.

For correlation we put a $\texttt{Beta}(2, 10)$ prior on the scale factor ρ. The prior favors a low correlation over maintaining relatively high sample efficiency for the posterior predictive samples.

Finally, we put a Dirichlet distribution, denoted Dir, of our mixture weights.

Bringing it all together, we can write the skewed model as

Procedure 2 Skewed Model

1: $\beta_{\mu_\phi} \sim \texttt{Uni}\,(0, 1)$
2: $\beta_{c_\phi} \sim \texttt{Gam}\,(1, 1/20)$
3: $\beta_{\mu_\psi} \sim \texttt{Uni}\,(0, 1)$
4: $\beta_{c_\psi} \sim \texttt{Gam}\,(1, 1/20)$
5: **for** $i \in [1, \ldots, m]$ **do** ▷ Mixing plate
6: $\quad \boldsymbol{\mu}_\phi^{(i)} \sim \texttt{vM}\,(\pi, 2)$
7: $\quad \boldsymbol{\mu}_\psi^{(i)} \sim \texttt{vM}\,(0, 0.1)$
8: $\quad \boldsymbol{\kappa}_\psi^{(i)} \sim \texttt{Beta}(\beta_{\boldsymbol{\mu}_\psi}\beta_{c_\psi}, (1 - \beta_{\boldsymbol{\mu}_\psi})\beta_{c_\psi})$
9: $\quad \boldsymbol{\kappa}_\phi^{(i)} \sim \texttt{Beta}(\beta_{\boldsymbol{\mu}_\phi}\beta_{c_\phi}, (1 - \beta_{\boldsymbol{\mu}_\phi})\beta_{c_\phi})$
10: $\quad \boldsymbol{\rho}^{(i)} \sim \texttt{Beta}(2, 10)$
11: $\quad \boldsymbol{\lambda}_\phi^{(i)} \sim \texttt{Uni}(-1, 1)$
12: $\quad \boldsymbol{\lambda}_\psi^{(i)} \sim \texttt{Uni}(-1 + |\boldsymbol{\lambda}_\phi^{(i)}|, 1 - |\boldsymbol{\lambda}_\phi^{(i)}|)$
13: $\quad \boldsymbol{\pi}^{(i)} \sim \mathrm{Dir}(1)$
14: $\quad \boldsymbol{\theta}^{(i)} \leftarrow (\boldsymbol{\mu}_\phi, \boldsymbol{\mu}_\psi, 70\boldsymbol{\kappa}_\phi, 70\boldsymbol{\kappa}_\psi, \boldsymbol{\rho}\sqrt{\boldsymbol{\kappa}_\phi\boldsymbol{\kappa}_\psi}, \boldsymbol{\lambda}_\phi, \boldsymbol{\lambda}_\psi)$
15: **end for**
16: **for** $j \in [1, \ldots, D]$ **do** ▷ Observation plate
17: $\quad (\boldsymbol{\phi}^{(j)}, \boldsymbol{\psi}^{(j)}) \sim \sum_{i=1}^{m} \boldsymbol{\pi}^{(i)}\, \mathrm{gss}^{(i)}\left(\phi, \psi; \boldsymbol{\theta}^{(i)}\right)$
18: **end for**

where m is number of mixture components, D is number angle pairs observations, and $\circ$ is the Hardamard product between real vectors. The Skewed model is presented graphically in Figure 4.5. Note the mixture plate (line 3) corresponds to the outer rectangle in Figure 4.5 and the observation plate (line 13) to the inner square in Figure 4.5.

The informative priors used in are somewhat ad-hoc, and more structured investigations into what constitutes good/optimal priors would be of interest for both the fields of Bioinformatics and Directional Statistics.

NumPyro caches support transformations (e.g., $h : [-1, 1] \to \mathbb{R}$) for distributions with constrained support. The caching poses a challenge for the uniform skewness prior. To see the problem, note that the support of λ_ψ in Algorithm 2 line 9 depends on the current value of λ_ϕ. Therefore, after the first accepted proposal, the support transformation is stale, i.e., referring to an earlier computation state invalid for the current state. We opted for the equivalent $\lambda_\psi \sim (1 - |\lambda_\phi|)\,\texttt{Uni}(-1, 1)$ in code as it makes the support static (i.e., always $[-1, 1]$). [9]

The full code example is distributed with the NumPyro framework.

[9] During integration into Pyro, Du Phat (a core Pyro-PPL developer) noted the stick-breaking transformation (Sethuraman, 1994) could be used for a more elegant and generalized solution.

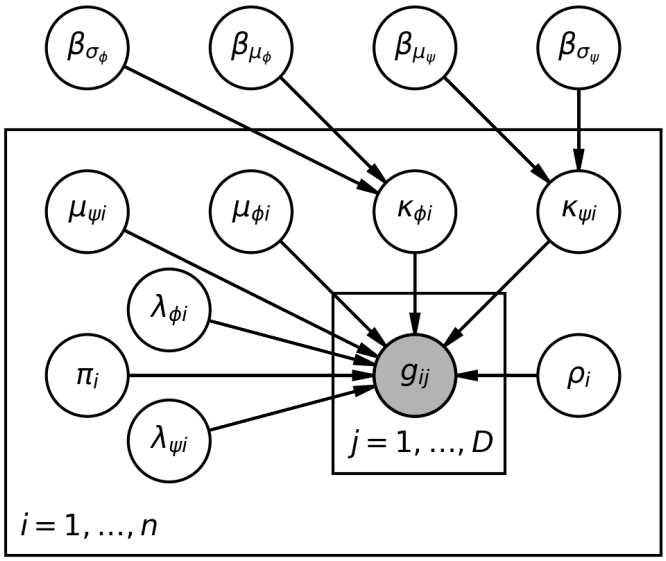


Figure 4.5: Probabilistic graphical model representation of the hierarchical Bayesian mixture model using the sine skewed sine distribution. The large plate (shown as rectangle) applies to the n mixture components and the small plate applies to the D (i.e., 1000) angles.

Amino Acid	ESS	$\hat{R}$	Divergences	warmup	samples	Inference time	Accept prob
Serine	1575.88	1	0	500	1500	7m23s	0.91
Glycine	1448.33	1	0	500	1500	6m40s	0.90
Proline	1384.20	1	0	500	1500	6m25s	0.93

Table 4.1: Summary statistics of Markov chains for models trained on individual amino acids. Statistics shown include mean effective sample size (ESS), mean Gelman-Rubin diagnostic $\hat{R}$, number of divergent samples, number of warmup samples, the total number of samples, the total time for sampling, and mean acceptance probability.

Inference

We infer parameters using iterative NUTS in NumPyro. NUTS was proposed by (Hoffman, Gelman, et al., 2014) and introduces a stop criterion for the trajectory of the Hamiltonian dynamics in HMC. Iterative NUTS is an extension to NUTS by (Phan, Pradhan, and Jankowiak, 2019a), which introduces a data-structure on the trajectory that makes NUTS work well with Jax.

We initialize the sine distribution locations with k-means clustering. We set the max tree depth for NUTS to seven, use target probability 0.8 and initial step length 1.71×10^{-2}. We run the chain for one thousand five hundred samples with five hundred burn-in, called warmup, samples. The time take

to sample the chains (inference time) is about seven minutes on our CPU[10]. The mean MH-ratio, called acceptance probability, is slightly greater than our target probability (0.8). However, both chain diagnostics (ESS and $\hat{R}$) indicate the chains have converged. The diagnostics are detailed in Section 4.6. All chain statistics are gathered in Table 4.1.

Results

Figure 4.4 shows that simulation from the posterior predictive distribution (bottom row) is visually close to the observation angles (top row). For glycine (middle row) the simulation has put some density in the forbidden region of the Ramachandran plot. The local density in the Ramachandran plot can be highly peaked, which poses a challenge for iterative NUTS. We handle this here by subsampling the data.

[10]Intel® Xeon® Silver 4114 Processor (13.75M Cache, 2.20 GHz).

4.5 Discussion

We presented efficient implementations of the sine distribution and the sine skewing procedure for the 2-torus in the PPLs Pyro and NumPyro. Currently, efficient implementations of directional distributions are generally scarce, with the univariate von Mises distribution as the notable exception. Therefore, we believe there is considerable untapped potential in making these methodologies more generally available to modern software running on parallel hardware. Another promising research direction is tailoring inference algorithms to work on non-Euclidean manifolds. In particular, implementing the so-called reparameterization trick (Kingma, Salimans, and Welling, 2015) for toroidal distributions would make these distributions available as directional priors in variational autoencoders (Xu and Durrett, 2018).

	mean	std	median	5.0%	95.0%	n_eff	$\hat{R}$
beta_mean_phi	0.98	0.03	0.99	0.94	1.00	7.01	1.22
beta_mean_psi	0.99	0.01	0.99	0.98	1.00	33.65	1.02
beta_prec_phi	18.11	18.51	12.27	0.44	47.61	9.83	1.08
beta_prec_psi	23.22	16.91	18.89	1.93	44.83	55.77	1.02
corr_scale[0]	0.02	0.01	0.02	0.00	0.04	81.95	1.00
corr_scale[1]	0.02	0.02	0.02	0.00	0.04	21.03	1.05
corr_scale[2]	0.01	0.01	0.01	0.00	0.02	126.25	1.00
corr_scale[3]	0.00	0.00	0.00	0.00	0.01	108.85	1.00
corr_scale[4]	0.08	0.04	0.08	0.01	0.13	17.91	1.01
corr_scale[5]	0.07	0.04	0.07	0.01	0.13	100.84	1.00
corr_scale[6]	0.05	0.03	0.05	0.00	0.09	46.31	1.09
corr_scale[7]	0.00	0.00	0.00	0.00	0.01	13.63	1.01

...

Number of Hamiltonian trajectory divergences: 0

4.6 Pyro PPL automatic MCMC summary statistics

Below is a summary of Pyro's and NumPyro's summary statistics for the MCMC chain run for 3000 steps with a mixture of fifteen sine models on the serine angle-pairs. The Rubin-Gelman Diagnostic $\hat{R}$ (Gelman, Rubin, et al., 1992) for two MCMC chains x and y of length n as

$$\hat{R} = \sqrt{\frac{1/n \sum_{i=1}^{n} \mathrm{Var}[(x_i, y_i)]}{(n-1)/n^2 \sum_{i=1}^{n} \mathrm{Var}[(x_i, y_i)] + \mathrm{Var}[(x_j + y_j)_{j=1}^{n}]}}.$$

For a single chain $\hat{R}$ is computed by comparing the first half to the second half of the chain. $1 \leq \hat{R} < 1.1$ is generally considered an indicator of convergence.

The effective sample size, n_eff, measures the increase in uncertainty due to within-chain auto-correlation. For m chains of length n the effective sample size can be estimated by

$$\text{n\_eff} = \frac{mn}{-1 + 2\sum_{t=0}^{m}(\rho_{2t} + \rho_{2t+1})}$$

where ρ_t is the estimated auto-correlation with lag t.

Divergence are registered when the energy difference between the head of the chain and the proposal is greater than one thousand.

5.0% and 95.0% denote the boundaries of the posterior density with 90% of the probability mass for each variable.

Chapter 5

Probabilistic Differentiable Molecular Simluation

Ola Rønning[1], Christophe Ley[2], and Thomas Hamelryck[13]

5.1 Introduction

The new frontier of protein structure prediction is finding the distribution of probable conformations (Lane, 2023). Current deep methods like AlphaFold2(Jumper et al., 2021), OmegaFold (Wu et al., 2022), and RoseTTAFold (Baek et al., 2021) give high-resolution snapshots from the native conformation ensemble. To reach the new frontier, the forecaster must reason about the ensemble rather than snapshots and adhere to experimental observations of the protein-water system and its dynamics. The challenge for current methods is that the methods embed the folding process in deep networks, which are black-box forecasters and often found overly confident in predictions (Nguyen, Yosinski, and Clune, 2015).

To overcome these challenges, we propose formulating the forecaster as a DPP using a rotation invariant heteroscedastic scoring function to learn an implicit solvent all-atoms energy function. The advantage of such an approach is threefold. First, the statistical models allow us to separate the folding dynamics from uncertainty in the experimental observations (native conformation ensemble). We propose using the Langevin equations (as detailed in Section 5.2)for accounting for stochasticity in the dynamics and the above-mentioned scoring function to account for all probable native conformations. Second, we will base our stochastic model on an empirical force field developed by (Irbäck and Mohanty, 2006; Irbäck, Mitternacht, and Mohanty, 2009) (known as ProFasi). ProFasi is a course-grained force field with 29 parameters

[1]Department of Computer Science, University of Copenhagen, Denmark
[2]Département Mathématiques, Université du Luxembourg, Luxembourg
[3]Department of Biology, University of Copenhagen, Denmark

that account for local and global interactions as detailed in Section 5.3. With a DPP formulation of ProFasi we can easily extend the expressibility (and granularity) of the force field by including (stochastic) neural networks–at the expense of interpretability. Third and finally, our approach is generative for the discretized folding trajectory and so allows us to infer statistical properties of both the trajectory and the native ensemble of conformations.

This chapter outlines the main components of our approach to protein folding but does not include an evaluation of our approach. Our approach relies on a flexible statistical inference engine which we believe we have developed with the EinStein library (see Chapter 2 and Chapter 3).

5.2 Statistical model

We model the folding of single peptides to their native conformation in an implicit solvent under constant temperature. The temperature is chosen such that the protein should reach the native conformation within a fixed folding time.

The native conformation likelihood We use the Bayesian version of the Theseus model (Moreta et al., 2019) for observed native conformations $\mathcal{D}$ given by

$$p(\mathcal{D}|\mathbf{M}, \mathbf{R}, \mathbf{U}) = \prod_{i}^{N} \mathcal{N}(\mathcal{D}_i|\mathbf{M}_i\mathbf{R}_i, \mathbf{U}),$$

where $\mathbf{M}_i$ is the noise free folded conformation at time T, $\mathbf{R}_i$ is the rotation that aligns $\mathcal{D}_i$ with $\mathbf{M}_i$ and $\mathbf{U}$ is the atom level covariance. Unlike conventional methods, the Theseus model can account for the rotational invariance of peptide conformations and the heteroscedasticity of atom positions in the peptide. We consider the alignment rotation $\mathbf{R}_i$ a random variable in SO3. We construct the distribution of $\mathbf{R}_i$ by transporting two projected Gaussian random variables to SO3 using a generalization of Perez-Sala et al., 2013 uniform distribution over the space of rotations. We assume atoms are pairs independent and use an empirical estimate of their variance derived from the b-factor of the observations (see Sun et al., 2019 for details).

Folding trajectory We model the dynamics of the folding trajectory $\mathcal{T}_t^{U_\theta}$ by the Langevin equations (Gillespie, 2000)

$$m\ddot{x} = \nabla_x U(x) + \lambda\dot{x} + \sqrt{2k_bT}\eta(t)$$

where $T \geq 0$ is the temperature in Kelvin, $U : \mathbb{R}^{3n} \to \mathbb{R}$ is the potential energy, k_b is the Boltzmann constant, $\lambda \geq 0$ is the friction constant and η is a stationary zero mean multivariate Gaussian process. By integrating with respect to time dt' we obtain configuration x_t by following the stochastic trajectory given by

$$\begin{aligned}
\mathcal{T}_t^{U_\theta} x_0 &= x_0 + \int_0^t dt' \nabla_x U_\theta(x(t')) + \int_0^t dt' \eta(t') && \eta(t) \sim N(\mathbf{0}, \sqrt{2k_b\lambda\tau}) \\
&= x_0 + \int_0^t dt' \nabla_x U_\theta(x(t')) + tN(\mathbf{0}, 2k_b\lambda T) && \left(z_0 = x_0 + tN(\mathbf{0}, \sqrt{2k_b\lambda\tau})\right) \\
&= z_0 + \int_0^t dt' \nabla_x U_\theta(x(t'))
\end{aligned}$$

with initial conformation x_0 and at temperature τ. We use that the stationary Gaussian process is independent of time to reduce the trajectory from a

stochastic differential equation to an ordinary differential equation (ODE). The simplifications allow us to compute the trajectory by Euler discretization (Ortega and Poole, 1981), which gives the recurrence

$$x_{t+1} = x_t + \Delta_t \nabla_x U_\theta(x(t)).$$

We can obtain the gradient with respect to θ reverse mode automatic differentiation. This will be computationally efficient. However, the memory overhead will be proportional to the size of the discretization. To overcome the memory overhead, we strip mine the discretization and use recomputation of the forward AD sweep as proposed by Schenck et al., 2022.

The conformation potential For potential, we use a variate of ProFasi (Irbäck and Mohanty, 2006; Irbäck, Mitternacht, and Mohanty, 2009), an all atom's potential with implicit solvent. The potential decomposes additively into terms that model local and global interactions. The potential is given by

$$U_{\text{tot}} = \sum_{i=1}^{3} E_{\text{loc}}^{(i)} + E_{\text{hb}} + E_{\text{hp}} + E_{\text{ch}} + E_{\text{ev}}.$$

The local interactions (denoted by $E_{\text{loc}}^{(\cdot)}$) model repulsion between charged atoms in consecutive peptide units, as well as the potential of the side-chain conformations. The global contribution includes a term modeling hydrogen bond formation (denoted E_{hb}), a term for the hydrophobic effect (E_{hp}), one for side chain charge (E_{ch}), and a term modeling steric clashes (E_{ev} where "ev" abbreviates excluded volume). The exact functional form we use varies slightly from Irbäck and Mohanty, 2006; Irbäck, Mitternacht, and Mohanty, 2009 due to our use of array programming and our choice of folding dynamic. In Section 5.3, we give a complete account of our ProFasi variate.

5.3 ProFasi potential

Irbäck, Mitternacht, and Mohanty, 2009 describes the implicit solvent all-atoms potential (called ProFasi) that we emulate. Due to the array programming style of most modern linear algebra libraries in Python (which includes `Jax`), our potential varies slightly from that available in the (`C++` based) ProFasi software package Irbäck and Mohanty, 2006. We detail the exact energy terms in our potential below.

Let $a_i, a_j \in \mathcal{P}$ be the (Cartesian) coordinates of the i'th (and j'th) atom in protein sequence $\mathcal{P}$ order from the N-terminus to C-terminus. We denote the euclidean distance as $\|a_i - a_j\|$ (measured in angstrøm) and use $\text{adj}_{i,j}$ for adjacency between atoms (adjacent atoms are fixed with respect to each other, details are given in Section 5.4). We denote free parameters by $w_\cdot \in \mathbb{R}$, and generally name them according to their associated energy term. Our potential has 29 free parameters in total.

Hydrogen bonding term In our potential we model hydrogen bonding between amino (NH) groups and carbonyl (CO) group. Hydrogen bonds occur between two backbone groups, or a backbone group and a charged side-chain group (i.e. Aspartic acid (Asp), Glutamatic acid (Glu), Lysine (Lys), and Arginine (Arg)).

The hydrogen bonding potential is given by

$$e_{\text{hb}} = \sum_{k<l} w_{\text{hb}}(k,l) \left[u\left(\|a^k_{\text{H}} - a^l_{\text{O}}\| \right) v(\angle_{k,l}\text{NH}\ldots\text{O}, \angle_{k,l}\text{H}\ldots\text{OC}) \right]$$

where k index the amino groups in $\mathcal{P}$, l index the carbonyl groups in $\mathcal{P}$, $w_{\text{hb}}(k,l)$ is a free parameter, $u(\cdot)$ mimics a Lennard-Jones potential, and $v(\cdot,\cdot)$ is a misalignment penalty. The functional form of the Lennard-Jones potential is

$$u(\Delta) = 5\left(\frac{\sigma_{\text{hb}}}{\Delta}\right)^{12} - 6\left(\frac{\sigma_{\text{hb}}}{\Delta}\right)^{10},$$

with van der Waals radius $\sigma_{\text{hb}} = 2$, and the misalignment penalty is given by

$$v(\angle_1, \angle_2) = \begin{cases} \sqrt{cos(\angle_1)cos(\angle_2)} & \text{if } \angle_1, \angle_2 > \pi/2 \\ 0 & \text{otherwise.} \end{cases}$$

Like Irbäck and Mohanty, 2006 we set the free parameter $w_{\text{hb}}(k,l)$ as

$$w_{\text{hb}}(k,l) = \begin{cases} w_{\text{bb}} & \text{if } l,k \text{ are both in the backbone} \\ 0 & \text{if } l,k \text{ are in both side-chains} \\ 0 & \text{if } l,k \text{ are in consecutive peptide-bond units} \\ w_{\text{sc}} & \text{otherwise,} \end{cases}$$

where Irbäck and Mohanty, 2006 uses $w_{\text{bb}}, w_{\text{sc}} = (3, 2.3)$.

Partial peptide charge term The partial peptide charge term represents interactions between partial charges in consecutive peptide bond units. We index the peptide units by n and order them from N-terminus to C-terminus. For a protein with N amino acids, we have $N-1$ peptide units (which we denote p_n). Each peptide units contains the four atoms $(\mathrm{C}_n, \mathrm{O}_n, \mathrm{N}_n, \mathrm{H}_n) \equiv r_n$, where C and O are in the n'th amino acid, and N and H are in the next amino acid. The partial peptide charge term is given by

$$e^{(1)}_{\text{loc}} = w^{(1)}_{\text{loc}} \sum_{n=1}^{N-2} \sum_{i \in p_n} \sum_{j \in p_{n+1}} \frac{q_i q_j}{\|a_i - a_j\|},$$

where $q \in (0.42, -0.42, 0.2, -0.2)$ is the partial charge for C, O, N, and H respectively.

Peptide oxygen-oxygen and hydrogen-hydrogen repulsion term The peptide OO and HH repulsion term provide a repulsion for neighbouring peptides when the OO or HH are exposed to each other. This makes the formation of double hydrogen bonds less likely. Glycine has different backbone potentials due to its lack of a C_β atom, so peptide units which share a Glycine are excluded. To simplify notation we therefore index by the order of the residue (denoted r_n), rather than the peptide units. The peptide units which share a Glycine are represented in Glycine ψ-angle penalty term. The functional form of the OO and HH repulsion term is

$$e^{(2)}_{\text{loc}} = w^{(2)}_{\text{loc}} \sum_{n | r_n \neq \text{Gly}} f\Delta^{(n)}_{\text{HH}} + f\Delta^{(n)}_{\text{OO}}$$

where f is the map $x \mapsto \max(0, \tanh 3x)$,

$$\Delta^{(n)}_{\text{HH}} = \left(\min\left(\|a^n_{\text{H}} - a^{n+1}_{\text{N}}\|, \|a^{n+1}_{\text{H}} - a^n_{\text{N}}\|\right) - \|a^n_{\text{H}} - a^{n+1}_{\text{H}}\|\right),$$

and

$$\Delta^{(n)}_{\text{OO}} = \left(\min\left(\|a^{n-1}_{\text{C}} - a^n_{\text{O}}\|, \|a^n_{\text{C}} - a^{n-1}_{\text{O}}\|\right) - \|a^{n-1}_{\text{O}} - a^n_{\text{O}}\|\right).$$

Note that when the peptide units cover a Proline we let $\Delta_{\text{HH}} = 0$, as it lacks the hydrogen.

Glycine ψ-angle penalty term The lack of a C_β atom in Glycine makes it considerably more flexible. However, Irbäck, Mitternacht, and Mohanty, 2009 notes that the observed Ramachandran plot for Glycine is not as spread as a steric consideration would suggest. The Glycine ψ-angle penalty term penalise $\psi = \angle \mathrm{NC}_\alpha \ldots \mathrm{CN}$ torsion angles around $\pm 2\pi/3$ radians. The term is given by

$$e^{G}_{\text{loc}} = w^{G}_{\text{loc}} \sum_{n | r_n = \text{Gly}} \cos\psi_n + 2\cos 2\psi_n.$$

Table 5.1: Side-chain torsion angle classifications $\kappa(\cdot,\cdot)$ (reproduced from Irbäck, Mitternacht, and Mohanty, 2009). χ_i denotes the i'th side-chain torsion angle ordered according to .

Residue	χ_1	χ_2	χ_3	χ_4
Ser, Cys, Thr, Val	I			
Ile, Leu	I	I		
Asp, Asn	I	IV		
His, Phe, Tyr, Trp	I	III		
Met	I	I	II	
Glu, Gln	I	I	IV	
Lys	I	I	I	I
Arg	I	I	I	III

Side-chain rotamer term The potential for the side-chain torsion angles (denoted $\chi_\cdot$) is expressed by the side-chain rotamer term. The distribution of angles follow steric considerations, so four prototypical angles are sufficient. Table 5.1 give the angle classifications according to the residue r and side-chain torsion angle index i (i.e. $\kappa(r,\chi) \in \{\mathrm{I},\mathrm{II},\mathrm{III},\mathrm{IV}\}$). The potential term is given by

$$e^{(3)}_{\mathrm{loc}} = \sum_n \sum_{i \in r_n} w^{(3)}_{\mathrm{loc}}(\kappa(r_n,\chi_i)) \cos n(\kappa(r_n,\chi_i))\chi_i$$

where $w^{(3)}_{\mathrm{loc}}(\cdot)$ and $n(\cdot)$ are step functions given by

$$w^{(3)}_{\mathrm{loc}}(c) = \begin{cases} 0.6 & \text{if } c = \mathrm{I} \\ 0.3 & \text{if } c = \mathrm{II} \\ 0.4 & \text{if } c = \mathrm{III} \\ -0.4 & \text{if } c = \mathrm{IV} \end{cases}, \quad n(c) = \begin{cases} 0.6 & \text{if } c = \mathrm{I} \\ 0.3 & \text{if } c = \mathrm{II} \\ 0.4 & \text{if } c = \mathrm{III} \\ -0.4 & \text{if } c = \mathrm{IV} \end{cases}.$$

Hydrophobic effect term The hydrophobic effect term models the hydrophobic interactions between the residues listed in Table 5.2. The term takes the functional form

$$e_{\mathrm{hp}} = -\sum_{n<m} C\left(r^{(hp)}_n, r^{(hp)}_m\right)\left(h(r^{(hp)}_n) + h(r^{(hp)}_m)\right)$$

where $C(\cdot,\cdot)$ denotes the contact measure between two residues and $h(\cdot)$ is the hydrophobic energy. We model the hydrophobic energy as additive to avoid

Table 5.2: Atoms used to compute the contact measure for the hydrophobic effect (reproduced from Irbäck, Mitternacht, and Mohanty, 2009)

Residue	$r^{(hp)}$
Pro	$\{C_\beta, C_\gamma, C_\delta\}$
Tyr	$\{C_\gamma, C_{\delta 1}, C_{\delta 2}, C_{\epsilon 1}, C_{\epsilon 2}, C_\zeta\}$
Val	$\{C_\beta, C_{\gamma 1}, C_{\gamma 2}\}$
Ile	$\{C_\beta, C_{\gamma 1}, C_{\gamma 2}, C_\delta\}$
Leu	$\{C_\beta, C_\gamma, C_{\delta 1}, C_{\delta 1}\}$
Met	$\{C_\beta, C_\gamma, S_\delta, C_\epsilon\}$
Phe	$\{C_\gamma, C_{\delta 1}, C_{\delta 2}, C_{\epsilon 1}, C_{\epsilon 2}, C_\zeta\}$
Trp	$\{C_\gamma, C_{\delta 1}, C_{\delta 2}, C_{\epsilon 3}, C_{\zeta 3}, C_{\eta 2}\}$
Arg	$\{C_\beta, C_\gamma\}$
Lys	$\{C_\beta, C_\gamma, C_\delta\}$

introducing more free parameters. The energy is given by

$$h(r) = \begin{cases} 0.3 & \text{if } r \in \{\text{Arg}\} \\ 0.4 & \text{if } r \in \{\text{Met}, \text{Lys}\} \\ 0.6 & \text{if } r \in \{\text{Val}\} \\ 0.8 & \text{if } r \in \{\text{Ile}, \text{Leu}, \text{Pro}\} \\ 1.1 & \text{if } r \in \{\text{Tyr}\} \\ 1.6 & \text{if } r \in \{\text{Phe}, \text{Trp}\} \end{cases}.$$

The contact measure run over the atoms given in Table 5.2 and its functional form is

$$C(r_n, r_m) = \frac{\min\left(\gamma(r_n, r_m)(|r_n| + |r_m|), \Gamma(r_n, r_m) + \Gamma(r_n, r_m)\right)}{\gamma(r_n, r_m)(|r_n| + |r_m|)},$$

where $|\cdot|$ denotes the count measure, $\gamma(\cdot, \cdot)$ is a step function given by

$$\gamma(r_i, r_j) = \begin{cases} 0.75 & \text{if } r_i \wedge r_j \in \{\text{Pro}, \text{Phe}, \text{Tyr}, \text{Ile}, \text{Leu}, \text{Val}\} \\ 1 & \text{otherwise} \end{cases}, \tag{5.1}$$

and Γ is a function. The Γ function is given by

$$\Gamma(r_i, r_j) = \sum_{a_k \in r_i} \min_{a_l \in r_j} \max\left[\min\left(\frac{-\|a_k - a_l\|^2}{\Delta^\top - \Delta_\perp} - \Delta^\top, 1\right), 0\right],$$

where $\Delta_\top$ is an upper distance threshold and $\Delta_\perp$ is a lower distance threshold. Irbäck and Mohanty, 2006 uses $\Delta_\perp = 3.7^2 \leq \|a_k - a_l\| \leq 4.5^2 = \Delta^\top$.

Table 5.3: Atoms used to compute the side-chain charge contact measure (reproduced from Irbäck, Mitternacht, and Mohanty, 2009)

Residue	$r^{(ch)}$
Arg	$\{N_\epsilon, C_\zeta, N_{\eta 1}, N_{\eta 2}\}$
Lys	$\{H^1_\zeta, H^2_\zeta, H^3_\zeta\}$
Asp	$\{O_{\delta 1}, O_{\delta 2}\}$
Glu	$\{O_{\epsilon 1}, O_{\epsilon 2}\}$

Side-chain charge term The side-chain charge term models interactions among residues with charged side chains. The residues contributing to the term are Arg, Lys, Asp, and Glu. The functional for of the side-chain charge term is

$$e_{\text{ch}} = -w_{\text{ch}} \sum_{n<m} q(r_n^{(ch)}, r_m^{(ch)}) C(r_n, r_m)$$

where $C(\cdot,\cdot)$ is the contact measure (which has functionally similar to Equation (5.1)) and $q(\cdot,\cdot)$ is the charge. Table 5.3 lists the atoms used by the contact measure. $C(\cdot,\cdot)$ varies from Equation (5.1) by letting $\gamma(\cdot,\cdot) = 1$. To compute the charge we use

$$q(r_i, r_j) = -s(r_i)s(r_j)$$

where $s(\cdot)$ is the charge sign with functional form

$$s(r) = \begin{cases} 1 & \text{if } r \in \{\text{Lys}, \text{Arg}\} \\ -1 & \text{if } r \in \{\text{Asp}, \text{Glu}\} \end{cases}.$$

Spring (bond length) energy term The ProFasi potential uses ideal covalent bond lengths. While folding a protein the proposed configurations maintain the covalent bond lengths by augmenting tosion angles. This update is not suitable for a differential simulator. To insure the conformations adhere to the ideal bond length we therefore introduce a square (spring) potential on all covalent bonds in the protein. The energy term is applied together with the bond angle energy term in a separate step from the rest of the potential. The spring energy is given by

$$e_{\text{sp}} = \sum_{i,j \in \text{adj}(\mathcal{P})} (r_{i,j} - c_{i,j})^2,$$

where $r_{i,j}$ is the distance between atoms i and j in protein $\mathcal{P}$ and $c_{i,j}$ is the ideal bond length between the two atoms. Note that e_{sp} is quadratic in $r_{i,j}$ and so we can obtain $e_{sp} = 0$ with a single step using Newton's method.

(Covalent) bond angle energy term

Excluded volume term The excluded volume potential energy is then given by

$$e_{\text{ev}} = w_{\text{ev}} \sum_{i<j} \lambda_{i,j} \begin{cases} \left(\frac{(\sigma_i+\sigma_j)}{\|a_i-a_j\|}\right)^{12} & \text{if } \neg \text{adj}_{i,j} \\ 0 & \text{otherwise} \end{cases},$$

where $w_{\text{ev}}, \lambda_{i,j} \in \mathbb{R}$ are free parameters (Irbäck and Mohanty, 2006 uses $w_{\text{ev}} = 0.1$) and σ_i is the i'th atoms radius (we model atoms as fixed size spheres). The $\lambda_{i,j}$ factor corrects the local potential when atoms are bonded by a triple covalent bond (Favrin et al., 2003; Irbäck, Mitternacht, and Mohanty, 2009), we use

$$\lambda_{i,j} = \begin{cases} 1 & \text{if i,j are triple bonded} \\ 3/4 & \text{otherwise} \end{cases}.$$

For atom radius we use $\sigma \in (1.77, 1.75, 1.53, 1.42, 1)$ for S, C, N, O and H respectively (Irbäck, Mitternacht, and Mohanty, 2009). Note that Irbäck and Mohanty, 2006 use a cutoff $\|a_i - a_j\|_2 \leq 4.3$ as a computational optimization. In the `Jax` framework we cannot use a radius cutoff as all indices must be known at run-time (that is index arrays must be concrete).

5.4 Pairwise fixed atoms

ProFasi operates with fixed bond lengths and some per residue fixed structure; however, it is not apparent from (Irbäck, Mitternacht, and Mohanty, 2009; Irbäck and Mohanty, 2006) exactly which atom pairs to consider fixed. To ascertain the fixed structures, we, therefore, simulate the dynamics of permutations of an (artificial) protein that contained all twenty amino acids for 5000 cycles in ProFasi and computed the variance of the pairwise distances of all atoms vgbfcdover time (the 5000 cycles). We choose the permutations, so every amino acid occurs at both terminals and once inside the protein.

Epilogue

Discussion of results

SVGD requires increasing particles to represent higher dimensional models (Ba et al., 2021). This is computationally burdensome, as SVGD scales quadratically in complexity with the number of particles. Nalisnick and Smyth, 2017 empirically demonstrate that we can improve performance on real-world (high-dimensional) problems while using fewer particles by lifting particles to densities. We generalize the method of Nalisnick and Smyth, 2017 using the Rényi α-divergence, and prove that their method corresponds to using $\alpha = 0$. This is important, as we show that using $\alpha = 1$ instead of $\alpha = 0$ leads to significantly better results. We show this for many problems, ranging from a regression in housing prices to image classification, using Bayesian neural networks and variational autoencoders. We also show that our method performs better than others, including MFVI, Laplace approximation, and SVGD. In addition, we demonstrate why our method does better: our method reduces the noise in the gradient estimation.

EinStein provides the latest techniques for SteinVI as an extension to Numpyro. Our results indicate that the library is substantially faster and more expressive than other available libraries for SteinVI. EinStein provides a familiar and efficient interface for practitioners working with the Pyro/NumPyro PPL and provides a unified code base to researchers for benchmarking new developments in SteinVI.

We presented efficient implementations of the sine distribution and the sine skewing procedure for the 2-torus in the PPLs Pyro and NumPyro. Currently, efficient implementations of directional distributions are generally scarce, with the univariate von Mises distribution as the notable exception. The distribution is central to making Thygesen et al., 2023 a statistically sound model of unfolded protein conformations by accounting for the correlation between torsion in the backbone.

Conclusions and perspectives for further research

We introduce a new algorithm called EwS based on a new connection between the inference of Stein mixtures and the Rényi variational bound. We demonstrate that EwS provides better gradient approximations than alternative algorithms, which results in better performance on real-world problems. EwS is integrated as a black box library in the NumPyro PPL. EinStein and NumPyro are distributed freely with an Apache License 2.0 license.

In future work, we intend to integrate EwS with doubly reparameterized gradient estimators (Tucker et al., 2018). Tucker et al., 2018 demonstrate that these can eliminate the degradation of the MC estimators of IWAE. Due to its relation to the update rule in EwS, we hypothesize that we would see similar effects for other α values. We plan to extend EinStein with Amortized SVGD (Feng, Wang, and Liu, 2017), Stein points (Chen et al., 2018) and Stein point MCMC (Chen et al., 2019), POVI (Feng, Wang, and Liu, 2017), ParVI (Liu et al., 2019), Equivariant SVGD (Jaini, Holdijk, and Welling, 2021) in subsequent work.

There are several other aspects of the Stein mixture that warrant investigation. We request funding to analyze Stein mixtures' finite time convergence properties in an open DFF project two grant proposal. The analysis will rely heavily on investigating properties of the kernel in the kernelized stein discrepancy. By understanding the kernels, we will develop automated methods for selecting kernels by program analysis of the stochastic program describing the model. We believe we can manipulate inference diagnostics from exact to VI methods as an interpolation between exact and approximate inference.

We presented efficient implementations of the sine distribution and the sine skewing procedure for the 2-torus in the PPLs Pyro and NumPyro. Currently, efficient implementations of directional distributions are generally scarce, with the univariate von Mises distribution as the notable exception. Therefore, there is considerable untapped potential in making these methodologies more generally available to modern software running on parallel hardware. Another promising research direction is tailoring inference algorithms to work on non-Euclidean manifolds. In particular, implementing the so-called reparameterization trick

Kingma, Salimans, and Welling, 2015 for toroidal distributions would make these distributions available as directional priors in variational autoencoders Xu and Durrett, 2018.

The sine distribution accounts for the two major backbone degrees of freedom (ϕ and ψ). To completely account for the backbone, we would need a distribution over the 3-torus (to include the cis-trans isomorphism often called ω). In future work, we look to develop a computationally efficient 3-torus distribution. To fully model amino acids, we need to account for the conformation of the side chain. The long-term goal of this line of research will be n-torus distributions with algebraic normalization constants. Note that distribution with this property for the 3-torus is ongoing research, poised to be presented this year. Directional toroidal distributions represent a considerable untapped potential for statistical chemistry modeling, but they require advances in modern software and hardware paradigms to be realized. The methods are central to realizing distributions over protein configurations, the new frontier of structural bioinformatics (Lane, 2023). They further impact astrophysics with applications for modeling orbits (Zouboulоglou, García-Portugués, and Marron, 2022) and solar flare events (Hallin, Liu, and Verdebout, 2022).

To overcome these challenges with deep models for protein structure forecasting, we propose formulating the forecaster as a DPP using a rotation invariant heteroscedastic scoring function to learn an implicit solvent all-atoms energy function. We outline the approach's advantages in Chapter 5. The research is going. From the prototype (under development), we can investigate introducing deep learning in three key aspects. First, we can parameterize the functional form of the energy terms. Second, we can parameterize the Langevin dynamics but parameterize the thermostat, timescale, and stochastic contributions. Finally, we can introduce finer-grained forcefields (like Wang et al., 2004) to model more physiochemical properties explicitly.

Bibliography

[1] Nathanael L Ackerman, Cameron E Freer, and Daniel M Roy. "Noncomputable conditional distributions". In: *2011 IEEE 26th Annual Symposium on Logic in Computer Science*. IEEE. 2011, pp. 107–116.

[2] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* (2016), arXiv–1605.

[3] Mohammed AlQuraishi. "End-to-end differentiable learning of protein structure". In: *Cell systems* 8.4 (2019), pp. 292–301.

[4] Mohammed AlQuraishi. "Machine Learning in Protein Structure Prediction". In: *Current Opinion in Chemical Biology* 65 (2021), pp. 1–8.

[5] Shun-ichi Amari. "Differential-Geometrical Methods in Statistics". In: *Springer Science & Business Media* 28 (2012).

[6] Jose Ameijeiras-Alonso and Christophe Ley. "Sine-Skewed Toroidal Distributions and Their Application in Protein Bioinformatics". In: *Biostatistics* (2021).

[7] Andreas Anastasiou, Alessandro Barp, François-Xavier Briol, Bruno Ebner, Robert E Gaunt, Fatemeh Ghaderinezhad, Jackson Gorham, Arthur Gretton, Christophe Ley, Qiang Liu, et al. "Stein's Method Meets Computational Statistics: A Review of Some Recent Developments". In: *arXiv preprint arXiv:2105.03481* (2021).

[8] Christian B Anfinsen. "Principles that govern the folding of protein chains". In: *Science* 181.4096 (1973), pp. 223–230.

[9] Jimmy Ba, Murat A Erdogdu, Marzyeh Ghassemi, Shengyang Sun, Taiji Suzuki, Denny Wu, and Tianzong Zhang. "Understanding the variance collapse of svgd in high dimensions". In: *International Conference on Learning Representations*. 2021.

[10] Minkyung Baek, Frank DiMaio, Ivan Anishchenko, Justas Dauparas, Sergey Ovchinnikov, Gyu Rie Lee, Jue Wang, Qian Cong, Lisa N Kinch, R Dustin Schaeffer, et al. "Accurate prediction of protein structures and interactions using a three-track neural network". In: *Science* 373.6557 (2021), pp. 871–876.

[11] Javad Behboodian. "On the modes of a mixture of two normal distributions". In: *Technometrics* 12.1 (1970), pp. 131–139.

[12] Alain Berlinet and Christine Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. 2011.

[13] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. "The Million Song Dataset". In: *International Conference on Music Information Retrieval* (2011).

[14] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. "Pyro: Deep Universal Probabilistic Programming". In: *J. Mach. Learn. Res.* 20 (2019), 28:1–28:6. URL: http://jmlr.org/papers/v20/18-403.html.

[15] Christopher Bishop, Neil Lawrence, Tommi Jaakkola, and Michael Jordan. "Approximating Posterior Distributions in Belief Networks Using Mixtures". In: *Advances in Neural Information Processing Systems* 10 (1997).

[16] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

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

[18] Paul Blomstedt, Diego Mesquita, Jarno Lintusaari, Tuomas Sivula, Jukka Corander, and Samuel Kaski. "Meta-analysis of Bayesian analyses". In: *arXiv preprint arXiv:1904.04484* (2019).

[19] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription". In: *arXiv preprint:1206.6392* (2012).

[20] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. "JAX: Composable Transformations of Python+NumPy Programs". Version 0.2.5. In: *http://github.com/google/jax* (2018).

[21] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

[22] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance Weighted Autoencoders". In: *arXiv preprint:1509.00519* (2015).

[23] Christopher Bystroff, Vesteinn Thorsson, and David Baker. "HMMSTR: a Hidden Markov Model for Local Sequence-Structure Correlations in Proteins". In: *Journal of molecular biology* 301.1 (2000), pp. 173–190.

[24] Wilson Ye Chen, Alessandro Barp, François-Xavier Briol, Jackson Gorham, Mark Girolami, Lester Mackey, and Chris Oates. "Stein point markov chain monte carlo". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1011–1021.

[25] Wilson Ye Chen, Lester Mackey, Jackson Gorham, François-Xavier Briol, and Chris Oates. "Stein points". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 844–853.

[26] Siddhartha Chib and Edward Greenberg. "Understanding the metropolis-hastings algorithm". In: *The american statistician* 49.4 (1995), pp. 327–335.

[27] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[28] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. "Gen: A General-purpose Probabilistic Programming System with Programmable Inference". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. ACM, 2019, pp. 221–236. ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314642.

[29] Gianluca Detommaso, Tiangang Cui, Youssef Marzouk, Alessio Spantini, and Robert Scheichl. "A Stein variational Newton method". In: *Advances in Neural Information Processing Systems* 31 (2018).

[30] Luc Devroye. "Nonuniform random variate generation". In: *Handbooks in operations research and management science* 13 (2006), pp. 83–121.

[31] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. "Tensorflow Distributions". In: *arXiv preprint:1711.10604* (2017).

[32] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. "Hybrid monte carlo". In: *Physics letters B* 195.2 (1987), pp. 216–222.

[33] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.7 (2011).

[34] Giorgio Favrin, Anders Irbäck, Björn Samuelsson, and Stefan Wallin. "Two-state folding over a weak free-energy barrier". In: *Biophysical journal* 85.3 (2003), pp. 1457–1465.

[35] Yihao Feng, Dilin Wang, and Qiang Liu. "Learning to draw samples with amortized stein variational gradient descent". In: *arXiv preprint arXiv:1707.06626* (2017).

[36] Ronald Aylmer Fisher. *Statistical methods for research workers*. Springer, 1992.

[37] Roy Frostig, Matthew Johnson, and Chris Leary. "Compiling machine learning programs via high-level tracing". In: *SysML 2018*. 2018. URL: http://www.sysml.cc/doc/2018/146.pdf.

[38] Zhe Gan, Chunyuan Li, Ricardo Henao, David E Carlson, and Lawrence Carin. "Deep Temporal Sigmoid Belief Networks for Sequence Modeling". In: *Advances in Neural Information Processing Systems* 28 (2015). Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett.

[39] Hong Ge, Kai Xu, and Zoubin Ghahramani. "Turing: A Language for Flexible Probabilistic Inference". In: *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*. 2018, pp. 1682–1690. URL: http://proceedings.mlr.press/v84/ge18b.html.

[40] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

[41] Andrew Gelman, Donald B Rubin, et al. "Inference from Iterative Simulation using Multiple Sequences". In: *Statistical science* 7.4 (1992), pp. 457–472.

[42] Samuel Gershman, Matt Hoffman, and David Blei. "Nonparametric Variational Inference". In: *arXiv preprint: 1206.4665* (2012).

[43] Daniel T Gillespie. "The chemical Langevin equation". In: *The Journal of Chemical Physics* 113.1 (2000), pp. 297–306.

[44] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[45] Jackson Gorham and Lester W. Mackey. "Measuring Sample Quality with Kernels". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1292–1301. URL: http://proceedings.mlr.press/v70/gorham17a.html.

[46] Peter J Green. "Reversible jump Markov chain Monte Carlo computation and Bayesian model determination". In: *Biometrika* 82.4 (1995), pp. 711–732.

[47] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. "A Kernel Two-Sample Test". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 723–773.

[48] Marc Hallin, Hang Liu, and Thomas Verdebout. "Nonparametric Measure-transportation-based Methods for Directional Data". In: *arXiv preprint arXiv:2212.10345* (2022).

[49] Thomas Hamelryck, Kanti V Mardia, and Jesper Ferkinghoff-Borg. *Bayesian Methods in Structural Bioinformatics*. Springer, 2012.

[50] Jun Han and Qiang Liu. "Stein variational gradient descent without gradient". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1900–1908.

[51] David Harrison Jr and Daniel L Rubinfeld. "Hedonic Housing Prices and the Demand for Clean Air". In: *Journal of Environmental Economics and Management* 5.1 (1978), pp. 81–102.

[52] Darrall Henderson, Sheldon Jacobson, and Alan Johnson. "The Theory and Practice of Simulated Annealing". In: Apr. 2006, pp. 287–319. DOI: 10.1007/0-306-48056-5_10.

[53] Jose Hernandez-Lobato, Yingzhen Li, Mark Rowland, Thang Bui, Daniel Hernández-Lobato, and Richard Turner. "Black-Box Alpha Divergence Minimization". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1511–1520.

[54] José Miguel Hernández-Lobato and Ryan Adams. "Probabilistic Back-propagation for Scalable Learning of Bayesian Neural Networks". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1861–1869.

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

[56] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. "Stochastic variational inference". In: *Journal of Machine Learning Research* (2013).

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

[58] John Ingraham, Adam J Riesselman, Chris Sander, and Debora S Marks. "Learning Protein Structure with a Differentiable Simulator." In: *ICLR* (2019).

[59] Anders Irbäck, Simon Mitternacht, and Sandipan Mohanty. "An effective all-atom potential for proteins". In: *PMC biophysics* 2.1 (2009), pp. 1–24.

[60] Anders Irbäck and Sandipan Mohanty. "PROFASI: a Monte Carlo simulation package for protein folding and aggregation". In: *Journal of computational chemistry* 27.13 (2006), pp. 1548–1555.

[61] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. "What Are Bayesian Neural Network Posteriors Really Like?" In: *International Conference on Machine Learning* (2021), pp. 4629–4640.

[62] Tommi S Jaakkola and Michael I Jordan. "Improving the Mean Field Approximation via the Use of Mixture Distributions". In: *Learning in Graphical Models*. Springer, 1998, pp. 163–173.

[63] Priyank Jaini, Lars Holdijk, and Max Welling. "Learning equivariant energy based models with equivariant stein variational gradient descent". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 16727–16737.

[64] Martin Jankowiak and Theofanis Karaletsos. "Pathwise Derivatives for Multivariate Distributions". In: *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 333–342. URL: http://proceedings.mlr.press/v89/jankowiak19a.html.

[65] Tony Jebara, Risi Kondor, and Andrew Howard. "Probability product kernels". In: *The Journal of Machine Learning Research* 5 (2004), pp. 819–844.

[66] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. "An introduction to variational methods for graphical models". In: *Machine learning* 37.2 (1999), pp. 183–233.

[67] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[68] John T Kent, Asaad M Ganeiber, and Kanti V Mardia. "A New Unified Approach for the Simulation of a Wide Class of Directional Distributions". In: *Journal of Computational and Graphical Statistics* 27.2 (2018), pp. 291–301.

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

[70] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), 307–392. ISSN: 1935-8245. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[71] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[72] Durk P Kingma, Tim Salimans, and Max Welling. "Variational Dropout and the Local Reparameterization Trick". In: *Advances in Neural Information Processing Systems* 28 (2015).

[73] Achim Klenke. *Probability theory: a comprehensive course*. Springer Science & Business Media, 2013.

[74] Andrey Kolmogoroff. "Grundbegriffe der wahrscheinlichkeitsrechnung". In: (1933).

[75] Rahul G Krishnan, Uri Shalit, and David Sontag. "Structured inference networks for nonlinear state space models". In: *arXiv preprint arXiv:1609.09869* (2016).

[76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[77] Andriy Kryshtafovych, Torsten Schwede, Maya Topf, Krzysztof Fidelis, and John Moult. "Critical assessment of methods of protein structure prediction (CASP)—Round XIII". In: *Proteins: Structure, Function, and Bioinformatics* 87.12 (2019), pp. 1011–1020.

[78] Brenden M Lake, Russ R Salakhutdinov, and Josh Tenenbaum. "One-Shot Learning by Inverting a Compositional Causal Process". In: *Advances in Neural Information Processing Systems* 26 (2013).

[79] Thomas J Lane. "Protein structure prediction has reached the single-structure frontier". In: *Nature Methods* (2023), pp. 1–4.

[80] Tuan Anh Le, Adam R Kosiorek, N Siddharth, Yee Whye Teh, and Frank Wood. "Revisiting Reweighted Wake-Sleep for Models With Stochastic Control Flow". In: *Uncertainty in Artificial Intelligence* (2020), pp. 1039–1049.

[81] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[82] Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. "Stochastic Expectation Propagation". In: *Advances in Neural Information Processing Systems* 28 (2015).

[83] Yingzhen Li and Richard E Turner. "Rényi Divergence Variational Inference". In: *arXiv preprint:1602.02311* (2016).

[84] Dorothee Liebschner, Pavel V Afonine, Matthew L Baker, Gábor Bunkóczi, Vincent B Chen, Tristan I Croll, Bradley Hintze, L-W Hung, Swati Jain, Airlie J McCoy, et al. "Macromolecular Structure Determination using X-rays, Neutrons and Electrons: Recent Developments in Phenix". In: *Acta Crystallographica Section D: Structural Biology* 75.10 (2019), pp. 861–877.

[85] Chang Liu, Jingwei Zhuo, Pengyu Cheng, Ruiyi Zhang, and Jun Zhu. "Understanding and accelerating particle-based variational inference". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4082–4092.

[86] Qiang Liu. "Stein Variational Gradient Descent as Gradient Flow". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, pp. 3115–3123. URL: http://papers.nips.cc/paper/6904-stein-variational-gradient-descent-as-gradient-flow.

[87] Qiang Liu, Jason Lee, and Michael Jordan. "A Kernelized Stein Discrepancy for Goodness-Of-Fit Tests". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 276–284.

[88] Qiang Liu and Dilin Wang. "Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett. 2016, pp. 2370–2378. URL: http://papers.nips.cc/paper/6338-stein-variational-gradient-descent-a-general-purpose-bayesian-inference-algorithm.

[89] Qiang Liu and Dilin Wang. "Stein variational gradient descent: A general purpose bayesian inference algorithm". In: *Advances in neural information processing systems* 29 (2016).

[90] Qiang Liu and Dilin Wang. "Stein Variational Gradient Descent as Moment Matching". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 8868–8877. URL: http://papers.nips.cc/paper/8101-stein-variational-gradient-descent-as-moment-matching.

[91] Qiang Liu and Dilin Wang. "Stein Variational Gradient Descent as Moment Matching". In: *Advances in Neural Information Processing Systems* 31 (2018).

[92] Kanti V Mardia. "Statistical Approaches to Three Key Challenges in Protein Structural Bioinformatics". In: *Journal of the Royal Statistical Society: SERIES C: Applied Statistics* (2013), pp. 487–514.

[93] Kanti V Mardia. "Statistics of Directional Data". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 37.3 (1975), pp. 349–371.

[94] Kanti V Mardia, Charles C Taylor, and Ganesh K Subramaniam. "Protein Bioinformatics and Mixtures of Bivariate von Mises Distributions for Angular Data". In: *Biometrics* 63.2 (2007), pp. 505–512.

[95] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. "An Introduction to Probabilistic Programming". In: *CoRR* abs/1809.10756 (2018). arXiv: 1809.10756. URL: http://arxiv.org/abs/1809.10756.

[96] Andrew C Miller, Nicholas J Foti, and Ryan P Adams. "Variational Boosting: Iteratively Refining Posterior Approximations". In: *International Conference on Machine Learning* (2017), pp. 2420–2429.

[97] Lys Sanz Moreta, Ahmad Salim Al-Sibahi, Douglas Theobald, William Bullock, Basile Nicolas Rommes, Andreas Manoukian, and Thomas Hamelryck. "A probabilistic programming approach to protein structure superposition". In: *2019 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. IEEE. 2019, pp. 1–5.

[98] John Moult, Jan T Pedersen, Richard Judson, and Krzysztof Fidelis. "A Large-Scale Experiment to Assess Protein Structure Prediction Methods". In: *Proteins: Structure, Function, and Bioinformatics* 23.3 (1995), pp. ii–iv.

[99] Eric Nalisnick and Padhraic Smyth. "Variational inference with Stein mixtures". In: *Advances in Approximate Bayesian Inference, NIPS 2017 Workshop*. 2017.

[100] Radford M Neal. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, 2012.

[101] Radford M Neal et al. "MCMC using Hamiltonian dynamics". In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.

[102] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 427–436.

[103] Fritz Obermeyer, Eli Bingham, Martin Jankowiak, Du Phan, and Jonathan P. Chen. "Functional Tensors for Probabilistic Programming". In: *arXiv preprint arXiv:1910.10775* (2020).

[104] Ola Rønning, Christophe Ley, Ahmad Salim Al-Sibahi and Thomas Hamelryck. "ELBO-ing Stein Mixtures". In: *Unpublished* (2023).

[105] James M Ortega and William G Poole. *An introduction to numerical methods for differential equations*. Pitman Publishing, 1981.

[106] Melissa E O'neill. "PCG: A family of simple fast space-efficient statistically good algorithms for random number generation". In: *ACM Transactions on Mathematical Software* (2014).

[107] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[108] Xavier Perez-Sala, Laura Igual, Sergio Escalera, and Cecilio Angulo. "Uniform sampling of rotations for discrete and continuous learning of 2D shape models". In: *Robotic vision: Technologies for machine learning and vision applications*. IGI Global, 2013, pp. 23–42.

[109] Du Phan, Neeraj Pradhan, and Martin Jankowiak. "Composable Effects for Flexible and Accelerated Probabilistic Programming in NumPyro". In: *arXiv preprint arXiv:1912.11554* (2019).

[110] Du Phan, Neeraj Pradhan, and Martin Jankowiak. "Composable Effects for Flexible and Accelerated Probabilistic Programming in NumPyro". In: *Program Transformations for Machine Learning, NeurIPS 2019 Workshop*. 2019.

[111] Tom Rainforth, Adam Kosiorek, Tuan Anh Le, Chris Maddison, Maximilian Igl, Frank Wood, and Yee Whye Teh. "Tighter Variational Bounds Are Not Necessarily Better". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4277–4285.

[112] Gopalasamudram N Ramachandran, Chandrasekharan Ramakrishnan, and Viswanathan Sasisekharan. "Stereochemistry of Polypeptide Chain Configurations". In: *Journal of Molecular Biology* 7 (1963), pp. 95–99.

[113] PS Rana. "Physicochemical Properties of Protein Tertiary Structure Data Set". In: *UCI Machine Learning Repository* (2013).

[114] Rajesh Ranganath, Sean Gerrish, and David Blei. "Black box variational inference". In: *Artificial intelligence and statistics*. PMLR. 2014, pp. 814–822.

[115] Rajesh Ranganath, Dustin Tran, Jaan Altosaar, and David Blei. "Operator Variational Inference". In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 496–504.

[116] Rajesh Ranganath, Dustin Tran, and David Blei. "Hierarchical Variational Models". In: *International Conference on Machine Learning* (2016), pp. 324–333.

[117] Alfréd Rényi. "On Measures of Entropy and Information". In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. Vol. 4. University of California Press. 1961, pp. 547–562.

[118] Daniel Ritchie, Paul Horsfall, and Noah D Goodman. "Deep amortized inference for probabilistic programs". In: *arXiv preprint arXiv:1610.05735* (2016).

[119] Louis-Paul Rivest. "A Distribution for Dependent Unit Vectors". In: *Communications in Statistics-Theory and Methods* 17.2 (1988), pp. 461–483.

[120] Carol A Rohl, Charlie EM Strauss, Kira MS Misura, and David Baker. "Protein Structure Prediction using Rosetta". In: *Methods in Enzymology* 383 (2004), pp. 66–93.

[121] Ola Rønning, Ahmad Salim Al-Sibahi, Christophe Ley, and Thomas Hamelryck. "EinSteinVI: General and Integrated Stein Variational Inference". In: *Unplublished* (2021).

[122] Ola Rønning, Christophe Ley, Kanti V. Mardia, and Thomas Hamelryck. "Time-efficient Bayesian Inference for a (Skewed) Von Mises Distribution on the Torus in a Deep Probabilistic Programming Language". In: *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. 2021, pp. 1–8. DOI: 10.1109/MFI52462.2021.9591184.

[123] Amit Sabne. *XLA : Compiling Machine Learning for Peak Performance*. 2020.

[124] Ruslan Salakhutdinov and Iain Murray. "On the Quantitative Analysis of Deep Belief Networks". In: *Proceedings of the 25th International Conference on Machine Learning*. ACM. 2008, pp. 872–879.

[125] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. "Probabilistic programming in Python using PyMC3". In: *PeerJ Comput. Sci.* 2 (2016), e55. DOI: 10.7717/peerj-cs.55. URL: https://doi.org/10.7717/peerj-cs.55.

[126] Robert Schenck, Ola Rønning, Troels Henriksen, and Cosmin E. Oancea. "AD for an Array Language with Nested Parallelism". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '22. Dallas, Texas: IEEE Press, 2022. ISBN: 9784665454445.

[127] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. "Improved Protein Structure Prediction using Potentials from Deep Learning". In: *Nature* 577.7792 (2020), pp. 706–710.

[128] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. "Protein structure prediction using multiple deep neural networks in the 13th Critical Assessment of Protein Structure Prediction (CASP13)". In: *Proteins: Structure, Function, and Bioinformatics* 87.12 (2019), pp. 1141–1148.

[129] Jayaram Sethuraman. "A Constructive Definition of Dirichlet Priors". In: *Statistica sinica* (1994), pp. 639–650.

[130] Joan-Emma Shea, Robert B Best, and Jeetain Mittal. "Physics-based Computational and Theoretical Approaches to Intrinsically Disordered Proteins". In: *Current Opinion in Structural Biology* 67 (2021), pp. 219–225.

[131] Rui Shu, Hung H Bui, Shengjia Zhao, Mykel J Kochenderfer, and Stefano Ermon. "Amortized Inference Regularization". In: *Advances in Neural Information Processing Systems* 31 (2018).

[132] Kim T Simons, Charles Kooperberg, Enoch Huang, and David Baker. "Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions". In: *Journal of molecular biology* 268.1 (1997), pp. 209–225.

[133] Harshinder Singh, Vladimir Hnizdo, and Eugene Demchuk. "Probabilistic model for two dependent circular variables". In: *Biometrika* 89.3 (2002), pp. 719–723.

[134] Zhoutong Sun, Qian Liu, Ge Qu, Yan Feng, and Manfred T Reetz. "Utility of B-factors in protein science: interpreting rigidity, flexibility, and internal motion and engineering thermostability". In: *Chemical reviews* 119.3 (2019), pp. 1626–1665.

[135] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).

[136] Christian B. Thygesen, Ahmad S. Al-Sibahi, Christian S. Steenmanns, Lys S. Moreta, Anders B. Sørensen, and Thomas Hamelryck. "Efficient Generative Modelling of Protein Structure Fragments using a Deep Markov Model". In: *International Conference on Machine Learning*. PMLR. 2021.

[137] Christian B. Thygesen, Ola Rønning, Christian Skjødt Steenmans, Anders Bundgård Sørensen, Kanti V. Mardia, John T. Kent, and Thomas Hamelryck. "A Multiscale Deep Generative Model of Protein Structure Using a Directional and a Procrustes Likelihood". In: *Unpublished* (2023).

[138] Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. "Edward: A Library for Probabilistic Modeling, Inference, and Criticism". In: *arXiv preprint:1610.09787* (2016).

[139] Constantino Tsallis. "Possible Generalization of Boltzmann-Gibbs Statistics". In: *Journal of Statistical Physics* 52.1 (1988), pp. 479–487.

[140] Athanasios Tsanas and Angeliki Xifara. "Accurate Quantitative Estimation of Energy Performance of Residential Buildings Using Statistical Machine Learning Tools". In: *Energy and Buildings* 49 (2012), pp. 560–567.

[141] George Tucker, Dieterich Lawson, Shixiang Gu, and Chris J Maddison. "Doubly Reparameterized Gradient Estimators for Monte Carlo Objectives". In: *arXiv preprint:1810.04152* (2018).

[142] Pınar Tüfekci. "Prediction of Full Load Electrical Power Output of a Base Load Operated Combined Cycle Power Plant Using Machine Learning Methods". In: *International Journal of Electrical Power & Energy Systems* 60 (2014), pp. 126–140.

[143] Kathryn Tunyasuvunakool, Jonas Adler, Zachary Wu, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, et al. "Highly accurate protein structure prediction for the human proteome". In: *Nature* 596.7873 (2021), pp. 590–596.

[144] David E Tyler. "Statistical analysis for the angular central Gaussian distribution on the sphere". In: *Biometrika* 74.3 (1987), pp. 579–589.

[145] Tim Van Erven and Peter Harremos. "Rényi divergence and Kullback-Leibler Divergence". In: *IEEE Transactions on Information Theory* 60.7 (2014), pp. 3797–3820.

[146] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[147] Dilin Wang, Yihao Feng, and Qiang Liu. "Learning to Sample Using Stein Discrepancy". In: *NeurIPS Workshop on Bayesian Deep Learning* 192 (2016).

[148] Dilin Wang and Qiang Liu. "Nonlinear Stein Variational Gradient Descent for Learning Diversified Mixture Models". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6576–6585.

[149] Dilin Wang and Qiang Liu. "Nonlinear Stein Variational Gradient Descent for Learning Diversified Mixture Models". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, 2019, pp. 6576–6585. URL: http://proceedings.mlr.press/v97/wang19h.html.

[150] Dilin Wang, Ziyang Tang, Chandrajit Bajaj, and Qiang Liu. "Stein Variational Gradient Descent With Matrix-Valued Kernels". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. 2019, pp. 7834–7844. URL: http://papers.nips.cc/paper/8998-stein-variational-gradient-descent-with-matrix-valued-kernels.

[151] Dilin Wang, Ziyang Tang, Chandrajit Bajaj, and Qiang Liu. "Stein variational gradient descent with matrix-valued kernels". In: *Advances in neural information processing systems* 32 (2019), p. 7834.

[152] Dilin Wang, Zhe Zeng, and Qiang Liu. "Stein Variational Message Passing For Continuous Graphical Models". In: *International Conference on Machine Learning* (2018), pp. 5219–5227.

[153] Dilin Wang, Zhe Zeng, and Qiang Liu. "Stein Variational Message Passing for Continuous Graphical Models". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 5206–5214. URL: http://proceedings.mlr.press/v80/wang18l.html.

[154] Junmei Wang, Romain M Wolf, James W Caldwell, Peter A Kollman, and David A Case. "Development and testing of a general amber force field". In: *Journal of computational chemistry* 25.9 (2004), pp. 1157–1174.

[155] Andrew Gordon Wilson. "The case for Bayesian deep learning". In: *arXiv preprint arXiv:2001.10995* (2020).

[156] Andrew Gordon Wilson and Pavel Izmailov. "Bayesian Deep Learning and a Probabilistic Perspective of Generalization". In: *arXiv preprint:2002.08791* (2020).

[157] Carter J Wilson, Wing-Yiu Choy, and Mikko Karttunen. "AlphaFold2: a role for disordered protein/region prediction?" In: *International Journal of Molecular Sciences* 23.9 (2022), p. 4591.

[158] Hao Wu, Andreas Mardt, Luca Pasquali, and Frank Noe. "Deep generative markov state models". In: *Advances in Neural Information Processing Systems* 31 (2018).

[159] Ruidong Wu, Fan Ding, Rui Wang, Rui Shen, Xiwen Zhang, Shitong Luo, Chenpeng Su, Zuofan Wu, Qi Xie, Bonnie Berger, et al. "High-resolution de novo structure prediction from primary sequence". In: *BioRxiv* (2022), pp. 2022–07.

[160] Jiacheng Xu and Greg Durrett. "Spherical Latent Spaces for Stable Variational Autoencoders". In: *arXiv preprint arXiv:1808.10805* (2018).

[161] I-C Yeh. "Modeling of Strength of High-Performance Concrete Using Artificial Neural Networks". In: *Cement and Concrete research* 28.12 (1998), pp. 1797–1808.

[162] Jingwei Zhuo, Chang Liu, Jiaxin Shi, Jun Zhu, Ning Chen, and Bo Zhang. "Message passing Stein variational gradient descent". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 6018–6027.

[163] Pavlos Zoubouloglou, Eduardo García-Portugués, and JS Marron. "Scaled torus principal component analysis". In: *Journal of Computational and Graphical Statistics* (2022), pp. 1–12.