Robust and Efficient Inference and Learning Algorithms for Generative Models

Kai Xu

Doctor of Philosophy

Institute for Adaptive and Neural Computation

School of Informatics

The University of Edinburgh

2021

Abstract

Generative modelling is a popular paradigm in machine learning due to its natural ability to describe uncertainty in data and models and for its applications including data compression (Ho et al., 2020), missing data imputation (Valera et al., 2018), synthetic data generation (Lin et al., 2020), representation learning (Kingma and Welling, 2014), robust classification (Li et al., 2019b), and more. For generative models, the task of finding the distribution of unobserved variables conditioned on observed ones is referred to as *inference*. Finding the optimal model that makes the model distribution close to the data distribution according to some discrepancy measures is called *learning*. In practice, existing learning and inference methods can fall short on *robustness* and *efficiency*. A method that is more robust to its hyper-parameters or different types of data can be more easily adapted to various real-world applications. How efficient a method is in regard to the size and the dimensionality of data determines at what scale the method can be applied. This thesis presents four pieces of my original work that improves these properties in generative models.

First, I introduce two novel *Bayesian inference* algorithms. One is called coupled multinomial Hamiltonian Monte Carlo (Xu et al., 2021a); it builds on Heng and Jacob (2019), which is a recent work in unbiased *Markov chain Monte Carlo* (MCMC) (Jacob et al., 2019b) and has been found to sensitive to hyper-parameters and less efficient compared to normal, biased MCMC. These issues are solved by establishing couplings to the widely-used multinomial Hamiltonian Monte Carlo, leading to a statistically more efficient and robust method. The other method is called roulette-based variational expectation (RAVE; Xu et al., 2019) that applies *amortised inference* to a model family called *Bayesian non-parametric* models, in which the number of parameters are allowed to grow unbounded as the data gets more complex. Unlike previous sampling-based methods that are slow or *variational inference* methods that rely on truncation, RAVE combines the advantages of both to achieve flexible inference that is also computational efficient. Second, I introduce two novel learning methods. One is called generative ratio-matching (Srivastava et al., 2019) which is a learning algorithm that makes deep generative models based on kernel methods applicable to high-dimensional data. The key innovation of this method is learning a projection of the data to a lower-dimensional space in which the density ratio is preserved such that learning can be done in the lower-dimensional space where kernel methods are effective. The other method is called *Bayesian symbolic physics* that combines Bayesian inference and *symbolic regression* in the context of naïve physics—the study of how humans understand and learn physics. Unlike classic generative models for which the structure of the generative process is predefined or deep generative models where the process is represented by data-hungry neural networks, Bayesian-symbolic generative processes are defined by functions over a hypothesis space specified by a context-free grammar. This formulation allows these models to incorporate domain knowledge in learning, which gives highly-improved sample efficiency. For all four pieces of work, I provide theoretical analyses and/or empirical results to validate that the algorithmic advances lead to improvements in robustness and efficiency for generative models.

Lastly, I summarise my contributions to free and open-source software on generative modelling. This includes a set of Julia packages that I contributed and are currently used by the Turing probabilistic programming language (Ge et al., 2018). These packages, which are highly reusable components for building probabilistic programming languages, together form a probabilistic programming ecosystem in Julia. An important package that is primarily developed by me is called ADVANCEDHMC.JL (Xu et al., 2020), which provides robust and efficient implementations of HMC methods and has been adopted as the backend of Turing. Importantly, the design of this package allows an intuitive abstraction to construct HMC samplers similarly to how they are mathematically defined. The promise of these open-source packages is to make generative modelling techniques more accessible to domain experts from various backgrounds and to make relevant research more reproducible to help advance the field.

Acknowledgements

First, I would like to thank my supervisor Charles Sutton for his all-sided support during Ph.D study at Edinburgh, and his indulgence on my wide research interests, fortunately most of which his coincides with. The guidance from Charles of how to conduct research as well as the technical support from him really transformed me from a student to a researcher.[1] I also want to thank my second supervisor, Iain Murray, for the support and useful discussions, from which I always learned something new.

Many thanks to my peers in Edinburgh, especially the *Cat Squad* members (Akash Srivastava, Cole Hurwitz), my office mates (Bowen Li, Yumo Xu, Liu Yang) and co-workers in Charles's Uncertain People (Lazar Valkov, Simao Eduardo). I am so glad I have you during the life and research in my PhD—it was the every meal I enjoyed with you and every research argument I had with you that enriched my PhD life.

I would like to acknowledge my collaborators of publications during my Ph.D who have not been mentioned earlier. It was a great learning experience working with all of you: Hong Ge, Zoubin Ghahramani, Tor E. Fjelde, Michael U. Gutmann, Dan Gutfreund, Tomer Ullman, Joshua B. Tenenbaum, Felix A. Sosa, Mohamed Tarek, Martin Trapp, Rajkarn Singh, Marco Fiore, Mahesh K. Marina, Hakan Bilen, Dae Hoon Park and Chang Yi. Special thanks to Zoubin and Hong who brings me to the research field of probabilistic machine learning.

I also would like to thank people from the Turing team and close developers from the Julia community: Cameron Pfiffer, David Widmann, Elizaveta Semenova, Qinliang Zhuo, Mohamed Tarek, Seth Axen, Will Tebbutt, Christopher Rackauckas and Andreas Noack. Without the efforts from everyone the Turing project will not go this far.

Finally, special thanks to my mother, Caiqin He, my father, Xuegen Xu, and my wife[2], Pinzhen Liao, for their care, encouragement and love for the nerdy Kai throughout these years.

[1]Also thanks to Charles' black Scottish cat, Daisy, who keeps our remote meetings lifeful and uncertain (in a good way).

[2]My wife and I married on 8th Dec 2022, which is exactly one week prior to my viva.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Kai Xu)*

Publications

This thesis contains some work that was previously published in the following venues.

Chapter 3 extends on[3]:

- Xu, K., Fjelde, T. E., Sutton, C., and Ge, H. (2021a). Couplings for multinomial Hamiltonian Monte Carlo. In *International Conference on Artificial Intelligence and Statistics*

Chapter 4 extends on:

- Xu, K., Srivastava, A., and Sutton, C. (2019). Variational Russian roulette for deep Bayesian nonparametrics. In *International Conference on Machine Learning*

Chapter 5 extends on[4]:

- Srivastava, A., Xu, K., Gutmann, M. U., and Sutton, C. (2019). Generative ratio matching networks. In *International Conference on Learning Representations*

Chapter 6 extends on:

- Xu, K., Srivastava, A., Gutfreund, D., Sosa, F. A., Tomer, U., Tenenbaumm, J. B., and Sutton, C. (2021b). A Bayesian-symbolic approach to reasoning and learning in intuitive physics. In *Advances in Neural Information Processing Systems*

Chapter 7 extends on:

- Ge, H., Xu, K., and Ghahramani, Z. (2018). Turing: A language for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics*
- Xu, K., Ge, H., Tebbutt, W., Tarek, M., Trapp, M., and Ghahramani, Z. (2020). AdvancedHMC.jl: A robust, modular and effcient implementation of advanced HMC algorithms. In *Symposium on Advances in Approximate Bayesian Inference*

[3]I (Kai Xu) and Tor E. Fjelde share the leading authorship of Xu et al. (2021a). I develop the method and conduct initial theoretical analysis as well as all the experiments, and Tor leads the final proofs and theoretical analysis that is presented in the published paper. The proof strategy presented in section 3.4.3 is my original work that was not included in the published version.

[4]Akash Srivastava and I (Kai Xu) share the leading authorship of Srivastava et al. (2019). Akash and I co-developed the prototype of the method, I experiment and study the method on synthetic data and Akash contributes all the experiments on large-scale data sets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Artificial intelligence, science and machine learning

Artificial intelligence (AI) is the long-standing mission of building machines that can behave or think like human beings. More specifically, the goal is to develop computer programs or algorithms that can solve tasks which were believed to only capable by humans, such as recognising animals from photos, understanding human speech, conducting conversations using natural languages with humans, etc. These tasks only have been successfully or partially solved by recent developments in AI, leading to the current ambition of developing **artificial general intelligence** (AGI) agents, which can understand, learn and perform *any* intellectual task just like humans do.

Science has also benefited from tools based on techniques from computer science, statistics and engineering. For example in natural science, statistical models are used by biologists to analyse genes—predicting traits or understanding diseases; such research usually requires massive computer simulation due to the large scale of genomics data (Larranaga et al., 2006). Another example is found in neuroscience where researchers analyse and interpret large-scale neural recordings using a variety of probabilistic models (Hurwitz et al., 2021). Similarly, in social science, quantitative approaches have been used to understand social interactions. Such analysis usually requires non-trivial computer engineering on large scale data to give reliable results (Peng et al., 2016).

Machine learning (ML) has become one of the most important methodologies for both AI and science due to the opportunity and challenge of **big data** that emerges from modern information technologies. The main distinction between ML and traditional computer sciences is the ability to learn from data. As such, engineering effort of programming machines what to do has been moved to teaching machines how to learn and provide them with learning signals, e.g. data. For example, pre-ML methods for hand-written digits recognition may require manually define heuristics of how each digit looks like in terms of templates while ML-based methods involves a data set of images of digits with the corresponding labels as well as a learning algorithm that teaches the program how to learn from such paired examples. After learning, the program is capable of predicting labels given images of digits.

Model-based ML is an ML paradigm in which models explicitly encodes the assumptions of the problem domain (Winn et al., 2020). One benefit of model-based ML is that more tailored solution to specific problem can be implemented through models; another benefit is that comparison between different hypotheses can be done through models encoded with different assumptions (Bishop, 2013). One type of model called **generative models** in the probabilistic machine learning (probabilistic ML) paradigm are especially interesting due to their probabilistic nature to deal with noise in data or uncertainty in models (Murphy, 2012). A wide range of applications have been built on generative models, including but not limited to data compression (Ho et al., 2020), missing data imputation (Valera et al., 2018), synthetic data generation (Lin et al., 2020), representation learning (Kingma and Welling, 2014), robust classification (Li et al., 2019b), etc.

The probabilistic ML framework that generative modelling takes is important because representing uncertainty is crucial. First, probability theory is *the* mathematical language for representing and manipulating uncertainty (Jaynes, 2003). The Cox axioms define some desirable properties of a formal way to represent beliefs or uncertainty, and indicate that degrees of belief, ranging from "impossible" to "absolutely certain", must follow the rules of probability theory (Cox, 1963; Jaynes, 2003). An example of how concise and useful this language is that, in the language of probability theory, the task of missing data imputation can be described as a conditional distribution of what to impute given what has been observed. Various problems in machine leaning such as noisy data, model learning and decision-making rely on careful representations of uncertainty (Ghahramani, 2015). The uncertainty in learning is rooted in the fact that observed data could be consistent with many models/hypotheses, which can mean the model parameter is uncertain, the model structure is uncertain or even the model/hypothesis itself is uncertain. Decision-making is a task that directly relies on uncertainty, which is grounded by the Dutch book theorem which states that an agent is willing to accept bets that will lose money unless the agent's belief is consistent with the rules of probabilities (De Finetti, 1937). When the decision-making problem is critical, such as disease diagnosis and policymaking, a careful and grounded treatment of uncertainty is needed, which cannot be well-handled by other machine learning paradigms.

In a high level, generative models are essentially probability distributions: Different models corresponds to different ways to represent probability distributions. The advances in generative modelling techniques can be generally categorised into three aspects: new representations, new inference algorithms and new learning algorithms. There are various representations for distributions. For example, a distribution can be represented by its *probability density function*, its *generative process*, etc. The task of finding the distribution of unobserved variables conditioned on observed ones is referred as *inference*. Finding the optimal model that makes the *model distribution*, i.e. the distribution that the model represents, close to the *data distribution*, i.e. the underlying distribution that generates the data, is called *learning*.

For successful application of generative models in practice, the corresponding inference and learning algorithms need to be robust and efficient. A method that is more robust to its hyper-parameters or different types of data can be more easily adapted to various applications. Robustness of the model determines how much human efforts are needed to make a successful application and is crucial for fully automated systems. How efficient a method is in regard to the size and dimensionality of data determines at what scale the method can be applied. Usually a large data set means more training signals and a lot of data like images, audio and videos live in high dimension, both of which requires high scalability for generative modelling. On the other hand, a method is more statistically efficient if it can perform well even a limit amount of data is provided.

In the rest of this chapter I will review necessary background to understand the main focus and the contributions of this thesis. A set of well-established, popular methods will be reviewed along with their advantages and limitations. In section 2.4, I will discuss how their limitations lead to practical concerns in robustness and efficiency. Finally, in section 1.2, I will sketch a set of solutions to these limitations solved by methods presented in this thesis as the outline of the thesis.

1.2 Thesis outline

The rest of this thesis first reviews some necessary background knowledge (chapter 2) and then presents four pieces of my original work that tackles the robustness and efficiency problems in the forementioned limitations in generative modelling.

Chapter 3 introduces coupled multinomial Hamiltonian Monte Carlo (CMHMC), an algorithm for robust and efficient unbiased Markov chain Monte Carlo (MCMC) estimation. Hamiltonian Monte Carlo (HMC) is a family of Bayesian inference algorithms that is widely used by practitioners for modelling in various science domains, such as biology, chemistry, pharmacology, cognitive science, etc., due to its superior statistical performance in differentiable models and its mature implementation in popular probabilistic programming languages, in which the inference methods are also known as inference engines. However, as a type of MCMC algorithm, HMC is only guaranteed to achieve unbiased inference in an asymptomatic manner (i.e. using an infinite amount of time) and is not easy to parallelise. Solving these two issues can make HMC more useful to Bayesian inference practitioners by providing robust and faster inference engines. A recent line of work (Jacob et al., 2019b; Heng and Jacob, 2019) tackles this problem by establishing unbiased MCMC estimation using couplings for HMC, showing the possibility of making HMC-based inference unbiased and parallelizable. In spite of this significant theoretical contribution, the proposed algorithm was found to be inferior to normal, biased HMC in terms of statistical efficiency, due to an unsatisfying trade-off between unbiasedness and variance. In my newly developed method CMHMC, I solve this problem by designing a coupled HMC kernel for multinomial HMC, a type of HMC variant that is more suitable for coupling, that encourages the chains to meet fast based on optimal transportation. Compared to the previously established coupled HMC algorithm in (Heng and Jacob, 2019), we find that CMHMC improves the computation cost thanks to the formulation of optimal transportation and improves the statistical efficiency and the robustness thanks to the role of multinomial HMC. Importantly, it achieves unbiasedness while maintaining statistical efficiency close to commonly-used HMC methods, indicating its potential to be used as the default inference engine in probabilistic modelling frameworks. All these improvements pave the way for wider use of unbiased MCMC estimation and hence more robust and efficient general-purpose Bayesian inference engines.

Chapter 4 introduces roulette-based amortised variational expectation (RAVE), an estimator used in flexible and efficient variational inference for (deep) Bayesian nonparametric models with stick-breaking priors. Bayesian nonparametric models are models in which the number of parameters is dependent on the complexity of data and could be potentially unbounded. For example, in an infinite latent feature model for a scene image, one would infer a different number of latent features, such as object identities and properties, based on the complexity of a given scene; it is not easy to set an upper bound for the number in practice as the scene could go as complex as it may need to be. While the nonparametric modelling approach is appealing, inference in such models is computationally challenging. For BNP models with stick-breaking priors, previous methods either use computationally expansive sampling-based approaches such as Gibbs sampling and slice sampling (Griffiths and Ghahramani, 2011; Teh et al., 2007) by faithfully treating the problem as nonparametric, or introduce an extra assumption on the maximum number of parameters that the model may take, leading to truncation-based variational inference methods (Chatzis, 2014; Singh et al., 2017). RAV E combines the benefits of both sampling-based method and variational based method by using a Russian roulette sampling within variational inference, leading to a new type of principled inference method that is fast while putting no additional assumption on the maximum number of parameters. Moreover, it is demonstrated that RAVE can be used together with amortised inference, a type of variational inference method that uses neural networks for accelerated computation. In fact, this leads to a type of variational autoencoders with a potentially infinitely large hidden size but can automatically infer the size during training. With both the advantages of sampling-based inference method—faithful nonparametric model complexity—and variational method—scalability due to the use of neural networks, RAVE paves the way for using truly nonparametric Bayesian methods in areas such as continual learning, scene understanding, etc.

Chapter 5 introduces generative ratio matching (GRAM), an algorithm that scales MMD-nets to high-dimensional data without introducing an unstable saddle-point optimisation. Deep generative models have become popular due to their capability of generating realistic natural images and are considered as an important approach to synthetic data generation that aims to solve privacy issue by replacing real data with statistical similar but private data generated by models. However, the most popular type of deep generative models, generative adversarial networks (GANs; Goodfellow et al., 2014), are known to be unstable and hard to train due to the existence of a saddle-point optimisation problem during training: it usually requires a careful hyper-parameter tuning (network architecture, learning rates, batch sizes, etc) to balance the capacity of the two involved components, a generator and a discriminator. Another recently proposed model called MMD-nets (Li et al., 2015; Dziugaite et al., 2015) that replaces the role of discriminator by a probability discrepancy measure called the maximum mean discrepancy (MMD) solves the stability issue but the method itself fails to generate data that is as high-dimensional as GANs can generate. To solve this issue, GRAM is designed to find a lower-dimensional projection of the data for which the density ratio of the model and data distribution is preserved such that training in the projected space via MMD is statistically efficient and still ensures the training is valid (as the ratio being preserved) in the original space. This formulation avoids introducing any saddle-point optimisation problem and the GRAM training is found to be stable to different choices of various hyper-parameters, and deep generators trained by GRAM also show improved modelling performance on high-dimensional data than existing GANs.

Chapter 6 introduces Bayesian-symbolic physics (BSP), a framework that incorporates symbolic components to generative modelling for physics learning to improve data efficiency. Understanding how humans learn physics is an important topic in cognitive science and an important question to answer in artificial intelligence. While there have been many methods proposed to learn physics from data, compared to what humans can do, they either fall short in adaptivity (i.e. the system can only deal with a limited number of laws that it is programmed to deal with and cannot understand novel physical phenomena; Smith et al., 2019; Ullman et al., 2018) or fall short in data efficiency (i.e. they require much more data to learn a physical law that humans would need; Sanchez-Gonzalez et al., 2019; Battaglia et al., 2016; Breen et al., 2019). BSP aims to fill the gap by providing a method that is adaptive to various, potentially unseen physical laws while only requiring limited data for the learning process. In BSP, I introduce a grammar of Newtonian physics that can be used to describe a wide range of Newtonian physics, and the overall data is assumed to be generated in a probabilistic manner by assuming unseen properties as latent variables, i.e. a generative model. By using physically inspired priors within the grammar, physics learning in BSP is made to be more data-efficient than alternatives based on neural networks. In order to learn physical laws, I extend the traditional symbolic regression setup to a bilevel optimisation problem so that the overall method is more robust to the local minima of the global constants which appear in common physical laws.

Lastly, chapter 7 describes some of my work and contributions to free and open-source software for generative modelling. This includes the development of the open-source ecosystem around TURING.JL in the Julia programming language, especially

ADVANCEDHMC.JL (AHMC) an open-source library that implements CMHMC along with other popular HMC methods in a modular and efficient manner.

The thesis ends with chapter 8 which gives a conclusion of the thesis and sketches some future directions of works contained in this thesis.

Chapter 2

Background

2.1 Generative modelling

I start by introducing generative models and the corresponding learning and inference tasks. Let the random variable X represent the data, which can be a row in the table, an image or a structured instance (e.g. a graph). Denote $\mathbf{x} \in \mathbb{R}^D$ as a realisation of X and suppose we are given a data set of N data points $\mathcal{D} = \{\mathbf{x}^i\}_{i=1}^N$, where $\mathbf{x}^i \sim p_{\mathcal{D}}$—the **data distribution**.[1] For generative modelling, we are interested in building *a generative model with parameter* θ *that describes the data set* $\mathcal{D}$, i.e. a probability distribution p_θ, the **model distribution**, that is close to $p_{\mathcal{D}}$, and in using this model to perform certain tasks that correspond to operations on the probability distribution p_θ, e.g. missing data imputation as inference of unobserved variables.

Representation There are various ways to represent a probability distribution p_θ.

- The most straightforward way is to represent the probability density function (PDF) $p_\theta(\mathbf{x})$ explicitly, i.e. the probability density for a given data point $\mathbf{x}$ has an explicit form and can be computed in a tractable manner. For example, one can make an assumption that the data is Gaussian distributed and define $p_\theta(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\theta = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ (the mean vector and the covariance matrix) is the model parameter and we explicitly have the PDF defined as

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) := \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right).$$

[1] In the scope of this thesis, we assume $\mathbf{x}$ is a vector-valued continuous variable and there are no labels in the data set, meaning the task is unsupervised.

However, the disadvantage of using explicit density functions is that the model would perform badly if the assumption of the explicit distribution fails.

- **Latent variable models** (LVMs) provides a recipe of building more flexible models given explicit probability density functions. In addition to X, a latent variable Z and its realisation $\mathbf{z} \in R^K$ are introduced, followed by two distributions: (i) the prior $p(\mathbf{z})$ and (ii) the likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$. One then has
$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \mathrm{d}\mathbf{z}.$$
Even in cases where both the prior and likelihood are explicit, p_θ can be more flexible and not equivalent to any distribution with an explicit density function. Apart from the expressiveness, one can also "bring" meanings to the latent variable by using a specific prior. For example, by defining $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}_K, \boldsymbol{I}_K)$ and $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{W}\mathbf{z}, \sigma^2 \boldsymbol{I}_D)$, where $K < D$, K is defined to be a low-dimensional representation of X; here $\mathbf{0}_K$ is a K-dimensional zero vector, $\boldsymbol{I}_K$ is a K-by-K identity matrix and $\theta = \{\boldsymbol{W}\}$ is the model parameter. This model is called probabilistic principal component analysis (PPCA) in the literature (Tipping and Bishop, 1999). The model is called PPCA because it is a probabilistic extension of principle component analysis (PCA)—we recover PCA with $\sigma \to \infty$.

- Another way to represent distributions beyond those with explicit density functions is to define the generative process instead. Suppose we are interested in modelling a 1-dimensional real-valued random variable X and we are given a uniformly distributed random variable U, we can define the generation of a realisation X of X given a realisation U of U as
$$X = T(U)$$
where $T : [0,1] \mapsto \mathbb{R}$ is the transformation function. Given such a transformation function, this generative process also defines a probability distribution p_θ. This is in fact called the **inverse transform sampling** and T corresponds to the inverse of the cumulative distribution function (CDF) of p_θ. Note that for any given T, it is not necessarily possible to compute the density $p_\theta(X)$ for a given X.

There are other ways to represent p_θ such as defining a diffusion process (Sohl-Dickstein et al., 2015), defining a set of invertible transformations on a noise variable with the same dimension as X (Papamakarios et al., 2019), defining an energy function as an unnormalised probability distribution (Hinton, 2002; Du and Mordatch, 2019), etc. I do not expand on them due to the focus of this thesis.

Inference Suppose we are given a model p_θ for images and an image instance with a missing patch, which we denote as $\mathbf{x} = \{\mathbf{x}_0, \mathbf{x}_1\}$ where $\mathbf{x}_0$ is the missing part and $\mathbf{x}_1$ are the observed part. The inference task, which is essentially missing data imputation, here is to find $\mathbf{x}_0$ given $\mathbf{x}_1$ based on our model, or more technically, to characterise $p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)$. Another common inference task arises in latent variable models. In cases where we are given a data point $\mathbf{x}$ and we would like to find the corresponding latent variable $\mathbf{z}$, the inference task is to characterise the so-called posterior distribution $p_\theta(\mathbf{x} \mid \mathbf{z})$, which I will expand in section 2.2.

Learning Suppose a model with unknown parameter θ is defined for a problem.[2] A model p_θ with randomly initialised θ will in no way be able to describe the data well and the process of making p_θ close to $p_\mathcal{D}$ is therefore called learning. To define this learning procedure, we need to use some probability divergence D that characterise the similarity between two distributions. For two distributions p and q, a probability divergence D is a function for which the following two properties hold

- $\mathrm{D}(p \| q) \geq 0$ for all p, q
- $\mathrm{D}(p \| q) = 0$ if and only if $p = q$

Given a divergence D, we can define the following optimisation problem

$$\theta^* = \arg\min_\theta \mathrm{D}(p_\mathcal{D} \| p_\theta).$$

Solving this optimisation problem is then called learning because after it, the model is able to describe the data, or in other words, the model learns from the data. Various forms of D correspond to different learning methods, which will be discussed in section 2.3.

I now review four types of generative models that are related to this thesis with details in the rest of this section.

2.1.1 Probabilistic graphical models

Probabilistic graphical models are probabilistic models in which the dependencies

[2]As mentioned in section 1.1, model learning can be in forms of learning the parameters, learning the structure or learning the model hypothesis. Here we mostly focus on the first type of learning while some of the methods or principles are generically applied. For example, one could encode the model structure as (discrete) parameters and treat the structure learning as parameter learning. With this being said, it is usually believed that hypothesis learning is the hardest, followed by structure learning then parameter learning.

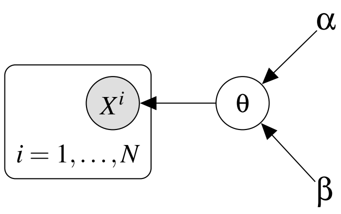


Figure 2.1: Plate diagram for the coin-flipping model

between random variables are represented by graphs. Such a graph could be an undirected graph, a directed acyclic graph, or even a cyclic directed graph. This thesis mostly focuses on directed graphical models while some techniques can also be applied to undirected ones. Directed graphical models are usually presented by either their generative process or in plate diagram notations. I give two examples next.

Coin flipping Suppose you are presented with a coin which you do not know whether it is fair or unfair. You want to know its fairness based on a few flips. One way to do so is by defining a coin-flipping model as follows. Define θ as the probability of flipping the coin gives a face, and $\{X^i\}_{i=1}^N$ is the data—the history of a few flip results for which $X^i = 1$ means the i-th flip is a head and $X^i = 0$ for a tail. The generative process of a coin-flipping models is as follows

$$\begin{aligned}\theta &\sim \mathrm{Beta}(\alpha,\beta)\\ X^i &\sim \mathrm{Ber}(\theta) \quad \text{for} \quad i \in 1,\dots,N\end{aligned}$$

where Beta is the beta distribution, α,β is the parameters of the beta distribution and Ber is the Bernoulli distribution. Note that as being a generative model, one can easily generate synthetic data from the model once defined. The plate diagram for this coin-flipping model is show in figure 2.1. In such plate diagrams, nodes without a circle are constants or (hyper-)parameters, circular nodes (without shades) are unobserved random variables, and circular nodes with shades are observed random variables, i.e. data. The benefit of such illustration is to clearly show the local and global dependencies: Random variables in the same plate are repeated and are independent and identically distributed (i.i.d.) given variables outside.

Bayesian logistic regression **Logistic regression** is a classic machine learning model for binary classification tasks such as spam email detection. For a data point $\boldsymbol{x} \in \mathbb{R}^D$ (e.g. an email), the model with parameter $\boldsymbol{w} \in \mathbb{R}^D$ predicts the probability $\hat{p}$ for $\boldsymbol{x}$ having

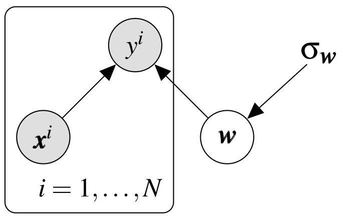


Figure 2.2: Plate diagram for Bayesian logistic regression

a positive label $y = 1$ (e.g. being a spam email) as

$$\hat{p} = \sigma(\boldsymbol{w}^\top \boldsymbol{x})$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function. Such a model is not generative but can be made so by specifying a proper prior and a corresponding likelihood for labels: For example, a standard Gaussian prior on the parameter $\boldsymbol{w}$, $\mathcal{N}(\boldsymbol{w}; \boldsymbol{0}, \sigma_{\boldsymbol{w}} \boldsymbol{I})$ (where $\sigma_{\boldsymbol{w}}$ is a hyper-parameter), and a Bernoulli likelihood for the label y, $\text{Ber}(y; \hat{p})$. The generative process for N labels *given N data points* (i.e. conditioned on $\mathbf{x}^1, \ldots, \mathbf{x}^N$) under this model is

$$\begin{aligned}
&\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma_{\boldsymbol{w}} \boldsymbol{I}) \\
&\text{for } \; i \in 1, \ldots, N \\
&\qquad \hat{p}^i = \sigma(\boldsymbol{w}^\top \boldsymbol{x}^i) \\
&\qquad y^i \sim \text{Ber}(\hat{p}^i)
\end{aligned}$$

This model is called **Bayesian logistic regression** (BLR) and model uncertainty is captured in the distribution of $\boldsymbol{w}$. The plate diagram for this model is show in figure 2.2. Compared to logistic regression, BLR captures the uncertainty in the model parameter $\boldsymbol{w}$, which is a type of *model uncertainty*, also referred as *epistemic uncertainty* (Gal, 2016). Such uncertainty is useful to access when the model is uncertain on its predictions.

2.1.2 Deep generative models

Different representations of generative models, in one way or another, relies on some flexible parametric functions. One way to construct flexible functions with learnable parameters are using neural networks (NNs). There are various ways to incorporate NNs in generative models, loosely corresponding to different generative model representations as discussed in the beginning of this section. For the interest of this thesis, I will focus on the basics of two of them in this section, after reviewing the basics of neural networks.

Neural networks **Neural networks**, or more precisely artificial neural networks, are layer-wise parametric functions inspired by humans' biological neural networks. The basic form of NNs are feedforward neural networks (FNNs) or multilayer perceptrons (MLPs), in which each layer consists a parametric transformation t_θ and an activation function a. The most simple example of t_θ is the **linear layer**: $t_\theta(\boldsymbol{x}) = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$ where $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$ is the parameter containing the weights and biases of this layer. There are a range of choices for a, e.g. $a(x) = \frac{1}{1+\exp(-x)}$, the sigmoid activation, $a(x) = \max(0, x)$, the **rectified linear unit** (ReLU) activation. We call the composition of some choice of t_θ and a as a layer, i.e. $f_\theta(\boldsymbol{x}) = a(t_\theta(\boldsymbol{x}))$. One can compose multiple layers of f to make the transformation more expressive.[3] In fact, it has been shown that under certain conditions, NNs are capable of modelling any function, which is usually referred as the universal approximation theorem (Cybenko, 1989; Pinkus, 1999). Therefore, NNs are used as a flexible parametric function in places where a complex unknown function is needed (to learn). I refer the reader to a standard text book such as Goodfellow et al. (2016) for a comprehensive review of the foundation of neural networks.

Variational autoencoders **Variational autoencoders** (VAEs; Kingma and Welling, 2014; Rezende et al., 2014) are latent variable models that make use of neural networks. VAEs usually consist of two components, a recognition network (also called the encoder) and a generative network (also called the decoder). The recognition network plays a role of inference, which we will explain in section 2.2.2. The generative network is essentially an explicit likelihood function parametrised by neural networks. For example, denote the latent variable as $\mathbf{z}$ and the observed variable as $\mathbf{x}$, a generative network for a Bernoulli likelihood can be written as $p(\mathbf{x} \mid \mathbf{z}) = \mathrm{Ber}(f_\theta(\mathbf{z}))$ where f_θ is a neural network with the weight parameter θ.

Generative adversarial networks **Generative adversarial networks** (GANs Goodfellow et al., 2014) are models inspired from the inverse transform sampling. They *deterministically* transform a random variable $\mathbf{z} \in \mathbb{R}^K$, called the noise, to the data $\mathbf{x} \in \mathbb{R}^D$, where usually $K < D$.[4] Importantly, such transformation is implemented by a neural network f_θ with parameter θ. Suppose we have a uniform noise distribution, the

[3]When the number of layers is large, NNs becomes deep NNs and the method is called as deep learning.

[4]Note that one difference between the noise variable in GANs and the latent variable in VAEs is that in GANs the noise variable *deterministically* decides the observation while the latent variable in VAEs *probabilistically* decides the observation.

sampling process of a data point $\mathbf{x}$ in GANs is

$$\begin{aligned} z_i &\sim \mathcal{U}(0,1) \quad \text{for} \quad i \in 1,\dots,K \\ \mathbf{x} &= f_\theta(\mathbf{z}) \end{aligned}.$$

As GANs directly define the sampling process, the generative network is also called *neural samplers*. Because there is no direct way to evaluate $p_\theta(\boldsymbol{x})$ for GANs, such models are also referred as implicit models and the learning is usually more involved (as maximum likelihood is not directly applicable), which will be discussed in section 2.3.

There are also works such as Donahue et al. (2016); Dumoulin et al. (2016); Tran et al. (2017) which bridge VAEs and GANs. I omit the discussion of them as they do not serve as necessary background for this thesis.

2.1.3 Bayesian nonparametric models

Bayesian nonparametric (BNP) models are Bayesian models in which the number of parameters grows with the complexity of the data. BNP models have the promise to automatically infer the complexity of the model based on the complexity of the data, which is a hard problem known as model selection. The formulation of BNP models usually involves stochastic processes. For example, mixtures models based on Dirichlet process (DP) or Chinese restaurant process (CRP) can automatically infer the number of mixtures that is needed to fit the data. In CRP, the clustering process is analogous to seating customers (= data points) at tables (= clusters) in a Chinese restaurant (Aldous, 1985). Imagine in a restaurant that has an infinite number of tables each with infinite capacity, customers come in according to the following procedure

1. Customer 1 comes and sits at table 1
2. Customer 2 comes and either sits at table 1 or a new table (table 2)
3. Customer *i* comes and either sits at a table already with customers or a new table
    - Customers choose sit at an occupied table with a probability proportional to the number of customers already there.

After seating, each customer has been assigned to a table and the clustering is done. As it can be seen, such process will give a number of clusters that has not predefined maximum value. As this process defines how a set of clusters are generated, one can associate each cluster with observations through a likelihood function. For instance, a Gaussian likelihood would mean that data points belong to the same cluster are deviated from the cluster mean with Gaussian additive noises. Thus, the CRP prior with a likelihood associated describes how data is generated, forming a representation of the generative model.

The inference problem arises in such models will naturally determine the number of clusters instead. While BNP models are defined to have such appealing properties, the learning and inference for them can be hard in general.

2.2 Inference as numerical integration

I review the basics of inference in generative models in this section. To contextualise the discussion, I mainly focus on Bayesian inference in a latent variable model $Z \to X$ with prior $p(\mathbf{z})$ and likelihood $p(\mathbf{x} \mid \mathbf{z})$ while the methodology could apply in other cases; note that I omit the model parameter θ in this section as it is assumed to be fixed during inference. According to the Bayes' rule

$$\mathbb{P}(B \mid A) = \frac{\mathbb{P}(A \mid B)\mathbb{P}(B)}{\mathbb{P}(A)},$$

we call the conditional distribution of Z given X the **posterior**

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} \tag{2.1}$$

where $p(\mathbf{x})$ is called the **marginal likelihood** of $\mathbf{x}$ or **model evidence**. The goal of (Bayesian/posterior) inference is to characterise the posterior $p(\mathbf{z} \mid \mathbf{x})$, which for example in PPCA describes, for a given data point, how its low-dimensional representation look like (Tipping and Bishop, 1999). Importantly, we usually assume $p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}$ is hard to compute or intractable because it involves integration (or summation if Z is discrete).

Usually, obtaining the posterior is not the end of the task and we need to use the posterior for prediction. Suppose we are given a function h that takes the latent variable $\mathbf{z}$ as inputs and makes predictions[5], the so-called Bayesian predictive is the weighted average of h under the posterior

$$H = \int h(\mathbf{z})p(\mathbf{z} \mid \mathbf{x})\mathrm{d}\mathbf{z} = \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z} \mid \mathbf{x})}[h(\mathbf{z})]. \tag{2.2}$$

[5]As in LVMs we assume $\mathbf{z}$ is either some meaningful, intrinsic latent variables (e.g. coin flip parameters) or some compact low-dimensional representation of $\mathbf{x}$ (e.g. as in PPCA), making predictions using $\mathbf{z}$ can be more effective and robust than using $\mathbf{x}$ directly.

One way to estimate H is by Monte Carlo methods if one has a way to (approximately) sample from p. Suppose we have $\{\mathbf{z}^i\}_{i=1}^N$ where $\mathbf{z}^i \sim p(\mathbf{z} \mid \mathbf{x})$, we can estimate H as

$$\hat{H} = \frac{1}{N}\sum_{i=1}^{N} h(\mathbf{z}^i) \approx H. \tag{2.3}$$

It is common to use algorithms to obtain such samples or approximate the posterior by a distribution that is easy to sample from, corresponding to the two algorithms, Markov chain Monte Carlo and variational inference, which I will review in this section.

In addition, instead of first characterising the posterior and then solving the integral, one may also frame the inference problem as approximating this integral in equation 2.2 directly. I will review how to obtain unbiased estimates of H in the end of this section.

2.2.1 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a sampling method to draw (approximate) samples from a target distribution $\pi(\mathbf{x}) = p^*(\mathbf{x})/Z$ (just like the form of $p(\mathbf{z} \mid \mathbf{x})$ in equation 2.1), where $Z = \int p^*(\mathbf{x})\mathrm{d}\mathbf{x}$, knowing only the unnormalised density function p^*. It is a good candidate for Bayesian inference as it avoids the computation of Z, which is usually intractable.

Suppose p is defined on a sampling space $\mathcal{X}$, MCMC works by defining a Markov kernel $\mathcal{K} : \mathcal{X} \mapsto \mathcal{X}$ that characterises the transition from $\mathbf{x}$ to $\mathbf{x}'$, denoted as $\mathcal{K}(\mathbf{x}' \mid \mathbf{x})$. With an initial distribution $p^{(0)}(\mathbf{x})$, one can simulate a Markov chain by first sampling from $p^{(0)}$, and then iteratively applying the kernel $\mathcal{K}$ to the current state, leading to the following intermediate distributions at iteration t ($t \geq 1$)

$$p^{(t)}(\mathbf{x}) = \int \mathcal{K}(\mathbf{x}' \mid \mathbf{x}) p^{(t-1)}(\mathbf{x})\mathrm{d}\mathbf{x}.$$

A MCMC method is valid if this Markov chain has following properties

1. The target distribution π is an invariant distribution of the Markov chain

$$\pi(\mathbf{x}') = \int \mathcal{K}(\mathbf{x}' \mid \mathbf{x})\pi(\mathbf{x})\mathrm{d}\mathbf{x}.$$

2. The chain must be ergodic:

$$p^{(t)}(\mathbf{x}) \to \pi(\mathbf{x}) \quad \text{as } t \to \infty, \text{ for any } p^{(0)}(\mathbf{x}).$$

If such two properties holds, after an infinitely-long MCMC simulation, the samples in the end can be used to construct the Monte Carlo estimate in equation 2.3 as they are from $\lim_{t\to\infty} p^{(t)}(\mathbf{x}) = \pi(\mathbf{x})$, i.e. the target distribution.[6] A sufficient but not necessary condition of the first property is **detailed balance**, which says

$$\pi(\mathbf{x})\mathcal{K}(\mathbf{x}' \mid \mathbf{x}) = \pi(\mathbf{x}')\mathcal{K}(\mathbf{x} \mid \mathbf{x}').$$

For the second property, it is usually enough to show that the initial distribution together with the kernel does not contain some isolated subsets nor have any periodic set (MacKay, 2003). The most popular MCMC methods are the Metropolis–Hastings (MH) algorithm and the Hamiltonian Monte Carlo (HMC) method.

It is important to notice that for a valid MCMC kernel, only running the chain infinitely long ensures the chain samples from the target in general. However, for any target π and a finite chain, it is unclear if the samples are from the stationary π. Thus, the estimate in equation 2.3 by MCMC samples is usually biased although the bias can be incrementally reduced by simulating a longer chain.

2.2.2 Variational inference

Instead of obtaining samples from the target posterior directly, **variational inference** (VI) learns a tractable distribution (tractable in the sensible of being easy to sample from) that approximates the target, and one can approximate H using samples from this approximate distribution. This distribution is usually called the variational distribution and the form of this distribution is usually referred as the variational approximation.

Consider a posterior $p(\mathbf{z} \mid \mathbf{x})$ that we want to characterise, we define the variational posterior $q_\phi(\mathbf{z})$ to be a parametrised distribution with parameter ϕ. We can measure how well $q_\phi(\mathbf{z})$ approximates $p(\mathbf{z} \mid \mathbf{x})$ by some probability discrepancy. The most popular choice is the Kullback–Leibler (KL) divergence due to its tractability. The KL divergence for two distributions p and q is defined as

$$\mathrm{D}_{\mathrm{KL}}(p \| q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \mathrm{d}\mathbf{x}.$$

[6]As an infinite simulation is not possible, practically a long-enough chain, whose length is based on some forms of convergence diagnosis, is run, and the samples during the burn-in period (the period that the samples are far from π) are discarded. This would typically give estimates that have small enough bias. With this being said, as there is no guarantee the samples are actually from π, there is no guarantee on the bias of the estimate. We will focus on this problem in chapter 3.

One way to make $q_\phi(\mathbf{z})$ approximates $p(\mathbf{z} \mid \mathbf{x})$ is by finding the parameter ϕ that minimise the KL divergence between them, equivalently the optimisation problem below

$$\begin{aligned}
\phi^* &= \arg\min_\phi \mathrm{D}_{\mathrm{KL}}\left[q_\phi(\mathbf{z}) \| p(\mathbf{z} \mid \mathbf{x})\right] \\
&= \arg\min_\phi \int q_\phi(\mathbf{z}) \log \frac{q_\phi(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \arg\min_\phi \int q_\phi(\mathbf{z}) \log \frac{q_\phi(\mathbf{z}) p(\mathbf{x})}{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})} \mathrm{d}\mathbf{z} && \text{(Bayes' rule)} \\
&= \arg\min_\phi \left\{\mathrm{D}_{\mathrm{KL}}\left[q_\phi(\mathbf{z}) \| p(\mathbf{z})\right] - \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p(\mathbf{x} \mid \mathbf{z})\right] + \log p(\mathbf{x})\right\} \\
&= \arg\max_\phi \left\{\underbrace{-\mathrm{D}_{\mathrm{KL}}\left[q_\phi(\mathbf{z}) \| p(\mathbf{z})\right] + \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p(\mathbf{x} \mid \mathbf{z})\right]}_{\text{ELBO}}\right\} && \text{(omit constant and flip sign)}
\end{aligned} \tag{2.4}$$

As the KL divergence is non-negative, according to the second to last line in equation 2.4 we also have $\mathrm{D}_{\mathrm{KL}}\left[q_\phi(Z) \| p(Z)\right] - \mathbb{E}_{q_\phi(z)}\left[\log p(X \mid Z)\right] + \log p(X) \geq 0$, which gives

$$\log p(X) \geq \underbrace{-\mathrm{D}_{\mathrm{KL}}\left[q_\phi(Z) \| p(Z)\right] + \mathbb{E}_{q_\phi(z)}\left[\log p(X \mid Z)\right]}_{\text{ELBO}} \tag{2.5}$$

As being a lower bound to the log-evidence $\log p(\mathbf{x})$, the term on the right-hand side of equation 2.5 is referred the evidence lower bound (ELBO). Note that it is also the optimisation objective as in equation 2.4. The most common choice of q_ϕ is the Gaussian distribution, i.e.

$$q_\phi(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where $\phi = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ consists the mean vector and the covariance matrix for $\mathbf{z}$.

Research on VI has been focus on how to allow more flexible representation of q_ϕ (Rezende and Mohamed, 2015; Mescheder et al., 2017a), how to make q_ϕ take account of posterior structures (Hoffman and Blei, 2015; Webb et al., 2017), how to make VI faster for Bayesian neural networks (Wu et al., 2018), etc. One exciting innovation in VI is to use neural networks in variational approximation. In such formation, the parameter of the variational distribution is computed by a neural network that takes the observation as inputs. In fact, this corresponds to the recognition network or the encoder in VAEs. For example, denote the recognition network as $f_\phi(\mathbf{x}) = [\boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}_\phi(\mathbf{x})]$ (ϕ is the neural network parameter and $[\cdot, \cdot]$ is for concatenation), and use Gaussian as the variational approximation, the variational distribution is then

$$q_\phi(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}_\phi(\mathbf{x})^2 \boldsymbol{I}).$$

The main advantage of such formulation is that the recognition network after training can be used to obtain variational parameters for new inputs without any optimisation such as equation 2.4. Therefore, the method is also called amortised inference, as the computation is mostly done during training and obtaining the parameter of the variational distribution at test time is almost free (Gershman and Goodman, 2014).

2.2.3 On Monte Carlo estimation of infinite summations

A range of inference problems consist or can be converted to computing infinite summations. This includes approaching asymptomatic property (i.e. properties that hold in infinite limits) of certain methods (e.g. MCMC in chapter 3) by converting infinite limits to infinite sums (section 2.2.3.4) or Bayesian nonparametric modelling (e.g. models with IBP priors in chapter 4). In this section I review some techniques for estimating infinite summations, and discuss what types of theoretical properties are required for successfully applying them in the end.

From now on, I will focus on reviewing how to compute the infinite summation S defined in the form below

$$S = \sum_{k=0}^{\infty} T_k,$$

where the summation S is assumed to be finite, i.e. $S < \infty$. As long as $T_k \neq 0$ for all $k = 1, \ldots, \infty$, there is no way to exactly compute S as it requires infinite computation. A naive work-around is to truncate this summation up-to some level K, leading to an estimate $\hat{S}_K = \sum_{k=0}^{K} T_k$. The concern of using $\hat{S}_K$ is that this estimate is biased with any choice of K, i.e. $\hat{S}_K \neq S$. The way I will focus on to tackle this bias issue is by Monte Carlo methods to get some estimate $\hat{S}$ with the unbiasedness property $\mathbb{E}[\hat{S}] = S$, where the expectation is taken over the randomness of the Monte Carlo method.

2.2.3.1 Crude Monte Carlo

Consider a random variable X. If T_k can be decomposed as a multiplication of a probability mass term $p_k := \mathbb{P}(X = k)$ and another term T'_k, i.e. $T_k = p_k T'_k$, then S can be written as an expectation $\mathbb{E}[T'_k]$ and estimated by crude Monte Carlo (a.k.a. mean-value Monte Carlo) using the following unbiased estimate

$$\hat{S}_{\mathrm{MC}} = T'_k \tag{2.6}$$

with probability $\mathbb{P}(X = k) = p_k$.

2.2.3.2 Importance sampling

When the decomposition $T_k = p_k T'_k$ is not directly available, one can still apply **importance sampling** to convert S into an expectation by introducing an importance distribution

$$S_{\mathrm{IS}} = \sum_{k=0}^{\infty} q_k \left(\frac{T_k}{q_k}\right) = \mathbb{E}[\frac{T_k}{q_k}] \tag{2.7}$$

where q_k is the probability of $X = k$ under the importance distribution. We can then estimate S_{IS} by Monte Carlo by the following estimate

$$\hat{S}_{\mathrm{IS}} = T_k / q_k \tag{2.8}$$

with probability $\mathbb{P}(X = k) = q_k$. This way of applying importance sampling is also referred as single term weighted truncation (Lyne et al., 2015). The variance of this estimator depends on the choice of importance distribution, which I will expand in section 2.2.3.6.

2.2.3.3 Russian roulette sampling

Russian roulette sampling (Lux and Koblinger, 1991; Carter and Cashwell, 1975; Lyne et al., 2015; Georgoulas et al., 2017) is a Monte Carlo technique for estimating large summations, including infinite ones. In Russian roulette sampling, S is estimated by randomly truncating the summation with τ (a random variable) terms. As a simple example, consider a two-term summation $S = T_1 + T_2$. We can estimate S using

$$\hat{S}_{\mathrm{RR}} = \begin{cases} T_1 & \text{with probability } \pi \\ T_1 + \frac{1}{1-\pi} T_2 & \text{with probability } 1-\pi \end{cases}.$$

It is easy to see that $\mathbb{E}[\hat{S}_{\mathrm{RR}}] = S$, meaning the estimate is unbiased. Applying this trick recursively yields the general Russian roulette estimate.

Let p_τ be the distribution of the truncation level τ; the support of p_τ is a subset of $\mathbb{N}$. We define the Russian roulette estimator $\hat{S}_{\mathrm{RR}}$ as the truncated summation

$$\hat{S}_{\mathrm{RR}} = \sum_{k=0}^{\tau'} \frac{T_k}{\mathbb{P}(\tau \geq k)}, \tag{2.9}$$

where $\tau' \sim p_\tau$. Note that as τ' is a random variable, $\hat{S}_{\mathrm{RR}}$ is also a random variable. $\mathbb{P}(\tau \geq k)$ is the probability that τ is greater than or equal to k, and dividing by this quantity has the effect of correcting for the fact that later terms are less likely to be included in the estimate. It can be shown that this estimate is unbiased for S, e.g. see the appendix of Lyne et al. (2015) for a proof.

A general way to parameterise p_τ is by defining $\mathbb{P}(\tau = k) = (1 - \rho_{k+1}) \prod_{s=0}^{k} \rho_s$ and $\mathbb{P}(\tau = 0) = \rho_0$. This is in fact a Bernoulli process, a stochastic process of tossing coins until getting a tail. In this process, we only make the k-th toss if the previous one turns out to be a head. The probability of the k-th coin being a head is denoted as ρ_k. Note that we also have the probability of starting tossing process being ρ_0. In this parameterisation, the probability vector $\boldsymbol{\rho} := [\rho_0, \rho_1, \rho_2, \ldots]$ is parameters of p_τ and the corresponding Russian roulette estimator can be written as

$$\hat{S}_{\boldsymbol{\rho}} = \sum_{k=0}^{\tau'} \frac{T_k}{\prod_{s=0}^{k-1} \rho_s}, \tag{2.10}$$

where $\tau' \sim p_\tau$ and $\prod_{s=0}^{k-1} \rho_s$ is the probability that $\tau \geq k$ ($\prod_{s=0}^{-1} \rho_s$ is defined as 1).

Note that the actual distribution of p_τ decides the expected computation and the variance of the estimator, the latter of which can potentially be infinite. I leave the related discussion to section 2.2.3.6.

2.2.3.4 The telescoping sum trick for infinite limits

One reason that unbiased estimation of infinite sums can be interesting is that, under certain conditions, any infinite limits can be converted to an infinite sum and thus be unbiased estimated. Consider the limit $L = \lim_{t \to \infty} S_t$, one can rewrite it as an summation of infinite terms of "differences" as below

$$\begin{aligned} L = \lim_{t \to \infty} S_t &= S_0 + (S_1 - S_0) + (S_2 - S_1) + \ldots \\ &= S_0 + \sum_{t=1}^{\infty} (S_t - S_{t-1}) \\ &= \sum_{t=0}^{\infty} \delta_t \end{aligned} \tag{2.11}$$

where the "differences" are defined as

$$\delta_t = \begin{cases} S_0 & \text{if } t = 0 \\ S_t - S_{t-1} & \text{if } t > 0 \end{cases}.$$

Once the infinite limit is written in the form of infinite sums, generic methods to estimate infinite sums, which are discussed earlier, can be used to estimate the limit, potentially in an unbiased manner. However, when each S_t is the distribution $q^{(t)}$ induced by a

Markov chain and an initial distribution as discussed in section 2.2.1, there is a special, low-variance way based on coupling to estimate such a limit with finite computation, which is reviewed next.

2.2.3.5 Unbiased MCMC via couplings

Coupling In probability theory, coupling is a concept defined for two distributions, μ defined on $\mathcal{X}$ and ν defined on $\mathcal{Y}$, and another distribution γ defined on $\mathcal{X} \times \mathcal{Y}$. All the couplings of μ and ν, $\Gamma(\mu,\nu)$, forms a set of distributions defined on $\mathcal{X} \times \mathcal{Y}$ for which their marginals are same as μ and ν, i.e.

$$\int \gamma(\mathbf{x},\mathbf{y})\mathrm{d}\mathbf{y} = \mu \quad \text{and} \quad \int \gamma(\mathbf{x},\mathbf{y})\mathrm{d}\mathbf{x} = \nu \quad \forall \gamma \in \Gamma(\mu,\nu).$$

We call any $\gamma \in \Gamma(\mu,\nu)$ a coupling of μ and ν. For a graduate-level introduction to couplings in the context of Monte Carlo methods, see the online course *Couplings and Monte Carlo* by Pierre Jacob at https://sites.google.com/site/pierrejacob/cmclectures.

Coupled MCMC Similarly, couplings can be defined for a pair of MCMC kernels. For two MCMC kernels $\mathcal{K}_1 : \mathcal{X} \mapsto \mathcal{X}$ and $\mathcal{K}_2 : \mathcal{Y} \mapsto \mathcal{Y}$, the couplings for them is a set of functions of $\mathcal{X} \times \mathcal{Y} \mapsto \mathcal{X} \times \mathcal{Y}$ for which the marginal kernels are same as $\mathcal{K}_1$ and $\mathcal{K}_2$.

Recall that when estimating $H = \int h(\mathbf{x})\pi(\mathbf{x})\mathrm{d}\mathbf{x}$ via Monte Carlo using MCMC samples, the estimate is usually biased because MCMC samples are only approximate—there is no guarantee that they are actually from the target distribution (stationary distribution) in finite time. Rhee and Glynn (2012) develops an unbiased estimate based on the telescoping sum trick reviewed in the previous section, and Jacob et al. (2019b) further improves the estimate based on coupled MCMC methods. This will make use of the property of valid MCMC chain X for which $\lim_{t\to\infty} p^{(t)}(\mathbf{x}) = \pi(\mathbf{x})$ holds. First, because of this limit, define $H_t := \int h(\mathbf{x})p^{(t)}(\mathbf{x})\mathrm{d}\mathbf{x} := \mathbb{E}[h(X_t)]$ we have

$$H = \lim_{t\to\infty} H_t = \lim_{t\to\infty} \mathbb{E}[h(X_t)].$$

We can then apply the telescoping sum trick to re-write it as an infinite sum

$$\begin{aligned} \lim_{t\to\infty} \mathbb{E}[h(X_t)] &= \mathbb{E}[h(X_0)] + (\mathbb{E}[h(X_1)] - \mathbb{E}[h(X_0)]) + (\mathbb{E}[h(X_2)] - \mathbb{E}[h(X_1)]) + \dots \\ &= \mathbb{E}[h(X_0)] + \textstyle\sum_{t=1}^{\infty} (\mathbb{E}[h(X_t)] - \mathbb{E}[h(X_{t-1})]) \end{aligned} \tag{2.12}$$

To estimate such infinite sums with finite computation and low variance (compared to the generic methods reviewed earlier), Jacob et al. (2019b) introduce another MCMC chain Y that coupled with X. This coupled chain Y has the following two properties

1. Y has the same *marginal* distribution as X
2. The chain (Y_{t-1}) meets with (X_t) and stays together after some iteration τ

Given such a chain Y, the unbiased estimate via coupled MCMC is derived as follows

$$\begin{aligned}
&\lim_{t\to\infty}\mathbb{E}[h(X_t)] \\
=&\mathbb{E}[h(X_0)]+\textstyle\sum_{t=1}^{\infty}(\mathbb{E}[h(X_t)]-\mathbb{E}[h(X_{t-1})]) && \text{(equation 2.12)}\\
=&\mathbb{E}[h(X_0)]+\textstyle\sum_{t=1}^{\infty}(\mathbb{E}[h(X_t)]-\mathbb{E}[h(Y_{t-1})]) && \text{(property 1)}\\
=&\mathbb{E}[h(X_0)]+\mathbb{E}[\textstyle\sum_{t=1}^{\infty}(h(X_t)-h(Y_{t-1}))] && \text{(exchange expectation and summation)}\\
=&\mathbb{E}[h(X_0)]+\mathbb{E}[\textstyle\sum_{t=1}^{\tau-1}(h(X_t)-h(Y_{t-1}))] && \text{(property 2)}
\end{aligned}$$

As so, the research on this estimate has focused on developing coupled MCMC methods for which the coupled chains meet fast, which affects the meeting time τ and thus the expected computation time.

2.2.3.6 Properties and requirements of mentioned infinite sum estimators

A set of requirements have to be satisfied in order to use the generic estimators, $\hat{S}_{\text{MC}}, \hat{S}_{\text{IS}}, \hat{S}_{\text{RR}}$, reviewed in this section.

1. The infinite summation itself should be finite valued, i.e. $S < \infty$. This is usually obvious in the problem itself or easy to check based on the form of the infinite sum. In general, this would require the sequence (T_k) decays in certain rates.
2. The expected computation cost should be finite. Suppose the evaluation of any term has a constant, finite cost C, this is easily true for $\hat{S}_{\text{MC}}$ and $\hat{S}_{\text{IS}}$, which are both single-sample estimators, i.e. only need to evaluate a single term. For $\hat{S}_{\text{RR}}$, the expected cost $\sum_{k=0}^{\infty} Ck\mathbb{P}(\tau=k) = C\sum_{k=0}^{\infty} k\mathbb{P}(\tau=k) = C\mathbb{E}_{p_\tau}[k]$ is required to be finite. As C is finite, this basically requires the truncation distribution p_τ to have an finite expectation. This usually means the probability mass decays in certain rates when k increases.
3. The estimator itself should have finite variance.

- For $\hat{S}_{\mathrm{MC}}$, it means $\sum_{k=0}^{\infty} p_k T_k'^2 < \infty$.
- For $\hat{S}_{\mathrm{IS}}$, it means $\sum_{k=0}^{\infty} q_k (\frac{T_k}{q_k})^2 = \sum_{k=0}^{\infty} \frac{T_k^2}{q_k} < \infty$.
- For $\hat{S}_{\mathrm{RR}}$, it means

$$\sum_{k=0}^{\infty} \mathbb{P}(\tau = k)(\sum_{k=0}^{\tau'} \frac{T_k}{\mathbb{P}(\tau \geq k)})^2 = \sum_{k=0}^{\infty} \left[(1-\rho_{k+1}) \prod_{s=0}^{k} \rho_k\right] (\sum_{k=0}^{\tau'} \frac{T_k}{\prod_{s=0}^{k-1} \rho_s})^2 < \infty$$

One important thing to notice here is that this variance, unlike the expected cost, also depends on T_k. Specific analysis is problem-dependent however usually it would mean the probability mass decays in a rate that is faster than the decay of the sequence (T_k^2).

For the unbiased MCMC estimation via couplings, the conditions are slightly more involved, which we will discuss in chapter 3.

2.3 Learning as divergence minimisation

I now review a few ways to learn generative models from data. In the rest of this section, we assume that we are equipped by a generative model p_θ with parameter θ. We are given a N-sized set of D-dimensional continuous data points $\mathcal{D} = \{\mathbf{x}^i\}_{i=1}^N$ i.i.d. from the data distribution $p_{\mathcal{D}}$, i.e. $\mathbf{x}^i \sim p_{\mathcal{D}}$. The goal of learning is to make p_θ similar to $p_{\mathcal{D}}$.

2.3.1 Kullback–Leibler divergence and maximum likelihood

Maximum likelihood is a general principle in statistics to learn probability distributions (James et al., 2013). For given p_θ and $\mathcal{D}$, the log-likelihood is defined as

$$\mathcal{L}_{\mathrm{ML}}(\theta, \mathcal{D}) = \log \prod_{\mathbf{x} \in \mathcal{D}} p_\theta(\mathbf{x}) = \sum_{\mathbf{x} \in \mathcal{D}} \log p_\theta(\mathbf{x})$$

and the maximum likelihood estimate (MLE) θ^*_{ML} is then defined as

$$\theta^*_{\mathrm{ML}} = \arg\max_\theta \mathcal{L}_{\mathrm{ML}}(\theta, \mathcal{D}). \tag{2.13}$$

The basic requirement of applying maximum likelihood is that the probability density function of the model is tractable. For example, if the model is defined as a Gaussian distribution with unknown mean and variance. However, directly solving equation 2.13 is not possible for latent variable models $Z \to X$ *in general* due to the intractable marginal likelihood which defined as an integral $p_\theta(\mathbf{x}) = \int p(\mathbf{z})p_\theta(\mathbf{x} \mid \mathbf{z})\mathrm{d}\mathbf{z}$. Fortunately, as reviewed for variational inference in section 2.2.2, such marginal likelihood has a tractable lower-bound

$$\log p_\theta(\mathbf{x}) \geq -\mathrm{D}_{\mathrm{KL}}\left[q_\phi(Z) \| p(Z)\right] + \mathbb{E}_{q_\phi(z)}\left[\log p_\theta(X \mid Z)\right] =: \mathcal{L}_{\mathrm{VI}}(\theta, \mathcal{D}) \tag{2.14}$$

where $q_\phi(\mathbf{z})$ is the variational distribution that is introduced to approximate the true posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$. Therefore, for LVMs, instead of maximising the log-likelihood directly, one can maximise a lower-bound of it—the ELBO

$$\theta^*_{\mathrm{VI}} = \arg\max_\theta \mathcal{L}_{\mathrm{VI}}(\theta, \mathcal{D}). \tag{2.15}$$

When both p_θ and q_ϕ are implemented as neural networks, the two networks forms the variational auto-encoder, and equation 2.15 is the standard training objective of VAEs, for which ϕ is also optimised using the same objective. It can be shown that when the bound in equation 2.14 is tight, the training is consistent with maximum likelihood estimation, i.e. $\theta^*_{\mathrm{VI}} = \theta^*_{\mathrm{ML}}$.

Another important fact for MLE is that solving equation 2.13 is equivalent to minimising the KL divergence between $p_\mathcal{D}$ and p_θ:

$$\begin{aligned}\arg\min_\theta \mathrm{D}_{\mathrm{KL}}[p_\mathcal{D} \| p_\theta] &= \arg\min_\theta \int p_\mathcal{D}(\mathbf{x}) \log \frac{p_\mathcal{D}(\mathbf{x})}{p_\theta(\mathbf{x})} \\ &= \arg\min_\theta \mathbb{E}_{p_\mathcal{D}}[\log p_\mathcal{D}(\mathbf{x})] - \mathbb{E}_{p_\mathcal{D}}[\log p_\theta(\mathbf{x})]. \\ &= \arg\max_\theta \mathbb{E}_{p_\mathcal{D}}[\log p_\theta(\mathbf{x})]\end{aligned} \tag{2.16}$$

When $\mathbb{E}_{p_\mathcal{D}}[\log p_\theta(\mathbf{x})]$ is estimated by Monte Carlo using the data set $\mathcal{D}$, we recover equation 2.13. This creates link between model learning and divergence minimisation, and one would expect different learning algorithms emerge by difference choices of divergences, which I will continue reviewing.

2.3.2 Jensen–Shannon divergence and adversarial learning

As mentioned in the previous section, maximum likelihood can only be applied to models for which the likelihood (or a lower bound of it) is tractable, which is not the case for neural samplers as reviewed in the beginning of section 2.1. Goodfellow et al. (2014) proposed a learning method called **adversarial learning** that can be used to learn neural samplers (also called implicit models), leading to a class of models called generative adversarial networks (GANs). Denote $G_\theta : \mathcal{Z} \mapsto \mathcal{X}$ as generator representing the generative process of p_θ and p_{noise} on $\mathcal{Z}$ as the noise distribution, adversarial learning introduces an auxiliary network $D_\phi : \mathcal{X} \mapsto [0,1]$ called the discriminator, which is a binary classifier implemented by a neural network. The discriminator is trained to predict whether a sample is from p_θ or $p_\mathcal{D}$, and can be used to provide learning signal to p_θ. The original adversarial learning (Goodfellow et al., 2014) iterates gradient-based parameter update for the following two optimisation steps

$$\text{Step 1} \quad \arg\max_\phi \mathbb{E}_{\mathbf{x}\sim p_\mathcal{D}}[\log D_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\text{noise}}}[\log\left(1 - D_\phi(G_\theta(\mathbf{z}))\right)] \tag{2.17}$$

$$\text{Step 2} \quad \arg\min_\theta \mathbb{E}_{\mathbf{z}\sim p_{\text{noise}}}[\log\left(1 - D_\phi(G_\theta(\mathbf{z}))\right)] \tag{2.18}$$

It can be shown that such learning corresponds to minimising the Jensen–Shannon divergence between p_θ and $p_\mathcal{D}$ (Goodfellow et al., 2014; Nowozin et al., 2016)

$$\theta^*_{\text{adv}} = \arg\min_\theta \mathrm{D}_{\text{JS}}[p_\mathcal{D} \| p_\theta]$$

where

$$\mathrm{D}_{\text{JS}}[p_\mathcal{D} \| p_\theta] := \frac{1}{2}\mathrm{D}_{\text{KL}}[p_\mathcal{D} \| \frac{p_\mathcal{D} + p_\theta}{2}] + \frac{1}{2}\mathrm{D}_{\text{KL}}[p_\theta \| \frac{p_\mathcal{D} + p_\theta}{2}].$$

Note that adversarial learning (Goodfellow et al., 2014) is closely related to prior works on density ratio estimation (Gutmann and Hyvärinen, 2010) due to the role of D_ϕ as effectively being a density ratio estimator..

While adversarial learning enables the training of neural samplers or implicit models, it has been found that the learning is sensitive to hyper-parameters (learning rate, network sizes, etc) due to the saddle-point optimisation problem rooted in the min-max formulation (equation 2.17 and equation 2.18).

2.3.3 Maximum mean discrepancy

One way to train neural samplers stably is using a discrepancy that does not reply on an auxiliary network as in adversarial learning. An example of this approach is by using the **maximum mean discrepancy** (MMD; Gretton et al., 2012) which measures the discrepancy between two distributions as the maximum difference between the expectations of a class of functions $\mathcal{F}$ in which $f : \mathcal{X} \mapsto \mathbb{R}$:

$$\text{MMD}_\mathcal{F}(\mu, \nu) = \sup_{f\in\mathcal{F}} \left(\mathbb{E}_\mu[f(\mathbf{x})] - \mathbb{E}_\nu[f(\mathbf{x})]\right). \tag{2.19}$$

Here $\mathbb{E}_\mu[f(\mathbf{x})]$ and $\mathbb{E}_\nu[f(\mathbf{x})]$ are usually referred as mean embeddings and equation 2.19 effectively computes the maximum difference for the mean embedding, which leads to the name maximum mean discrepancy.

If $\mathcal{F}$ is chosen to be a rich enough class, then MMD is a valid probability discrepancy, i.e., $\mathrm{MMD}(\mu,\nu) = 0$ implies that $\mu = \nu$. Gretton et al. (2012) show that it is sufficient to choose $\mathcal{F}$ to be a unit ball in an reproducing kernel Hilbert space (RKHS) $\mathcal{R}$ with a characteristic kernel k. Given samples $\mathbf{x}_1,\ldots,\mathbf{x}_N \sim \mu$ and $\mathbf{y}_i,\ldots,\mathbf{y}_M \sim \nu$, we can estimate $\mathrm{MMD}_\mathcal{R}$ as

$$\hat{\mathrm{MMD}}^2_\mathcal{R}(\mu,\nu) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{i'=1}^{N} k(\mathbf{x}_i,\mathbf{x}_{i'}) - \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} k(\mathbf{x}_i,\mathbf{y}_j) + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{j'=1}^{M} k(\mathbf{y}_j,\mathbf{y}_{j'}). \tag{2.20}$$

MMD networks (MMD-nets; Li et al., 2015; Dziugaite et al., 2015) are neural samplers for which the learning is done by minimising the empirical estimate of MMD in equation 2.20. MMD-nets are found stable to train but, on their own, not much empirical success on high-dimensional image data, which GANs are capable of modelling.

2.3.4 Wasserstein distance and optimal transportation

Lastly, another way to measure the discrepancy between two probability distributions is through the theory of optimal transportation (OT). Although in this thesis OT is not used to learn generative models, it is used for other purposes (e.g. improving couplings) in chapter 3. For this reason, I review OT in this section.

Intuitively, suppose we have two distributions μ and ν defined on the same space $\mathcal{X}$. They both describe the allocation a unit amount of earth is distributed on $\mathcal{X}$. Let's also define $d(\boldsymbol{x},\boldsymbol{x}')$ as the cost of moving earth between location $\boldsymbol{x}$ and $\boldsymbol{x}'$. The *Wasserstein distance* is defined to be the minimal cost of turning the allocation defined by μ to that defined by ν. Because of this earth moving analogy, this distance is also called *earth mover's distance* (EMD). More formally, the Wasserstein-p distance (W_p) is defined as

$$W_p(\mu,\nu) = \left(\inf_{\gamma\in\Gamma(\mu,\nu)} \int\int \gamma(\mathbf{x},\mathbf{x}')d(\mathbf{x},\mathbf{x}')^p \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{x}'\right)^{1/p},$$

where $\Gamma(\mu,\nu)$ is all couplings between μ and ν. One could imagine using this optimisation problem to find the coupling that makes two marginal distribution close in some choice of d, which is how OT is used in chapter 3.

The Wasserstein distance is a proper probability discrepancy thus can also be used to train neural samplers, e.g. in Arjovsky et al. (2017b). In fact, both the Wasserstein distance and the MMD are instances of a family of probability discrepancies called the integral probability metrics (IPMs) while divergences like Kullback–Leibler divergence and Jensen-Shannon divergence belong to the family of f-divergence (Sriperumbudur et al., 2009). Both families have different theoretical and practical properties; see Sriperumbudur et al. (2009) for details.

2.4 Limitations in robustness and efficiency

However, forementioned algorithms for generative modelling have many limitations and fall shorts in terms of robustness and efficiency.

Unbiased MCMC estimation are sensitive to hyper-parameters and statistically inefficient (*robustness and statistically efficiency*). In order to achieve fast meeting (which corresponds to small computation cost), coupled HMC methods have been developed. However, it has been found that the computation time of this estimate is very sensitive to the two important parameters of the coupled HMC kernel. Moreover, the estimates based on coupled HMC are statistically less efficient than those based on normal HMC, making the use of unbiased estimation less appealing.

Inference for BNP models are either restricted to truncation or unable to scale to large data sets (*computational efficiency*). Traditionally inference for BNP models are either based on truncated VI or sampling-based methods. For truncated VI, a maximum model complexity need to be set, which is not desirable because BNP models are aimed to automatically find the model complexity. For sampling-based methods such as Gibbs sampling and slice sampling, the inference is usually slow compared to VI and cannot benefit from modern hardware acceleration.

GANs are unstable to train and MMD-nets fail to model high-dimensional data (*robustness and statistically efficiency*). The convergence of adversarial training is sensitive to hyper-parameters like network architecture or learning rates due to the existence of saddle-point optimisation. The efficiency of using MMD as the training objective is limited to the nature of kernel methods, which is known to be less effective for high-dimensional data unless there are flexible enough kernels. How to train neural samplers that are both robust and effective in high dimension remains a challenge.

Symbolic generative models provides the advantage of incorporating symbolic priors for data efficiency but they can be sensitive to local optima for learnable constants (*robustness and data efficiency*). Unlike neural methods or statistical methods, symbolic methods such as symbolic regression need special care of incorporating numerical values, which is refereed as constant learning, when the symbolic form is learned. In the context of generative modelling, applying standard learning framework to learn such constant as latent variables can make the learning sensitive to local optima, and it requires some other formulation to alleviate this drawback.

Part I

Inference

Chapter 3

Couplings for Multinomial Hamiltonian Monte Carlo

3.1 Introduction

Markov chain Monte Carlo (MCMC) is a standard tool to draw samples from target distributions known up to a normalising constant (Metropolis et al., 1953; Geman and Geman, 1984). Such samples are commonly used to estimate an integral of interest. Specifically, for a probability distribution π on $\mathbb{R}^d$ and a measurable function of interest $h : \mathbb{R}^d \mapsto \mathbb{R}$, we want to estimate

$$H = \int \pi(\mathbf{x}) h(\mathbf{x}) \mathrm{d}\mathbf{x} = \mathbb{E}_{\mathbf{x}\sim\pi}[h(\mathbf{x})]. \tag{3.1}$$

Approximating this integral H in equation 3.1 is at the core of many statistics and machine learning problems. For example in Bayesian inference, Monte Carlo samples are used to estimate some posterior predictive distribution, or perform model comparison (Gelman et al., 2013). Or in energy-based modelling, MCMC samples are to estimate gradients used to update model parameters (Teh et al., 2003; Xie et al., 2016; Qiu et al., 2019). A particular popular MCMC algorithm is Hamiltonian Monte Carlo (HMC), which is widely applicable to differentiable π and has been shown to scale well to high-dimensional problems due to the use of underlying geometric information (Neal, 2011).

Under the framework of MCMC, a Markov chain is simulated to obtain correlated samples from π, and then Monte Carlo integration is used to estimate H using these samples. However, such estimators are unbiased only when the underlying Markov chain has converged to the equilibrium, which is challenging to verify in practice. Therefore, convergence analysis of the chain is performed to ensure the bias is sufficiently small.

MCMC with couplings has attracted research attention recently thanks to its ability to debias Monte Carlo estimators (Jacob et al., 2019b). In particular, Heng and Jacob (2019) focused on the Metropolis-Hastings (MH) adjusted HMC variant, which proposes the end-point of a simulated Hamiltonian trajectory as the new state, followed by an MH correction step. We refer to this HMC variant as *coupled Metropolis HMC*. Heng and Jacob (2019) noticed that coupled Metropolis HMC is sensitive to the choice of HMC parameters such as integrator step sizes and Hamiltonian trajectory lengths. More specifically, parameters (e.g. trajectory lengths) optimal for sampling efficiency (e.g. effective sample size) can require numerous HMC iterations to achieve meeting; on the other hand, optimal parameters for coupling can lead to poor mixing (Heng and Jacob, 2019).

Building upon the recent work of Heng and Jacob (2019), we propose two novel couplings based on a different, more robust implementation of HMC—*multinomial HMC*. Unlike Metropolis HMC that only considers the end-point of a simulated trajectory with a risk of getting rejection, multinomial HMC samples from the entire simulated trajectory and is thus more robust. We refer to our methods as *coupled multinomial HMC* and demonstrate several advantages of these methods. First, coupled multinomial HMC meets faster in general. Intuitively, like all MH algorithms, the previous coupled HMC method can only propose a point from the initial or the last integration step, which leads to two drawbacks for couplings: (i) it may well be that intermediate points are the closest between two chains and (ii) rejection rates of proposals are quite sensitive to step sizes of Hamiltonian dynamics solvers. Multinomial coupling allows coupled chains to accept intermediate points that are potentially closer together, so they meet quicker. We therefore design couplings to minimise the expected distance between coupled chains within each transition to encourage faster meeting. Second, coupled multinomial HMC is less sensitive to Hamiltonian integration step sizes. For Metropolis HMC, a small enough step size has to be used to ensure a large enough acceptance probability in the MH adjustment step. However, for multinomial HMC, intermediate points can be proposed even though the end-points would have been rejected in Metropolis HMC. We argue that this robustness is crucial for practical use of coupled HMC algorithms. Thirdly, we prove that the meeting time of coupled multinomial HMC decays geometrically, which is a sufficient condition to use the unbiased estimator from Jacob et al.

Algorithm 3.1 Sample a pair of coupled chains

Input: A kernel $\mathcal{K}$, its coupled kernel $\bar{\mathcal{K}}$, an initial distribution π_0 and a coupling of π_0

Output: A pair of coupled chains (X,Y)k

```
1: Sample $(X_0, Y_0) \sim \bar{\pi}_0$ ($\bar{\pi}_0$ is a coupling of $\pi_0$)
2: Sample $X_1 \sim \mathcal{K}(X_0, \cdot)$
3: Set $N = 1$
4: while $X_N \neq Y_{N-1}$ do
5:     Sample $(X_{N+1}, Y_N) \sim \bar{\mathcal{K}}((X_N, Y_{N+1}), \cdot)$
6:     Set $N = N + 1$
7: end while
8: Set $\tau = N$ and output $\{(X_n)_{n=0}^{\tau}, (Y_n)_{n=0}^{\tau-1}\}$
```

(2019b). Finally, we perform extensive simulations to verify the improved meeting and robustness of our proposed method. The improvements in efficiency and robustness together pave the way for a wider and more practical use of coupled HMC methods.

3.2 Background

3.2.1 Unbiased MCMC with couplings

For two distributions p and q, we denote $\Gamma(p,q)$ as their *couplings*, i.e. for any $\gamma \in \Gamma(p,q)$, the marginals of γ are p and q. For a π-invariant Markov kernel $\mathcal{K}$ defined on $\left(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d)\right)$, its coupled kernel $\bar{\mathcal{K}}$, defined on $\left(\mathbb{R}^d \times \mathbb{R}^d, \mathcal{B}(\mathbb{R}^d) \times \mathcal{B}(\mathbb{R}^d)\right)$, by construction has $\mathcal{K}$ as its marginals, where $\mathcal{B}$ denotes the Borel σ-algebra. Additionally, given an initial distribution π_0 and some $\bar{\pi}_0 \in \Gamma(\pi_0, \pi_0)$, a pair of coupled chains $X = (X_n)_{n \geq 0}$, $Y = (Y_n)_{n \geq 0}$ that share the same equilibrium distribution π can be simulated by algorithm 3.1 (Jacob et al., 2019b) until meeting at iteration $\tau := \inf\{n \geq 1 : X_n = Y_{n-1}\}$ (the *meeting time*). The main design choice in this algorithm is the construction of $\bar{\mathcal{K}}$. Jacob et al. (2019b) established that if $\bar{\mathcal{K}}$ satisfies certain conditions/assumptions, a pair of coupled chains X, Y from algorithm 3.1 can be used to obtain unbiased estimates of equation 3.1 with finite variance and finite computation cost as

$$H_k(X,Y) = h(X_k) + \sum_{n=k+1}^{\tau-1} \{h(X_n) - h(Y_{n-1})\} \tag{3.2}$$

where $k \in \mathbb{N}$ is a parameter to choose.[1] The first term in equation 3.2 is a standard, single-sample MCMC estimate and the second term can be seen as a debiasing term to correct the bias introduced by non-converged chains. This estimator is built on the pioneering works from Glynn and Rhee (2014), derived using telescoping sums. In practice, we use a time-averaged version of equation 3.2, which is still unbiased but with lower variance, e.g. in Section 3.5. The three assumptions on $\bar{\mathcal{K}}$ to use equation 3.2 are

Assumption 1 (Convergence of marginal chain)**.** As $n \to \infty$, $\mathbb{E}[h(X_n)] \to H$. Furthermore, $\exists\, \eta > 0$ and $D < \infty$ such that $\mathbb{E}[|h(X_n)|^{2+\eta}] \leq D$ for all $n \in \mathbb{N}$.

Assumption 2 (Tail of meeting time)**.** The chains are such that the *meeting time* $\tau := \inf\{n \geq 1 : X_n = Y_{n-1}\}$ satisfies $\mathbb{P}(\tau > n) \leq C\lambda^n$ for all $n \in \mathbb{N}$ for some constants $C < \infty$ and $\lambda \in (0,1)$.

Assumption 3 (Faithfulness)**.** The coupled chains are faithful (Rosenthal, 1997)—they stay together after meeting, i.e. $X_n = Y_{n-1}$ for all integers $n \geq \tau$.

Assumption 1 basically requires the marginal chains to be valid, i.e. converging to the target and the function h has finite variance; assumption 2 requires the coupling to be "fast-enough" such that the estimate has finite computation time. Under these assumptions, using a pair of coupled chains (X,Y) from algorithm 3.1, we can obtain an unbiased estimate of equation 3.1 with finite variance (Glynn and Rhee, 2014; Jacob et al., 2019b)

3.2.2 Hamiltonian Monte Carlo

In an HMC kernel, new states are proposed by simulating Hamiltonian dynamics (Neal, 2011). For a Hamiltonian system with a position variable $q \in \mathbb{R}^d$ and a momentum variable $p \in \mathbb{R}^d$, the trajectory $\mathbf{t} := (q(t), p(t))_{t \in \mathbb{R}_+}$ can be described by the following ordinary differential equations

$$\begin{aligned} \frac{\mathrm{d}q}{\mathrm{d}t} &= +\nabla_p \mathcal{E}\left(q(t), p(t)\right), \\ \frac{\mathrm{d}p}{\mathrm{d}t} &= -\nabla_q \mathcal{E}\left(q(t), p(t)\right) = -\nabla U\left(q(t)\right) \end{aligned} \tag{3.3}$$

[1] When k happens to be smaller than $\tau - 1$, the second term can be seen as a debiasing term; otherwise, the debiasing term could be non-existent, which is counter-intuitive. To understand this, recognise the fact that τ itself is a random variable that has a different realisation for each MCMC simulation and the probability of $k < \tau - 1$ (depending on the convergence rate of MCMC chains) is non-zero—meaning that in expectation the debiasing term helps remove the bias.

where the *potential* $U : \mathbb{R}^d \mapsto \mathbb{R}_+$ is chosen s.t. the target $\pi(q) \propto \exp(-U(q))$, the *kinetic* term $K : \mathbb{R}^d \mapsto \mathbb{R}_+$ is assumed to have a form of $K(p) = \frac{1}{2}p^\top Mp$, where M is the mass matrix, and the *Hamiltonian* is defined as $\mathcal{E}(q,p) := U(q) + K(p)$.[2] The extended target $\bar{\pi}$ for *phase points* $z := (q,p)$ on the *phase space* $\mathbb{R}^d \times \mathbb{R}^d$ is then defined as having density $\propto \exp(-\mathcal{E}(q,p))$.

To describe the dynamics more succinctly, we consider the *flow* map $\Phi_t(q_0,p_0) = (q(t),p(t))$ for equation 3.3 initialised at $(q_0,p_0) := \big(q(0),p(0)\big) \in \mathbb{R}^d \times \mathbb{R}^d$. Following Heng and Jacob (2019), we write $\Phi_t^\circ(q_0,p_0) = q(t)$ and $\Phi_t^*(q_0,p_0) = p(t)$ for the flow projected onto its position and momentum spaces, respectively. The flow map Φ_t is in general not available in closed form and requires discretisation in time via numerical integrators as approximations. A standard choice for HMC is the *leapfrog integrator* that, given an initial phase point (q_0,p_0), iterates:

$$\begin{aligned} p_{\ell+1/2} &:= p_\ell - \frac{\varepsilon}{2}\nabla U(q_\ell) \\ q_{t+1} &:= q_\ell + \varepsilon p_{\ell+1/2} \\ p_{t+1} &:= p_{\ell+1/2} - \frac{\varepsilon}{2}\nabla U(q_{\ell+1}) \end{aligned}$$

for $\ell = 0,\dots,L-1$ with a step size $\varepsilon > 0$ and leapfrog steps $L \in \mathbb{N}$. We denote $\hat{\Phi}_{\varepsilon,\ell}(q_0,p_0) := (q_\ell,p_\ell)$ as the numerical flow map approximated by a leapfrog integrator with a step size ε for ℓ steps, and similarly $\hat{\Phi}^\circ_{\varepsilon,\ell}$ and $\hat{\Phi}^*_{\varepsilon,\ell}$ for projected maps onto position and momentum, respectively. For more properties of Hamiltonian dynamics and numerical integration, see appendix 3.A.

Metropolis HMC One can design an MCMC kernel by proposing the *end-point* of a Hamiltonian trajectory in equation 3.3. In practice, a discretised trajectory is obtained by leapfrog integration. Due to numerical errors in the simulation, in order to ensure the kernel π-invariant, the proposal needs to be adjusted by a Metropolis-Hasting step (Metropolis et al., 1953; Neal, 2011). Denoting $\mathcal{N}_d$ as the d-dimensional standard Gaussian, the kernel $Q \sim \mathcal{K}^{\mathrm{MH}}_{\varepsilon,L}(Q_0,\cdot)$ for Metropolis HMC follows

$$P_0 \sim \mathcal{N}_d, \quad (q_L,p_L) = \hat{\Phi}_{\varepsilon,L}(Q_0,P_0),$$
$$Q = \begin{cases} q_L & \text{with prob. } \min\{1,\exp(\Delta_{\mathcal{E}})\} \\ Q_0 & \text{otherwise} \end{cases} \tag{3.4}$$

[2]Unless otherwise specified we let $M^{-1} = I_d$ throughout, though we note that M can be chosen using existing adaption methods, e.g. (Carpenter et al., 2017), or as in Riemannian HMC (Girolami and Calderhead, 2011).

where $\Delta_{\mathcal{E}} := -\mathcal{E}(q_L, p_L) + \mathcal{E}(Q_0, P_0)$ is the energy difference between the origin and the proposal.

Note here (and across this chapter) we use lower case q, p to refer to the states of numerical simulation/trajectory and capital Q, P to refer to MCMC states.

Multinomial HMC Betancourt (2018) describes a trajectory variant of HMC, which we refer as *multinomial* HMC and denote $\mathcal{K}^{\text{Mult}}_{\varepsilon,L}(Q_0, \cdot)$. In multinomial HMC, all intermediate points of a numerical trajectory can be proposed as the next state:

$$P_0 \sim \mathcal{N}_d, \mathbf{t} \sim \mathbb{P}_{\varepsilon,L}(\cdot \mid Q_0, P_0), (Q, P) \sim \mathbb{P}(\cdot \mid \mathbf{t}) \tag{3.5}$$

where $\mathbf{t} := [(q_{-L_\text{b}}, p_{-L_\text{b}}), \ldots, (Q_0, P_0), \ldots, (q_{L_\text{f}}, p_{L_\text{f}})]$. The *trajectory sampling* $\mathbf{t} \sim \mathbb{P}_{\varepsilon,L}(\cdot \mid Q_0, P_0)$ follows

$$\begin{aligned} L_\text{f} &\sim \mathcal{U}(\{0, \ldots, L\}), \quad L_\text{b} = L - L_\text{f}, \\ (q_\ell, p_\ell) &= \begin{cases} \hat{\Phi}_{\varepsilon,\ell}(Q_0, +P_0) & \text{for } \ell = 1, \ldots, L_\text{f} \\ \hat{\Phi}_{\varepsilon,\ell}(Q_0, -P_0) & \text{for } \ell = 1, \ldots, L_\text{b} \end{cases} \end{aligned} \tag{3.6}$$

The *intra-trajectory sampling* $Q, P \sim \mathbb{P}(\cdot \mid \mathbf{t})$ follows a multinomial distribution (Betancourt, 2018) as

$$\mathbb{P}\left((Q, P) = (q_\ell, p_\ell) \mid \mathbf{t}\right) = \sigma\left((q_\ell, p_\ell), \mathbf{t}\right) \tag{3.7}$$

where $\sigma(z, \mathbf{t}) := \exp\left(-\mathcal{E}(z)\right) / \sum_{z' \in \mathbf{t}} \exp\left(-\mathcal{E}(z')\right)$. Note that sampling from equation 3.7 can be equivalently defined by sampling from a categorical distribution followed by an indexing operation as

$$\begin{gathered} i \sim \bar{P}_\ell \\ (q_\ell, p_\ell) = \mathbf{t}_i = (q_i, p_i) \end{gathered}$$

where $\bar{P}_\ell$ is a categorical with probability mass $\text{Cat}(\ell = i) = \sigma(\mathbf{t}_\ell, \mathbf{t})$.

3.2.3 Coupled MCMC kernels

The coupled HMC kernel in (Heng and Jacob, 2019) *and* the coupled kernels proposed in this work can be unified through algorithm 3.2. Specifically, letting (i) In Line 2, $\mathbb{P}_{L_\text{f}}(L_\text{f} = L) = 1$ and (ii) In Line 6, $(i, j) \mid (\mathbf{t}^1, \mathbf{t}^2) \sim \bar{P}_\ell$ in algorithm 3.2, we recover the

Algorithm 3.2 Coupled HMC kernels

```
INPUT: A pair of current states $(Q_0^1, Q_0^2)$ and a HMC kernel $\mathcal{K}_{\varepsilon,L}$ with step size $\varepsilon$ and number $L$
OUTPUT: A pair of next states $(Q^1, Q^2)$
1: Sample $P_0 \sim \mathcal{N}_d$
2: Sample $L_f \sim \mathbb{P}_{L_f}$ and set $L_b = L - L_f$
3: for $c = 1, 2$ do
4:     Use the leapfrog integrator to simulate the trajectory $\mathbf{t}^c = [\hat{\Phi}_{\varepsilon,-L_b}(Q_0^c, -P_0), \dots (Q_0^c, P_0), \dots \hat{\Phi}_{\varepsilon,L_f}(Q_0^c, P_0)]$
5: end for
6: Sample next state indices $(i, j) \mid (\mathbf{t}^1, \mathbf{t}^2) \sim \bar{P}_\ell$
7: Set $(Q^1, P^1) = \mathbf{t}_i^1, (Q^2, P^2) = \mathbf{t}_j^2$
8: Output $(Q^1, Q^2)$
```

$\bar{\mathcal{K}}_{\varepsilon,L}^{\mathrm{MH}}$ from Heng and Jacob (2019), where $\bar{P}_\ell$ follows the generative process

$$u \sim \mathcal{U}([0,1]),$$

$$i = \begin{cases} L & \text{if } u < \alpha^1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad j = \begin{cases} L & \text{if } u < \alpha^2 \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha^c = \exp\left(-\mathcal{E}(\mathbf{t}_L^c) + \mathcal{E}(\mathbf{t}_0^c)\right)$ for $c = 1, 2$. This corresponds to using *common random number* (CRN) in the MH correction steps in equation 3.4. For coupled multinomial HMC kernels studied in this work, which we denote $\bar{\mathcal{K}}_{\varepsilon,L}^{\gamma}$, we make different choices for Line 2 and Line 6 in algorithm 3.2. In short, Line 2 will be a coupled version of equation 3.6 and Line 6 will correspond to a coupling γ of two multinomial distributions as equation 3.7. We will discuss them in detail in section 3.3.

Although coupled HMC kernels can bring two chains within a small neighbourhood of each other, the probability of *exact* meeting is zero, thus failing to satisfy the faithfulness condition for using equation 3.2. To alleviate this issue, Heng and Jacob (2019) instead propose a mixture of coupled random-walk Metropolis-Hastings (RWMH) $\bar{\mathcal{K}}_\sigma$ and coupled HMC $\bar{\mathcal{K}}_{\varepsilon,L}$ to trigger "exact meeting". The coupled RWMH kernel $\bar{\mathcal{K}}_\sigma$ with proposal variance $\sigma^2 I_d$ uses maximal coupling (Johnson, 1998; Jacob et al., 2019b) to encourage two chains meet exactly when they are close. For completeness, we provide it in algorithm 3.3 where we slightly abuse notation, writing $\mathcal{K}_\sigma(X,Y)$ to mean denote the probability *density* of the probability measure $\mathcal{K}_\sigma(X,\cdot)$ evaluated at Y, where X and Y are random variables. Intuitively speaking, at each iteration of the coupled

Algorithm 3.3 Coupled RWMH kernel with maximal coupling (Jacob et al., 2019b)

Input: A pair of current states (X_0, Y_0) and a RWMH kernel $\mathcal{K}_\sigma$ with variance $\sigma^2 I_d$

Output: A pair of next states (X', Y')

```
 1: Sample $X^* \sim \mathcal{K}_\sigma(X_0, \cdot)$
 2: Sample $w \mid X \sim \mathcal{U}([0, \mathcal{K}_\sigma(X_0, X^*)])$
 3: if $w \leq \mathcal{K}_\sigma(Y_0, X^*)$ then
 4:     Set $Y^* = X^*$
 5: else
 6:     repeat
 7:         Sample $Y^* \sim \mathcal{K}_\sigma(Y_0, \cdot)$
 8:         Sample $w^* \mid Y^* \sim \mathcal{U}([0, \mathcal{K}_\sigma(Y_0, Y^*)])$
 9:     until $w^* > \mathcal{K}_\sigma(X_0, Y^*)$
10: end if
11: Sample $u \sim \mathcal{U}([0,1])$
12: Set $X = X_0$ and $Y = Y_0$
13: if $u \leq \min\{1, \pi(X^*)/\pi(X_0)\}$ then
14:     Set $X = X^*$
15: end if
16: if $u \leq \min\{1, \pi(Y^*)/\pi(Y_0)\}$ then
17:     Set $Y = Y^*$
18: end if
19: Output $(X, Y)$
```

RWMH kernel, the kernel is designed as the maximal coupling of the two (conditional) Gaussians—with this we maximise the probability of transiting to a state that the two chains meet. Also note that this ignores the MH correction following the random walk; for a method that considers the maximal coupling of the random walk together with the corrections, see O'Leary et al. (2020).

The overall mixture kernel, denoted $\bar{\mathcal{K}}_{\varepsilon,L,\sigma}$, is then defined as

$$\bar{\mathcal{K}}_{\varepsilon,L,\sigma}\left(\bar{x}, \bar{A}\right) = (1-\alpha)\bar{\mathcal{K}}_{\varepsilon,L}\left(\bar{x}, \bar{A}\right) + \alpha\bar{\mathcal{K}}_{\sigma}\left(\bar{x}, \bar{A}\right) \tag{3.8}$$

for $\alpha \in (0,1)$, $\bar{x} := (x,y) \in \mathbb{R}^d \times \mathbb{R}^d$ and $\bar{A} := (A,B) \in \mathcal{B}(\mathbb{R}^d) \times \mathcal{B}(\mathbb{R}^d)$. That is, with probability α we use the coupled RWMH kernel and with probability $1-\alpha$ we use the HMC kernel. Heng and Jacob (2019) proves that, under certain assumptions, if the *relaxed meeting time* $\tau_\delta := \inf\{n \geq 0 : \|X_n - Y_{n-1}\| \leq \delta\}$ of the coupled HMC kernel

$\bar{\mathcal{K}}_{\varepsilon,L}$ has geometric tails for any $\delta > 0$, the chains meet exactly with non-zero probability under $\bar{\mathcal{K}}_{\varepsilon,L,\sigma}$ for any $\alpha \in (0,1)$, warranting the use of equation 3.2.

The key conditions that ensure the unbiasedness, finite variance and finite computation cost of equation 3.2 are (i) the coupled chains marginally converge to the target and (ii) the two chains meet sufficiently quickly and stay together after meeting; see Jacob et al. (2019b) for explicit definitions. Suppose our proposed HMC kernels satisfy (i) by construction, to ensure the method satisfies (ii) it is sufficient to prove that the relaxed meeting time has geometric tails. This we establish in section 3.4

3.3 Optimal transport couplings for multinomial HMC

Recall that in order to use the multinomial HMC kernel $\mathcal{K}_{\varepsilon,L}^{\text{Mult}}$ in algorithm 3.2, we need to specify how Line 2 and Line 6 are performed. First, the number of leapfrog steps forward and backward sampled in Line 2 follows equation 3.6, inheriting from multinomial HMC, and is shared between the two chains. In other words, both chains simulate forward and backward for the same number of steps, making them "aligned in-time". Second, Line 6 correspond to a coupling of the intra-trajectory multinomial sampling step in equation 3.7. To ensure that the marginal chains are equivalent to the original multinomial kernel, it is sufficient to sample (i, j) such that the corresponding marginal *categorical distributions* $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ of equation 3.7 for indices i, j are preserved

$$\boldsymbol{\mu} : \text{Cat}(\ell = i) = \sigma(\mathbf{t}_\ell^1, \mathbf{t}^1), \quad \boldsymbol{\nu} : \text{Cat}(\ell = j) = \sigma(\mathbf{t}_\ell^2, \mathbf{t}^2)$$

Here we overload the notations $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ also to refer to their corresponding *probability vectors*.

To this end, our method is fully specified by providing an algorithm to sample (i, j) such that $i \sim \boldsymbol{\mu}$ and $j \sim \boldsymbol{\nu}$. The collection of such joint distributions for (i, j) are couplings of $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$, i.e. $\Gamma(\boldsymbol{\mu}, \boldsymbol{\nu})$.

3.3.1 Optimal transport couplings

To repeat, our aim is to construct coupled kernels in which the coupled chains from algorithm 3.1 meet in a relatively small number of MCMC steps, i.e. short meeting time. Unfortunately, it is not clear how to directly minimise the meeting time. Intuitively, one might expect a kernel which, informally, "brings chains closer" to also have an

Algorithm 3.4 Sampling from a discrete joint J

Input: A $M \times N$ matrix J that represents the joint of two categorical distributions

Output: A pair of indices $(i, j) \sim J$

```
1: for $i = 1, \dots, M, j = 1, \dots, N$ do
2:     Compute $k = M(i-1) + j$
3:     Set $u_k = (i, j)$ and $\boldsymbol{\nu}_k = J_{ij}$
4: end for
5: Sample $k \sim \mathrm{Cat}(\boldsymbol{\nu})$
6: Output $u_k$
```

improved meeting time. Naturally this brings us to consider the following problem:

$$\gamma := \arg\min_{\gamma'} \sum_{i,j} \gamma'_{ij} D_{ij} \;\text{ s.t. }\; \gamma' \in \Gamma(\boldsymbol{\mu}, \boldsymbol{\nu}) \tag{3.9}$$

where $D_{ij} = d(\mathbf{t}_i^1, \mathbf{t}_j^2)$ is the distance in the *position space* between the i-th point in the first trajectory and the j-th point in the second. This is an example of a *Kantorovich problem*, a well-studied family of problems from optimal transport (Villani, 2003). In the case where $d_2^p(x,y) = \|x-y\|_2^p$, we will refer to the minimiser as the W_p*-coupling* due to the role it plays in the Wasserstein distance w.r.t. Euclidean metric $W_p(\boldsymbol{\mu}, \boldsymbol{\nu}) = \left(\inf_{\gamma \in \Gamma(\boldsymbol{\mu}, \boldsymbol{\nu})} \mathbb{E}_{(X,Y)\sim\gamma} \|x-y\|_2^p\right)^{1/p}$.

In this work we will consider two different choices for the metric d: 1) Euclidean distance d_2 which gives rise to the W_2*-coupling*, and 2) 0-1 distance d_I which gives rise to the *maximal coupling*.

3.3.2 W_2-coupling

Arguably the most natural choice of metric d in equation 3.9 is the squared Euclidean distance $d_2^2(\mathbf{t}_i^1, \mathbf{t}_j^2) = \left\| q_i^1 - q_j^2 \right\|_2^2$, whose solution we denote γ°. Once we obtain γ°, sampling (i, j) is straightforward. For completeness, we provide an algorithmic description of how to sample a pair of indices given their joint probability matrix in algorithm 3.4.

Computationally, the optimisation in equation 3.9 can be solved by generic linear programming solvers or more specialised methods, e.g. as in Bonneel et al. (2011). Such solvers in general have a time complexity $O(K^3)$ where K is the length of the probability vectors $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$. This can be alleviated by using an approximate solver which could introduce biases. Therefore, similarly to Jacob et al. (2016), we also describe a debiasing method that allows the use of approximate solvers in section 3.3.2.1.

3.3.2.1 Debiasing marginal-non-preserving joints

A side effect of using fixed-point iteration solvers or even approximate solvers (Cuturi, 2013) to solve equation 3.9 is that the solution does not belong to $\Gamma(\boldsymbol{\mu},\boldsymbol{\nu})$. We denote such solutions as J°, which indicates it is a $\underline{\text{j}}$oint probability matrix rather than a proper coupling. Therefore we need a way to ensure that when using J°, we still have $i\sim\boldsymbol{\mu}$ and $j\sim\boldsymbol{\nu}$ exactly, which we refer as a *debiasing* step. Inspired by the mixture view of the maximal coupling, the result of our debiasing algorithm, the debiased W_2-coupling $\hat{\gamma}^{\circ}$, can be as well viewed as a mixture

$$\hat{\gamma}^{\circ}=\alpha J^{\circ}+(1-\alpha)J^{d}$$

where α is the probability of sampling from J°, and J^{d} is the $\underline{\text{d}}$ebiasing joint probability matrix ("d" is underlined to indicate it is the superscript in J^{d}). The algorithm aims to find the maximal probability α such that $\hat{\gamma}^{\circ}\in\Gamma(\boldsymbol{\mu},\boldsymbol{\nu})$, together with the corresponding debiasing matrix J^{d}. First, to find the maximal α, we see that $\hat{\gamma}^{\circ}\in\Gamma(\boldsymbol{\mu},\boldsymbol{\nu})$ implies

$$\boldsymbol{\mu}=\alpha\boldsymbol{\mu}^{\circ}+(1-\alpha)\boldsymbol{\mu}^{d},\quad \boldsymbol{\nu}=\alpha\boldsymbol{\nu}^{\circ}+(1-\alpha)\boldsymbol{\nu}^{d} \tag{3.10}$$

where $\boldsymbol{\mu}^{\circ}$ and $\boldsymbol{\nu}^{\circ}$ are marginals of J° and $\boldsymbol{\mu}^{d}$ and $\boldsymbol{\nu}^{d}$ are marginals of J^{d}. Since $\boldsymbol{\mu}^{d}$ and $\boldsymbol{\nu}^{d}$ are K-length probability vectors, we have $\mu_i^{d}>0$ and $\nu_i^{d}>0$ for all $i=1,\dots,K$, which implies a set of constrains on α

$$\mu_i\geq\alpha\mu_i^{\circ},\quad\text{and}\quad \nu_i\geq\alpha\nu_i^{\circ}\quad\text{for all } i=1,\dots,K$$

Therefore, the maximal value of α is given by

$$\alpha=\min\{1,\frac{\mu_1}{\mu_1^{\circ}},\dots,\frac{\mu_K}{\mu_K^{\circ}},\frac{\nu_1}{\nu_1^{\circ}},\dots,\frac{\nu_K}{\nu_K^{\circ}}\}. \tag{3.11}$$

With α found, we can solve equation 3.10 to find $\boldsymbol{\mu}^{d}$ and $\boldsymbol{\nu}^{d}$, and J^{d} can be chosen as any coupling of them, i.e. $J^{d}\in\Gamma(\boldsymbol{\mu}^{d},\boldsymbol{\nu}^{d})$, including the *independent coupling* that simply samples as $i\sim\boldsymbol{\mu}^{d}, j\sim\boldsymbol{\nu}^{d}$. We summarise in algorithm 3.5 a sampling procedure of $\hat{\gamma}^{\circ}$ resulting from this debiasing approach. It is not hard to see that by construction, the approach satisfies equation 3.10 and yields $\hat{\gamma}^{\circ}\in\Gamma(\boldsymbol{\mu},\boldsymbol{\nu})$, which, as a result, yields a coupled HMC kernel whose marginal kernels converge to the target. Also, when there is no bias, i.e. $J^{\circ}\in\Gamma(\boldsymbol{\mu},\boldsymbol{\nu})$, we have $\alpha=1$ from equation 3.11 and the algorithm reduces to exact W_2-coupling.

Algorithm 3.5 Maximally sampling from a joint $\hat{\gamma}$ that is not a coupling while ensuring marginals to be $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$

Input: A $K \times K$ probability matrix $\hat{\gamma}$ and two K-length probability vectors $\boldsymbol{\mu}, \boldsymbol{\nu}$ to target

Output: A pair of indices (i, j) with $i \sim \boldsymbol{\mu}$ and $j \sim \boldsymbol{\nu}$ while maximally using $\hat{\gamma}$

```
 1: Compute $\boldsymbol{\mu}^\circ$ and $\boldsymbol{\nu}^\circ$ as marginals of $\hat{\gamma}$
 2: Compute $\alpha$ according to equation 3.11
 3: Sample $U \sim \mathcal{U}([0,1])$
 4: if $U < \alpha$ then
 5:     Sample $(i, j) \sim \hat{\gamma}$ using algorithm 3.4
 6: else
 7:     Compute $\boldsymbol{\mu}^d$ and $\boldsymbol{\nu}^d$ by solving equation 3.10
 8:     Sample $i \sim \boldsymbol{\mu}^d$ and $j \sim \boldsymbol{\nu}^d$
 9: end if
10: Output $(i, j)$
```

3.3.3 Maximal coupling

For general choices of d, we do not have analytical solutions for equation 3.9, but for the particular choice $d_I(\mathbf{t}_i^1, \mathbf{t}_j^2) = \mathbb{1}(i \neq j)$ we do. In this case, the solution is the well-known maximal coupling γ^* of two categorical distributions—the coupling that maximising the probability that the two categoricals having the same index, which can be represented in its mixture view as

$$\gamma^* = \omega \frac{\boldsymbol{\mu} \wedge \boldsymbol{\nu}}{Z} + (1 - \omega) \frac{\boldsymbol{\mu} - (\boldsymbol{\mu} \wedge \boldsymbol{\nu}) + \boldsymbol{\nu} - (\boldsymbol{\mu} \wedge \boldsymbol{\nu})}{1 - Z} \tag{3.12}$$

where $\wedge$ is the point-wise minimum operation, $\omega = \mathbb{P}(i = j) = 1 - \mathrm{D}_{\mathrm{TV}}(\boldsymbol{\mu}, \boldsymbol{\nu})$ (where D_{TV} is the total variation distance) and $Z = \sum_i (\boldsymbol{\mu} \wedge \boldsymbol{\nu})_i$. Here in equation 3.12 the first term/mixture is trying to putting as much probability on letting $i = j$ happen as possible, and the second term/mixture serves as a debiasing term to preserve the marginals. As it is possible to compute the joint probability matrix using equation 3.12, sampling from γ^* is tractable and straightforward. For completeness, we provide an algorithmic description of how to sample a pair if indices from the maximal coupling of two categorical distribution in algorithm 3.6.

By definition, for a maximal coupling γ^*, the probability of choosing pairs with the same time-index in two trajectories is maximised; we refer to such pairs with same indices as "index-aligned" pairs. As we will see in section 3.4, this property allows us

Algorithm 3.6 Maximal coupling of $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$

Input: Two categorical distributions $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ **Output**: A pair of indices $(i,j) \sim \gamma^*$

```
1: Compute $\omega = 1 - \mathrm{D_{TV}}(\boldsymbol{\mu},\boldsymbol{\nu})$ and $Z = \sum_i (\boldsymbol{\mu} \wedge \boldsymbol{\nu})_i$
2: Sample $u \sim \mathcal{U}([0,1])$
3: if $u \leq \omega$ then
4:     Sample $i \sim \mathrm{Cat}(\frac{\boldsymbol{\mu} \wedge \boldsymbol{\nu}}{Z})$ and set $j = i$
5: else
6:     Sample $i \sim \mathrm{Cat}(\frac{\boldsymbol{\mu} - (\boldsymbol{\mu} \wedge \boldsymbol{\nu})}{1-Z})$, $j \sim \mathrm{Cat}(\frac{\boldsymbol{\nu} - (\boldsymbol{\mu} \wedge \boldsymbol{\nu})}{1-Z})$
7: end if
8: Output $(i,j)$
```

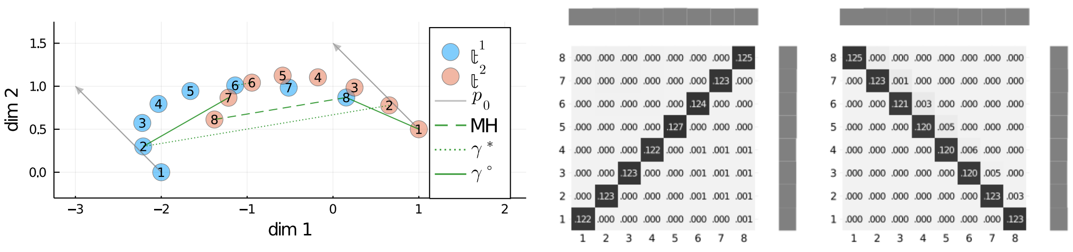
to exploit Lemma 1 in Heng and Jacob (2019) to show that the distance between the two coupled chains decreases with non-zero probability when the potential is strongly convex, or, equivalently, the target is strongly log-concave.

Though the idea of index-aligned pairs is useful to establish the theoretical results, it is not necessarily so in practice. Note that as the approximation of Hamiltonian simulation by numerical integration becomes more accurate when step sizes become smaller, the joint γ^* converges to the diagonal uniform distribution, i.e. $\gamma_{ii} \approx 1/K$ and $\gamma_{ij} \approx 0$ for $i \neq j$ for large K. It is easy to construct examples where this leads to sub-optimal behaviour when the goal is to minimise distance between the proposed states; figure 3.1 illustrates this nicely.

Computationally, maximal couplings in algorithm 3.6 are much cheaper than OT couplings as the maximal couplings for categoricals are easy to compute.

3.3.4 An illustration of different couplings

We now illustrate how different intra-trajectory couplings behave using a 2D Gaussian with zero mean and unit diagonal covariance. We start by simulating two Hamiltonian trajectories from $q_0^1 = [0.5, 2.0]$ and $q_0^2 = [0.5, -1.0]$ using the same momentum $p_0 = [1.0, 1.0]$ for 7 steps, obtaining two trajectories $\mathbf{t}^1$ and $\mathbf{t}^2$ in figure 3.1a, where the arrows represent the initial momentum p_0. We then sample from our two couplings to generate $100,000$ pairs of indices to estimate the joint distributions and to compute the marginals, which are shown in figure 3.1b and figure 3.1c. Note how the joint distributions differs: the ordering of the pairings are "reversed". This intuitively makes sense when looking at figure 3.1a, in which, e.g. the closest point for $\mathbf{t}_1^1$ (state 1 in the first (blue) trajectory)



(a) Coupled trajectories in 2D (b) Maximal coupling (c) W_2-coupling

Figure 3.1: An illustration of different HMC couplings: trajectories of simulated coupled chains (figure 3.1a) and their joints (figure 3.1b and figure 3.1c with marginals on top and right sides). Green lines in figure 3.1a indicate possible pairs from different methods. For coupled Metropolis HMC, the dashed line pairs the end-points of two trajectories, which has a relative large distance. The dotted line is for multinomial HMC with maximal coupling. Though there is a change that the 6-th points of two trajectories are paired, other index-aligned pairs are equally likely (figure 3.1b), e.g. the pair of 2-th points has a large distance. In contrast, all pairs from multinomial HMC with W_2-coupling (solid lines) have relatively small distances, resulting in a small distance on average. To see this, we calculate the expected distances: they are 1.37 for W_2-coupling and 1.97 for maximal coupling, where the former is clearly smaller, as expected. Note that the marginals in figure 3.1b and figure 3.1c are close to uniform because the quantisation error is small in this example.

is $\mathbf{t}_8^2$ (state 8 inn the second (red) trajectory). Finally, note that this U-turn example is chosen to highlight the differences between the two couplings. If there was no U-turn, the differences between the two couplings could potentially be smaller.

3.4 Theoretical analysis

We now establish geometric tails for the meeting time for the mixture kernel in equation 3.8 with the proposed coupled HMC kernels as the HMC component, thus satisfying the necessary conditions to use the estimator equation 3.2.

Proof sketch To prove geometric tails it turns out that it is sufficient to prove that the methods satisfy the conditions for Proposition 1 in Heng and Jacob (2019). Informally, the proposition states that once the chains enter a region S in the state space where the target density is strongly log-concave, there is a non-zero probability that the chains will end up in a δ-neighbourhood of each other in some $n_0 \in \mathbb{N}$ steps. The proof presented in Heng and Jacob (2019) obtains this statement for the coupled Metropolis HMC by arguing directly about the probabilities of such an event conditioned on the initial states being in S. Here we instead prove a slightly stronger property, *local contractivity*, from which the proposition follows immediately. Informally, local contractivity ensures that the distance between the chains will decrease on average when initialised in some region. We first prove that this holds for the maximal coupling by exploiting the fact that it maximises the probability of picking index-aligned pairs, which, as mentioned before, is guaranteed to decrease the distance compared to the initial positions for strongly log-concave targets. Once this has been established, local contractivity for the W_2-coupling follows immediately since by definition W_2-coupling has a smaller expected distance than the maximal coupling. The remainder of the proof is essentially identical to Heng and Jacob (2019) where excursions from the set S is controlled with a geometric drift condition, from which we obtain geometric tails for the meeting time and thus validity of the methods.

Following Heng and Jacob (2019), we make two assumptions on the potential function $U : \mathbb{R}^d \mapsto \mathbb{R}$.

Assumption 4 (Regularity and growth of potential)**.** The potential U is twice continuously differentiable and its gradient $\nabla U : \mathbb{R}^d \mapsto \mathbb{R}^d$ is globally β-Lipschitz, i.e. there exists $\beta > 0$ such that $\|\nabla U(q) - \nabla U(q')\| \leq \beta \|q - q'\|$ for all $q, q' \in \mathbb{R}^d$.

Assumption 5 (Local strong convexity of potential)**.** There exists a compact set $S \in \mathcal{B}(\mathbb{R}^d)$, with positive Lebesgue measure, s.t. the restriction of the potential U to S is α-strongly convex, i.e., $\exists\, \alpha > 0$ s.t. $(q - q')^\top (\nabla U(q) - \nabla U(q')) \geq \alpha \|q - q'\|^2$ for all $q, q' \in S$.

Unless otherwise specified, we will let S denote the set in Assumption 5, $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$ denote a coupled HMC kernel as described in algorithm 3.2 with (i) shared momentum, (ii) shared forward and backward simulation steps and (iii) $(i, j) \sim \gamma$ for intra-trajectory sampling, and $\mathrm{pr}^{\gamma}_{\varepsilon,L}$ denote the law of a coupled HMC kernel $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$. For functions $f : \mathbb{R}^d \to \mathbb{R}$, we will also use the notation $L_\ell(f) = \left\{x \in \mathbb{R}^d : f(x) \leq \ell\right\}$ for the level sets of f and $f_A := f\big|_A$ for the restriction of f to $A \in \mathcal{B}(\mathbb{R}^d)$.

3.4.1 Geometric tails via local contractivity

We first state the definition of local contractivity and Proposition 1 from Heng and Jacob (2019).

Condition 1 (Local contractivity)**.** Given a compact set $S \in \mathcal{B}(\mathbb{R})$ with positive Lebesgue measure, we say the kernel $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$ is *locally contractive* on S with rate $\rho \in (0,1)$ if there exists $m \geq 1$ such that for any given $k_0 > 0$ there exists $\bar{\varepsilon} > 0$, $\bar{L} \in \mathbb{N}$ s.t.

$$\mathbb{E}_{(l_1,l_2)\sim\gamma}\left\|\hat{\Phi}^{\circ}_{\varepsilon,l_1}(q^1,p) - \hat{\Phi}^{\circ}_{\varepsilon,l_2}(q^2,p)\right\|^m \leq \rho^m \left\|q^1 - q^2\right\|^m \tag{3.13}$$

for all $\varepsilon \in (0,\bar{\varepsilon})$, $L \in \mathbb{N}$ such that $\varepsilon L < \bar{\varepsilon}\bar{L}$ and for all $(q^1,q^2,p) \in S \times S \times L_{k_0}(K)$.

Informally, this is saying that there exists a step size and integration time such that a single application of $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$ decreases the distance between the two states on average. Furthermore, this property is preserved when decreasing the step size or the integration time. This last part is important since different parts of the analysis will require possibly smaller step sizes and/or integration times. Thus, by ensuring that all statements hold for all smaller step sizes and integration times, we can combine the statements by simply choosing the minimum of the step sizes and/or integration times required by the different statements.

Proposition 3.4.1 (Proposition 1, Heng and Jacob (2019))**.** Suppose that the potential U satisfies Assumptions 4 and 5. Then for any $\delta > 0$, $u_0 > \inf_{q\in S} U(q)$, and $u_1 < \sup_{q\in S} U(q)$ with $u_0 < u_1$, there exists $\bar{\varepsilon} > 0$ and $\bar{L} \in \mathbb{N}$ such that for any $\varepsilon \in (0,\bar{\varepsilon})$ and $L \in \mathbb{N}$ satisfying $\varepsilon L < \bar{\varepsilon}\bar{L}$, there exists $v_0 \in (u_0,u_1)$, $n_0 \in \mathbb{N}$ and $\omega \in (0,1)$ such that

$$\inf_{q^1,q^2\in S_0} \bar{\mathcal{K}}^{\gamma,n_0}_{\varepsilon,L}\left((q^1,q^2),D_\delta\right) \geq \omega \tag{3.14}$$

where $S_0 = L_{v_0}(U_S)$ is compact with positive Lebesgue measure,

$$\bar{\mathcal{K}}^{\gamma,n}_{\varepsilon,L}\left((q^1,q^2),A^1\times A^2\right) = \mathrm{pr}^{\gamma}_{\varepsilon,L}\left((Q^1_n,Q^2_n)\in A^1\times A^2 \mid (Q^1_0,Q^2_0) = (q^1,q^2)\right)$$

denotes the n-step transition probabilities of the coupled chain, and $D_\delta = \left\{(q,q') \in \mathbb{R}^d \times \mathbb{R}^d : \|q-q'\| \leq \delta\right\}$.

As noted earlier, this proposition is a key step in the proof of the coupled Metropolis HMC kernel in Heng and Jacob (2019). This ensures that once we reach the set $S_0 \subset S$, there is a non-zero probability that within some finite number of steps the chains will be δ-close, i.e. meet in the relaxed sense. If we can then also ensure that this set S_0 will be entered by the coupled chains sufficiently often, then we bound the tails of distribution over meeting times.

We now establish that indeed, for coupled multinomial HMC kernels, Condition 1 implies Proposition 3.4.1.

Lemma 3.4.1. If $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$ satisfies Condition 1, then $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$ satisfies the conditions of Proposition 3.4.1.

Proof. Observe that

$$\begin{aligned}
&\mathrm{pr}_{\varepsilon,L}\left(\|Q_1^1 - Q_1^2\| \leq \rho \|Q_0^1 - Q_0^2\| \mid (Q_0^1, Q_0^2) = (q^1, q^2)\right) \\
&= \mathbb{E}_{\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}}[\mathbb{1}\left\{\|Q_1^1 - Q_1^2\| \leq \rho \|q^1 - q^2\|\right\}] \\
&= \mathbb{E}_{P\sim\mathcal{N}(0,I)}[\mathbb{E}_{(l_1,l_2)\sim\gamma}[\mathbb{1}(R_{q^1,q^2,P}) \mid P]]
\end{aligned}$$

where we have let $R_{q^1,q^2,p}$ denote the set of events where we have contraction, i.e.

$$R_{q^1,q^2,p} = \left\{\|\hat{\Phi}^{\circ}_{\varepsilon,l_1}(q^1,p) - \hat{\Phi}^{\circ}_{\varepsilon,l_2}(q^2,p)\| \leq \rho \|q^1 - q^2\|\right\}$$

By Condition 1, $\mathbb{E}[(l_1,l_2)\sim\gamma]\mathbb{1}(R_{q^1,q^2,p}) > 0$ for any $(q^1,q^2,p) \in S \times S \times L_{k_0}(K)$, where $k_0 > 0$ is to be decided, since otherwise equation 3.13 would not hold. Hence a single application of the kernel $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$ will have a non-zero probability of decreasing the distance between the states if we are in S. The remainder of the proof ensures that parameters can be chosen such that there is a non-zero probability of staying within the set $S_0 \subset S$ for some $n_0 := \inf\left\{n \in \mathbb{N} : \rho^n \|q^1 - q^2\| \leq \delta\right\}$ applications of the kernel, i.e. $(Q_k^1, Q_k^2) \in S_0 \times S_0$ for all $k = 1, \ldots, n_0$. This finally allows us to conclude that there is a non-zero probability of entering D_δ if we are currently in the set S_0. See the appendix 3.B.1 for the full proof. □

Theorem 3.4.1 (Theorem 2, Heng and Jacob (2019))**.** Suppose that the potential U satisfies Assumptions 4 and 5. Suppose that there exists $\bar{\varepsilon} > 0$ and $\bar{\sigma} > 0$ such that for any $\varepsilon \in (0,\bar{\varepsilon})$, $L \in \mathbb{N}$ and $\sigma \in (0,\bar{\sigma})$, there exists a measurable function $V : \mathbb{R}^d \to [1,\infty)$, $\lambda \in (0,1)$, $b < \infty$ and $\mu > 0$ such that

$$\mathcal{K}_{\varepsilon,L}(V)(x) \leq \lambda V(x) + b, \quad Q_\sigma(V)(x) \leq \mu(V(x)+1)$$

for all $x \in \mathbb{R}^d$, $\pi_0(V) < \infty$, $\lambda_0 = (1-\gamma)\lambda + \gamma(1+\mu) < 1$ and $\left\{x \in \mathbb{R}^d : V(x) \leq \ell_1\right\} \subseteq \{x \in S : U(x) \leq \ell_0\}$, for some $\ell_0 \in \{\inf_{x\in S} U(x), \sup_{x\in S} U(x)\}$ and $\ell_1 > 1$ satisfying $\lambda_0 + 2\left((1-\gamma)b + \gamma\mu\right)(1-\lambda_0)^{-1}(1+\ell_1)^{-1} < 1$. Then there exists $\varepsilon_0 \in (0,\bar{\varepsilon})$, $L_0 \in \mathbb{N}$ and $\sigma_0 > 0$ such that for any $\varepsilon \in (0,\varepsilon_0)$, $L \in \mathbb{N}$ satisfying $\varepsilon L < \varepsilon_0 L_0$ and $\sigma \in (0,\sigma_0)$, we have

$$\mathrm{pr}^{\gamma}_{\varepsilon,L,\sigma}(\tau > n) \leq C_0 \kappa_0^n$$

for some $C_0 \in \mathbb{R}_+$ and $\kappa_0 \in (0,1)$ and for $n \in \mathbb{N}_0$, where $\mathrm{pr}^{\gamma}_{\varepsilon,L,\sigma}$ denotes the law of the kernel $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L,\sigma}$ for a given coupled HMC kernel $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$.

Proof. The proof is identical to Heng and Jacob (2019) via Lemma 3.4.1. □

3.4.2 Local contractivity for W_2-coupling and maximal coupling

Now we establish Condition 1 for coupled multinomial HMC kernels with maximal coupling γ^* and W_2-coupling γ°, ensuring that Theorem 3.4.1 applies to the resulting mixture kernels $\bar{\mathcal{K}}^*_{\varepsilon,L,\sigma}$ and $\bar{\mathcal{K}}^\circ_{\varepsilon,L,\sigma}$.

We first restate a slight variation of Lemma 1 from Heng and Jacob (2019), which tells us that the states reached by *exact* flows with shared momentum is closer than the initial states for sufficiently small integration times.

Lemma 3.4.2. Suppose that the potential U satisfies Assumptions 4 and 5. For any compact set $A \subset S \times S \times \mathbb{R}^d$, there exists a trajectory length $T > 0$ such that

$$\left\|\Phi^\circ_t(q^1,p) - \Phi^\circ_t(q^2,p)\right\| \leq \rho\left\|q^1 - q^2\right\| \tag{3.15}$$

for all $t \in [-T,T] \setminus \{0\}$ and all $(q^1,q^2,p) \in A$.

Proof. See appendix 3.B.2 for the detailed proof. □

Note that Lemma 3.4.2 is a statement about the distance between the integrated states *at the same integration time t*. As an immediate consequence the expected distance with respect to a joint distribution with probability mass only along the diagonals satisfies a similar property, controlling for numerical errors (see appendix 3.B.3). Therefore, our strategy in proving local contractivity for $\bar{\mathcal{K}}^*_{\varepsilon,l}$ and $\bar{\mathcal{K}}^\circ_{\varepsilon,l}$ is to ensure that as we decrease the stepsize the probability mass on the diagonals, i.e. $\mathbb{P}(i = j)$, can be made close to 1.

Maximal coupling To establish Condition 1 for coupled multinomial HMC with maximal coupling γ^*, $\bar{\mathcal{K}}^*_{\varepsilon,l}$, we first introduce a bound on the total variation distance between the trajectory distributions $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$.

Proposition 3.4.2. Suppose that U satisfies Assumptions 4 and 5. For any $\delta > 0$, there exists $\varepsilon_0 > 0$, $L_0 \in \mathbb{N}$ s.t. for all $\varepsilon \in (0,\varepsilon_0)$, $L \in \mathbb{N}$ satisfying $\varepsilon L < \varepsilon_0 L_0$, we have

$$\mathrm{D_{TV}}(\boldsymbol{\mu}\|\boldsymbol{\nu}) = \mathbb{P}(i \neq j) < \delta. \tag{3.16}$$

Proof. The proof uses the 1-Lipschitz property of the softmax function (Gao and Pavel, 2018) in equation 3.7 to bound the probability differences introduced by numerical errors. See appendix 3.B.4. □

With Proposition 3.4.2, we can establish local contractivity for coupled multinomial HMC kernels with γ^*.

Lemma 3.4.3. $\bar{\mathcal{K}}^*_{\varepsilon,l}$ satisfies Condition 1.

Proof. Due to Proposition 3.4.2, for a given integration time, we can choose step size arbitrary small to increase probability of picking parallel-in-time pairs, whose contractivity is established in Proposition 3.B.1. See appendix 3.B.5 for the complete proof. □

W_2-**coupling** Similarly to Lemma 3.4.3, for coupled multinomial HMC with γ°, $\bar{\mathcal{K}}^\circ_{\varepsilon,l}$, we have:

Lemma 3.4.4. $\bar{\mathcal{K}}^\circ_{\varepsilon,l}$ satisfies Condition 1.

Proof. The definition of γ° from equation 3.9 implies

$$\begin{aligned}\mathbb{E}_{(i,j)\sim\gamma^\circ}\left\|\hat{\Phi}^\circ_{\varepsilon,l_i}(q^1,p)-\hat{\Phi}^\circ_{\varepsilon,l_j}(q^2,p)\right\|^2 \leq \\ \mathbb{E}_{(i,j)\sim\gamma^*}\left\|\hat{\Phi}^\circ_{\varepsilon,l_i}(q^1,p)-\hat{\Phi}^\circ_{\varepsilon,l_j}(q^2,p)\right\|^2\end{aligned} \tag{3.17}$$

The rest of the proof follows that of Lemma 3.4.3. □

Lemmas 3.4.3 and 3.4.4 together with Theorem 3.4.1 then establishes geometric tails for the meeting time of the resulting mixture kernels $\bar{\mathcal{K}}^*_{\varepsilon,l}$ and $\bar{\mathcal{K}}^\circ_{\varepsilon,l}$, respectively.

3.4.3 A simplified proof for relaxed meeting time

Previous parts of this section establish a proof of the geometric tails of exact meeting time by modifying Heng and Jacob (2019). The proof is complete but can be involved due to the complexity of the proof strategy taken by Heng and Jacob (2019). Here I present an alternative, simplified proof for the most important part of the property we want to establish—geometric tails of the relaxed meeting time. Note that although this proof, on its current form, cannot show the geometric tails for the exact meeting time, it is simpler and gives intuitive on how the (relaxed) meeting time is controlled by different conditions.

A proof strategy for geometric tails This specific proof relies on a proof strategy that is commonly used to deal with the drift condition of MCMC algorithms (Jacob, 2020). I summarise it as the theorem below.

Theorem 3.4.2. If there exist a set $C \subseteq \mathcal{X}$ where $\mathcal{X}$ is the space that the Markov kernel $\mathcal{K}$ defined on, a parameter $\lambda \in (0,1)$ and a function $V : C \mapsto [1,\infty]$ satisfying

$$\mathbb{E}_{\mathcal{K}}\{V(x_{t+1}) \mid x_t\} \leq \lambda V(x_t)$$

for all $x_t \notin C$. Then let $\tau_C := \inf\{t \geq 0 : x_t \in C\}$, we have

$$\mathrm{pr}(\tau_C > t) \leq a\lambda^t$$

for some constant $a < \infty$.

Note that I will expand the proof of this theorem when it is used in context.

Proof sketch With Theorem 3.4.2 available, the job left is to show that such a set C and function V exist for the *coupled* kernel we are interested in. This will rely on the local contractivity we have shown for the two coupled kernels (Lemma 3.4.3 and Lemma 3.4.4) as well as a similar assumption on the existence of a global Lyapunov function that Theorem 3.4.1 also assumes.

3.4.3.1 Geometric tails of relaxed meeting time

I now establish a modified version of Theorem 1 from (Heng and Jacob, 2019) that is more suitable to our problem by relating the theorem and proof directly with contractivity instead of the probability of relaxed meeting which is established in (Proposition 1, Heng and Jacob, 2019).

Remark. Difference and similarity. In (Heng and Jacob, 2019), they first establish the probability of n-step transition probability to relaxed meeting in (Proposition 1, Heng and Jacob, 2019) and use it in the proof of (Theorem 1, Heng and Jacob, 2019), which is *exactly* what has been done in (Proposition 3.4, Jacob et al., 2019b) as for generic Metropolis–Hastings. The difference here is that I want to replace this n-step transition probability with a contractivity condition. The motivation here is to relate the theorem and proof directly to the contractivity rate, which is more suitable to our case because of how the method is related to OT.[3]

Theorem 3.4.3. Suppose that there exists $\varepsilon_0 > 0$ and $L_0 \in \mathbb{N}$ such that for any $\varepsilon \in (0, \varepsilon_0)$ and $L \in \mathbb{N}$ satisfying $\varepsilon L < \varepsilon_0 L_0$ such that the kernel $\mathcal{K}_{\varepsilon,L}$ satisfies

[3] I believe that (Heng and Jacob, 2019) can also state something similar here. However, the end-point HMC itself they work with is very related to MH. Therefore, adapting the proof of (Proposition 3.4, Jacob et al., 2019b) makes a lot of sense to them.

1. For the coupled Markov kernel $\bar{\mathcal{K}}$ and the local contractivity set $S \in \mathcal{X}$, there exists a contractivity rate $\lambda \in (0,1)$ such that for $(x,y) \in S \times S$
$$\mathbb{E}_{\bar{\mathcal{K}}}\{|x'-y'| \mid x,y\} \leq \lambda|x-y|, \tag{3.18}$$
where λ is the contractivity rate.

2. For the Markov kernel $\mathcal{K}$, there exists a measurable function $V : \mathcal{X} \to [1,\infty)$, $\lambda \in (0,1), b < \infty$ such that for all $x \in \mathcal{X}$ and $\mathbb{E}_{x\sim\pi_0}\{V(x)\} < \infty$,
$$\mathbb{E}_{\mathcal{K}}\{V(y) \mid x\} \leq \lambda V(x) + b. \tag{3.19}$$
and $\{q \in \mathbb{R}^d : V(q) \leq \ell_1\} \subseteq \{q \in S : U(q) \leq \ell_0\}$ for some $\ell_0 \in \{\inf_{q\in S} U(q), \sup_{q\in S} U(q)\}$ and $\ell_1 > 1$ satisfying $\lambda + 2b(1+\ell_1)^{-1} < 1$.

Then for any $\delta > 0$ with $\varepsilon \in (0,\varepsilon_0)$ and $L \in \mathbb{N}$ satisfying $\varepsilon L < \varepsilon_0 L_0$,
$$\mathrm{pr}_{\varepsilon,L}(\tau_\delta > n) \leq C_0 \kappa^n(\lambda) \quad (\forall n \in \mathbb{N}) \tag{3.20}$$
for some $C_0 \in \mathbb{R}_+$ and $\kappa(\lambda) \in (0,1)$. In particular, $\kappa(\lambda)$ is monotonically increasing.

Remark. Theorem3.4.3 is consistent with (Theorem 1, Heng and Jacob, 2019) but replaces (Proposition 1, Heng and Jacob, 2019) with a local contractivity condition and makes this condition explicit as an assumption. For the drift condition, the reason it looks more complex than the drift condition in (Proposition 3.4, Jacob et al., 2019b) is because of the interaction of the local convexity set S. Besides, the condition on ℓ_1 in the end of the drift condition is less stricter compared to that of (Heng and Jacob, 2019), which is $\lambda + 2b(1-\lambda)^{-1}(1+\ell_1)^{-1} < 1$.

Proof. We first establish the excursion set from the drift condition, following the proof of (Theorem 1, Heng and Jacob, 2019). For any $\varepsilon \in (0,\varepsilon_0)$ and $L \in \mathbb{N}$ satisfying $\varepsilon L < \varepsilon_0 L_0$, it follows from the second assumption that the coupled transition kernel $\bar{\mathcal{K}}_{\varepsilon,L}$ satisfies the geometric drift condition
$$\mathbb{E}_{\bar{\mathcal{K}}_{\varepsilon,L}}\{\bar{V}_1(x',y') \mid x,y\} \leq \lambda_0 \bar{V}_1(x,y) + b$$
for all $x',y' \in \mathbb{R}^d$ with $\bar{V}_1(x,y) = \{V(x)+V(y)\}/2$ as the bivariate Lyapunov function. Define level sets $L_\ell(f) := \{x \in \Omega : f(x) \leq \ell\}$, we have for $(x,y) \notin L_{\ell_0}(U_S) \times L_{\ell_0}(U_S)$ which implies $(x,y) \notin L_{\ell_1}(V) \times L_{\ell_1}(V)$, we have $\bar{V}_1(x,y) \geq (1+\ell_1)/2$. Hence
$$\mathbb{E}_{\bar{\mathcal{K}}_{\varepsilon,L}}\{\bar{V}_1(x',y') \mid x,y\} \leq \lambda_1 \bar{V}_1(x,y) \tag{3.21}$$

with $\lambda_1 = \lambda_0 + 2b(1+\ell_1)^{-1} < 1$ for all $(x,y) \notin L_{\ell_0}(U_S) \times L_{\ell_0}(U_S) := \bar{C}$.

We then look at the excursion set from the local contractivity condition. For any $\varepsilon \in (0, \varepsilon_0)$ and $L \in \mathbb{N}$ satisfying $\varepsilon L < \varepsilon_0 L_0$, it follows local contractivity assumption that the coupled trainsition kernel $\bar{\mathcal{K}}_{\varepsilon,L}$ satisfies

$$\mathbb{E}_{\bar{\mathcal{K}}_{\varepsilon,L}}\{|x'-y'| \mid x,y\} \leq \lambda_2 |x-y|$$

for all $(x,y) \in S \times S$. Define $D_\delta = \{(x,y) \in \mathbb{R}^d \times \mathbb{R}^d : |x-y| < \delta\}$ and $\bar{V}_\delta(x,y) = |x-y|/\delta$, we have

$$\mathbb{E}_{\bar{\mathcal{K}}_{\varepsilon,L}}\{\bar{V}_\delta(x',y') \mid x,y\} \leq \lambda_2 \bar{V}_\delta(x,y)$$

for which $\bar{V}_\delta(x',y') \in [1,\infty)$ for all $(x,y) \in \{(x,y) \in S \times S : (x,y) \notin D_\delta\}$.

Now with $\lambda := \max(\lambda_1, \lambda_2)$ we have

$$\mathbb{E}_{\bar{\mathcal{K}}_{\varepsilon,L}}\{\bar{V}(x',y') \mid x,y\} \leq \lambda \bar{V}(x,y) \tag{3.22}$$

with

$$\bar{V}(x,y) := \begin{Bmatrix} \bar{V}(x,y) & (x,y) \notin S \times S \\ \bar{V}_\delta(x,y) & (x,y) \in S \times S \end{Bmatrix} \geq 1 \tag{3.23}$$

for all $(x,y) \notin \bar{C}_\delta$ where $\{\bar{C}_\delta := L_{\ell_0}(U_S) \times L_{\ell_0}(U_S)\} \cap \bar{D}_\delta \subset \bar{D}_\delta$. Now we can use the "standard" proof strategy for drift condition. For completeness, it is detailed below.

First define $\tau_C = \inf\{t \geq 0 : (x_t, y_t) \in C\}$. As $\bar{C}_\delta \subseteq D_\delta$, $\tau_{D_\delta} \leq \tau_{\bar{C}_\delta}$, which means $\mathrm{pr}_{\varepsilon,L}\{\tau_{D_\delta} > t\} \leq \mathrm{pr}_{\varepsilon,L}\{\tau_{\bar{C}_\delta} > t\}$. We now consider establish geometric tails for the latter. Notice that

$$\begin{aligned}
\lambda^{-t}\mathrm{pr}_{\varepsilon,L}\{\tau_{\bar{C}_\delta} > t\} &= \lambda^{-t}\sum_{s>t}\mathrm{pr}_{\varepsilon,L}\{\tau_{\bar{C}_\delta} = s\} \\
&\leq \sum_{s>t}\lambda^{-s}\mathrm{pr}_{\varepsilon,L}\{\tau_{\bar{C}_\delta} = s\} \\
&= \mathbb{E}_{\bar{q}}\{\lambda^{-\tau_{\bar{C}_\delta}}\mathbb{1}(\tau_{\bar{C}_\delta} > t)\} =: \mathbb{E}_{\bar{q}}\{M_t\} \\
&\leq \mathbb{E}_{\bar{q}}\{\lambda^{-\tau_{\bar{C}_\delta}}\} =: \mathbb{E}_{\bar{q}}\{M\}
\end{aligned}$$

where $\bar{q} := (x,y)$, $\mathbb{E}_{\bar{q}}$ is expectation over the coupeld initial state $\bar{\pi}_0$ and the first inequality comes from the fact that $\lambda^{-s} > \lambda^{-t}$ due to $s > t$. Equivalently, we have

$$\mathrm{pr}_{\varepsilon,L}\{\tau_{\bar{C}_\delta} > t\} \leq \mathbb{E}_{\bar{q}}\{M\}\lambda^t. \tag{3.24}$$

In order to complete our proof, the last step is to show that $\mathbb{E}_{\bar{q}}\{M\}$ is bounded above. We first need to prove the following statement which says that for all $t \geq 1, \bar{q} \notin \bar{C}_\delta$

$$\bar{V}(\bar{q}) \geq \lambda^{-t}\mathbb{E}_{\bar{q}}\{V(\bar{q})\mathbb{1}(\tau_{\bar{C}_\delta} > t)\} + \sum_{s=1}^{t}\lambda^{-s}\mathrm{pr}_{\varepsilon,L}(\tau_{\bar{C}_\delta} = s). \tag{3.25}$$

The proof can be done by induction. For $t = 1$, the proof is done by applying the law of total expectation to equation 3.22. Now suppose equation 3.25 is true for some $t > 1$, we can use the following inequality to induce equation 3.25 is also true for $t+1$.

$$\begin{aligned}\mathbb{E}_{\bar{q}}\{\bar{V}(\bar{q})\mathbb{1}(\tau_{\bar{C}_\delta} > t)\} &\geq \lambda^{-t}\mathbb{E}_{\bar{q}}\{\bar{V}(\bar{q})\mathbb{1}(\tau_{\bar{C}_\delta} > t)\} \\ &\geq \lambda^{-t}\mathbb{E}_{\bar{q}}\{\bar{V}(\bar{q})\mathbb{1}(\tau_{\bar{C}_\delta} > t+1)\} + \lambda^{-t}\mathrm{pr}_{\varepsilon,L}(\tau_{\bar{C}_\delta} = t)\end{aligned}$$

which completes the proof for equation 3.25. As for any $\bar{q} \notin \bar{C}_\delta$, we have $\bar{V}(\bar{q}) \geq 1$, equation 3.25 implies for all $t \geq 1$

$$\begin{aligned}\bar{V}(\bar{q}) &\geq \lambda^{-t}\mathbb{E}_{\bar{q}}\{\mathbb{1}(\tau_{\bar{C}_\delta} > t)\} + \underbrace{\sum_{s=1}^{t}\lambda^{-s}\mathrm{pr}_{\varepsilon,L}(\tau_{\bar{C}_\delta} = s)}_{>0} \\ &\geq \lambda^{-t}\mathbb{E}_{\bar{q}}\{\mathbb{1}(\tau_{\bar{C}_\delta} > t)\}\end{aligned}$$

or equivalently $\mathrm{pr}_{\varepsilon,L}\{\tau_{\bar{C}_\delta} > t\} \leq \bar{V}(\bar{q})\lambda^t$. Taking $t \to \infty$, it gives that $\mathrm{pr}_{\varepsilon,L}\{\tau_{\bar{C}_\delta} = \infty\} = 0$, which means $\tau_{\bar{C}_\delta}$ is almost surely finite. Now take $t \to \infty$ in equation 3.25, we have

$$\bar{V}(\bar{q}) \geq \sum_{s=1}^{\infty}\lambda^{-s}\mathrm{pr}_{\varepsilon,L}(\tau_{\bar{C}_\delta} = s) = \mathbb{E}_{\bar{q}}\{\lambda^{-\tau_{\bar{C}_\delta}}\} = \mathbb{E}_{\bar{q}}\{M\}$$

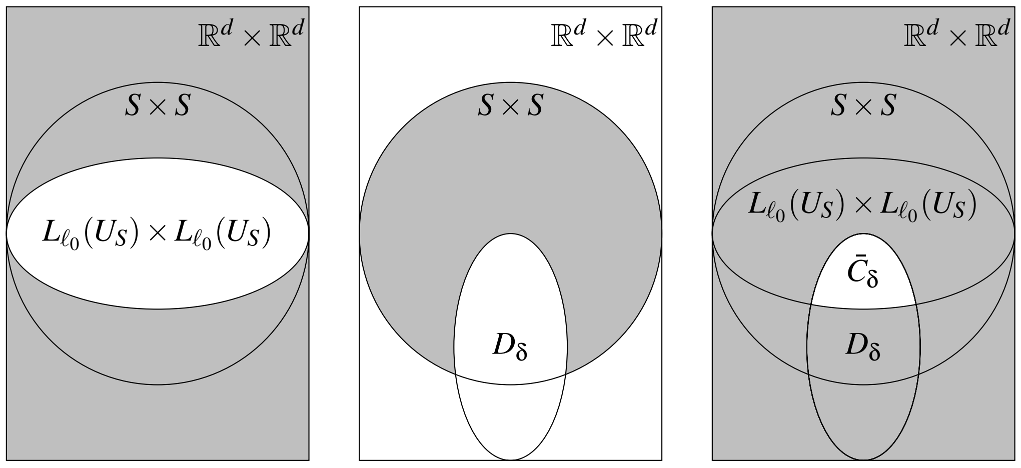
of which the involved equality is valid because $\tau_{\bar{C}_\delta}$ is almost surely finite. Now we can conclude that there exists a finite constant $C_0 \geq \sup_t \left\{\mathbb{E}_{\bar{q}}\{M\}\right\}$ such that

$$\mathrm{pr}_{\varepsilon,L}\{\tau_{\bar{C}_\delta} > t\} \leq C_0\lambda^t.$$

□

Remark. For the last step of the proof, we could also find a more explicit upper bound via establishing $(M_t, \mathcal{F}_t)_{t\geq 0}$ as a super-margintingale where $\mathcal{F}$ the required σ-algebra; see the second part of the proof of (Theorem 1, Heng and Jacob, 2019).

Remark. Figure 3.2 visualize the relationship between the global drift condition and the local contractivity condition using Venn diagrams. As it can be seen, the way the geometric tails are established is by combining the global drift condition and the local contractivity condition. Each of they correspond to a different set in the coupled state space and the intersection between them is one in $S \times S$ with a corresponding Lyapunov function on the coupled space, providing the necessary conditions to use the proof strategy stated in Theorem 3.4.2.


$\mathbb{R}^d \times \mathbb{R}^d$
$S \times S$
$L_{\ell_0}(U_S) \times L_{\ell_0}(U_S)$
$\mathbb{R}^d \times \mathbb{R}^d$
$S \times S$
D_δ
$\mathbb{R}^d \times \mathbb{R}^d$
$S \times S$
$L_{\ell_0}(U_S) \times L_{\ell_0}(U_S)$
$\bar{C}_\delta$
D_δ

(a) Global drift (b) Local contractivity (c) Combined

Figure 3.2: Controlling excursions outside of sets

3.5 Experiments

In this section, we evaluate the performance of the proposed coupled HMC kernels. The coupled Metropolis HMC by Heng and Jacob (2019) is used as a baseline. Following Heng and Jacob (2019), we combine coupled HMC kernels with a coupled RWMH kernel to obtain exact couplings: we take the standard deviation of the RWMH kernel to be $\sigma = 10^{-3}$ and the mixture coefficient (i.e. probability of using RWMH) to be $\alpha = 1/20$. For the estimation task, we consider a more efficient (still unbiased) *time-averaged* variant of equation 3.2:

$$H_{k:m}(X,Y) = \frac{1}{m-k+1} \sum_{i=k}^{m} H_i(X,Y) \tag{3.26}$$

where hyper-parameters k, m are positive integers to choose; we discuss some heuristics to choose these parameters in section 3.5.3. This estimate still requires simulation of a coupled chain X, Y and essentially is an average of H_k in equation 3.2 using samples from k to m. Practically, equation 3.26 can be averaged over multiple simulations of coupled chains For this, we run R independent pairs of coupled chains (X^r, Y^r), $r = 1, \dots, R$, and estimate $H^\dagger$ as $\hat{H} = R^{-1} \sum_{r=1}^{R} H_{k:m}(X^{(r)}, Y^{(r)})$. For $x \in \mathbb{R}^d$, we consider h as the first and second moments of x, i.e. $h_i(x) = x_i$ and $h_{d+i}(x) = x_i^2$ for $i = 1, \dots, d$.

We consider three target distributions. The first target is a **1,000D Gaussian**. The second one is the posterior of a **Bayesian logistic regression** model on the German credit dataset (Asuncion and Newman, 2007). We apply the same pre-processing as in Heng and Jacob (2019), which results in a sampling space of $\mathbb{R}^{302}$. The last model considered is a **log-Gaussian Cox point process** that models tree locations in a forest.

We discretise the forest using a 16×16 grid, resulting in a sampling space on $\mathbb{R}^{256}$. Note that the first two targets meet necessary conditions from section 3.4. More details of these targets can be found in section 3.5.1.

3.5.1 Target distributions

We follow the pre-processing steps in Heng and Jacob (2019) for the German credit dataset (Asuncion and Newman, 2007) and the Finnish pine saplings dataset (Møller et al., 1998) used in logistic regression and log-Gaussian Cox point process respectively.

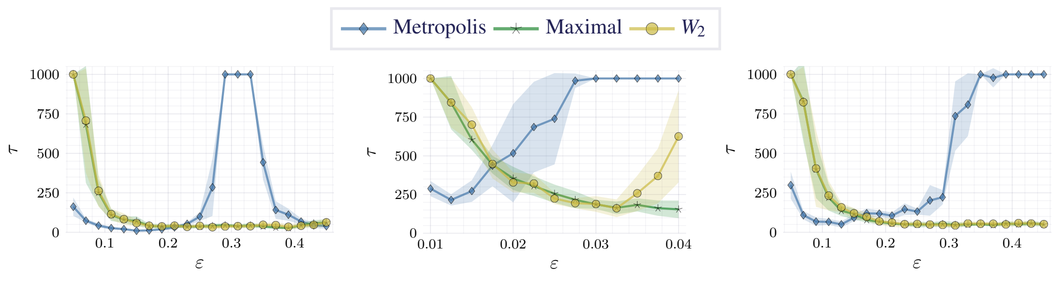
Bayesian logistic regression We combine features in the German credit dataset with all of their standardized pairwise interactions, resulting in a design matrix in $\mathbb{R}^{300\times 1{,}000}$. Denoting an Exponential distribution with rate λ as $\text{Exp}(\lambda)$, the Bayesian logistic regression follows the following generative process: $s^2 \sim \text{Exp}(\lambda), a \sim \mathcal{N}(0, s^2), b \sim \mathcal{N}_{300}$, where the variance $s^2 \in \mathbb{R}$, the intercept $a \in \mathbb{R}$ and the coefficients $b \in \mathbb{R}^{300}$, giving a total dimension $d = 302$.

Log-Gaussian Cox point process Firstly, the plot of the forest is discretised into an $n \times n$ grid. For $i \in \{1, \dots, n\}^2$, the number of points in each grid cell $y_i \in \mathbb{N}$ is assumed to be conditionally independent given a latent intensity variable Λ_i and follows a Poisson distribution with mean $a\Lambda_i$, where $a = n^{-2}$ is the area of each cell. We denote the logarithm of Λ as X and put a Gaussian process prior with mean $\mu \in \mathbb{R}$ and exponential covariance function $\Sigma_{i,j} = s^2 \text{Exp}\left(-|i-j|/(nb)\right)$ on it, where s^2, b and μ are hyperparameters. The generative process of the number of grid cell points follows $X \sim \mathcal{GP}(\mu, \Sigma)$, $\forall\, i \in \{1, \dots, n\}^2 : \Lambda_i = \text{Exp}(X_i)$, $y_i \sim \mathcal{P}\text{oisson}(a\Lambda_i)$. Following (Møller et al., 1998), we use a dataset of 126 Scot pine saplings in a natural forest in Finland, and adapt the parameters $s^2 = 1.91$, $b = 1/33$ and $\mu = \log(126) - s^2/2$.

Our implementation is based on ADVANCEDHMC.JL Xu et al. (2020) and is available at `https://github.com/TuringLang/CoupledHMC.jl`, which also contains scripts to reproduce results in this chapter.

3.5.2 Meeting time comparisons

We first investigate how the meeting time τ of our method changes under different step sizes ε and numbers of leapfrog steps L. For this purpose, we run all coupled HMC



(a) $1,000$D Gaussian

(b) Logistic regression

(c) Log-Gaussian Cox point process

Figure 3.3: Meeting time τ with different ε (x-axis) and $L = 10$ out of $R = 10$ runs with lines for average and shade for 1 standard deviation. Overall, coupled multinomial HMC attains smaller meeting time and is more robust to ε. Note that the range of x-axes depends on the actual parameter sweeps; see main texts for details.

methods initialised at a random draw from $\mathcal{N}(0, I)$ for 1,000 iterations. For each method, we use different step sizes ε and leapfrog steps L: $(\varepsilon, L) \in \{0.05, 0.07, \ldots, 0.45\} \times \{5, 10, 15\}$ for 1,000D Gaussians, $(\varepsilon, L) \in \{0.01, 0.0125, \ldots, 0.04\} \times \{10, 20, 30\}$ for logistic regression and $(\varepsilon, L) \in \{0.05, 0.07, \ldots, 0.45\} \times \{10, 20, 30\}$ for log-Gaussian Cox point processes. Furthermore, we repeat each experiment for $R = 10$ times to estimate standard derivation. Figure 3.3 shows resulting meeting time together with standard derivation for varying ε and a fixed $L = 10$; figures for other L are similar so we defer those to appendix 3.5.4. It is worth noting that τ equal to 1,000 should be interpreted as coupled chains *did not meet within 1,000 iterations*.

Figure 3.3 shows clearly that both maximal coupling and W_2-coupling achieve smaller meeting time than the baseline for large step sizes. This robustness against large step sizes is useful in practice since it allows us to simulate a trajectory of a given length with less computation, by using larger ε rather than larger L. However, when the step size is sufficiently small Metropolis HMC will almost always accept the end-point, thus travel the full integration length T at every step. In contrast, multinomial HMC will put uniform mass on intermediate states which means that it travels $1/4T$ in expectation. Therefore, Metropolis HMC will move towards the typical set faster and thus have a smaller meeting time compared to multinomial HMC. It is also worth noting the surge in meeting time for coupled Metropolis HMC in figure 3.3a around $\varepsilon = 0.3$ can be explained by the similar phenomena observed in figure 3.1, large trajectory length can lead to end-points close to their starting points, thus never meet. In particular, the trajectory length $3 = 0.3 \times 10$ is around $\pi \approx 3.14$, in which case trajectories are basically full circles ending close to where they start, a special case for Gaussians.

Furthermore, for logistic regression (figure 3.3b), optimal parameters of HMC ($\varepsilon = 0.03, L = 10$) leads to excessively long meeting time. This result is consistent with those in Heng and Jacob (2019). This is clearly undesirable: optimal parameters for sampling efficiency leads to non-contractive coupled chains. Besides, it is worth noting that maximal coupling is more robust to large step sizes than W_2-optimal coupling for the logistic regression model. To understand this, recall that W_2-coupling takes a local greedy approach but there is *no guarantee* it can lead to faster meeting through multiple transitions. With large step sizes, numerical errors in simulation are enlarged, leading to more probabilities assigned to non-diagonal entries in the coupling matrix, equivalently more freedom in the W_2-coupling. In such cases, the greedy effect of W_2-coupling is also enlarged but such greedy approach *turns out* to be less effective than maximal coupling for the logistic regression model, a target that satisfies Assumptions 4 and 5. In short, whether the greedy approach is preferable or not is target-dependent. Specifically, when a target satisfies Assumptions 4 and 5, one would expect maximal coupling to work well enough; when such assumptions fail, W_2-coupling can be more efficient, as seen in appendix 3.5.5.

Finally, as motivated earlier, one can use existing adaption techniques to choose parameters ε, L. For example, one can use the adapted parameters from preliminary runs of NUTS. As a concrete example, NUTS-adapted ε, L for logistic regression are 0.022 and 22, and those for log-Gaussian Cox point processes are 0.28 and 16, which allows our method to meet relatively fast: 114 and 118 for the first model and 50 and 51 for the second one, for the two proposed kernels respectively.

3.5.3 Estimator efficiency comparisons

Although Monte Carlo estimates by equation 3.26 are unbiased, it can have large variances due to the use of coupled but often short Markov chains. In other words, making equation 3.26 unbiased comes at a cost of increased variance. Therefore, it is helpful to study the efficiency, or *inefficiency*, of the estimator under a joint effect of removed bias but increased variance, which we define next.

For a vector-valued function h, the variance of estimator $\hat{H}$ for coupled HMC is defined as $\sum_d \nu(h_d)$ where $\nu(h) = \mathbb{V}_r\left(H_{k:m}(h, X^r, Y^r)\right)$. Here r is the index of repeated runs. *Asymptotic inefficiency* is defined as $\sum_d i(h_d)$ where $i(h) = \hat{C}\nu(h)$ (Glynn and

Whitt, 1992). Here $\hat{C} = \mathbb{E}[r]2(\tau^r - 1) + \max(1, m+1-\tau^r)$ is the *expected cost* over R runs and τ^r is the meeting time for the r-th run. The asymptotic variance of (non-coupled) HMC can be approximated with the `spectrum0.ar` function of the `coda` R package (Plummer et al., 2006) using a long chain: 10,000 samples after a burn-in of 1,000 using $(\varepsilon, L) = (0.03, 10)$ for logistic regression, and $(0.3, 10)$ for the log-Gaussian Cox point process model. Relative inefficiency is then defined as the ratio of asymptotic inefficiency (of coupled HMC) over asymptotic variance (of the corresponding non-coupled HMC).

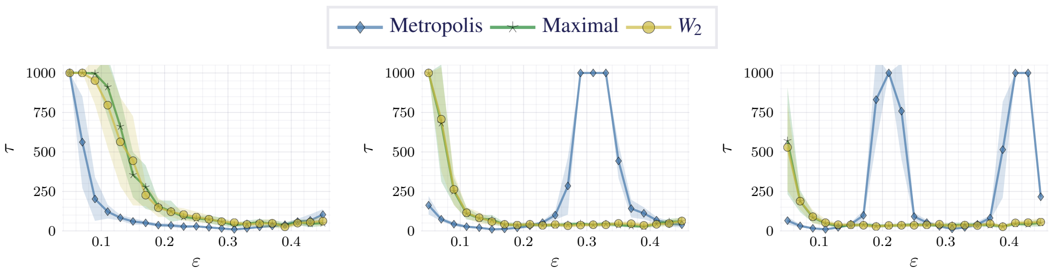
We study inefficiency using logistic regression and log-Gaussian Cox point processes, both widely used in practice. Following Heng and Jacob (2019), we set ε and L to values resulting in the smallest meeting time in section 3.5.2. We first perform 100 runs of coupled HMC kernels to get an empirical distribution of meeting time τ, then we use this distribution to determine k and m following heuristics from Heng and Jacob (2019): take k as either the median or the 90% sample quantile of τ and m as a product of k and a constant, e.g. $5k$ or $10k$—setting a large m here is helpful to get a lower variance with an increase in computation. We then perform $R = 100$ independent runs of coupled chains with different combinations of k and m—we expect a longer chain to have a smaller variance but larger computation budget. Also bear in mind that an ideal asymptotic inefficiency should be close to 1.

Table 3.1 shows asymptotic inefficiencies for varying k and m.[4] For logistic regression, with suitable choices of k, m, the relative inefficiency can be made close to 1 (W_2-coupling); for the other model, the best (3.83) is attained by maximal coupling, both of which are superior to best of Metropolis. Overall, it demonstrates that optimal transport couplings can obtain better bias-variance trade-off than the baseline. Note that both optimal transport couplings *seem* to be inefficient for $m = 5k$ with k being the median. This is because the chains meet quickly compared to Metropolis, and are thus further from the stationary distribution. Table 3.1 also confirms that larger m reduces asymptotic inefficiency at the cost of more computation. Variance reduction can also be achieved by parallel execution.

[4]We kindly note that relative inefficiencies reported here for Metropolis on logistic regression differ from Heng and Jacob (2019) by a factor ≈ 2.0, as confirmed by the authors.

k	m	Metropolis	Maximal	W_2
median	$5k$	2.40	17.69	3.18
	$10k$	2.39	7.37	3.73
90% quantile	$5k$	2.36	2.14	2.88
	$10k$	2.32	1.90	**0.94**
median	$5k$	6.03	4.84	7.62
	$10k$	4.81	4.01	6.00
90% quantile	$5k$	5.26	4.58	6.86
	$10k$	4.61	**3.83**	5.80

Table 3.1: Relative inefficiency with different k and m for logistic regression (top half) and log-Gaussian Cox point processes (bottom half). Bold indicates the one most close to 1. Note that for each method, k is different thus inefficiencies across different coupled kernels (across columns) are *not directly comparable*. Instead, we aim to study if the relative inefficiency can be made close to 1 with suitable parameters (across rows).

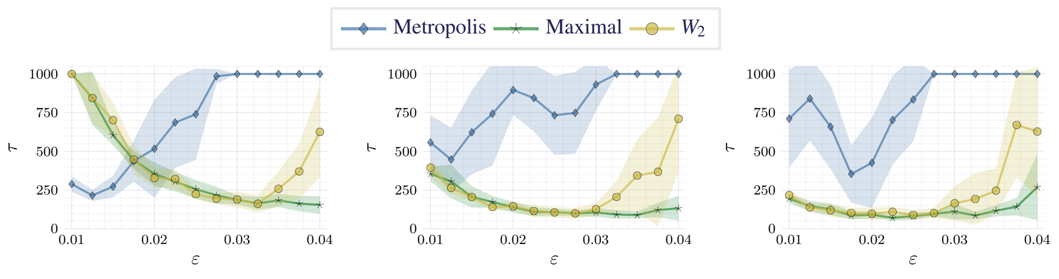


(a) $L = 5$ (b) $L = 10$ (c) $L = 15$

Figure 3.4: Averaged meeting time $\bar{\tau}$ with different ε and L for 1,000D Gaussian.

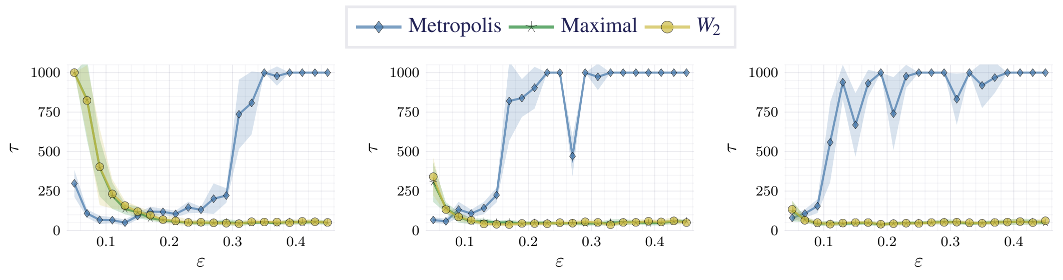
3.5.4 Robustness: meeting time with more parameter sweeps

Figure 3.4, 3.5 and 3.6 provide a wider range of parameter sweep under the same experimental setup as Section 3.5.2. We can see the proposed method with both kernels in general has a better meeting time than coupled Metropolis HMC over a wider range of parameters. However, not only there is no clear winner between maximal and W_2 couplings but also, maybe counter-intuitively, maximal coupling performs pretty well on the logistic regression target. In general, we do expect the maximal coupling perform well on log-concave targets (with mild assumptions)—this is in fact the conditions for which we proved the convergence of our method (with maximal coupling first, followed by W_2 coupling). With this being said, for more complex targets, there is no such



(a) $L = 10$ (b) $L = 20$ (c) $L = 30$

Figure 3.5: Averaged meeting time $\bar{\tau}$ with different ε and L for logistic regression.



(a) $L = 10$ (b) $L = 20$ (c) $L = 30$

Figure 3.6: Averaged meeting time $\bar{\tau}$ with different ε and L for log-Gaussian Cox point process.

guarantee any more. The next section shows two examples of such, which are targets with multi-modality and targets that are high non-convex, in both of which W_2 couplings show consistent advantages.

3.5.5 Multi-modal and highly non-convex targets

We first study how proposed methods behave on multi-modal distributions. Specifically, we want to know if the coupled chains can meet in a short time given the target is multi-modal. We consider a mixture of Gaussians on $\mathbb{R}^2$ with three components $\mathcal{N}([-1,-1], 0.25^2 I)$, $\mathcal{N}([0,0], 0.25^2 I)$, $\mathcal{N}([1,1], 0.25^2 I)$ weighted by 0.25, 0.4 and 0.35 respectively. We initialise chains from $\mathcal{U}([0,1]^2)$, covering two of the modes. We simulate $R = 500$ pairs of chains and check if they meet within 100 iterations. Denoting the number of chains which meet as N_τ, we report $i_\tau = N_\tau / R$ as a measure of efficiency in meeting. Regarding the choice of ε, L, it is known that HMC is sensitive to the total trajectory length εL in multi-modal distributions: it requires the Hamiltonian simulation long enough to allow jumps between modes. Therefore, starting with $(\varepsilon, L) = (0.1, 10)$, we consider two ways of increasing εL: sweeping $\varepsilon \in \{0.1, 0.15, \ldots, 0.3\}$ and sweeping

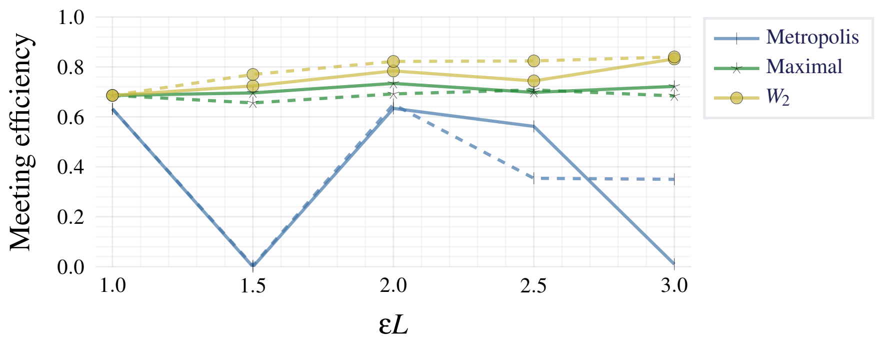


Figure 3.7: Meeting efficiency on the mixture of Gaussians target with the total trajectory length εL increasing. Solid lines are from increasing ε and dashed ones from increasing L.

Momentum	Metropolis	Maximal	W_2
Shared	136.6 ± 95.8	112.4 ± 74.9	103.8 ± 76.5
Contractive	$\mathbf{39.7 \pm 18.9}$	$\mathbf{81.3 \pm 56.3}$	$\mathbf{77.2 \pm 48.1}$

Table 3.2: Effect of different momentum coupling methods on meeting time for the Banana target.

$L \in \{10, 15, \ldots, 30\}$, equivalently providing a range of total lengths between 1 and 3. While both means increase the trajectory length, the first approach doesn't introduce additional computation but might lead to larger simulation errors, which may then affect the overall performance. Figure 3.7 provides i_τ under such changes of total trajectory lengths for all methods. First, by increasing εL, our proposed methods overall improve the meeting efficiency, which is not the case for coupled Metropolis HMC. This can be explained by the following: for coupled Metropolis HMC, meetings can only happen if two chains are proposed to the same mode. However, for coupled multinomial HMC, as long as the trajectories explore common modes, there is a chance for meeting. Especially with W_2-coupling, this chance is further increased by utlizing the actual distances between pairs to find coupling, making it the best in the figure. Second, regarding the two ways of increasing εL, for our proposed methods, increasing L appears to be better as we expected. That said, the gap is relatively small—coupled multinomial HMC tends to be robust against large ε, which is practically useful as it allows the use of a smaller amount of computation comparing to increasing L. Note that we do not claim or indicate our methods improve the mixing in multi-modal distributions, which by itself is an important and unsolved issue for HMC.

Second, to examine the proposed methods on highly non-convex distributions, we consider a banana-shaped distribution on $\mathbb{R}^2$, of which the potential is given by the Rosenbrock function $U(x_1,x_2) = (1-x_1)^2 + 10(x_2 - x_1^2)^2$ $(x_1, x_2 \in \mathbb{R})$. As it is done in (Heng and Jacob, 2019), we also take this chance to study the effect of other methods for coupling the initial momentums rather than simply sharing them. Specifically, we consider the contractive coupling from (Bou-Rabee et al., 2020), in which the initial momentums P^1, P^2 are sampled based on the current positions Q^1, Q^2 as follow

$$P^1 \sim \mathcal{N}(0, I),$$
$$P^2 = \begin{cases} P^1 + \kappa\Delta & \text{with prob. } \frac{\mathcal{N}(\bar{\Delta}^\top P^1 + \kappa|\Delta|;0,1)}{\mathcal{N}(\bar{\Delta}^\top P^1;0,1)} \\ P^1 - 2(\bar{\Delta}^\top P^1)\bar{\Delta} & \text{otherwise} \end{cases}$$

where $\kappa > 0$ is a tuning parameter, $\Delta = Q^1 - Q^2$ is the difference in position space and $\bar{\Delta}$ is the corresponding normalised difference. With initial states sampled from $\mathcal{U}([0,1]^2)$, we simulated $R = 500$ pairs of coupled chains with $(\varepsilon, L) = (1/50, 50)$ for maximally 500 iterations with two momentum coupling methods: shared momentum and contractive coupling with $\kappa = 1$. We summarise means and standard deviations of τ from R runs in table 3.2.

First of all, all method with two momentum coupling methods can meet within 150 iterations in such high non-convex setup. Also, it can be seen that our methods can also benefit from contractive coupling, even though it is derived as a maximal coupling (Thorisson, 2000) for Metropolis HMC. This is the reason why coupled Metropolis HMC is largely improved by it. That is to say, contractive coupling is an orthogonal method of ours rather than a replacement. Note that the table should not be used to compare coupled multinomial HMC against coupled Metropolis HMC in terms of meeting time because they have different optimal parameters for meeting in this target.

3.6 Related Work

Research on couplings for MCMC methods has a long history (Devroye, 1990; Johnson, 1996, 1998; Rosenthal, 1997; Meyn and Tweedie, 2012; Rowland et al., 2018; Nuesken and Pavliotis, 2018; Jacob et al., 2019a; Biswas et al., 2019). Couplings for HMC has been more recently focusing on Metropolis HMC, e.g. Neal (2017). Most closely related to our work is Heng and Jacob (2019), in which coupling for Metropolis HMC is established. Bou-Rabee et al. (2020) studied the convergence of Metropolis HMC, and also proposed a new way to couple momentum variables, called *contractive coupling*, that does not rely on simply sharing them.

Developing or improving parallel MCMC methods has surged an interest in recent years due to cheaper parallel computing resources. For example, Neiswanger et al. (2013) proposed to distribute the computation of the complete dataset into subsets; Goodman and Weare (2010) develops a method that running multiple MCMC chains and combine them every few iterations. A separate direction of scaling up MCMC methods to large data is to use stochastic information via mini-batches at each iteration only (Korattikara et al., 2014) which also has been done in the context of HMC (Welling and Teh, 2011; Chen et al., 2014).

Our analysis in section 3.4 is also related to works on convergence analysis of HMC on log-concave targets (Mangoubi and Smith, 2017; Chen and Vempala, 2019).

Appendix

3.A Additional Background

3.A.1 Properties of Hamiltonian flow

The flow map Φ_t has the following properties:

1. (Reversibility). $\forall\, t \in \mathbb{R}_+$, the inverse flow map Φ_t^{-1} satisfies $\Phi_t^{-1} = R \circ \Phi_t \circ R$, where $R(q,p) = (q,-p)$ denotes the momentum reversal operation.
2. (Energy conservation). The Hamiltonian $\mathcal{E}$ of the system satisfies $\mathcal{E} \circ \Phi_t = \mathcal{E}$.
3. (Measure preservation). For any $t \in \mathbb{R}_+$ and $A \in \mathcal{B}(\mathbb{R}^{2d})$, we have $\mathrm{Leb}_{2d}\left(\Phi_t(A)\right) = \mathrm{Leb}_{2d}(A)$, where Leb_d denotes the Lebesgue measure on $\mathbb{R}^d$.

Together the properties ensures that the Markov kernel defined by the Hamiltonian flow leaves the extended target distribution $\bar{\pi}$ invariant.

3.A.2 Properties of leapfrog integration

The numerical flow map $\hat{\Phi}_{\varepsilon,L}$ enjoys the following two inequalities due to the symplecticity of order-two leapfrog integrators (Hairer et al., 2006)

$$\left\|\hat{\Phi}_{\varepsilon,L}(q_0,p_0) - \Phi_{\varepsilon L}(q_0,p_0)\right\| \leq C_a(q_0,p_0,L)\varepsilon^2 \tag{3.27}$$

$$\left\|\mathcal{E}\left(\hat{\Phi}_{\varepsilon,L}(q_0,p_0)\right) - \mathcal{E}(q_0,p_0)\right\| \leq C_b(q_0,p_0,L)\varepsilon^2 \tag{3.28}$$

for some positive constants C_a and C_b. These two inequalities are used in several places through our theoretical analysis, e.g. in Section 3.B.2 and Section 3.B.4.

3.B Technical Details

3.B.1 Proof of Lemma 3.4.1

Proof. Suppose $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$ satisfies Condition 1 on the set S for some $\bar{\varepsilon} > 0$, $\bar{L} \in \mathbb{N}$.

First observe that

$$
\begin{aligned}
&\mathrm{pr}^{\gamma}_{\varepsilon,L}\left(\|Q^1_1 - Q^2_1\| \leq \rho\|Q^1_0 - Q^2_0\| \mid (Q^1_0, Q^2_0) = (q^1, q^2)\right) \\
&= \mathbb{E}_{\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}} \mathbb{1}\left\{\|Q^1_1 - Q^2_1\| \leq \rho\|q^1 - q^2\|\right\} \\
&= \mathbb{E}_{P\sim\mathcal{N}(0,I)}\left[\mathbb{E}_{(l_1,l_2)\sim\gamma}\mathbb{1}(R_{q^1,q^2,P}) \mid P\right]
\end{aligned}
$$

where we have let $R_{q^1,q^2,p}$ denote the set of events where we have contraction, i.e.

$$
R_{q^1,q^2,p} = \left\{\|\hat{\Phi}^{\circ}_{\varepsilon,l_1}(q^1,p) - \hat{\Phi}^{\circ}_{\varepsilon,l_2}(q^2,p)\| \leq \rho\|q^1 - q^2\|\right\}
$$

By Condition 1 we know that there exists $\omega_1 \in (0,1)$ such that

$$
\mathbb{P}_{(l_1,l_2)\sim\gamma}\left(R_{q^1,q^2,p}\right) \geq \omega_1 \tag{3.29}
$$

for all $(q^1, q^2, p) \in S \times S \times L_{k_0}(K)$, where $k_0 > 0$. By the tower property of expectation, this immediately implies that

$$
\begin{aligned}
&\mathbb{E}_{P\sim\mathcal{N}(0,I)}\left[\mathbb{E}_{(l_1,l_2)\sim\gamma}\mathbb{1}(R_{q^1,q^2})\,\mathbb{1}\{K(P) \leq k_0\} \mid P\right] \\
&\geq \mathbb{E}_{P\sim\mathcal{N}(0,I)}\omega_1\,\mathbb{1}\{K(P) \leq k_0\} \\
&= \omega_1\mathbb{P}_{P\sim\mathcal{N}(0,I)}(\{K(P) \leq k_0\}) \\
&> 0
\end{aligned}
$$

where the last inequality follows from the fact that the level sets $L_{k_0}(K)$ are closed for any $k_0 > 0$ since K is continuous and bounded and therefore compact, in addition to having positive Lebesgue measure. Since equation 3.29 holds for all $(q^1, q^2, p) \in S \times S \times L_{k_0}(K)$ with $\omega_1 > 0$, we have

$$
\begin{aligned}
\inf_{q^1,q^2\in S} \mathrm{pr}^{\gamma}_{\varepsilon,L}\big(&\{\|Q^1_1 - Q^2_1\| \leq \rho\|Q^1_0 - Q^2_0\|\} \\
&\cap\{K(P) \leq k_0\} \mid (Q^1_0, Q^2_0) = (q^1, q^2)\big) \\
&\geq \omega_1\omega_2 \\
&> 0
\end{aligned} \tag{3.30}
$$

where we have let $\omega_2 = \mathbb{P}_{P\sim\mathcal{N}(0,I)}(K(P) \leq k_0)$.

In words, for any initial points $(q^1, q^2) \in S \times S$, a single application of the kernel $\bar{\mathcal{K}}^{\gamma}_{\varepsilon,L}$ decreases the distance with non-zero probability. Equipped with this, proving the desired statement is just a matter of ensuring that we can indeed apply equation 3.30 repeatedly to get the states sufficiently close to each other. A straightforward approach to this is to simply choose the stepsize to be sufficiently small such that even when taking the required number of steps to get within the desired δ-ball, every step taken is still within a set where equation 3.30 holds. This is exactly the approach taken in Heng and Jacob (2019) and so the rest of the proof is essentially identical to the last paragraph in the proof of Proposition 1 in Heng and Jacob (2019).

Consider $u_0 > \inf_{q \in S} U(q)$, and $u_1 < \sup_{q \in S} U(q)$ with $u_0 < u_1$, and let $A_\ell := L_\ell(U_S) \times L_{u_1 - \ell}(K) \subset L_{u_1}(\mathcal{E})$ for $\ell \in (u_0, u_1)$. Since continuity and convexity of U_S imply that this is a closed function, its level sets $L_\ell(U_S)$ are closed. Moreover, under the assumptions on U and S, it follows that these level sets are compact with positive Lebesgue measure. Note that if $(q, p) \in A_\ell$, due to energy conservation and continuity of U, the mapping $t \mapsto \Phi^\circ_t(q, p)$ imply that $\Phi^\circ_t(q, p) \in L_{u_1}(U_S)$ for any $t \in [-T, T]$. Due to time discretisation, using equation 3.28 and compactness of A_ℓ we can only conclude that there exists $\eta_0 > 0$ such that $\hat{\Phi}^\circ_{\varepsilon,l}(q, p) \in L_{u_1+\eta_0}(U)$ for all $(q, p) \in A_\ell$ and $l = L_b, \ldots, L_f$. Let $n_0 = \min\{n \in \mathbb{N} : \rho^n B \leq \delta\}$, where $B := \sup_{q^1, q^2 \in S} \|q^1 - q^2\|$. By choosing $v_0 \in (u_0, u_1)$, $k_0 > 0$, and $\eta_0 > 0$ small enough such that

$$v_0 + (n_0 + 1)k_0 + n_0\eta_0 < u_1$$

holds, we have $Q^1_k, Q^2_k \in S$ for all $k = 1, \ldots, n_0$. Hence, by repeated application of equation 3.30,

$$\inf_{q^1, q^2 \in S_0} \mathrm{pr}_{\varepsilon,L}\left(\|Q^1_{n_0} - Q^2_{n_0}\| \leq \delta \mid (Q^1_0, Q^2_0) = (q^1, q^2)\right) > 0$$

with $S_0 = L_{v_0}(U_S)$, exactly as in Proposition 3.4.1. □

3.B.2 Proof of Lemma 3.4.2

Lemma 3.B.1. Suppose that the potential U satisfies Assumptions 4 and 5. For any compact set $A \subset S \times S \times \mathbb{R}^d$, there exists a trajectory length $T > 0$ and a step size $\varepsilon_1 > 0$ s.t. for any $\varepsilon \in (0, \varepsilon_1]$ and any $t \in [-T, T] \setminus \{0\}$ with $l := t/\varepsilon \in \mathbb{Z}$, there exists $\rho \in [0, 1)$ satisfying

$$\|\hat{\Phi}^\circ_{\varepsilon,l}(q^1_0, p_0) - \hat{\Phi}^\circ_{\varepsilon,l}(q^2_0, p_0)\| \leq \rho \|q^1_0 - q^2_0\| \tag{3.31}$$

for all $(q^1_0, q^2_0, p_0) \in A$.

Proof. As the leapfrog integrator is of order two (Hairer et al., 2006; Bou-Rabee et al., 2020), for any sufficently small step size ε and number of step l states above, we have $\left\|\hat{\Phi}_{\varepsilon,l}(q_0,p_0)-\Phi_t(q_0,p_0)\right\| \leq C_1(q_0,p_0,t)\varepsilon^2$ and similar for its position-projected correspondence

$$\left\|\hat{\Phi}^{\circ}_{\varepsilon,l}(q_0,p_0)-\Phi^{\circ}_t(q_0,p_0)\right\| \leq C_1(q_0,p_0,t)\varepsilon^2 \tag{3.32}$$

where $C_1(q_0,p_0,t)$ is some constant that only depends on q_0,p_0 and t.

By (Lemma 1, Heng and Jacob, 2019), with some fixed T, we have $\rho' \in [0,1)$ satisfying

$$\left\|\Phi^{\circ}_t(q_0^1,p_0)-\Phi^{\circ}_t(q_0^2,p_0)\right\| \leq \rho'\left\|q_0^1-q_0^2\right\| \tag{3.33}$$

for any $t \in (0,T]$ and all $(q_0^1,q_0^2,p_0) \in A$. Since $\Phi^{\circ}_t(q_0^1,-p_0)=\Phi^{\circ}_{-t}(q_0^1,p_0)$, applying (Lemma 1, Heng and Jacob, 2019) again with the momentum variable negated, we have equation 3.33 for $t \in [-T,0)$. Therefore equation 3.33 holds for $t \in [-T,T]\setminus\{0\}$.

With these two intermediate results, we can now bound the left-hand side (LHS) of equation 3.31 for any $t \in [-T,T]\setminus\{0\}$ with $l=t/\varepsilon \in \mathbb{Z}$ and all $(q_0^1,q_0^2,p_0) \in A$

$$\begin{aligned}
&\left\|\hat{\Phi}^{\circ}_{\varepsilon,l}(q_0^1,p_0)-\hat{\Phi}^{\circ}_{\varepsilon,l}(q_0^2,p_0)\right\| \\
=&\left\|\hat{\Phi}^{\circ}_{\varepsilon,l}(q_0^1,p_0)-\Phi^{\circ}_t(q_0^1,p_0)-\right. \\
&\left.\hat{\Phi}^{\circ}_{\varepsilon,l}(q_0^2,p_0)+\Phi^{\circ}_t(q_0^2,p_0)+\Phi^{\circ}_t(q_0^1,p_0)-\Phi^{\circ}_t(q_0^2,p_0)\right\| \\
\leq&\left\|\hat{\Phi}^{\circ}_{\varepsilon,l}(q_0^1,p_0)-\Phi^{\circ}_t(q_0^1,p_0)\right\|+ \\
&\left\|\hat{\Phi}^{\circ}_{\varepsilon,l}(q_0^2,p_0)-\Phi^{\circ}_t(q_0^2,p_0)\right\|+\left\|\Phi^{\circ}_t(q_0^1,p_0)-\Phi^{\circ}_t(q_0^2,p_0)\right\| \\
\leq&\left(C(q_0^1,p_0,t)+C(q_0^2,p_0,t)\right)\varepsilon^2+\rho'\left\|q_0^1-q_0^2\right\|
\end{aligned}$$

where the third line is a result of the triangle inequality and the last line comes from equation 3.32 and equation 3.33 respectively. As $\lim_{\varepsilon\to 0}\left(C(q_0^1,p_0,t)+C(q_0^2,p_0,t)\right)\varepsilon^2=0$, for any $\rho \in (\rho',1)$, there exists a step size $\varepsilon_1>0$ such that for any $\varepsilon \leq \varepsilon_1$, equation 3.31 holds. □

3.B.3 Proof of Proposition 3.B.1

For the sake of presentation, in this section we only consider Condition 1 for $m=1$. To prove that γ^* satisfies Condition 1 for $m>1$ follows the exact reasoning since equation 3.15 in Lemma 3.4.2 still holds when both sides are raised to some positive power m.

Proposition 3.B.1. Suppose that U satisfies Assumptions 4 and 5. For any compact set $A \subset S \times S \times \mathbb{R}^d$ and any parallel-in-time joint $J^{\|} \in \mathbb{R}^{K\times K}$, there exists a trajectory length $T > 0$, a step size $\varepsilon_1 > 0$ s.t. for any $\varepsilon \in (0, \varepsilon_1]$ and any $L_1, L_2 \in \mathbb{N}$ with $L_1 + L_2 = K - 1$ and $\varepsilon L_1, \varepsilon L_2 < T$, there exists $\tilde{\rho} \in (0,1)$ satisfying

$$\mathbb{E}_{(i,j)\sim J^{\|}} \left\| \hat{\Phi}^{\circ}_{\varepsilon,l_i}(q^1,p) - \hat{\Phi}^{\circ}_{\varepsilon,l_j}(q^2,p) \right\| \le \tilde{\rho} \left\| q^1 - q^2 \right\| \tag{3.34}$$

for all $(q^1, q^2, p) \in A$, where l_k is the k-th entry of the vector $[-L_1, \dots, 0, \dots, L_2]$.

Proof. By definition, $J^{\|}$ has only diagonal entries, thus $(i, j) \sim J^{\|}$ is equivalent to (i,i) with $i \sim \text{diag}(J^{\|})$. Denote the left-hand side of equation 3.34 as A_1, expanding and rearranging A_1 and applying Lemma 3.4.2, we have

$$\begin{aligned}
A_1 &= \sum_{k=0}^{L_1+L_2+1} \mathbb{P}(i=k) \left\| \hat{\Phi}^{\circ}_{\varepsilon,l_k}(q^1,p) - \hat{\Phi}^{\circ}_{\varepsilon,l_k}(q^2,p) \right\| \\
&= \sum_{k\neq L_1+1} \mathbb{P}(i=k) \left\| \hat{\Phi}^{\circ}_{\varepsilon,l_k}(q^1,p) - \hat{\Phi}^{\circ}_{\varepsilon,l_k}(q^2,p) \right\| \\
&\qquad + \mathbb{P}(i = L_1+1) \left\| \hat{\Phi}^{\circ}_{\varepsilon,0}(q^1,p) - \hat{\Phi}^{\circ}_{\varepsilon,0}(q^2,p) \right\| \\
&\le \sum_{k\neq L_1+1} \mathbb{P}(i=k) \rho_{l_k} \left\| q^1 - q^2 \right\| \\
&\qquad + \mathbb{P}(i = L_1+1) \left\| q^1 - q^2 \right\| \\
&= \mathbb{E}_i[\rho_{l_i}] \left\| q^1 - q^2 \right\| \\
&:= \tilde{\rho} \left\| q^1 - q^2 \right\|
\end{aligned}$$

where we let $\rho_0 = 1$. As $\rho_l \in (0,1)$ for $l \neq 0$ and $\rho_0 = 1$, $\mathbb{E}_i \rho_{l_i} = \sum_k \mathbb{P}(i=k) \times \rho_{l_k} \in (0,1)$ by the property of convex combination. In other words, we have $\tilde{\rho} \in (0,1)$. □

3.B.4 Proof of Proposition 3.4.2

Proof. For two length-K Hamiltonian trajectories $\mathbf{t}^1$ and $\mathbf{t}^2$, denote $\boldsymbol{x} = [\mathcal{E}(\mathbf{t}^1_1), \dots, \mathcal{E}(\mathbf{t}^1_K)]$ and $\boldsymbol{y} = [\mathcal{E}(\mathbf{t}^2_1), \dots, \mathcal{E}(\mathbf{t}^2_K)]$ as vectors of the Hamiltonian energy of all phasepoints. With the softmax function $\sigma(\boldsymbol{x})_i = \exp(-\boldsymbol{x}_i)/\sum_{i'} \exp(-\boldsymbol{x}_{i'})$, the entries of $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ can be expressed as

$$\mu_i = \sigma(\boldsymbol{x})_i \quad \nu_j = \sigma(\boldsymbol{y})_j$$

By the Cauchy–Schwarz inequality, we have $\|\sigma(\boldsymbol{x})-\sigma(\boldsymbol{y})\|_1 \leq \sqrt{K}\|\sigma(\boldsymbol{x})-\sigma(\boldsymbol{y})\|$. With this, we can then upper-bound $\mathrm{D_{TV}}(\boldsymbol{\mu},\boldsymbol{\nu})$ as

$$\begin{aligned}\mathrm{D_{TV}}(\boldsymbol{\mu},\boldsymbol{\nu}) &= \mathrm{D_{TV}}(\sigma(\boldsymbol{x}),\sigma(\boldsymbol{y}))\\ &= \frac{1}{2}\|\sigma(\boldsymbol{x})-\sigma(\boldsymbol{y})\|_1\\ &\leq \frac{1}{2}\sqrt{K}\|\sigma(\boldsymbol{x})-\sigma(\boldsymbol{y})\|\end{aligned}$$

Denote the energy of the initial phase points in each trajectory (q_0^1, p_0) and (q_0^2, p_0) as $\mathcal{E}_0^1$ and $\mathcal{E}_0^1$ and let $\mathcal{E}_i^1 := \boldsymbol{x}_i$ and $\mathcal{E}_j^1 := \boldsymbol{y}_j$; note that for some $i_0 \in \{1,\ldots,K\}$ we have $\mathcal{E}(\mathbf{t}_{i_0}^c) = \mathcal{E}_0^c$ for $c = 1,2$, i.e. i_0 represents the initial time-index which is shared between the two. As the leapfrog integrator is of order two (Hairer et al., 2006; Bou-Rabee et al., 2020), for any sufficiently small step size $\varepsilon = T/L$, we have

$$|\mathcal{E}_0^c - \mathcal{E}(\mathbf{t}_i^c))| \leq C_2(q_0^c, p_0)\, t_i\, \varepsilon^2 \leq C_2(q_0^c, p_0)\, T\, \varepsilon^2 \tag{3.35}$$

for $c = 1,2$, where t_i denotes the corresponding integration time for the i-th phase point from the first phase point. Denote the energy differences as $\Delta_i^1 = \mathcal{E}(\mathbf{t}_i^1) - \mathcal{E}_0^1$ and $\Delta_j^1 = \mathcal{E}(\mathbf{t}_j^2) - \mathcal{E}_0^2$ and observe that

$$\sigma(\boldsymbol{x}) = \sigma([\Delta_1^1,\ldots,\Delta_K^1]) \quad \sigma(\boldsymbol{y}) = \sigma([\Delta_1^2,\ldots,\Delta_K^2])$$

Using the fact that the softmax function is 1-Lipschitz (Gao and Pavel, 2018) and applying equation 3.35, we have

$$\begin{aligned}\|\sigma(\boldsymbol{x})-\sigma(\boldsymbol{y})\| &= \left\|\sigma([\Delta_1^1,\ldots,\Delta_K^1]) - \sigma([\Delta_1^2,\ldots,\Delta_K^2])\right\|\\ &\leq \left\|[\Delta_1^1,\ldots,\Delta_K^1] - [\Delta_1^2,\ldots,\Delta_K^2]\right\|\\ &\leq \sqrt{\sum_{k=1}^{K} C_2(q_0^1,p_0)C_2(q_0^2,p_0)\, T^2\, \varepsilon^4}\\ &= \sqrt{KC_2(q_0^1,p_0)C_2(q_0^2,p_0)}\, T\, \varepsilon^2\end{aligned}$$

Substituting back into our bound on $\mathrm{D_{TV}}(\boldsymbol{\mu},\boldsymbol{\nu})$,

$$\mathrm{D_{TV}}(\boldsymbol{\mu},\boldsymbol{\nu}) \leq \frac{1}{2}K\sqrt{C_2(q_0^1,p_0)C_2(q_0^2,p_0)}\, T\, \varepsilon^2$$

Since T is fixed, $\varepsilon = T/L$ and $K = L+1$, we have

$$\begin{aligned}\mathrm{D_{TV}}(\boldsymbol{\mu},\boldsymbol{\nu}) &\leq \frac{1}{2}\sqrt{C_2(q_0^1,p_0)C_2(q_0^2,p_0)}\, T^3\, \frac{L+1}{L^2}\\ &\leq \sqrt{C_2(q_0^1,p_0)C_2(q_0^2,p_0)}\, T^3 L^{-1}\\ &\leq \sqrt{C_2(q_0^1,p_0)C_2(q_0^2,p_0)}\, T^2\, \varepsilon.\end{aligned} \tag{3.36}$$

Finally, note that the upper-bound decreases in with ε and T, hence for any given $\delta > 0$, there exists $\varepsilon_0 > 0$, $L_0 \in \mathbb{N}$ such that $D_{\mathrm{TV}}(\boldsymbol{\mu}, \boldsymbol{\nu}) \leq \delta$ for all $\varepsilon \in (0, \varepsilon_0)$ and $L \in \mathbb{N}$ satisfying $\varepsilon L < \varepsilon_0 L_0 = T$. □

3.B.5 Proof of Lemma 3.4.3

Similarly to in appendix 3.B.3 we only consider Condition 1 with $m = 1$ as the case of $m > 1$ follows similarly.

To prove Lemma 3.4.3 we first restate a more detailed version of the lemma, which we then prove.

Lemma 3.B.2. Suppose that the potential U satisfies Assumptions 4 and 5. For a maximal coupling γ^*, there exists a trajectory length $T > 0$ and a step size $\varepsilon_2 > 0$ such that for any $\varepsilon \in (0, \min\{\varepsilon_1, \varepsilon_2\}]$ and any $t \in [-T, T] \setminus \{0\}$ with $l := t/\varepsilon \in \mathbb{Z}$, there exists $\rho_2 \in (0, 1)$ satisfying

$$\mathbb{E}_{(l_1,l_2)\sim\gamma^*} \left\|\hat{\Phi}^{\circ}_{\varepsilon,l_1}(q^1, p) - \hat{\Phi}^{\circ}_{\varepsilon,l_2}(q^2, p)\right\| \leq \rho \left\|q^1 - q^2\right\| \tag{3.37}$$

for all $(q^1, q^2) \in S \times S$, where $\bar{\mathcal{K}}^*_{\varepsilon,l}$ is the coupled kernel in algorithm 3.2 with (i) shared momentum, (ii) shared forward and backward simulation steps and (iii) $(i, j) \sim \gamma^*$ for intra-trajectory sampling.

Proof. We first decompose γ^* into its "diagonal" and "non-diagonal" components

$$\gamma^* = \omega J^{\parallel} + (1-\omega) J^{\nparallel}$$

where $1 - \omega = \mathbb{P}(i \neq j)$ and $J^{\nparallel}$ is defined to be the residual with normalisation. Thus we have

$$\begin{aligned} A_2 &:= \omega \mathbb{E}_{J^{\parallel}} \left\|\hat{\Phi}^{\circ}_{\varepsilon,l_i}(q^1, p) - \hat{\Phi}^{\circ}_{\varepsilon,l_j}(q^2, p)\right\| \\ &\quad + (1-\omega)\mathbb{E}_{J^{\nparallel}} \left\|\hat{\Phi}^{\circ}_{\varepsilon,l_i}(q^1, p) - \hat{\Phi}^{\circ}_{\varepsilon,l_j}(q^2, p)\right\| \\ &\leq \omega \tilde{\rho} \left\|q^1 - q^2\right\| \\ &\quad + (1-\omega)\mathbb{E}_{J^{\nparallel}} \left\|\hat{\Phi}^{\circ}_{\varepsilon,l_i}(q^1, p) - \hat{\Phi}^{\circ}_{\varepsilon,l_j}(q^2, p)\right\| \end{aligned} \tag{3.38}$$

for $T > 0$, $\varepsilon \in (0, \varepsilon_1]$ and $\tilde{\rho} \in (0, 1)$ in Proposition 3.B.1. As $\mathbb{E}_{J^{\nparallel}} \left\|\hat{\Phi}^{\circ}_{\varepsilon,l_i}(q^1, p) - \hat{\Phi}^{\circ}_{\varepsilon,l_j}(q^2, p)\right\|$ is finite, by Proposition 3.4.2, the limit of the upper bound goes to $\tilde{\rho}\left\|q^1 - q^2\right\|$ as $\varepsilon \to 0$. In other words, for any $\rho \in (\tilde{\rho}, 1)$, there exists a step size $\varepsilon_2 > 0$ such that for any $\varepsilon \in (0, \min\{\varepsilon_1, \varepsilon_2\}]$,

$$A_2 \leq \rho \left\|q^1 - q^2\right\|$$

which is exactly what we wanted to prove. $\square$

Chapter 4

Variational Russian Roulette for Deep Bayesian Nonparametrics

4.1 Introduction

A major challenge in unsupervised learning is to infer the complexity of the latent structure, such as the number of clusters or the size of a continuous representation, that is necessary to describe a data set. A principled way from statistics to choose the complexity of a model is provided by Bayesian nonparametric (BNP) methods (Walker et al., 1999; Müller and Quintana, 2004; Orbanz and Teh, 2010; Gershman and Blei, 2012). BNP methods allow for the size of the inferred model to automatically adapt to the complexity of data, so that simpler models are preferred for smaller data sets, and more complex models are preferred for larger data sets. For example, a latent feature model with an Indian buffet process prior is a representation learning method that infers a latent binary vector for each data point, where the number of binary features is chosen adaptively based on the data. Within machine learning, Bayesian nonparametric methods have been applied within models as diverse as clustering (Antoniak, 1974; Görür and Rasmussen, 2010; Teh et al., 2005), topic modelling (Teh et al., 2006), and infinite deep neural networks (Adams et al., 2010; Abbasnejad et al., 2017).

While Bayesian nonparametric models are appealing, inference in such models can be computationally challenging. Among the most common inference algorithms are Markov chain Monte Carlo based methods, which includes Gibbs sampling and slice sampling (Griffiths and Ghahramani, 2011; Teh et al., 2007). These methods are flexible but slow, making them hard to apply to large data sets. Amortised variational methods, which is a type of variational inference method (Kingma and Welling, 2014; Rezende et al., 2014; Ranganath et al., 2014; Mnih and Gregor, 2014), are an appealing option, because they exploit the smoothing properties of deep neural networks to accelerate inference. For Bayesian nonparametric models, however, amortised inference is challenging because the dimensionality of the latent space is not fixed. Previous methods for variational inference in such models rely on a truncated approximation, which places an upper bound on the size of the latent space under the approximate posterior (Blei and Jordan, 2004; Doshi-Velez et al., 2009). Similarly, recent work on amortised inference in Bayesian nonparametrics relies on truncation, albeit sometimes within an outer loop that searches over the truncation size (Miao et al., 2017; Nalisnick and Smyth, 2017; Chatzis, 2014; Singh et al., 2017).

However, the use of truncated approximation has several drawbacks. If the truncation level is chosen too small, the accuracy of the approximation degrades, whereas if the truncation level is chosen too large, then inference will be slow, removing one of the main advantages of variational inference. Besides, the truncation level can interact poorly with amortised inference, because of the well-known component collapse issue (Dinh and Dumoulin, 2016; van den Oord et al., 2017), which is also called over-pruning (Burda et al., 2015; Yeung et al., 2017). This refers to the problem when the inferred latent representation includes components whose conditional distribution given a test data point tends to remain very similar to the prior.[1] Perhaps more importantly, it loses the main benefit of nonparametric approach for automatic complexity inference.

In this work, we overcome these limitations using a new dynamic variational approximation, which we call **roulette-based amortised variational expectation** (RAV E). The goal of RAVE is to allow the approximate variational posterior to adapt its size over the course of the optimisation within the variational inference framework. However, realising this causes the problem that expectations for the evidence lower-bound (ELBO) then require computing an infinite summation which cannot be tackled using the reparameterisation trick (Williams, 1992; Kingma and Welling, 2014; Rezende et al., 2014). To surmount this problem, we use a different Monte Carlo approximation, namely, the Russian roulette sampling method from statistical physics (Lux and Koblinger, 1991; Carter and Cashwell, 1975; Lyne et al., 2015), which allows to approximate this sum by

[1]This effect is similar to the truncated approximation in that it seems to *automatically* determine the number of dimensions in the latent representation. However, it is considered as a failure case of training because otherwise the model can either manage (1) to use all the capacity (e.g. by scheduled annealing on the KL term to use all provided latent dimensions) or (2) to avoid perform unnecessary computation for dimensions that collapse to the prior as these dimensions do not help explain (or reconstruct) the data.

a sample from a Markov chain. This leads to an unbiased estimate of the gradient of the ELBO which can be maximised using stochastic gradient ascent.

We demonstrate RAVE on an infinite variational autoencoder Chatzis (2014), which assigns each data point to a continuous representation whose size is automatically inferred from the data set. The prior on the number of components is given by an Indian buffet process prior. We show empirically that previous amortised variational methods suffer from the component collapsing problem and tend to infer useless components, whereas RAVE leads to a model with many fewer components, while the overall model has a similar explanatory power. Importantly, RAVE achieves so within the variational inference framework without any outer-loop optimisation, which means it enjoys the theoretical benefits of variational inference in terms of consistency, convergence, etc.

4.2 Background

In this section, we review materials from nonparametric Bayesian statistics and variational inference that are used in developing and demonstrating RAVE.

4.2.1 Indian buffet process

An important problem in representation learning is to learn to represent each data item by a binary vector whose elements indicate latent *features* underlying the data. If the necessary number of features is unknown, we can take a Bayesian approach, and place a prior distribution over all possible latent feature matrices. One such prior distribution is the Indian buffet process (IBP), denoted $\mathbf{Z} \sim \mathrm{IBP}(\alpha)$, which is a probability distribution over sparse binary matrices with a finite number of rows and an unbounded number of columns (Griffiths and Ghahramani, 2011). We define the IBP using the stick-breaking construction (SBC) of Teh et al. (2007), which defines a distribution over $\mathbf{Z}$ by the following generative process for each entry z_{nk} of the matrix

$$\nu_k \sim \mathrm{Beta}(\alpha, 1), \quad \pi_k = \prod_{j=1}^{k} \nu_j, \quad z_{nk} \sim \mathrm{Ber}(\pi_k) \tag{4.1}$$

for $n \in 1 \ldots N$ and $k \in 1, 2, \ldots, \infty$, where z_{nk} is the n-th row and k-th column of $\mathbf{Z}$. Intuitively, we start with a stick of length 1 and break it at random to obtain a new stick of length ν_1. We then break this new stick at another random proportion ν_2 to obtain a new stick of length $\nu_1\nu_2$. We write $\mathbf{Z} \sim \mathrm{SBC}(\alpha, N, K)$ to indicate the distribution of the binary matrix $\mathbf{Z}$ if this process is stopped after K columns, where N is the number of data points which should be clear in context. We denote by IBP(α) the stochastic process that results from SBC(α, N, K) as $K \to \infty$.

4.2.2 Latent feature model

The IBP can be used as a prior over sparse latent representation $\mathbf{Z} = [\mathbf{z}_1 \dots \mathbf{z}_N] \in \mathbb{R}^{K\times N}$ of data $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_N] \in \mathbb{R}^{N\times D}$, where K is the size of the latent representation that can potentially go to infinity. Using this prior, we model the data as

$$\mathbf{Z} \sim \text{IBP}(\alpha), \quad \mathbf{A} \sim \mathcal{N}(0, \sigma_A^2 I), \quad \mathbf{X} \sim p_\theta(\mathbf{X} \mid \mathbf{Z}, \mathbf{A}), \tag{4.2}$$

where $\mathbf{A}$ is a feature matrix; a popular model arises when $\mathbf{A}$ is a matrix with K rows and D columns, and when $p_\theta(\mathbf{X} \mid \mathbf{Z}, \mathbf{A}) = \mathcal{N}(\mathbf{X} \mid \mathbf{ZA}, \sigma_{\mathbf{X}}^2 I)$. This is the well-studied linear Gaussian model. For this linear case, the prior on $\mathbf{A}$ is usually omitted (Chatzis, 2014; Singh et al., 2017) when doing amortised inference, and instead optimisation is done for $\mathbf{A}$, treating it as model parameters; I follow this choice in the experiments. As the Gaussian likelihood model has the same role of decoder in a variational autoencoder, I also refer to it as the linear decoder.

Alternatively, we can use a deep network to parameterised the likelihood $p_\theta(\mathbf{X} \mid \mathbf{Z}, \mathbf{A})$. Specifically, choose $p_\theta(\mathbf{X} \mid \mathbf{Z}, \mathbf{A}) = \mathcal{N}(\mathbf{X} \mid \mu_\theta(\mathbf{H}), \sigma_\theta(\mathbf{H}))$ or $p_\theta(\mathbf{X} \mid \mathbf{Z}, \mathbf{A}) = \text{Ber}(\mathbf{X} \mid p_\theta(\mathbf{H}))$ where the hidden representation $\mathbf{H} := \mathbf{Z} \odot \mathbf{A}$ and $\odot$ is the Hadamard product (Chatzis, 2014; Singh et al., 2017), μ_θ, σ_θ, and p_θ are multi-layer neural networks with weights θ. We refer to these three neural networks as a deep decoder. This choice of decoder leads to the infinite variational autoencoder, which we describe next.

4.2.3 Infinite variational autoencoders

The *infinite variational autoencoder* (infinite VAE) arises when apply variational inference to the deep latent feature model of the previous section (Chatzis, 2014). In that model, the posterior distribution over the latent variables $p(\mathbf{Z}, \mathbf{A}, \boldsymbol{\nu} \mid \mathbf{X})$ is intractable, so one popular approximate inference is variational inference. Singh et al. (2017) present a structured variational inference method for this model, based on the method of Hoffman and Blei (2015), which performs better than the more common mean-field approximation, as it introduces dependencies in the approximate posterior distribution, between $\mathbf{Z}$ and $\boldsymbol{\nu}$. The variational posterior from Singh et al. (2017) has the form

$$q(\mathbf{Z}, \mathbf{A}, \boldsymbol{\nu}_{(1:K)}) = q(\mathbf{A})q(\boldsymbol{\nu}_{(1:K)}) \prod_{n=1}^{N} \prod_{k=1}^{K} q(z_{nk} \mid \boldsymbol{\nu}_{(1:K)}),$$

where K is the truncation level. Each component of q has parameters, called *variational parameters*, which are optimised to make $q(\mathbf{Z}, \mathbf{A}, \boldsymbol{\nu}_{(1:K)})$ as close as possible to the true posterior $p(\mathbf{Z}, \mathbf{A}, \boldsymbol{\nu}_{(1:K)}|\mathbf{X})$ as measured by the KL-divergence. This is accomplished by optimising a lower bound called the *evidence lower-bound (ELBO)*. Optimising the ELBO requires sampling from a Monte Carlo estimate, which is designed to be differentiable with respect to the model parameters and the variational parameters. This can be made possible with the reparameterisation trick.[2] Singh et al. (2017) employs reparameterisation of the Beta distribution (Nalisnick and Smyth, 2017) and the Bernoulli distribution (Jang et al., 2016; Maddison et al., 2016). Singh et al. (2017) notices that using the reparameterisation tricks leads to better training than REINFORCE (Williams, 1992; Chatzis, 2014) for VAEs with IBP priors.

4.3 Roulette-based amortised variational expectation

Now we introduce RAVE, an amortised variational inference method based on random truncation. For concreteness, we describe RAVE in the context of a deep latent factor model with an IBP prior, but the method can be generally applied to models with a beta-Bernoulli process prior. First, we introduce a variational family in which the number of latent dimensions is random, governed by its own variational parameters (section 4.3.1); this is essentially an infinite mixture of truncated variational distributions. Then, we present the ELBO over all the variational parameters, showing that it can be written as an infinite sum (section 4.3.2). Then we show how Russian roulette sampling can be used to obtain an unbiased Monte Carlo estimate the gradient of this sum (section 4.3.3). Finally, we put all of these ideas together into a stochastic gradient optimisation algorithm that works on a finite representation of the infinite number of parameters (section 4.3.4).

[2] Alternatives include REINFORCE (Williams, 1992), which uses log-derivative trick to compute gradient and other reparameterisation tricks such as generalised reparameterisation gradient (Ruiz et al., 2016), and automatic differentiation variational inference (Kucukelbir et al., 2017).

4.3.1 Infinite-sized variational family

We start by describing the variational family of approximate posterior distributions that we consider in RAVE. Unlike previous amortised variational methods, the dimensionality is not bounded a priori, but is controlled by continuous variational parameters. We define the variational family using the stick-breaking construction as

$$\begin{aligned} &\nu_k \sim \mathrm{Beta}(\alpha_k, \beta_k),\ \pi_k = \prod_{j=1}^{k} \nu_j, \\ &K^* = k \ \text{ with probability } \ m_k = (1-\rho_{k+1}) \prod_{i=1}^{k} \rho_i, \\ &z_{nk} \sim \mathrm{Ber}(f_\phi(\pi_k, \mathbf{x}_n) \cdot \delta\{k \le K^*\}) \end{aligned} \tag{4.3}$$

for $n \in 1, \ldots, N$ and $k \in 1, 2, \ldots, \infty$.[3] We denote a single variational distribution in this family as $q(\boldsymbol{\nu}, K^*, \mathbf{Z} \mid \alpha, \beta, \phi, \rho)$. We refer this generative process as the modified stick-breaking process (MSBC) for the IBP. In this process, α_k, β_k, ρ_k, and ϕ are the variational parameters, and the neural network f_ϕ is an inference network that amortises the approximation of the posterior distribution. Also note that f_ϕ (the whole inference network) has a static-sized input and a varied-sized output, and f_θ (the whole generative network) has a varied-sized input and a static-sized output. One can also view this variational distribution as a mixture of infinitely many truncated ones.

In the rest of presentation, the only parameters we amortise are those for the variational distribution of $\mathbf{Z}$. The parameters of the inference network are an infinite sequence of vectors $\phi = (\phi_0, \phi_1, \ldots)$ with $\phi_k \in \mathbb{R}^{D+1}$. Then our inference network is

$$f_\phi(\pi_k, \mathbf{x}_n) = \sigma(\mathrm{logit}(\pi_k) + \phi_k^\top [\mathbf{x}_n, 1]), \tag{4.4}$$

where logit is the logit function and σ is the sigmoid function.[4] Note that f_ϕ outputs a scalar, essentially the approximate posterior distribution for a single hidden unit.

For the Beta and Bernoulli distributions, we use the Monte Carlo "reparameterisation trick" during training for gradient. We use the Kumaraswamy reparameterisation for the Beta distribution (Nalisnick and Smyth, 2017) and the Concrete reparameterisation for the Bernoulli distribution (Jang et al., 2016; Maddison et al., 2016), following Singh et al. (2017).

[3]Note that if we let $\rho_k = 1$ for all k, we recover the SBC for IBP(α) by letting $\beta = 1$.

[4]The notation $[\mathbf{x}_n, 1]$ represents vector concatenation, so that the dot product implicitly incorporates the bias term.

We have defined this variational family to have an infinite number of parameters: all of the variational parameters α, β, ρ and ϕ are infinite sequences. In order to optimise these parameters practically, observe for any integer k the conditional distribution $q(\boldsymbol{\nu}, \mathbf{Z} \mid K^* = k, \alpha, \beta, \phi, \rho)$ depends only on the first k variational parameters. It is this property that we use to approximate the ELBO in the next section.

This completes the description of the approximate posterior distribution $q(\boldsymbol{\nu}, \mathbf{Z}, K^*)$ for the parameters of the IBP prior. We also need a variational distribution $q(\mathbf{A})$ over the parameters $\mathbf{A}$ of the observation model, which we choose to be Gaussian. It is only amortised when using deep decoders while learned by optimisation for linear decoders (Chatzis, 2014; Singh et al., 2017).

Lazy dense layer A *lazy dense layer* is a dense layer that can take arbitrary size of input and make arbitrary size of output. Consider a normal dense layer $\ell(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W}$ has infinitely many rows and columns and $\mathbf{b}$ is an infinite long vector. One can make this layer take arbitrary dimension of $\mathbf{x} \in \mathbb{R}^{I \times B}$, where I is the input dimension and B is the batch size, by using the sub-matrix of $\mathbf{W}$ that has I columns. Similarly, one can make this layer output an arbitrary dimension O by using the sub-matrix of $\mathbf{W}$ that has O rows. In practise, $\mathbf{W}$ cannot be infinitely many rows and columns, but the rows and columns can be initialised (using standard Xavier initialisation (Glorot and Bengio, 2010)) as required during training.[5] This type of lazy dense layers are used to implement the inference and generative networks.[6]

4.3.2 Approximating the ELBO gradient

To find the best approximate posterior distribution, we maximise the ELBO

$$\begin{aligned}\mathcal{L} = &- \mathrm{D}_{\mathrm{KL}}[q(\boldsymbol{\nu}) \| p(\boldsymbol{\nu})] - \mathrm{D}_{\mathrm{KL}}[q(\mathbf{A}) \| p(\mathbf{A})] \\ &+ \mathbb{E}_{q_{\boldsymbol{\nu}}}\left[\mathbb{E}_{q_{\mathbf{Z}}}\left[\mathbb{E}_{q_{\mathbf{A}}}\left[\log \frac{p(\mathbf{X} \mid \mathbf{Z}, \mathbf{A}) p(\mathbf{Z} \mid \boldsymbol{\nu})}{q(\mathbf{Z} \mid \boldsymbol{\nu})}\right]\right]\right],\end{aligned} \tag{4.5}$$

[5]Note that there is a non-zero probability that, at inference time, a new "feature" can be initialised at random. This appears to have little effect to the overall method because the "importance" of that feature is really small.

[6]Note that an alternative to the proposed lazy dense layers here are recurrent neural networks. However, during initial investigation, they were found to be much harder to train thus not used in the final method.

which can be derived from the KL-divergence between the marginal distribution $q(\boldsymbol{\nu},\mathbf{Z},\mathbf{A})$[7] and the true posterior $p(\boldsymbol{\nu},\mathbf{Z},\mathbf{A}|\mathbf{X})$.[8] Optimising this function is challenging for several reasons. First, there are an infinite number of variational parameters, so we need to obtain a finite representation. Second, the distribution $q(\mathbf{Z} \mid \boldsymbol{\nu})$ in the third term is not easy to compute, because it is a marginal distribution $q(\mathbf{Z} \mid \boldsymbol{\nu}) = \sum_{k=0}^{\infty} q(\mathbf{Z}, K^* = k \mid \boldsymbol{\nu})$. Finally, optimising with respect to ρ is particularly challenging, intuitively because ρ determines the stochastic control flow of q; see algorithm 4.1. Specifically, there is no gradient for the parameters ρ_k of the auxiliary stopping variables from the terms under the expectation of $q_{\mathbf{Z}}$ (the second expectation in second line of equation 4.5), because they vanish once we apply the Monte Carlo approximation.

Computing the first two terms of equation 4.5 is straightforward (see appendix 4.A). For the third term, we re-write this expectation using the tower property (a.k.a. the law of total expectation)

$$\begin{aligned} &\mathbb{E}_{q_{\mathbf{A}}}\left[\mathbb{E}_{q_{\mathbf{Z}}}\left[\log \frac{p(\mathbf{X} \mid \mathbf{Z},\mathbf{A})p(\mathbf{Z} \mid \boldsymbol{\nu})}{q(\mathbf{Z} \mid \boldsymbol{\nu})}\right]\right] \\ =&\mathbb{E}_{q_{\mathbf{A}}}\left[\sum_{k=0}^{\infty} m_k \mathbb{E}_{q_{\mathbf{Z}}}\left[\log \frac{p(\mathbf{X} \mid \mathbf{Z},\mathbf{A})p(\mathbf{Z} \mid \boldsymbol{\nu})}{q(\mathbf{Z} \mid \boldsymbol{\nu})} \mid K^* = k\right]\right]. \end{aligned} \tag{4.6}$$

Because the expectation now conditions on $K^* = k$, we know that $\mathbf{Z}$ will have at most k nonzero columns, and so the numerator within the expectation is now computable. The denominator $q(\mathbf{Z} \mid \boldsymbol{\nu})$ is still challenging, because it marginalises out K^*, and still contains an infinite sum. We can obtain a slightly looser variational lower bound using the inequality

$$q(\mathbf{Z} \mid \boldsymbol{\nu}) = \sum_{j=1}^{\infty} m_j q(\mathbf{Z} \mid K^* = j, \boldsymbol{\nu}) \leq q(\mathbf{Z} \mid K^* = K^\dagger, \boldsymbol{\nu}), \tag{4.7}$$

where $K^\dagger := \max\{k \mid \exists n, z_{nk} \neq 0\}$, the maximum column index for which that column of $\mathbf{Z}$ is not all 0s. The inequality comes from two facts. First, $q(\mathbf{Z}|K^* = j, \boldsymbol{\nu}) = 0$ for $j < K^\dagger$ because it is impossible to generate more than j features if the process is truncated at j. Second, $q(\mathbf{Z}|K^* = j, \boldsymbol{\nu})$ is a monotonically decreasing function for $j \geq K^\dagger$ because observing more columns of zeros will only decrease the probability under the Bernoulli distribution. A more detailed proof is provided in the appendix 4.B.

[7]We have omitted the dependence of q on the variational parameters for brevity.

[8]Note that here we present the formulation in which $\mathbf{A}$ is treated as a latent variable, which is the case we assume for deep decoder. For the linear model, we do optimisation for $\mathbf{A}$ thus the corresponding KL term and expectation disappear.

Combining equation 4.5–equation 4.7, we obtain the training objective

$$\tilde{\mathcal{L}} = \sum_{k=0}^{\infty} m_k \tilde{\mathcal{L}}^k, \tag{4.8}$$

where

$$\begin{aligned} \tilde{\mathcal{L}}^k &= -\mathrm{D}_{\mathrm{KL}}[q(\boldsymbol{\nu}) \| p(\boldsymbol{\nu})] - \mathrm{D}_{\mathrm{KL}}[q(\mathbf{A}) \| p(\mathbf{A})] \\ &+ \mathbb{E}_{q_{\mathbf{A}}}\left[\mathbb{E}_{q_{\boldsymbol{\nu}}}\left[\mathbb{E}_{q_{\mathbf{Z}}}\left[\log \frac{p(\mathbf{X} \mid \mathbf{Z}, \mathbf{A}) p(\mathbf{Z} \mid \boldsymbol{\nu})}{q(\mathbf{Z} \mid K^* = K^\dagger, \boldsymbol{\nu})} \mid K^* = k\right]\right]\right]. \end{aligned} \tag{4.9}$$

Note that we also move the KL terms for $\boldsymbol{\nu}$ and $\mathbf{A}$ inside the infinite summation. This is valid as the infinite summation is an expectation. This expectation cannot be computed exactly, but we will present a method for approximating it in the next section.

When RAVE is used with the IBP, we will refer to the overall method as RRS-IBP, where RR stands for Russian roulette and S stands for either structured or sampling, at the reader's option.

Interpretation as a mixture of random truncation We give another interpretation of the ELBO in equation 4.8. During training, $\tilde{\mathcal{L}}^k$ is exactly the ELBO for the truncated variational method with truncation level k (Singh et al., 2017). Therefore, the lower bound $\tilde{\mathcal{L}}$ that we use can be interpreted as the expectation of the truncated ELBO, where the expectation is taken over our variational distribution $q(K^* = k) = m_k$ over the truncation level.

4.3.3 Russian roulette estimation of the ELBO gradient

Finally, we describe how we optimise the ELBO $\tilde{\mathcal{L}}$. To simplify the presentation, we introduce the notation $\psi_k = (\alpha_k, \beta_k, \phi_k, \theta_k)$[9], the vector of the variational and model parameters for component k, except for ρ_k, and we define the matrix $\psi_{1:k} = (\psi_1 \ldots \psi_k)$. Then, $\tilde{\mathcal{L}}$ has the form

$$\tilde{\mathcal{L}} = \sum_{k=1}^{\infty} m_k T_k(\psi_{1:k}), \tag{4.10}$$

where $m_k = (1 - \rho_{k+1}) \prod_{i=1}^{k} \rho_i$ depends only on $\rho_{1:k+1}$, and T_k depends only on the other parameters $\psi_{1:k}$. This can be optimised by stochastic gradient ascent if we can obtain an unbiased estimate of its gradient.

[9]We assume ϕ_k or θ_k subsumes the corresponding parameter of $q(\mathbf{A})$ when it is modelled, for the deep and linear model respectively.

First, the gradient with respect to ψ_k is

$$\partial_{\psi_k} := \frac{\partial \tilde{\mathcal{L}}}{\partial \psi_k} = \sum_{i=k}^{\infty} m_i \frac{\partial T_i}{\partial \psi_k}. \tag{4.11}$$

We assume each $\frac{\partial T_i}{\partial \psi_k}$ can be computed by standard automatic differentiation techniques. To estimate this, we use a Russian roulette estimate $\hat{\partial}^R_\psi$ with probabilities m_t. More specifically, we sample τ with probability $\mathbb{P}(\tau = t) = m_t$, and then return the estimate $\hat{\partial}^R_{\psi_k} = \hat{\partial}^\tau_{\psi_k}$, where

$$\hat{\partial}^\tau_{\psi_k} = \sum_{i=k}^{\tau} \frac{m_i}{\left(\prod_{j=1}^{i} \rho_j\right)} \frac{\partial T_i}{\partial \psi_k} = \sum_{i=k}^{\tau} (1 - \rho_{i+1}) \frac{\partial T_i}{\partial \psi_k}. \tag{4.12}$$

To derive the derivatives for ρ_k, first the chain rule yields

$$\partial_{\rho_k} := \frac{\partial \tilde{\mathcal{L}}}{\partial \rho_k} = \frac{\partial \mathcal{L}}{\partial m} \frac{\partial m}{\partial \rho_k} = \sum_{i=1}^{\infty} \frac{\partial \tilde{\mathcal{L}}}{\partial m_i} \frac{\partial m_i}{\partial \rho_k} = \sum_{i=1}^{\infty} T_i \frac{\partial m_i}{\partial \rho_k}, \tag{4.13}$$

where $\frac{\partial \mathcal{L}}{\partial m_i} = T_i$ and

$$\frac{\partial m_i}{\partial \rho_k} = \begin{cases} 0 & i < k-1 \\ -\frac{m_i}{1-\rho_k} & i = k-1 \\ \frac{m_i}{\rho_k} & i > k-1 \end{cases}. \tag{4.14}$$

This yields

$$\partial_{\rho_k} = \sum_{i=k-1}^{\infty} m_i w_i T_i, \quad w_i = \begin{cases} \frac{1}{\rho_k - 1} & i = k-1 \\ \frac{1}{\rho_k} & i > k-1 \end{cases}. \tag{4.15}$$

where we use the fact that $\frac{\partial m_i}{\partial \rho_k} = 0$ for $k \geq i$.

Finally, the Russian roulette estimate $\hat{\partial}^R_{\rho_k}$ is

$$\hat{\partial}^\tau_{\rho_k} = \sum_{i=k-1}^{\tau} \frac{m_i w_i T_i}{\left(\prod_{j=1}^{i} \rho_j\right)} = \sum_{i=k-1}^{\tau} (1 - \rho_{i+1}) w_i T_i \tag{4.16}$$

with probability $\mathbb{P}(\tau = t) = m_t$.

Now, each T_i is still difficult to compute, because it contains the expectation

$$\mathbb{E}_{q_{\mathbf{A}}}\left[\mathbb{E}_{q_{\boldsymbol{\nu}}}\left[\mathbb{E}_{q_{\mathbf{Z}}}\left[\log \frac{p(\mathbf{X} \mid \mathbf{Z}, \mathbf{A}) p(\mathbf{Z} \mid \boldsymbol{\nu})}{q(\mathbf{Z} | K^* = K^\dagger, \boldsymbol{\nu})} \mid K^* = i\right]\right]\right].$$

We obtain a Monte Carlo approximation of both of these expectations using the standard reparameterisation tricks for beta-Bernoulli processes.

In general, the Russian roulette estimation can have high variance, but what makes it so nice for variational inference is that because we are using a stick breaking construction, the earlier terms in the summation are the most important to include in the estimate.[10]

4.3.4 Stochastic gradient algorithm

Now that we have Monte Carlo estimates of the necessary derivatives, we can define the stochastic gradient algorithm. The key point is how we operate with only a finite representation of the variational parameters; essentially, we lazily instantiate only the finite subset of variational parameters that receive stochastic gradient updates. More specifically, at every point in the optimisation algorithm, we maintain a finite representation of the variational parameters $\psi = (\psi_0 \dots \psi_L)$ and $\rho = (\rho_0 \dots \rho_L)$. These matrices will lazily grow in size as needed, that is, L will grow over the course of the optimisation. At the beginning of the algorithm $L = 0$, each iteration of stochastic gradient ascent computes new parameters (ψ', ρ') from the current values (ψ, ρ). To do this, first we sample τ from the distribution $\mathbb{P}(\tau)$ defined in the previous section. Importantly, it can happen that $\tau > L$, which means that the current iteration will introduce new parameters, which are initialised to default values. To make this clear, see algorithm 4.1. Given a value of τ, we make the gradient updates

$$\begin{aligned} \psi'_k &\leftarrow \psi_k + \varepsilon_0 \hat{\partial}^{\tau}_{\psi_k}, \quad & k \in 1, 2, \dots \tau \\ \rho'_k &\leftarrow \rho_k + \varepsilon_1 \hat{\partial}^{\tau}_{\rho_k}, \quad & k \in 2, 3, \dots \tau + 1 \end{aligned}$$

where ε_0 and ε_1 are step sizes. We only need perform the updates for $k \in 1, 2, \dots \tau$ for ψ_k because $\hat{\partial}^{\tau}_{\psi_k} = 0$ if $k > \tau$, and similarly for $\hat{\partial}^{\tau}_{\rho_k}$. We enforce that $\rho_1 = 1$ to ensure $\tau > 0$. Note that the gradient steps for ρ are important to the method, because this determines the inferred number of features. Finally, in practice each gradient is an average over M independent roulette samples; Appendix 4.D describes how to reuse computation over the samples.

[10] It is also possible to estimate equation 4.11 and equation 4.14 via naive MC estimation as both of them can be written as an expectation under m. However, this estimate is of high variance which cannot be easily overcame by using more samples. We empirically illustrate this in appendix 4.C with comparison to our Russian roulette estimates.

Algorithm 4.1 Sampling the truncation level τ during variational optimisation, with lazy parameter initialisation.

Input: The IBP parameter α

```
1:  $t \leftarrow 0$
2:  loop
3:      $t \leftarrow t+1$
4:      if $t > L$ then
5:          Sample initial weights $\varepsilon_0 \sim \mathcal{N}(0, I)$
6:          $\rho_{t+1} \leftarrow 0.5$, $\alpha_t \leftarrow \alpha$, $\beta_t \leftarrow 1.0$, $\phi_t \leftarrow \varepsilon_0$, $\theta_t \leftarrow \varepsilon_0$
7:          $L \leftarrow L+1$
8:      end if
9:      return $\tau = t$ with probability $1 - \rho_{t+1}$
10: end loop
```

4.4 Experiments

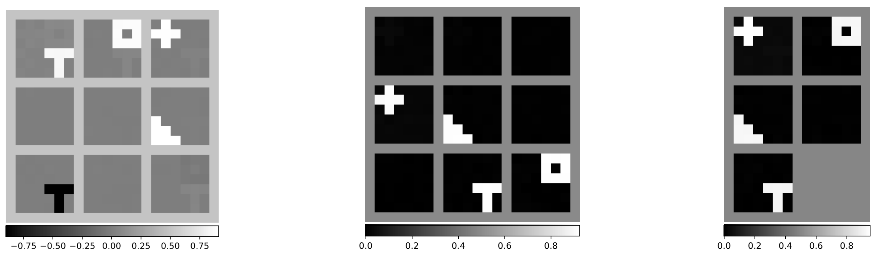
To evaluate the RRS-IBP method, we compare it with two previous amortised inference approaches for IBP models: mean-field Indian buffet process (MF-IBP) (Chatzis, 2014) and structured Indian buffet process (S-IBP) (Singh et al., 2017). Following Singh et al. (2017), a multiplier greater than 1 is put on the KL term for $\boldsymbol{\nu}$ during training for structured variational methods, to encourage adhering to the IBP prior. See appendix 4.E for other training details. Source code of the implementation of our method is also available at https://github.com/xukai92/RAVE.jl.

4.4.1 Synthetic data

First, in order to check the correctness of the inference, we compare the different inference methods on synthetic dataset, for which the true data distribution and number of components are known. This data set, proposed by Griffiths and Ghahramani (2011), contains 6×6 gray-scale images, each of which are generated as a linear combination of four black and white images with random weights, plus Gaussian noise $\mathcal{N}(0, 0.1^2)$. We sample 2,400 images for training and 400 held-out images for testing. [11]

All inference methods are applied to the linear-Gaussian IBP model (section 4.2.1).

[11]Different from Griffiths and Ghahramani (2011) in which each feature is activated with probability 0.5, we sample an activation matrix $\mathbf{Z} \sim \mathrm{SBC}(4.0, 2800, 4)$ (equation 4.1) and generate the dataset as $\mathbf{X} = \mathbf{ZA}$, where $\mathbf{A}$ is the predefined feature matrix.



(a) MF-IBP (b) S-IBP (c) RRS-IBP

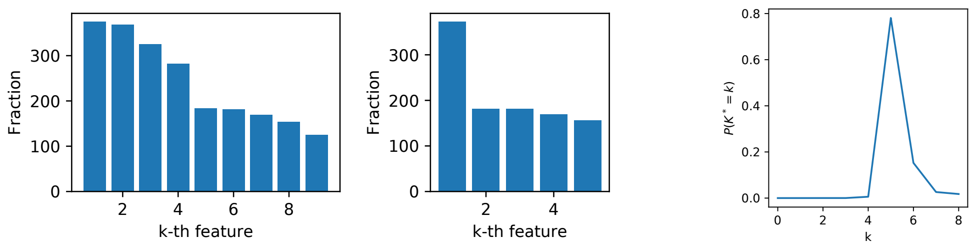
Figure 4.1: Features learned by VAEs ($\alpha = 4.0$)

We set $\alpha = 4$ so that the expected number of components is not the same as the true distribution. For MF-IBP and S-IBP, the truncation level is set to 9, greater than the number of true features in the data.

First, we are interested in whether the inference methods identify the correct number of features for this data. One way to measure this is by the expected number of features per images under the posterior distribution, which we define as $M = N^{-1}\sum_{n=1}^{N}\mathbb{E}_{q_{\mathbf{Z}}}\left[\sum_k z_{nk} \mid \mathbf{x}_n\right]$, where $\mathbf{x}_n$ are the test images. For the true generating process, $M = 2.275$. Because the training data set is large, and the model family contains the generating distribution, the variational approximations should identify a similar number of features. We find that $M = 5.647$ for MF-IBP and $M = 6.021$ for S-IBP, while $M = 3.464$ for RRS-IBP. In other words, both MF-IBP and S-IBP infer many more features than are present in the true distribution, while RRS-IBP infers a value that is closer to the true one.

It may be surprising that the methods are inferring such different values for the number of features, because all three approximate the same posterior distribution. To understand this better, we visualise the learned features in figure 4.1, which is simply the matrix $\mathbf{A}$ in the linear decoder. As we can see, S-IBP and RRS-IBP recover four black-and-white image features, which indeed are the images that were used to generate the data. MF-IBP recovers these four features up to a scaling of the image intensities, but also introduces an unnecessary negative feature (lower left in figure 4.1a). Additionally, both MF-IBP and SS-IBP infer several useless features, which we will refer as "dummy features".

Such dummy features do no harm if they are never activated, that is, if their corresponding column in $\mathbf{Z}$ is always zero. However, in 4.2a, we plot the activation frequencies, i.e. the number of features activated from a single sample of $q(\mathbf{Z} \mid \boldsymbol{\nu})$, inferred by S-IBP, for the testing set. This figure shows that the some top features



(a) S-IBP (b) RRS-IBP (c) $\mathbb{P}(K^* = k)$

Figure 4.2: Number activations per feature and truncation probability

from S-IBP are almost always activated, and in fact some are the black features in figure 4.1b. In other words, *meaningful* features in S-IBP do not necessarily come in order, as the method can infer "dummy" features.[12] On the other hand, RRS-IBP mostly avoids this issue (figure 4.2b), as it learns only one dummy feature and this feature is not always being activated. As we can see the four meaningful features for S-IBP and RRS-IBP actually has similar activation probabilities. We hypothesise that this activation of dummy feature phenomena comes from the fact that when training with a high truncation level, there are many local maxima.

The plot of the probability mass function of the stopping level in figure 4.2c shows that after training, the variational posterior for the truncation level puts most of its mass on the right level of truncation. This plot is not available for the other methods because their truncation level is fixed. "Dummy" features are less likely to be learned, compared with methods that require a large fixed truncation level because, in our method, only features under the truncation level get the training signal, and the truncation level is only optimised to be higher if during random truncation, it finds that having more features tends to be helpful.

4.4.2 Image data

Now we compare the inference methods on benchmark image data sets, namely the MNIST dataset of handwritten digits (LeCun, 1998; LeCun et al., 1998) and Fashion-MNIST, a dataset of images of products (Xiao et al., 2017). We use a deep decoder, where the prior is $p(\mathbf{A}) = \mathcal{N}(\mathbf{A}; 0, 1)$, and a Bernoulli observation distribution. Both the encoder and the decoder are two layer neural networks with 500 hidden units and rectified linear unit (ReLU) activation function. We also report the performance of the

[12]This was also observed by Singh et al. (2017) on a different synthetic data set (see their appendix).

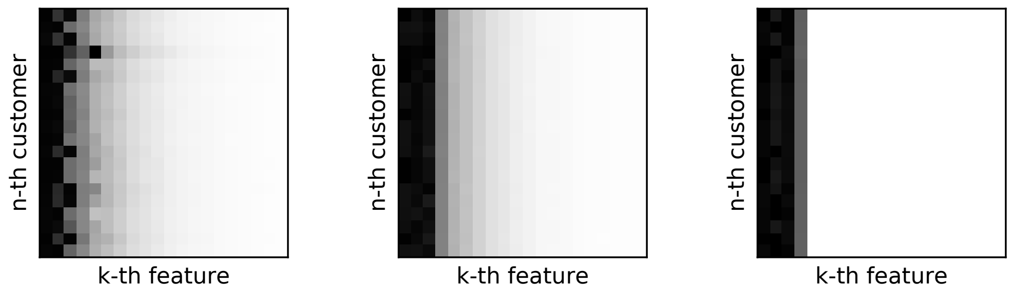
Table 4.1: IWAE for different VAEs under the IBP prior on various datasets.

Method	SYNTH			MNIST			Fashion-MNIST		
$(\alpha, K_{\max})$	$(4,20)$			$(10,50)$			$(20,50)$		
	Training	Test	$\tilde{K}$	Training	Test	$\tilde{K}$	Training	Test	$\tilde{K}$
MF-IBP	43.91	43.28	20	-111.17	-111.45	50	-241.99	-244.15	50
S-IBP	42.99	42.47	20	-103.83	-104.28	50	-239.24	-241.52	50
RRS-IBP	45.73	44.93	4	-103.54	-104.54	14	-237.84	-240.34	9

same model and architecture on the synthetic data from the last section; for that simpler data, we use hidden layer of size 50 and a Gaussian observation distribution.

We measure both the quality of the inferred models and the number of inferred features. To measure model quality, we estimate the marginal probability of the test using the variational bound from the importance weighted autoencoder (IWAE) (Burda et al., 2015), which is a tighter lower bound than the standard ELBO by using importance sampling. For S-IBP, we use the modification of the IWAE from Singh et al. (2017). For RRS-IBP, in order to compare against S-IBP using the same IWAE, we compute the mean of variational distribution of the truncation level, $\bar{m} = \mathbb{E}_{k \sim m_k}[k]$, and use $\lceil \bar{m} \rceil$ as the truncation level when computing the IWAE, where $\lceil \cdot \rceil$ is the ceiling function. To measure the number of inferred features, we report the number of features that have a non-negligible posterior activation probability, defined as $\tilde{K} = \sum_{k=1}^{K_{\max}} \max_{n \in \{1 \ldots N\}} \delta\{q(z_{nk} = 1) > \varepsilon\}$ with $\varepsilon = 0.01$. We will call $\tilde{K}$ the *number of activated features*. The results are shown in table 4.1. MF-IBP has the worst IWAE over all datasets, consistent with the results of Singh et al. (2017). The model quality of S-IBP and RRS-IBP are mostly similar. However, looking at $\tilde{K}$, we see that RRS-IBP infers many fewer features than the other methods, even though the overall model is of similar or better quality.

Visualising activation posterior probability We first verify our finding by visualising the activation posterior probability, $q_{\mathbf{Z}}$, for models on the synthetic dataset. We choose synthetic dataset for visualisation because the truncation level for other datasets are too high to populate readable plots. We visualise $q_{\mathbf{Z}}$ on a subset of the synthetic dataset by plotting the probabilities in grey-scale, which is shown in figure 4.3. We confirm our previous finding on the visualisation: MF-IBP spreads the activation prob-



(a) MF-IBP (b) S-IBP (c) RRS-IBP

Figure 4.3: Posterior feature activation probability in grey-scale for VAEs with deep decoders on a subset of SYNTH; darker the higher. The plot for RRS-IBP is padded to the same number of features for comparison.

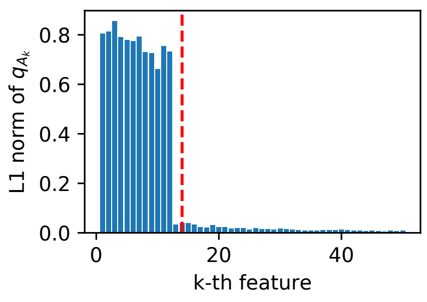
ability after its dark region on the left widely. S-IBP has a more compact dark region and the spread is much less. RRS-IBP has the smallest dark region and spreading effect.

Truncating collapsed components Another way to gain insight into whether the additional features inferred by S-IBP are indeed meaningless dummy features is to visualise the approximate posterior distributions for the S-IBP model. First, to understand $\mathbf{A}$, on the MNIST dataset[13] we plot the absolute value of the posterior mean of each feature k, $\mathbb{E}_{q_{\mathbf{A}}}[\mathbf{A}_k]$ ($\mathbf{A}_k$ for the k-th feature) in the order inferred by the model[14], averaged over the test set. This is shown in figure 4.4a. We see that the first 13 features have a posterior distribution $q_{\mathbf{A}_k}$ with a mean that is not close to 0; for other features, the posterior mean is very close to 0. This is component collapsing (Dinh and Dumoulin, 2016; van den Oord et al., 2017), a known phenomenon in VAEs, which means that some hidden units collapse to the prior and hence convey no information to the decoder. The vertical line in all the plots in figure 4.4 indicates the number of features inferred by RRS-IBP, the mode of $q(K^*)$. It is striking that RRS-IBP learns to truncate the model right after where S-IBP stops producing informative features.

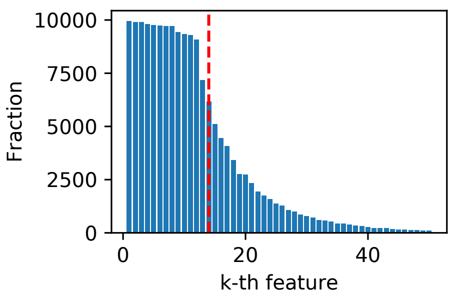
Now, to understand $\mathbf{Z}$, we show the activation frequencies, i.e. the number of times each feature is used, for the test set (figure 4.4b). This confirms that many of the uninformative features are activated for a substantial fraction of data points. This is similar to what we have visually seen in section 4.4.1 for the synthetic data.

[13]We conducted the same analysis on Fashion-MNIST and observed qualitatively similar results (see plots in appendix 4.F).

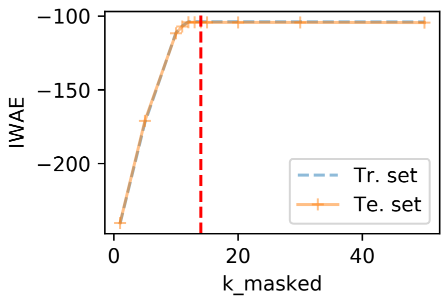
[14]Observe that because all the inference procedures are based on the stick breaking construction, this order is meaningful; the inferred features are automatically sorted by π_k.



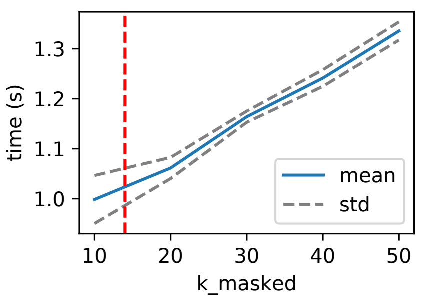
(a) Mean absolute value of $q_{\mathbf{A}}$



(b) Activation frequency of $\mathbf{Z}$



(c) IWAE with first k features



(d) Running time v.s. k

Figure 4.4: Effects of truncation level for S-IBP with a deep decoder. All plots are computed from the whole test set of MNIST dataset. The vertical line in red is the corresponding mode of the truncation distribution inferred by RRS-IBP.

As a final test of whether these uninformative features contribute to the quality of the inferred model, we report the test set IWAE of the S-IBP model if after training is complete, the model is truncated after k features (figure 4.4c). Figure 4.4c confirms our statement that the dummy features conveys no information to the decoder because the IWAE reaches its maximum right after all non-collapsed features are used. It is clear that the additional features do not help improve the inference results, and is a waste of computation, shown in figure 4.4d. As we have seen, with a trained S-IBP, it is hard for one to decide the optimal truncation level by only looking at the activation frequencies of $\mathbf{Z}$, without doing experiments like in figure 4.4a or figure 4.4c. However, RRS-IBP avoids this problem due to its nature of automatic truncation and the inferred truncation level (vertical line in red) is just the optimal level of truncation under our post analysis. Additionally, compared with a fixed truncation chosen at 50, the inferred network adapted with size 14 has ~ 1.35 times speed-up.

We can understand why S-IBP activates dummy features by considering the effect of dummy features on the ELBO. The additional KL penalty for the dummy features is very small, as $q_{\mathbf{A}_k}$ for the dummy features collapses to the prior. Second, the IBP prior we set encourages the inference network to use more features. E.g. under IBP(α), the expected number of total features is αH_N, where $H_N = \sum_{j=1}^{N} \frac{1}{j}$ and N is the data points, which is very large compared to the features all methods currently use. For this reason, activating more features would decrease the KL terms for $\boldsymbol{\nu}$ and $\mathbf{Z}$. This is actually the mode collapsing for IBP prior, because the latent variables from the inference network do not help the decoder at all. The only way to reduce this part of KL penalty for the IBP prior is to use a smaller truncation level, because in a truncated variational posterior, we assume $q_{\boldsymbol{\nu}}$ and $q_{\mathbf{Z}}$ beyond the truncation level collapse to the prior but they have no affect on the actual activations because the activations are set to 0 by truncation. However, for S-IBP the truncation level is fixed. Furthermore, the reconstruction term of the ELBO does not discourage dummy features too much because the dummy features mostly model noise and are ignored by the decoder. Also, for datasets in high dimension, the reconstruction dominates the ELBO, e.g. in MNIST, the reconstruction term is order of magnitude higher than the KL term for $\mathbf{A}$ and $\mathbf{Z}$ and 2 order of magnitudes higher than the KL term for $\boldsymbol{\nu}$, which also explains why there are obvious gaps in IWAEs only for the synthetic dataset in table 4.1 (the gaps for the other two datasets are marginal). As a result, the ELBO does not provide a large penalty for including these features. As an aside, we found that optimisation becomes difficult for MF-IBP and S-IBP with a large truncation. E.g. on SYNTH, we were unable to train these methods with a fixed truncation of 50. We hypothesise this is due to an increased number of local maxima introduced by the larger inference and generation networks. As a result, there is only small or no gradient to remove these dummy features during training for S-IBP.

As figure 4.4c indicates, the VAE only needs around 15 features to reach almost optimal IWAE, which can be also seen from figure 4.4a as features after the 15-th one collapsed to the prior. However, the inference network for $\mathbf{Z}$ still choose to activate those collapsed features. This is unwanted because activating a collapsed feature does not help with the inference quality nor the generative quality, which means it only can add more KL penalty without gaining any decrease in reconstruction. Having a way to large truncation is also a waste of computation, as shown in figure 4.4d.

4.5 Related work

IBP has been used to model choice behaviour, predict links, model dyadic data via binary matrix factorisation, extract features from similarity judgements, model protein interactions and in independent components analysis and sparse factor analysis; see Griffiths and Ghahramani (2011) for a thorough discussion. Recently, Valera et al. (2018) proposes a model with an IBP prior for tabular data for automatic data science; the model can infer data types and perform missing value imputation automatically. The IBP has been used in other areas like motion capture segmentation (Fox et al., 2014).

Variational inference for IBP latent feature models is proposed by Doshi-Velez et al. (2009) and relies on the stick-breaking construction from (Teh et al., 2007). Based on the construction, Teh et al. (2007) develops a slice sampler for IBP latent feature models. The sampler maintains a finite representation of SBC that can be used to compute the conditional probabilities required during the sampling. Our method is also based on the stick-breaking construction and in fact shares a similar spirit with the slice sampling: we maintain a finite representation that can be used to compute the unbiased gradient of ELBO during optimisation.

We are aware of only a few papers that apply amortised inference to IBP models. Chatzis (2014) uses black box variational inference (BBVI, Ranganath et al. (2014)) to train a VAE with an IBP prior using the mean-field variational approximation; BBVI is used due to the existence of Beta and Bernoulli distributions in SBC. Later Singh et al. (2017) propose a more accurate approximation by using the Kumaraswamy and Gumbel reparameterisation instead of BBVI, and by introducing a structured variational approximation instead of mean-field. We follow all of those innovations in this work. However, both of these papers rely on finite truncation in the variational distribution, which we avoid in this work. Another type of nonparametric variational autoencoder is proposed by Abbasnejad et al. (2017), who introduce an infinite mixture of finite-sized variational autoencoders.

In other Bayesian nonparametric models, some authors have reduced the impact of truncation by combining truncated variational inference with other optimisation methods; however, these methods perform discrete search over the truncation level outside the VI optimisation loop. In fact, discrete search for truncation levels based on various heuristics can be viewed as using a roulette probability that only has mass on truncation levels around the mode in our method. Such heuristics may introduce some bias to the overall learning but also have the merits of low-variance in many cases. For example, Bryant and Sudderth (2012); Hughes et al. (2015) use split and merge steps to change the size of the variational model, which results in a complex heuristic search algorithm. It is also possible to use heuristics to adaptively change the truncation level. Hu et al. (2015); Hughes and Sudderth (2013) also propose methods that adapt the truncation levels outside the VI optimisation loop in the context of memorised variational inference for Dirichlet process models. Similarly, Nalisnick and Smyth (2017) mention some initial experiments on making the level of stick-breaking adaptive by putting a threshold on the percentage of the remaining stick but reports that this approach is slow. We omit works in MCMC approaches with adaptive truncation levels as we focus only on works that use VI. Some MCMC approach use adaptive truncation levels (Teh et al., 2007; Griffiths and Ghahramani, 2011).

Within Bayesian statistics, Russian roulette sampling has been previously used within MCMC algorithms for doubly-intractable distributions (Lyne et al., 2015; Wei and Murray, 2016; Georgoulas et al., 2017). We are unaware of previous work applying Russian roulette sampling within variational methods. Indeed, to our knowledge, ours is the first variational inference method for Bayesian nonparametrics (whether amortised or the more traditional mean-field approach) that avoids a truncated variational approximation.

Appendix

4.A KL terms for $\mathbf{A}$ and $\boldsymbol{\nu}$

The computation for the KL terms for $\mathbf{A}$ and $\boldsymbol{\nu}$ in equation 4.5 is omitted in the main paper and we present here to show how to compute them.

The KL term for $\mathbf{A}$ is the KL between two Normal distributions $q_{\mathbf{A}} = \mathcal{N}(\mu_\phi(\mathbf{x}), \sigma_\phi(\mathbf{x}))$[15] and $p_{\mathbf{A}} = \mathcal{N}(0, I)$. We use its closed-form expression adapted from the appendix of Kingma and Welling (2014)

$$\mathrm{D}_{\mathrm{KL}}[q(\mathbf{A}) \| p(\mathbf{A})] = -\frac{1}{2} \sum_{k=1}^{\infty} \left(1 + \log\left((\sigma_k)^2\right) - (\mu_k)^2 - \sigma_k)^2\right) \tag{4.17}$$

where $\mu_k := \mu_{\phi_k}(\mathbf{x})$ and $\sigma_k := \sigma_{\phi_k}(\mathbf{x})$.

The KL term for $\boldsymbol{\nu}$ is the KL between the Kumaraswamy and Beta distributions $q(\nu_k) = \text{Kumaraswamy}(a_k, b_k)$ and $p(\nu_k) = \text{Beta}(\alpha, 1)$. We use its closed-form expression adapted from appendix of Nalisnick and Smyth (2017)

$$\mathrm{D}_{\mathrm{KL}}[q(\boldsymbol{\nu}) \| p(\boldsymbol{\nu})] = \sum_{k=1}^{\infty} \mathrm{D}_{\mathrm{KL}}[\text{Kumaraswamy}(a_k, b_k) \| \text{Beta}(\alpha, 1)], \tag{4.18}$$

where

$$\begin{aligned} &\mathrm{D}_{\mathrm{KL}}[\text{Kumaraswamy}(a, b) \| \text{Beta}(\alpha, \beta)] \\ &= \frac{a-\alpha}{a}\left(-\gamma - \Psi(b) - \frac{1}{b}\right) + \log ab + \log B(\alpha, \beta) - \frac{b-1}{b} + (\beta - 1) b \sum_{m=1}^{\infty} \frac{1}{m + ab} B\left(\frac{m}{a}, b\right). \end{aligned} \tag{4.19}$$

We approximate the infinite sum in equation 4.19 using its first 11 terms as suggested by Nalisnick and Smyth (2017).

[15]Note that here both μ_ϕ and σ_ϕ are implemented by the lazy dense layer such that the dimension of their outputs can change.

4.B Proof of the inequality in equation 4.7

Equation 4.7 states the inequality that

$$q(\mathbf{Z} \mid \mathbf{v}) = \sum_{j=1}^{\infty} m_j q(\mathbf{Z} \mid K^* = j, \mathbf{v}) \leq q(\mathbf{Z} \mid K^* = K^\dagger, \mathbf{v}), \tag{4.7}$$

where $K^\dagger := \max_k\{\exists n, z_{nk} \neq 0\}$, the maximum column index for which that column of $\mathbf{Z}$ is not all 0s.

To prove this, we begin with the definition of $q_{\mathbf{Z}}$ given a truncation level j

$$q(\mathbf{Z}|K^* = j, \mathbf{v}) := \delta\{j \geq K^\dagger\} \prod_{n=1}^{N} \prod_{k=1}^{j} \pi_k^{z_{nk}} (1-\pi_k)^{(1-z_{nk})}. \tag{4.20}$$

First,$q(\mathbf{Z} \mid K^* = j, \mathbf{v}) = 0$ for $j < K^\dagger$ because of the delta function that comes from the truncation. Second, $q(\mathbf{Z}^k \mid K^* = j, \mathbf{v})$ is a monotonically decreasing function for $j \geq K^\dagger$. To see this, consider $q(\mathbf{Z} \mid K^* = l, \mathbf{v})$ for which $K^\dagger \leq j < l$

$$\begin{aligned}
q(\mathbf{Z} \mid K^* = l, \mathbf{v}) &= \delta\{l \geq K^\dagger\} \prod_{n=1}^{N} \prod_{k=1}^{l} \pi_k^{z_{nk}} (1-\pi_k)^{(1-z_{nk})} \\
&= \prod_{n=1}^{N} \left(\prod_{k=1}^{j} \pi_k^{z_{nk}} (1-\pi_k)^{(1-z_{nk})} \prod_{k=j+1}^{l} \pi_k^{z_{nk}} (1-\pi_k)^{(1-z_{nk})} \right) \\
&= \left(\prod_{n=1}^{N} \prod_{k=1}^{j} \pi_k^{z_{nk}} (1-\pi_k)^{(1-z_{nk})} \right) \left(\prod_{n=1}^{N} \prod_{k=j+1}^{l} \pi_k^{z_{nk}} (1-\pi_k)^{(1-z_{nk})} \right) \\
&= q(\mathbf{Z}|K^* = j, \mathbf{v}) \left(\prod_{n=1}^{N} \prod_{k=j+1}^{l} \pi_k^{z_{nk}} (1-\pi_k)^{(1-z_{nk})} \right) \\
&=: q(\mathbf{Z}|K^* = j, \mathbf{v}) Q \\
&\leq q(\mathbf{Z} \mid K^* = j, \mathbf{v})
\end{aligned} \tag{4.21}$$

The last step comes from the fact that $Q <= 1$ since $\pi_k \in [0,1]$, $\forall k = 1, \ldots, \infty$. Now we can show

$$\begin{aligned}
q(\mathbf{Z} \mid \mathbf{v}) &= \sum_{j=1}^{\infty} m_j q(\mathbf{Z} \mid K^* = j, \mathbf{v}) \\
&= \sum_{j=1}^{K^\dagger - 1} m_j q(\mathbf{Z} \mid K^* = j, \mathbf{v}) + \sum_{j=K^\dagger}^{\infty} m_j q(\mathbf{Z} \mid K^* = j, \mathbf{v}) \\
&\leq \sum_{j=K^\dagger}^{\infty} m_j q(\mathbf{Z} \mid K^* = K^\dagger, \mathbf{v}) \\
&\leq \sum_{j=1}^{\infty} m_j q(\mathbf{Z} \mid K^* = K^\dagger, \mathbf{v}) = q(\mathbf{Z} \mid K^* = K^\dagger, \mathbf{v})
\end{aligned}$$

which completes the proof. □

4.C Comparison of Russian roulette estimation against naive Monte Carlo estimation

As a way to motivate the Russian roulette sampler, in this section we show the empirical evidence that, when estimating an infinite summation like equation 4.11 or equation 4.14, naive Monte Carlo can have very high variance which cannot be easily overcome by using more MC samples.

Suppose that we want to estimate $S = \lim_{K^* \to \infty} S_{K^*}$ where $S_{K^*} = \sum_{k=1}^{K^*} \mathbb{P}(K = k) T(k)$. In this illustration, we set $T(k) = T^*(k) + \mathcal{N}(0,1)$ where

$$T^*(k) = \begin{cases} (k-25)^2 & k < 25 \\ 0.01(k-25) & k \geq 25 \end{cases}.$$

in order to mimic a common loss curve for different sized truncation levels, and set $\mathbb{P}(K = k) = m_k = (1 - \rho_{k+1}) \prod_{i=1}^{k} \rho_i$ where

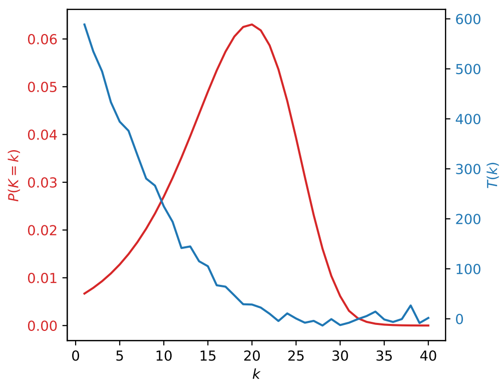
$$\rho_k = \begin{cases} \frac{1}{1+\exp(-5(k-1)/29)} & k < 30 \\ 0.5 & k \geq 30 \end{cases}.$$

in order to mimic a distribution of truncation level which is still away from an optimal one. Figure 4.C.1 shows $T(k)$ and $\mathbb{P}(K = k)$ (figure 4.C.1a) as well as how S_{K^*} changes with different level of K^* (figure 4.C.1b). As can be seen from figure 4.C.1b, S is approximately 96.

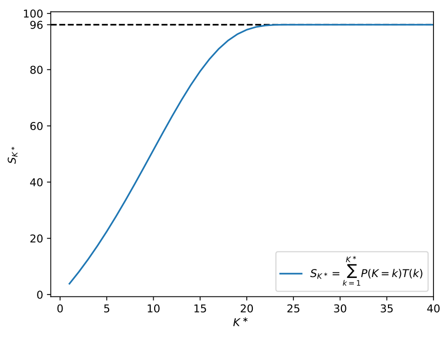
We compare to the native/crude Monte Carlo estimate in which we sample $k_1 \dots k_N$ independently from $\mathbb{P}(K)$ and compute

$$\hat{S}_{\text{MC}} = \frac{1}{N} \sum_{i=1}^{N} T(k_i).$$

We run each estimation for 100 times with the number of samples varying from 1 to 200. We report the mean and variance over the 100 runs. We also report the *efficiency* (Pharr et al., 2016) which is defined as $1/vt$ where v is the variance and t is the time required by the estimation. These results are given in figure 4.C.2. As one can see from the top plot for the mean, both estimates are unbiased towards the approximated true value. However, the variance of the naive Monte Carlo estimator is much higher.



(a) $\mathbb{P}(K=k)$ and $T(k)$



(b) S_{K^*}

Figure 4.C.1: Functions and the target summation in the toy example

This is shown more clearly in the plot in the middle, which shows how the variance asymptotically changes with the number of samples. It can be seen that the Russian roulette estimates have much lower variance than naive Monte Carlo. In fact, the naive Monte Carlo estimation retains a variance of 63.5 even with 200 samples while Russian roulette reaches a variance of around 10 with 15 samples. The final plot in the bottom compares the efficiency of the two estimators and it can be seen that the efficiency for the naive Monte Carlo one increases very slowly with more samples. Thus, we can conclude that using a naive Monte Carlo estimation for an infinite summation like equation 4.11 or equation 4.14 is of very high variance and the variance cannot be easily overcome by using more MC samples.

4.D Algorithmic description for effective RAVE

A naive implementation requires running the encoder and decoder once for each Russian roulette sample, but in this appendix we show how computation can be reused across samples. When we average over M independent Russian roulette samples, of each the truncation level is τ_m, the gradient estimate for ρ_k becomes

$$\hat{\partial}^M_{\rho_k} := \frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{\tau_m} a_i T_i, \tag{4.22}$$

where $T_i = \tilde{\mathcal{L}}^i$ and we define

$$a_i = (1-\rho_{i+1})w_i. \tag{4.23}$$

A naive way to compute the weighted summation of $\hat{\partial}^M_{\rho_k}$ is by running the encoder and decoder multiple times for each truncation level τ_m required and sum up, but this involves much wasted computation.

Instead, a re-weighting trick can be applied to reuse computation. We first draw M samples $\{\tau_m\}_{m=1}^M$ of the truncation level from the distribution defined by $\mathbb{P}(\tau_m = k) = m_k$. Then we compute $\tau^* = \max_m \tau_m$. We run the encoder once at the truncation level of $K^* = \tau^*$ and run the decoder multiple times (with truncation level $K^* = 1, \ldots, \tau^*$), and compute the final estimate as the weighted sum

$$\hat{\partial}^M_{\rho_k} = \sum_{i=1}^{\tau^*} b_i T_i, \tag{4.24}$$

where

$$b_i = \frac{1}{M} \sum_{m=1}^{M} a_i \delta(\tau_m \geq i). \tag{4.25}$$

It can be seen by reordering terms that this is equal to equation 4.22.

A similar re-weighting method applies to computing $\hat{\partial}^M_{\psi_k}$ with M samples, for which one only needs to re-weight each term in the forward pass of the EBLO computation as

$$\tilde{\mathcal{L}}^M = \sum_{i=1}^{\tau^*} c_i T_i, \text{ where } c_i = \frac{1}{M} \sum_{m=1}^{M} (1 - \rho_{i+1}) \delta(\tau_m \geq i) \tag{4.26}$$

and automatic differentiation can compute the gradient for all the parameters in the inference and generative network in equation 4.12.

Note that for the KL term, $\mathcal{K}_{1:i}$, in each $\tilde{\mathcal{L}}^i$ for $i = 1, \ldots, \tau^*$, as $\mathcal{K}_{1:i}$ is a sum of independent KL terms for each feature, $\mathcal{K}_j$, this term can be computed after the single run of encoder by $\mathcal{K}_{1:i} = \sum_{j=1}^{i} \mathcal{K}_j$, where $\mathcal{K}_j$ is the KL term of the j-th feature.

The complete algorithm that uses this re-weighting trick is given in algorithm 4.2. This requires a single run of the encoder and τ^* runs of the decoder. Additionally, equation 4.24 for all ks (i.e. Line 10 in algorithm 4.2) can be vectorised and implemented as matrix multiplications. We omit the details here and refer readers to our source code for concrete ways of vectorisation.

4.E Training details

For optimisation, we use Adam (Kingma and Ba, 2014) with a learning rate of 0.001 and momentum parameters 0.99 and 0.999, for all parameters except for ρ. For ρ we use a

Algorithm 4.2 Effective RAVE with re-weighting trick.

Input: a batch of B data points, $\{\mathbf{X}_i\}_{i=1}^{B}$, and the number of Russian roulette samples to use M

```
1:  for $i = 1, \dots, B$ do
2:      Sample $M$ samples $\{\tau_m\}_{m=1}^{M}$ from the truncation distribution $m_k$
3:      Compute $\tau^* = \max_m \tau_m$
4:      Compute $\{b_k\}_{k=1}^{\tau^*}$ following equation 4.25
5:      Encode $\mathbf{X}_i$ into variational distributions with a truncation level of $\tau^*$
6:      Compute KL between variational posterior and prior for each level $\{\mathcal{K}_{1:k}\}_{k=1}^{\tau^*}$
7:      for $k = 1, \dots, \tau^*$ do
8:          Compute the expected reconstruction term $\mathcal{R}_k$ in $T_k$ under variational distributions
9:      end for
10:     Compute ELBO $\tilde{\mathcal{L}}^i$ for each level $i$: $\{\tilde{\mathcal{L}}^i = \mathcal{R}_k - \mathcal{K}_{1:k}\}_{k=1}^{\tau^*}$
11:     Compute $\hat{\partial}_{\rho_k}^{M}$ for $k = 1, \dots, \tau^*$ using equation 4.24
12:     Compute and return the weighted ELBO $\tilde{\mathcal{L}}^M$ in equation 4.26 to automatic differentiation
13:     Obtain $\{\hat{\partial}_{\psi_k}^{M}\}_{k=1}^{\tau^*}$ from automatic differentiation
14:     Update $\{\rho_k, \psi_k\}_{k=1}^{\tau^*}$ using $\{\hat{\partial}_{\rho_k}^{M}, \hat{\partial}_{\psi_k}^{M}\}_{k=1}^{\tau^*}$ via gradient optimization methods
15: end for
```

stochastic gradient descent optimiser (Robbins and Monro, 1985) with a learning rate of 0.002. During training, the temperature of Concrete reparameterisation is set to 0.1. We found that in order to to use a low temperature, it is necessary to use high-precision 64-bit floating-point numbers. Using 32-bit floating point numbers with a temperature of 0.1 frequently results in numerical errors.

Note again that we use a multiplier on the KL term for $\boldsymbol{\nu}$ for structured variational methods during training to encourage adhering to the IBP prior, following Singh et al. (2017). It is set to 1,000 on both MNIST and FMNIST datasets. This has a similar effect of using a large α in the IBP prior. This increases the number of hidden units used, encouraging a better IWAE.

4.F Visualisation of component collapsing of S-IBP on Fashion-MNIST

For Fashion-MNIST, the corresponding component collapsing visualisation of figure 4.4 is given in figure 4.F.1. Note that worse than MNIST, there are even dummy features that are activated as frequently as active features, e.g. the 7-th features in figure 4.F.1a is shown to be frequently activated in figure 4.F.1. The number of active features shown in figure 4.F.1a is 8, which is the mode of the truncation distribution inferred by RRS-IBP, indicated in vertical red lines in both plots, which means RRS-IBP successfully inferred the same number of active features.

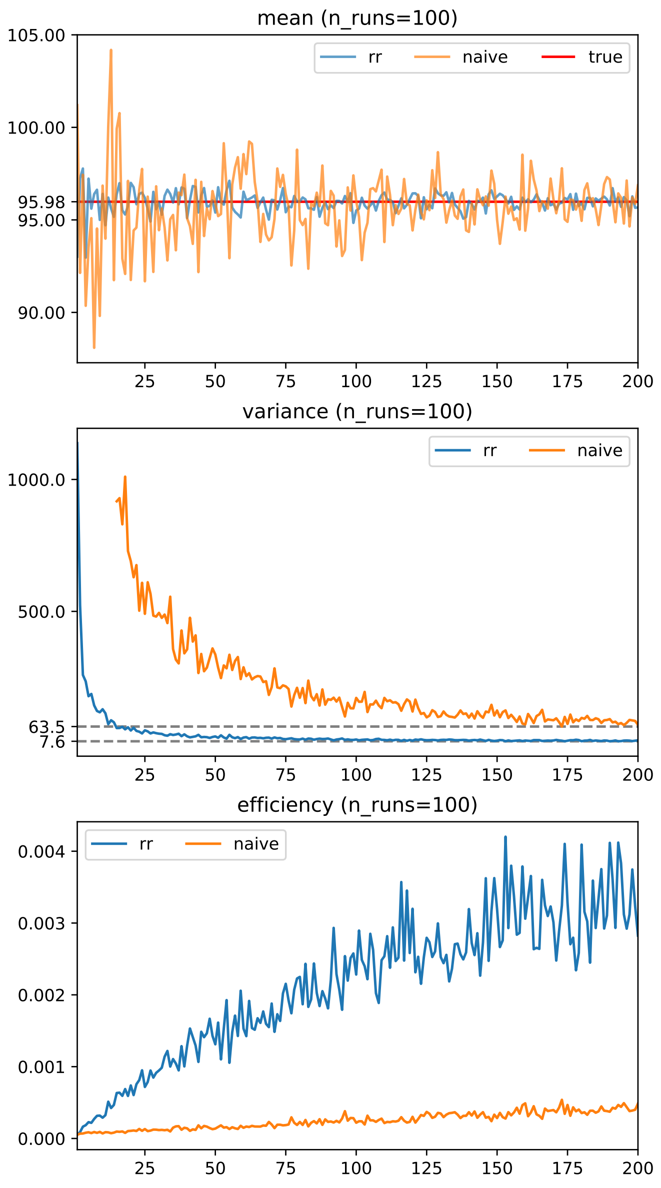
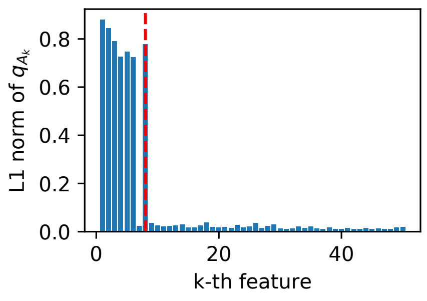
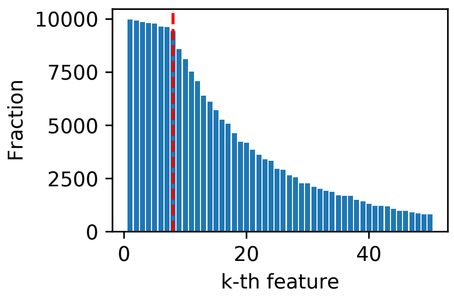


Figure 4.C.2: Mean, variance and efficiency for Russian roulette and naive Monte Carlo estimation. In all plots, *rr* stands for Russian roulette and *naive* stands for naive Monte Carlo. In the first plot, *true* represents the value of S that we are estimating.



(a) Mean absolute value of $q_{\mathbf{A}}$



(b) Activation frequency of $\mathbf{Z}$

Figure 4.F.1: Effects of truncation level for S-IBP with a deep decoder. All plots are computed from the whole testing set of Fashion-MNIST dataset. The vertical line at 8.0 in red is the corresponding truncation level learned by RRS-IBP.

Part II

Learning

Chapter 5

Generative Ratio Matching Networks

5.1 Introduction

Deep generative models (Kingma and Welling, 2014; Rezende et al., 2014; Goodfellow et al., 2014; Kingma and Dhariwal, 2018) have been shown to learn to generate realistic-looking images. These methods train a deep neural network, called a generator, to transform samples from a noise distribution to samples from the data distribution. Most methods use adversarial learning (Goodfellow et al., 2014), in which the generator is pitted against a critic function, also called a discriminator, which is trained to distinguish between the samples from the data distribution and from the generator. Upon successful training the two sets of samples become indistinguishable with respect to the critic.

Maximum mean discrepancy networks (MMD-nets; Li et al., 2015; Dziugaite et al., 2015) are a class of generative models that are trained to minimise the MMD (Gretton et al., 2012) between the true data distribution and the model distribution. MMD-nets are similar in spirit to generative adversarial networks (GANs; Goodfellow et al., 2014; Nowozin et al., 2016), in the sense that the MMD is defined by maximising over a class of critic functions. However, in contrast to GANs, where finding the right balance between generator and critic is difficult, training is simpler for MMD-nets because using the kernel trick the MMD can be estimated without the need to numerically optimise over the critic function. This avoids the need in GANs to numerically solve a saddle-point problem.

Unfortunately, although MMD-nets work well on low dimensional data, these networks have not on their own matched the generative performance of adversarial methods on higher dimensional datasets, such as natural images (Dziugaite et al., 2015).

Several authors (Li et al., 2017; Bińkowski et al., 2018) suggest that a reason is that MMD is sensitive to the choice of kernel. Li et al. (2017) propose a method called MMD-GAN, in which the critic maps the samples from the generator and the data into a lower-dimensional representation, and MMD is applied in this transformed space. This can be interpreted as a method for learning the kernel in MMD. The critic is learned adversarially by maximising the MMD at the same time as it is minimised with respect to the generator. This is much more effective than MMD-nets, but training MMD-GANs can be challenging, because in this saddle-point optimisation problem, the need to balance training of the learned kernel and the generator can create a sensitivity to hyper-parameters like network sizes or learning rates. In practice, it is necessary to introduce several additional penalties to the loss function in order for training to be effective.

In this work, we present a novel training method that builds on MMD-nets' insight to use kernels as fixed adversaries in order to avoid saddle-point optimisation based training for the critic and the generator. Our goal is for the critic to map the samples into a lower-dimensional space in which the MMD-net estimator will be more effective. Our proposal is that the critic should be trained to preserve density ratios, namely, the ratio of the true density to the model density. If the critic is successful in this, then matching the generator to the true data in the lower dimensional space will also match the distributions in the original space. We call networks that have been trained using this criterion ***g**enerative **ra**tio **m**atching* (GRAM) networks, or GRAM-nets[1]. We show empirically that our method is not only able to generate high quality images but by virtue of avoiding a zero-sum game (in critic and generator) it avoids saddle-point optimisation and hence is more stable to train and robust to the choice of hyper-parameters.[2]

5.2 Background

Given data $\mathbf{x}_i \in \mathbb{R}^D$ for $i \in \{1 \ldots N\}$ from a distribution of interest with density p_X, the goal of deep generative modelling is to learn a parameterised function $G_\gamma : \mathbb{R}^h \mapsto \mathbb{R}^D$, called a generator network, that maps samples $\mathbf{z} \in \mathbb{R}^h$ where $h < D$ from a noise distribution p_Z to samples from the model distribution. Since G_γ defines a new random variable, we denote its density function by q_X, and also write $\mathbf{x}^q = G_\gamma(\mathbf{z})$, where we

[1]Interestingly, the training of GRAM-nets heavily relies on the use of kernel Gram matrices.

[2]Official implementations are available at `https://github.com/GRAM-nets`

suppress the dependency of $\mathbf{x}^q$ on γ. The parameters γ of the generator are chosen to minimise a loss criterion which encourages q_X to match p_X.

5.2.1 Density ratio estimation

Given i.i.d samples from the data distribution p_X and a generative model of the data q_X, many learning and inference methods can be reduced to the estimation and thereafter the optimisation of the ratio $r(\mathbf{x}) = \frac{p_X(\mathbf{x})}{q_X(\mathbf{x})}$ of the data distribution p_X and the model distribution q_X. The task of estimating this ratio is called *density ratio estimation*. As surveyed by Sugiyama et al. (2012), a well known way of estimating this ratio is to train a classifier to distinguish between the samples from p_X and q_X. Upon convergence, as the classifier is Bayesian optimal, the ratio is simply a function of the classifier logits. Seen in this light, GANs essentially use the discriminator to ascertain a function of the ratio of the data and model densities. Then the generator parameters are trained to minimise the Jensen-Shannon divergence between p_X and q_X, which is in fact simply a function of the ratio that the discriminator estimates.

Huang et al. (2006); Sugiyama et al. (2012) present a density ratio estimator for the ratio between the densities p and q that has a closed form solution, based on kernel mean embedding. This estimator for $r(\mathbf{x}) = \frac{p(\mathbf{x})}{q(\mathbf{x})}$ only needs samples from p and q and is derived by optimising

$$\hat{r}(\mathbf{x}) = \underset{r \in \mathcal{R}}{\arg\min} \left\| \int k(\mathbf{x};.)p(\mathbf{x})\mathrm{d}\mathbf{x} - \int k(\mathbf{x};.)r(\mathbf{x})q(\mathbf{x})\mathrm{d}\mathbf{x} \right\|_{\mathcal{R}}^2, \tag{5.1}$$

where k is a kernel function. It is easy to see that at the minimum, we have $r = p/q$. In the so-called fixed design setup, in which we only seek for the ratio estimates for a set of samples from q rather than actually building a (parametric) model for $r(\mathbf{x})$, this estimate has a closed form solution (Huang et al., 2006; Sugiyama et al., 2012)

$$\hat{\mathbf{r}}_q = \mathbf{K}_{q,q}^{-1}\mathbf{K}_{q,p}\mathbf{1}$$

where $\mathbf{K}_{q,q}$ and $\mathbf{K}_{q,p}$ denote the kernel Gram matrices defined by $[\mathbf{K}_{q,q}]_{i,j} = k(\mathbf{x}_i^q, \mathbf{x}_j^q)$ and $[\mathbf{K}_{q,p}]_{i,j} = k(\mathbf{x}_i^q, \mathbf{x}_j^p)$ for some kernel k. Note that this closed-form estimator is not guaranteed to be non-negative nor guaranteed to provide normalised estimator (for a normalised density ratio estimator $\hat{r}$ we have $\mathbb{E}_{\mathbf{x}\sim q}[\hat{r}(\mathbf{x})] = 1$). Alternative, a positivity or a normalisation constraint can be imposed, for which equation 5.1 can to solved by standard constrained solvers.

Dimensionality reduction for density ratio estimation Sugiyama et al. (2011) suggest that density ratio estimation for distributions p and q over $\mathbb{R}^D$ can be more accurately done in lower dimensional sub-spaces $\mathbb{R}^K$. They propose to first learn a linear projection to a lower dimensional space by maximising an f-divergence between the distributions $\bar{p}$ and $\bar{q}$ of the projected data and then estimate the ratio of $\bar{p}$ and $\bar{q}$ (using direct density ratio estimation). They showed that the projected distributions preserve the original density ratio. Our method builds on this insight, generalising it to non-linear projections and incorporating it into a method for deep generative modelling.

5.3 Generative ratio matching

Our aim will be to enjoy the advantages of MMD-nets, but to improve their sample quality by mapping the data ($\mathbb{R}^D$) into a lower-dimensional space ($\mathbb{R}^K$), using a critic network f_θ, before computing the MMD criterion. Because MMD with a fixed kernel performs well for lower-dimensional data (Li et al., 2015; Dziugaite et al., 2015), we hope that by choosing $K < D$, we will improve the performance of the MMD-net. Instead of training f_θ using an adversarial criterion like MMD-GAN, we aim at a stable training method that avoids the saddle-point optimisation for training the critic.

More specifically, unlike the MMD-GAN type methods, instead of maximising the same MMD criterion that the generator is trained to minimise, we train f_θ to minimise the *squared ratio difference*, that is, the difference between density ratios in the original space and in the low-dimensional space induced by f_θ (section 5.3.1). More specifically, let $\bar{q}$ be the density of the transformed simulated data, i.e., the density of the random variable $f_\theta(G_\gamma(\mathbf{z}))$, where $\mathbf{z} \sim p_Z$. Similarly let $\bar{p}$ be the density of the transformed data, i.e., the density of the random variable $f_\theta(\mathbf{x})$. The squared ratio difference is minimised when θ is such that p_X/q_X equals $\bar{p}/\bar{q}$. The motivation is that if density ratios are preserved by f_θ, then matching the generator to the data in the transformed space will also match it in the original space (section 5.3.2). The reduced dimension of f_θ should be chosen to strike a trade-off between dimensionality reduction and ability to approximate the ratio. If the data lie on a lower-dimensional manifold in $\mathbb{R}^D$, which is the case for e.g. natural images, then it is reasonable to suppose that we can find a critic that strikes a good trade-off. To compute this criterion, we need to estimate density ratio $\bar{p}/\bar{q}$, which can be done in closed form using MMD (section 5.2.1). The generator is trained as an MMD-net to match the transformed data $\{f_\theta(\mathbf{x}_i)\}$ with transformed outputs from the generator $\{f(G_\gamma(\mathbf{z}_i))\}$ in the lower dimensional space (section 5.3.2). Our method performs stochastic gradient (SG) optimisation on the critic and the generator jointly (section 5.3.3).

5.3.1 Training the critic using squared ratio difference

Our principle is to choose f_θ so that the resulting densities $\bar{p}$ and $\bar{q}$ preserve the density ratio between p_X and q_X. We will choose f_θ to minimise the distance between the two density ratio functions

$$r_X(\mathbf{x}) = p_X(\mathbf{x})/q_X(\mathbf{x}) \qquad r_\theta(\mathbf{x}) = \bar{p}(f_\theta(\mathbf{x}))/\bar{q}(f_\theta(\mathbf{x})).$$

One way to measure how well f_θ preserves density ratios is to use the squared distance

$$D^*(\theta) = \int q_X(\mathbf{x}) \left(\frac{p_X(\mathbf{x})}{q_X(\mathbf{x})} - \frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))} \right)^2 \mathrm{d}\mathbf{x}. \tag{5.2}$$

This objective is minimised only when the ratios match exactly, that is, when $r_X = r_\theta$ for all x in the support of q_X. Clearly a distance of zero can be trivially achieved if $K = D$ and if f_θ is the identity function.But non-trivial optima can exist as well. For example, suppose that p_X and q_X are "intrinsically low dimensional" in the following sense. Suppose $K < D$, and consider two distributions p_0 and q_0 over $\mathbb{R}^K$, and an injective map $T : \mathbb{R}^K \to \mathbb{R}^D$. Suppose that T maps samples from p_0 and q_0 to samples from p_X and q_X, by which we mean $p_X(\mathbf{x}) = \boldsymbol{J}(T)p_0(T^{-1}(\mathbf{x}))$, and similarly for q_X. Here $\boldsymbol{J}(T)$ denotes the Jacobian matrix $\boldsymbol{J}(T) = \sqrt{|\delta T \delta T^\top|}$ of T. Then we have that D^* is minimised to 0 when $f_\theta = T^{-1}$.

However, it is difficult to optimise $D^*(\theta)$ directly because density ratio estimation in high dimension, where data lives, is known to be hard, i.e., the term $\frac{p_X(\mathbf{x})}{q_X(\mathbf{x})}$ in equation 5.2 is difficult to estimate. We will show how to alleviate this issue next.

Avoiding density ratio estimation in data space To avoid computing the term $\frac{p_X(\mathbf{x})}{q_X(\mathbf{x})}$ in equation 5.2, we expand the square in equation 5.2, apply the law of the unconscious statistician and cancel terms out (see appendix 5.A for detailed steps), which yields

$$\begin{aligned} D^*(\theta) &= C + \int \bar{q}(f_\theta(\mathbf{x})) \left(\frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))} \right)^2 d f_\theta(\mathbf{x}) - 2 \int \bar{p}(f_\theta(\mathbf{x})) \frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))} d f_\theta(\mathbf{x}), \\ &= C' - \left[\int \bar{q}(f_\theta(\mathbf{x})) \left(\frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))} \right)^2 d f_\theta(\mathbf{x}) - 1 \right] \end{aligned} \tag{5.3}$$

where C and $C' = C - 1$ does not depend on θ. This implies that minimising D^* is equivalent to maximising the Pearson divergence

$$\mathrm{PD}(\bar{p},\bar{q}) = \int \bar{q}(f_\theta(\mathbf{x}))\left(\frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))}\right)^2 df_\theta(\mathbf{x}) - 1 = \int \bar{q}(f_\theta(\mathbf{x}))\left(\frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))} - 1\right)^2 df_\theta(\mathbf{x}) \tag{5.4}$$

between $\bar{p}$ and $\bar{q}$, which justifies our terminology of referring to f_θ as a critic function. So we can alternatively interpret our squared ratio distance objective as preferring f_θ so that the low-dimensional distributions $\bar{p}$ and $\bar{q}$ are maximally separated under Pearson divergence. Therefore, D^* can be minimised empirically using samples $\mathbf{x}_1^q \dots \mathbf{x}_M^q \sim q_X$ to maximise the critic loss function

$$\mathcal{L}(\theta) = \frac{1}{M}\sum_{i=1}^{M}\left[r_\theta(\mathbf{x}_i^q) - 1\right]^2. \tag{5.5}$$

Optimising this requires a way to estimate $r_\theta(\mathbf{x}_i^q)$. For this purpose we use the density ratio estimator introduced in section 5.2.1. Notice that to compute equation 5.5, we need the value of r_θ *only for* the points $\mathbf{x}_1^q \dots \mathbf{x}_M^q$. In other words, we need to approximate the vector $\mathbf{r}_{q,\theta} = [r_\theta(\mathbf{x}_1^q) \dots r_\theta(\mathbf{x}_M^q)]^T$. Following Huang et al. (2006); Sugiyama et al. (2012), we replace the integrals in equation 5.1 with Monte Carlo averages over the points $f_\theta(\mathbf{x}_1^q) \dots f_\theta(\mathbf{x}_M^q)$ and over points $f_\theta(\mathbf{x}_1^p) \dots f_\theta(\mathbf{x}_N^p) \sim \bar{p}$; the minimising values of $\mathbf{r}_{q,\theta}$ can then be computed as

$$\hat{\mathbf{r}}_{q,\theta} = \mathbf{K}_{q,q}^{-1}\mathbf{K}_{q,p}\mathbf{1}. \tag{5.6}$$

Here $\mathbf{K}_{q,q}$ and $\mathbf{K}_{q,p}$ denote the kernel Gram matrices defined by $[\mathbf{K}_{q,q}]_{i,j} = k(f_\theta(\mathbf{x}_i^q), f_\theta(\mathbf{x}_j^q))$ and $[\mathbf{K}_{q,p}]_{i,j} = k(f_\theta(\mathbf{x}_i^q), f_\theta(\mathbf{x}_j^p))$.

Substituting these estimates into equation 5.5 and adding a positivity constraint $\hat{\mathbf{r}}_{q,\theta}^T.\mathbf{1}$ for using the MMD density ratio estimator (Sugiyama et al., 2012), we get

$$\hat{\mathcal{L}}(\theta) = \frac{1}{M}\sum_{i=1}^{M}\left[r_\theta(\mathbf{x}_i^q) - 1\right]^2 + \lambda\hat{\mathbf{r}}_{q,\theta}^T.\mathbf{1}, \tag{5.7}$$

where λ is a parameter to control the constraint, being set to 1 in all our experiments.[3] This objective can be maximised to learn the critic f_θ. We see that this is an estimator of the Pearson divergence $\mathrm{PD}(\bar{p},\bar{q})$ in that we are both, averaging over samples from q_X, and approximating the density ratio. Thus maximising this objective leads to the preservation of density ratio (Sugiyama et al., 2011).

[3]As it is pointed out by one of the reviewers, instead of adding the regularisation term, another way to resolve the non-negativity issue of the ratio estimator is to simply clipping it the estimator be positive. This in fact works well in practice and can further improves the stability of training.

Algorithm 5.1 Generative ratio matching

```
1: while not converged do
2:     Sample a mini-batch of data $\{\mathbf{x}_i^p\}_{i=1}^N \sim p_X$ and generated samples $\{\mathbf{x}_i^q\}_{i=1}^M \sim q_X$
3:     Using $f_\theta$ to transform data as $\{f_\theta(\mathbf{x}_i^p)\}_{i=1}^N$ and generated samples as $\{f_\theta(\mathbf{x}_i^q)\}_{i=1}^M$
4:     Compute the Gram matrix $\mathbf{K}$ under the kernel of choice in the transformed space
5:     Compute $\hat{\mathcal{L}}(\theta)$ via equation 5.6 and equation 5.7, and $\hat{\mathcal{L}}(\gamma)$ via equation 5.8 using the same $\mathbf{K}$
6:     Compute the gradients $g_\theta = \nabla_\theta \hat{\mathcal{L}}(\theta)$ and $g_\gamma = \nabla_\gamma \hat{\mathcal{L}}(\gamma)$
7:     $\theta \leftarrow \theta + \eta g_\theta$; $\gamma \leftarrow \gamma - \eta g_\gamma$        ▷ Perform SG optimisation for $\theta$ and $\gamma$
8: end while
```

5.3.2 Generator loss

To train the generator network G_γ, we minimise the MMD in the low-dimensional space, where both the generated data and the true data are transformed by f_θ. In other words, we minimise the MMD between $\bar{p}$ and $\bar{q}$. We sample points $\mathbf{z}_1, \ldots, \mathbf{z}_M \sim p_Z$ from the input distribution of the generator. Then using the empirical estimate equation 2.20 of the MMD, we define the generator loss function as

$$\begin{aligned}\hat{\mathcal{L}}^2(\gamma) =& \frac{1}{N^2}\sum_{i=1}^{N}\sum_{i'=1}^{N} k(f_\theta(\mathbf{x}_i), f_\theta(\mathbf{x}_{i'})) - \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} k(f_\theta(\mathbf{x}_i), f_\theta(G_\gamma(\mathbf{z}_j))) \\ &+ \frac{1}{M^2}\sum_{j=1}^{M}\sum_{j'=1}^{M} k(f_\theta(G_\gamma(\mathbf{z}_j)), f_\theta(G_\gamma(\mathbf{z}_{j'})))\end{aligned}, \qquad (5.8)$$

which we minimise with respect to γ.

5.3.3 The generative ratio matching algorithm

Finally, the overall training of the critic and the generator proceeds by jointly performing SG optimisation on $\hat{\mathcal{L}}(\theta)$ and $\hat{\mathcal{L}}(\gamma)$., which is shown in algorithm 5.1. Unlike WGAN (Arjovsky et al., 2017a) and MMD-GAN (Li et al., 2017), we do not require the use of gradient clipping, feasible set reduction or autoencoding regularisation terms. Our algorithm is a simple iterative process. By default for continuous data, we use the Gaussian kernel or a mixture of Gaussian kernels with a set of bandwidths in Line 4. We fount GRAM is insensitive to the absolute value of bandwidths as the projection network has the ability to change the scale of the projected space. Therefore, it is enough to choose the set of bandwidths with distinct relative values, such as $(0.1, 1, 10)$ or $(0.2, 1, 5)$.

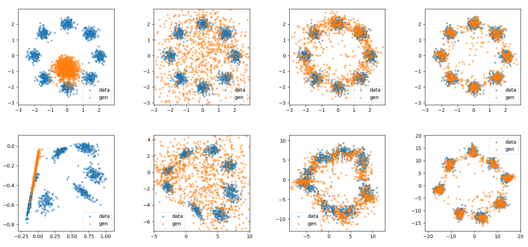
Convergence If we succeed in matching the generator to the true data in the low-dimensional space, then we have also matched the generator to the data in the original space, in the limit of infinite data. To see this, suppose that we have γ^* and θ^* such that $D^*(\theta^*) = 0$ and that $M_y = \text{MMD}(\bar{p}, \bar{q}) = 0$. Then for all $\mathbf{x}$, we have $r_X(\mathbf{x}) = r_{\theta^*}(\mathbf{x})$ because $D^*(\theta^*) = 0$, and that $r_{\theta^*}(\mathbf{x}) = 1$, because $M_y = 0$. This means that $r_X(\mathbf{x}) = 1$, so we have that $p_X = q_X$.

5.3.4 Stability of GRAM-nets

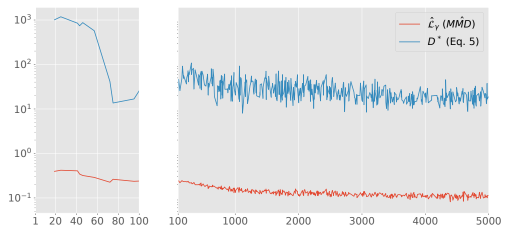
Unlike GANs, the GRAM formulation avoids the saddle-point optimisation which leads to a very stable training of the model. In this section we provide a battery of controlled experiments to empirically demonstrate that GRAM training relying on the empirical estimators of MMD and Pearson Divergence (PD), i.e. equations equation 5.5 and equation 5.7. While the MMD estimator is very extensively studied, our novel empirical estimator of PD is not. Therefore we now show that in fact maximizing our estimator for PD can be reliably used for training the critic, i.e. it minimizes equation equation 5.2.

Stability of adversarial methods has been extensively studied before by using a simple dataset of eight axis-aligned Gaussian distributions with standard deviations of 0.01, arranged in a ring shape (Roth et al., 2017; Mescheder et al., 2017b; Srivastava et al., 2017). Therefore we train a simple generator using GRAM on this 2D ring dataset to study the stability and accuracy of GRAM-nets. We set the projection dimension $K = D = 2$ in order to facilitate visualisation. Both the generator and the projection networks are implemented as two-layer feed-forward neural networks with two hidden layers of size of 100 and ReLU activations.

The generator output (orange) is visualised over the real data (blue) in figure 5.1a at 10,100,1000 and 10,000th training iterations. The top row visualises the observation space ($\mathbb{R}^D$) and the bottom row visualises the projected space ($\mathbb{R}^K$). Note, as the training progress the critic (bottom row) tries to find projections that better separate the data and the generator distributions (especially noticeable in columns 1 and 3). This provides a clear and strong gradient signal to the generator optimisation that successfully learns to minimise the MMD of the projected distributions and eventually the data and the model distributions as well (as shown in the last column). Notice, how in the final



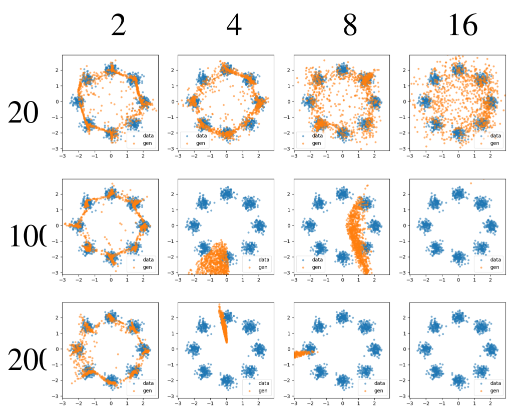
(a) Data and samples in the original (top) and projected space (bottom) during training; four plots are at iteration 10, 100, 1000 and 10,000 respectively. Notice how the projected space separates $\bar{p}$ and $\bar{q}$.



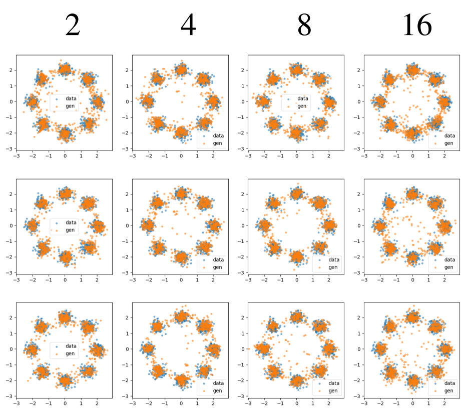
(b) Trace of $\hat{\mathcal{L}}_\gamma$ and D^* (equation equation 5.3) during training. The left plot is for iteration 1 to 100 and the right plot is for 100 to 5,000, with the same y-axes in the log scale.

Figure 5.1: Training results with projected dimension fixed to 2.

column the critic is no longer able to separate the distributions. Throughout this process the critic and the generator objectives are consistently minimised. This is clearly shown in figure 5.1b which records the values of equation 5.8 for the generator (orange) loss and equation 5.3 for the critic objective (blue). Next, we compare GRAM-based training against classical adversarial training that involves a zero-sum game. For this purpose we train a GAN with the same exact architecture and hyper-parameter as those used for the GRAM-nets in the previous experiments. Adversarial models are known to be unstable on the 2D ring dataset (Roth et al., 2017; Mescheder et al., 2017b) as the critic can often easily outperform the generator in this simple setting. Figure 5.2 summarises the results of this comparison. Both models are trained on three different levels of generator capacity and four different dimensions of the input noise variable (h). GANs are known to be unstable at both low and high dimensional noise but they additionally tend to mode-collapse (Srivastava et al., 2017) on high dimensional noise. This is confirmed in our experiments; GANs failed to learn the true data distribution in every case and in most cases the optimisation also diverged. In sharp contrast to this, GRAM training successfully converged in all 12 settings. Adversarial training needs a careful balance between the generator and the critic capacities. In the plot we can see that, as the capacity of the generator becomes larger, the training become more unstable in GAN. This is not the case for GRAM-nets, which train well without requiring to make any adjustments in other hyper-parameters. This enlists as an important advantage of GRAM, as one can use the most powerful model (generator and critic) given their computational budget instead of worrying about balancing the optimisation. We also provide the corresponding results for MMD-nets and MMD-GANs in appendix 5.B.2,



(a) GANs



(b) GRAM-nets

Figure 5.2: Training after 2,000 epochs by varying noise dimension h and the hidden layer size of critic model. For each model, each row is a different layer size in $[20, 100, 200]$ and each column is a different h in $[2, 4, 8, 16]$. Half of the GAN training diverges while all GRAM training converges.

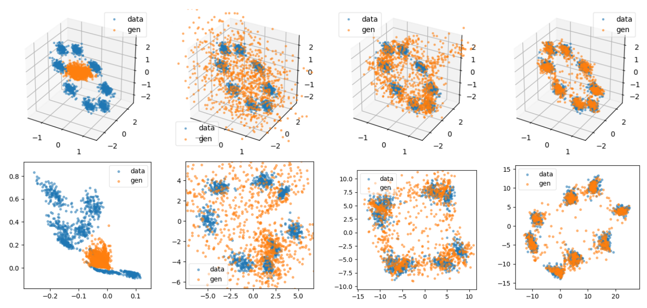
in which one can see MMD-nets can also capture all modes, but the learned distribution is less separated (or sharp) compared to GRAM-nets and MMD-GANs tend to produce distributions which are too sharp.

5.3.4.1 3D ring dataset

The same experiment as in figure 5.1a is repeated with a 3-dimensional data space and a 2-dimensional projection space to provide further insights of how the projection network works. We create a 3D ring dataset by augmenting the third dimension with Gaussian noise (0 mean and 0.1 standard deviation), which is then rotated by 60 degrees along the second axis. We repeat the experiments in figure 5.1a using this dataset. The results are shown in figure 5.3. Unlike the 2D ring example, the optimal choice of the projection function learned here is no longer the identity function. However, our method can stil, easily learn a low-dimensional manifold that tends to preserve the density ratio.

5.3.4.2 MNIST dataset

In this section, we show the results of training GRAM-nets on a more realistic dataset—the MNIST dataset; the hyperparameters for this experiment can be found in appendix 5.B.3. Figure 5.4 shows how the generated samples change during the phase of training. This results confirms that GRAM is not only applicable synthetic datasets



(a) Data and samples in the original (top) and projected space (bottom) during training; four plots are at iteration 0, 100, 1000 and 10,000 respectively. Notice how the projected space separates $\bar{p}$ and $\bar{q}$.

Figure 5.3: Training results of GRAM-nets with projected dimension fixed to 2 on the 3D ring dataset.

that are shown before but also to real dataset like MNIST. Figure 5.5 shows how the generator loss and the ratio matching objective are being optimised simultaneously. As for dataset that has a similar dimensionality of MNIST, kernel methods are still statistically effective thus we can still estimate the squared difference in density ratios in equation 5.2. Figure 5.5 confirms that during training, this squared difference decreases, meaning the projection network is learning a space that matches the density ratio.

5.3.5 Computation graphs

Figure 5.6 visualises the computational graph for GAN, MMD-net, MMD-GAN and GRAM-net. Solid arrows describe the direction of the computation, **K** denotes the kernel gram matrix and most importantly dashed lines represent the presence of saddle-point optimisation. In case of GAN and MMD-GAN, the dashed lines imply that by formulation, training the critic adversarially affects the training of the generator as they are trained by minimising and maximising the same objective i.e., $L_\gamma = -L_\theta$. Both MMD-nets and GRAM-nets avoid this saddle-point problem. In GRAM-nets, the critic and generator do not play a zero-sum game as they are trained to optimise different objectives, i.e. the critic learns to find a projection in which the density ratios of the pair of input distributions are preserved after the projection and the generator tries to minimise the MMD in the projected space.

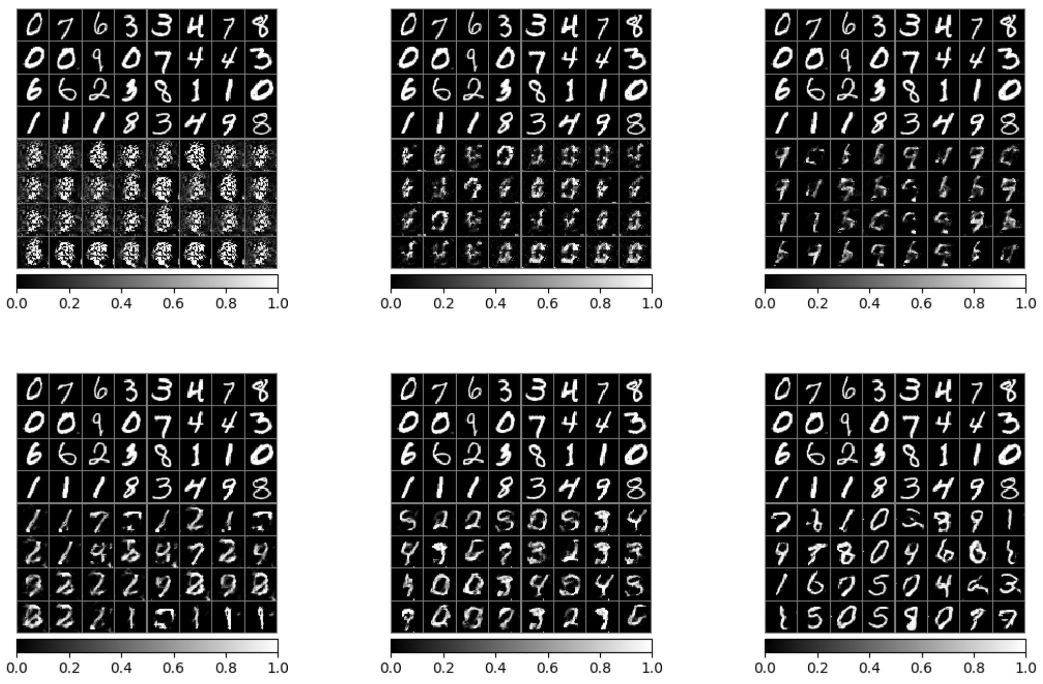


Figure 5.4: Data and samples (top and bottom half in each plot) during training at iteration 100, 250, 500 (top row) and at 750, 1,000, 4,000 (bottom row). The orders for each row are from left to right.

5.4 Experiments

In this section we empirically compare GRAM-nets against MMD-GANs and vanilla GANs, on the Cifar10 and CelebA image datasets. Please note that while we have tried to include maximum number of evaluations in this section itself, due to space limitations, some results are made available in the appendix. To evaluate the sample quality and resilience against mode dropping, we used Frechet Inception Distance (FID; Heusel et al., 2017).[4] Like the Inception Score (IS), FID also leverages a pre-trained Inception Net to quantify the quality of the generated samples, but it is more robust to noise than IS and can also detect intra-class mode dropping (Lucic et al., 2017). FID first embeds both the real and the generated samples into the feature space of a specific layer of the pre-trained Inception Net. It further assumes this feature space to follow a multivariate Gaussian distribution and calculates the mean and covariance for both sets of embeddings. The sample quality is then defined as the Frechet distance between the

[4]We use a standard implementation available from https://github.com/bioinf-jku/TTUR

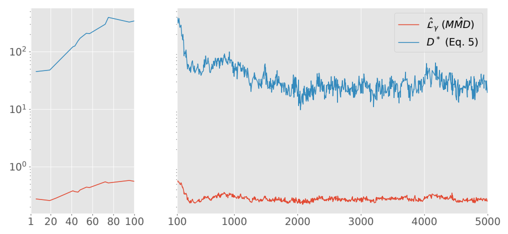


Figure 5.5: Trace of $\hat{\mathcal{L}}_\gamma$ and D^* (equation equation 5.3) during training. The left plot is for iteration 1 to 100 and the right plot is for 100 to 5,000, with the same y-axes in the log scale.

$$\begin{tikzcd}
\mathbf{z} \sim p_Z \arrow[d, "\gamma"] & \\
\mathbf{x}^q \arrow[d, "\theta"] \arrow[dr, "\theta"] & \mathbf{x}^p \sim p_X \arrow[d, dashed, "\theta"] \\
\mathcal{L}_\gamma \arrow[r, dash, dotted] & \mathcal{L}_\theta
\end{tikzcd}$$

(a) GAN

$$\begin{tikzcd}
\mathbf{z} \sim p_Z \arrow[d, "\gamma"] & \mathbf{x}^p \sim p_X \arrow[d] \\
\mathbf{x}^q \arrow[r] & \mathbf{K} \arrow[dl] \\
\mathcal{L}_\gamma &
\end{tikzcd}$$

(b) MMD-net

$$\begin{tikzcd}
\mathbf{z} \sim p_Z \arrow[d, "\gamma"] & \mathbf{x}^p \sim p_X \arrow[d, "\theta"] \\
\mathbf{x}^q \arrow[d, "\theta"] & f_\theta(\mathbf{x}^p) \arrow[d] \\
f_\theta(\mathbf{x}^q) \arrow[r] & \mathbf{K} \arrow[dl] \arrow[d] \\
\mathcal{L}_\gamma \arrow[r, dash, dotted] & \mathcal{L}_\theta
\end{tikzcd}$$

(c) MMD-GAN

$$\begin{tikzcd}
\mathbf{z} \sim p_Z \arrow[d, "\gamma"] & \mathbf{x}^p \sim p_X \arrow[d, "\theta"] \\
\mathbf{x}^q \arrow[d, "\theta"] & f_\theta(\mathbf{x}^p) \arrow[d] \\
f_\theta(\mathbf{x}^q) \arrow[r] & \mathbf{K} \arrow[dl] \arrow[d] \\
\mathcal{L}_\gamma & \mathcal{L}_\theta
\end{tikzcd}$$

(d) GRAM-net

Figure 5.6: Computation graphs of GAN, MMD-net, MMD-GAN and GRAM-net. **K** is the kernel Gram matrix. Solid arrows represent the flow of the computation and dashed lines represents min-max relationship between the losses, i.e. saddle-point optimisation in which minimising one loss maximises the other. Therefore, in the zero-sum game case (GAN, MMD-GAN) the two objectives (L_γ and L_θ) cannot be optimised simultaneously (Mescheder et al., 2017b).

two Gaussian distributions, which is

$$\mathrm{FID}(\mathbf{x}_p, \mathbf{x}_q) = \|\mu_{\mathbf{x}_p} - \mu_{\mathbf{x}_q}\|_2^2 + \mathrm{Tr}(\Sigma_{\mathbf{x}_p} + \Sigma_{\mathbf{x}_q} - 2(\Sigma_{\mathbf{x}_p}\Sigma_{\mathbf{x}_q})^{\frac{1}{2}}),$$

where $(\mu_{\mathbf{x}_p}, \Sigma_{\mathbf{x}_p})$, and $(\mu_{\mathbf{x}_q}, \Sigma_{\mathbf{x}_q})$ are the mean and covariance of the sample embeddings from the data distribution and model distribution. We report FID on a held-out set that was not used to train the models. We run all the models three times from random initializations and report the mean and standard deviation of FID over the initializations.

Architecture We test all the methods on the same architectures for the generator and the critic, namely a four-layer DCGAN architecture (Radford et al., 2015), because this has been consistently shown to perform well for the datasets that we use. Additionally, to study the effect of changing architecture, we also evaluate a slightly weaker critic,

Table 5.1: Sample quality (measured by FID; lower is better) of GRAM-nets compared to GANs. Numbers after ± are standard deviations.

Arch.	**Dataset**	**MMD-GAN**	**GAN**	**GRAM-net**
DCGAN	Cifar10	40.00 ± 0.56	26.82 ± 0.49	$\mathbf{24.85} \pm \mathbf{0.94}$
Weaker	Cifar10	210.85 ± 8.92	31.64 ± 2.10	$\mathbf{24.82} \pm \mathbf{0.62}$
DCGAN	CelebA	41.105 ± 1.42	30.97 ± 5.32	$\mathbf{27.04} \pm \mathbf{4.24}$

Table 5.2: FID with fully convolutional architecture originally used by Li et al. (2017). Numbers after ± are standard deviations.

Dataset	**MMD-GAN**
Cifar10	38.39 ± 0.28
CelebA	40.27 ± 1.32

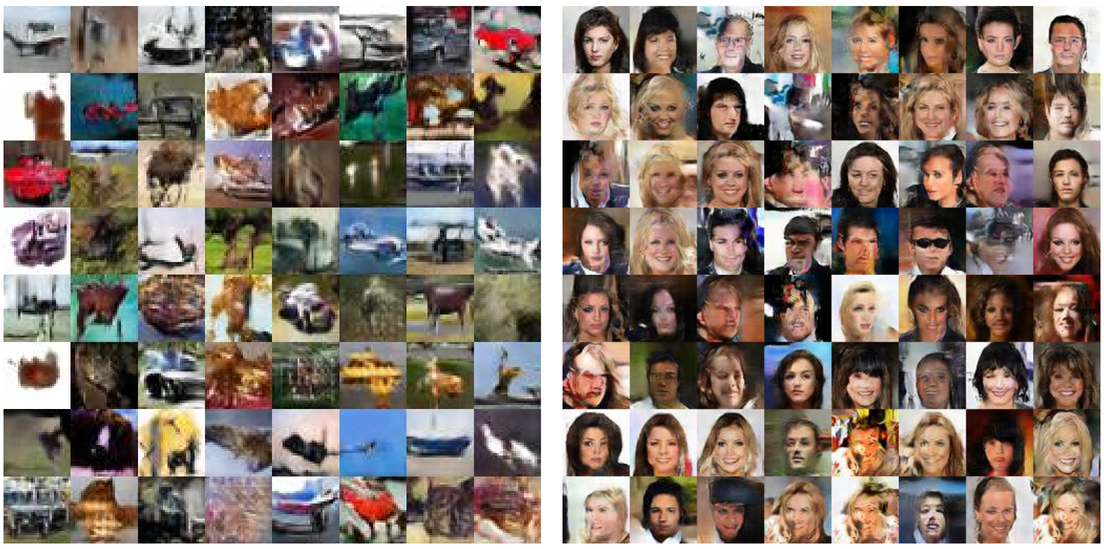
with the same number of layers but half the number of hidden units. Details of the architectures are provided in appendix 5.D.

Hyper-parameters To facilitate fair comparison with MMD-GAN we set all the hyperparameters shared across the three methods to the values used in Li et al. (2017). Therefore, we use a learning rate of $5e^{-5}$ and set the batch size to 64. For the MMD-GAN and GRAM-nets, we used the same set of RBF kernels that were used in Li et al. (2017). We used the implementation of MMD-GANs from Li et al. (2017).[5] We leave all the hyper-parameters that are only used by MMD-GAN to the settings in the authors' original code. For GRAM-nets, we choose $K = h$, i.e. the projected dimension equals the input noise dimension. We present an evaluation of hyperparameter sensitivity in section 5.4.2.

5.4.1 Image quality

We now look at how our method competes against GANs and MMD-GANs on sample quality and mode dropping on Cifar10 and CelebA datasets. Quantitative results for comparison are shown in table 5.1 with some of the randomly generated samples from our method in figure 5.1. Clearly, GRAM-nets outperform both baselines. For CelebA,

[5] Available at `https://github.com/OctoberChang/MMD-GAN`.



(a) CIFAR10 (b) CelebA

Figure 5.1: Random Samples from a randomly selected epoch (>100).

we do not run experiments using the weaker critic, because this is a much larger and higher-dimensional dataset, so a low-capacity critic is unlikely to work well.

It is worth noting that while the difference between the FIDs of GAN and GRAM-net is relatively smaller, it is quite significant that GRAM-net outperforms GAN on both datasets. As shown in a large scale study of adversarial generative models (Lucic et al., 2017), GANs in general perform very well on FID when compared to the state-of-the-art methods such as WGAN (Arjovsky et al., 2017a). Interestingly, in the case CIFAR10, GANs are the state-of-art on FID performance.

To provide evidence that GRAM-nets are not simply memorizing the training set, we note that we measure FID on a held-out set, so a network that memorized the training set would be likely to have poor performance. For additional qualitative evidence of this, see figure 5.2. This figure shows the five nearest neighbours from the training set for 20 randomly generated samples from the trained generator of our GRAM-net. None of the generated images have an exact copy in the training set, and qualitatively the 20 images appear to be fairly diverse.

Note that our architecture is different from that used in the results of Li et al. (2017). That work uses a fully convolutional architecture for both the critic and the generator, which results in an order of magnitude more weights. This makes a large comparison more expensive, and also risks overfitting on a small dataset like Cifar10. However, for completeness, and to verify the fairness of our comparison, we also report the FID

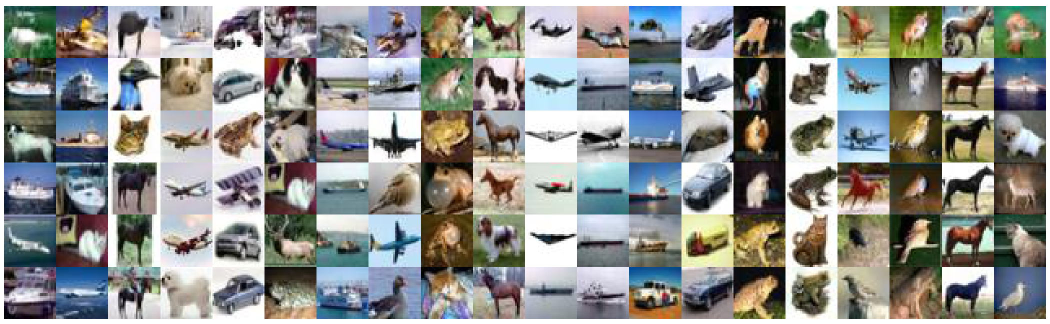
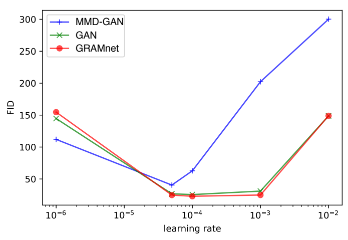
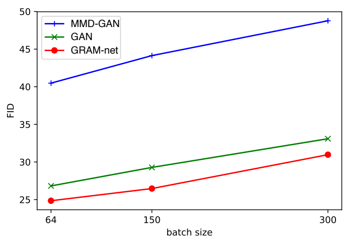


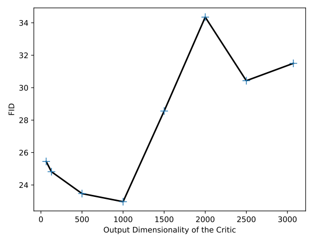
Figure 5.2: Nearest training images to samples from a GRAM-net trained on Cifar10. In each column, the top image is a sample from the generator, and the images below it are the nearest neighbors.



(a) FID vs Learning Rate



(b) FID vs Batch Size



(c) FID vs critic output dimension for GRAM-nets.

Figure 5.3: Hyper-parameter sensitivity of MMD-GAN, GAN and GRAM-net on Cifar10 dataset. Sample quality measured by FID.

that we were able to obtain with MMD-GAN on this fully-convolutional architecture in table 5.2. Compared to our experiments using MMD-GAN to train the DCGAN architecture, the performance of MMD-GAN with the fully convolutional architecture remains unchanged for the larger CelebA dataset. On Cifar10, not surprisingly, the larger fully convolutional architecture performs slightly better than the DCGAN architecture trained using MMD-GAN. The difference in FID between the two different architectures is relatively small, justifying our decision to compare the generative training methods on the DCGAN architecture.

5.4.2 Sensitivity to hyper-parameters and effect of the critic dimensionality

GAN training can be sensitive to the learning rate and the batch size used for training (Lucic et al., 2017). We examine the effect of learning rates and batch sizes on the performance of all three methods. Figure 5.3a compares the performance as a function of the learning rates. We see that GRAM-nets are much less sensitive to the learning rate than MMD-GAN, and are about as robust to changes in the learning rate as a vanilla GAN. MMD-GAN seems to be especially sensitive to this hyper-parameter. We suggest that this might be the case since the critic in MMD-GAN is restricted to the set of k-Lipschitz continuous functions using gradient clipping, and hence needs lower learning rates. Similarly, figure 5.3b shows the effect of the batch size on three models. We notice that all models are slightly sensitive to the batch size, and lower batch size is in general better for all methods.

We examine how changing the dimensionality K of the critic affects the performance of our method. We use the Cifar10 data. Results are shown in figure 5.3c. Interestingly, we find that there are two regimes: the output dimensionality steadily improves the FID until $K = 1000$, but larger values sharply degrade performance. This agrees with the intuition in section 5.3.1 that dimensionality reduction is especially useful for an "intrinsically low dimensional" distribution. For more inspections of the stability of MMD-GANs, see appendix 5.C.

5.5 Related work

Li et al. (2015) and Dziugaite et al. (2015) independently proposed MMD-nets, which use the MMD criterion to train a deep generative model. Unlike f-divergences, MMD is well-defined even for distributions that do not have overlapping support, which is an important consideration for training generative models (Arjovsky et al., 2017a). Therefore, MMD-nets use equation 2.20 in order to minimise the discrepancy between the distributions q_X and p_X with respect to G_γ. However, the sample quality of MMD-nets generally degrades for higher dimensional or color image datasets (Li et al., 2015).

To address this problem, Li et al. (2017) introduce MMD-GANs, which use a critic $f_\theta : \mathbb{R}^D \mapsto \mathbb{R}^K$ to map the samples to a lower dimensional space $\mathbb{R}^K$, and train the generator to minimise MMD in this reduced space. This can be interpreted as learning the kernel function for MMD, because if f_θ is injective and k_0 is a kernel in $\mathbb{R}^K$, then $k(\mathbf{x}, \mathbf{x}') = k_0(f_\theta(\mathbf{x}), f_\theta(\mathbf{x}'))$ is a kernel in $\mathbb{R}^D$. This injectivity constraint on f_θ is imposed by introducing another deep neural network f'_ϕ, which is trained to approximately invert f_θ using an auto-encoding penalty. Though it has been shown before that inverting f_θ is not necessary for the method to work. We also confirm this in experiments (See appendix).

The critic f_θ is trained using an adversarial criterion (maximizing equation equation 2.20 that the generator minimizes), which requires numerical saddle-point optimization, and avoiding this was one of the main attractions of MMD in the first place. Due to this, successfully training f_θ in practice required a penalty term called feasible set reduction on the class of functions that f_θ can learn to represent. Furthermore, f is restricted to be k-Lipschitz continuous by using a low learning rate and explicitly clipping the gradients during update steps of f akin to WGAN (Arjovsky et al., 2017a). Recently, Bińkowski et al. (2018); Li et al. (2019a) have proposed improvements to stabilize the training of the MMD-GAN method. But these methods still rely on solving the same saddle-point problem to train the critic.

Appendix

5.A Derivation details

The derivation from equation 5.2 - equation 5.4 relies on the law of unconscious statistician (LOTUS), and therefore no Jacobian correction is needed. Here we show how we applied LOTUS in the derivation, with more detailed steps.

Let $\bar{r}(\mathbf{x}) = \frac{\bar{p}(\mathbf{x})}{\bar{q}(\mathbf{x})}$. Expanding equation 5.2 we have,

$$\begin{aligned}
D^*(\theta) &= \int q_X(\mathbf{x}) \left(\frac{p_X(\mathbf{x})}{q_X(\mathbf{x})}\right)^2 \mathrm{d}\mathbf{x} - 2\int q_X(\mathbf{x}) \frac{p_X(\mathbf{x})}{q_X(\mathbf{x})} \frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))} \mathrm{d}\mathbf{x} + \int q_X(\mathbf{x}) \left(\frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))}\right)^2 \mathrm{d}\mathbf{x} \\
&= \int p_X(\mathbf{x}) \frac{p_X(\mathbf{x})}{q_X(\mathbf{x})} \mathrm{d}\mathbf{x} - 2\int p_X(\mathbf{x}) \frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))} \mathrm{d}\mathbf{x} + \int q_X(\mathbf{x}) \left(\frac{\bar{p}(f_\theta(\mathbf{x}))}{\bar{q}(f_\theta(\mathbf{x}))}\right)^2 \mathrm{d}\mathbf{x} \\
&= \int p_X(\mathbf{x}) \frac{p_X(\mathbf{x})}{q_X(\mathbf{x})} \mathrm{d}\mathbf{x} - 2\int p_X(\mathbf{x}) \bar{r}(f_\theta(\mathbf{x})) \mathrm{d}\mathbf{x} + \int q_X(\mathbf{x}) \bar{r}^2(f_\theta(\mathbf{x})) \mathrm{d}\mathbf{x}
\end{aligned}$$

We obtain the second line by cancelling and rearranging terms and the third line, by plugging in $\bar{r}(\mathbf{x})$.

Applying Theorem 3.6.1 from Bogachev (2007) (pg. 190) on the third term, $\int q_X(\mathbf{x}) \bar{r}^2(f_\theta(\mathbf{x}))$, with $g = \bar{r}^2$ and $f = f_\theta$, we get

$$\int q_X(\mathbf{x}) \bar{r}^2(f_\theta(\mathbf{x})) \mathrm{d}\mathbf{x} = \int \bar{q}(f_\theta(\mathbf{x})) \bar{r}^2(f_\theta(\mathbf{x})) d f_\theta(\mathbf{x}).$$

Similarly, for the second term, with $g = \bar{r}$ and $f = f_\theta$, we obtain

$$2\int p_X(\mathbf{x}) \bar{r}(f_\theta(\mathbf{x})) \mathrm{d}\mathbf{x} = 2\int \bar{p}(f_\theta(\mathbf{x})) \bar{r}(f_\theta(\mathbf{x})) d f_\theta(\mathbf{x}).$$

Thus,

$$D^*(\theta) = \int p_X(\mathbf{x}) \frac{p_X(\mathbf{x})}{q_X(\mathbf{x})} \mathrm{d}\mathbf{x} - 2\int \bar{p}(f_\theta(\mathbf{x})) \bar{r}(f_\theta(\mathbf{x})) d f_\theta(\mathbf{x}) + \int \bar{q}(f_\theta(\mathbf{x})) \bar{r}^2(f_\theta(\mathbf{x})) d f_\theta(\mathbf{x}),$$

which is the first line of equation 5.3.

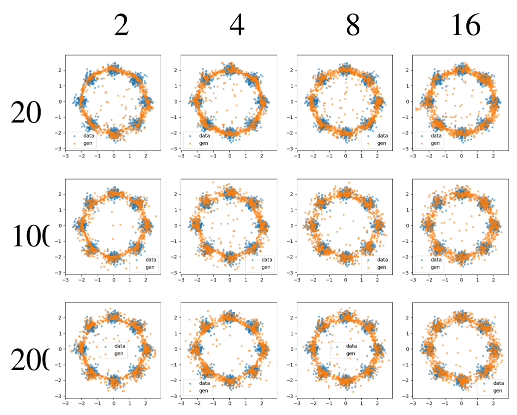
Note that we use LOTUS "in reverse" of its usual application.

5.B Experimental details and more results for section 5.3.4

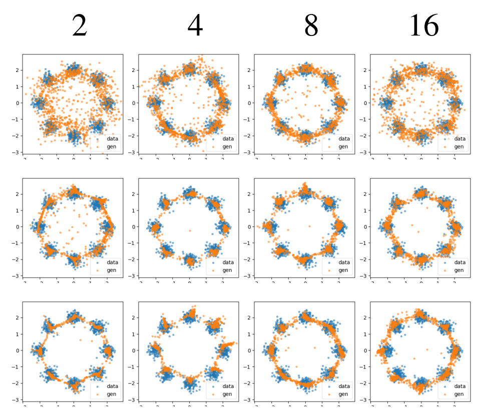
5.B.1 Experimental details

Below list the hyper-parameters used in section 5.3.4, except from those being varied in the experiments.

- Number of epochs: 2,000
- Noise distribution: Gaussian
- Activation between hidden layers: ReLU
- Batch normalisation: not used
- Batch size: 200
- Batch size for generated samples: 200
- GAN
  - Optimizer: ADAM
  - Learning rate: 1e-4
  - Momentum decay: 0.5
  - Critic architecture: 2-100-100-10
- MMD-net
  - Optimizer: RMSprop
  - Learning rate: 1e-3
  - RBF kernel bandwidth: 1
- GRAM-net
  - Optimizer: ADAM
  - Learning rate: 1e-3



(a) MMD-nets



(b) MMD-GANs

Figure 5.B.1: Corresponding plots to figure 5.2 for MMD-nets and MMD-GANs.

- Momentum decay: 0.5
- RBF kernel bandwidth: 1
- Critic architecture: 2-100-100-10

Note that we also provide the experimental details for MMD-nets where we will show the results in appendix 5.B.2.

5.B.2 Stability of MMD-nets and MMD-GANs

In this section we repeat the same stability-related experiments that we conducted on GANs and GRAM-nets in section 5.3.4, for MMD-nets and MMD-GANs. Results are shown in figure 5.B.1a and 5.B.1b. One can see MMD-nets can capture all the modes, but the learned distribution is less separated (or sharp) compared to GRAM-nets. This is because the MMD is computed with a fixed kernel and in a fixed space, which can only distinguish distributions, subject to the set of kernels being used. Figure 5.B.1b shows the results on MMD-GAN models that are trained for 4,000 epochs, each with 5 steps for the critic network and 1 step for the generative network.[6] Compared to figure 5.2b, in which GRAM-nets only takes 2,000 training epochs with 1 joint step for both networks, even with longer training, the quality of MMD-GAN is visually worse than GRAM-nets in this synthetic example. Notice, how the generated samples of MMD-GAN are similar to the successful runs for GAN in figure 5.2a: generated

[6]When training MMD-GAN, both the auto-encoding loss and the feasible set reduction loss are used. Parameter clipping is done with -0.1 (lower) and 0.1 (upper). Learning rate is set to 5e-5 and other parameters are the same as GRAM-nets.

samples tend to be too concentrated around the mode of the individual clusters. Our method on the other hand, with shorter training time, is clearly able to recover all the 8 clusters along with their spreads.

Note that, unlike GRAM-nets which are robust to changing dimensionality of the projected spaces (2, 4, 8, 16), MMD-GAN training easily diverges for 2-dimensional projected spaces. As a result, for MMD-GAN we only show the results up to the iterations before the training diverges. It's likely that any change in the neural network size (output dimension) requires a different set of hyperparameters to make MMD-GAN converge. On the other hand, GRAM-nets are robust to these changes, out of the box.

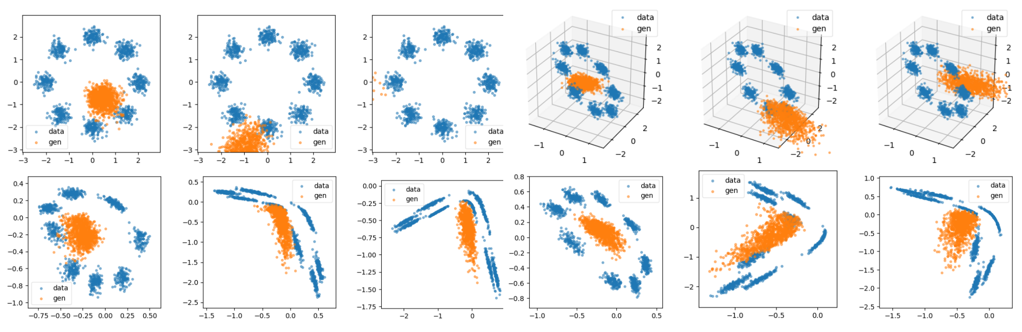
5.B.3 GRAM-nets on the MNIST dataset

The generator used in this experiment is a multilayer perceptron (MLP) of size 100-600-600-800-784 with ReLU activations between hidden layers and sigmoid activation for the output layer; the noise distribution is a multivariate uniform distribution between $[-1, 1]$ with 100 dimensions. For the critic, we use a convolution architecture specified in table 5.B.1. For the optimisation, we use ADAM with a learning rate of 1e-3,

Op	Input	Output	Filter	Pooling	Padding
Reshape	784	(-1,28,28,1)	-	-	-
Conv2D + BatchNorm	1	16	3	2	1
Conv2D + BatchNorm	16	32	3	2	1
Conv2D + BatchNorm	32	32	3	2	1
Reshape	(-1,3,3,32)	(-1,288)	-	-	-
Linear	288	100	-	-	-

Table 5.B.1: Critic architecture for MNIST. All BatchNorm are followed by a ReLU activation.

momentum decay of 0.5, and batch size of 200 for both data and generated samples. For the RBF kernels, we use bandwidths of $[0.1, 1, 10, 100]$



(a) 2D Ring (b) 3D Ring

Figure 5.C.1: Training of MMD-GAN with projected dimension fixed to 2 *before diverging*. Data and samples in the original (top) and projected space (bottom) during training; four plots are at iteration 100, 500 and $1,000$ respectively. Notice how the projected space separates $\bar{p}$ and $\bar{q}$.

5.C Stability of MMD-GANs

In addition to figure 5.B.1b, which evaluates the stability of MMD-GANs in terms of the projected dimension and the generator capacity, we perform qualitative evaluation by visualizing the projected space during training as well as the effect of stabilization techniques in this section.

5.C.1 Projected space during training

We first perform the qualitative evaluation (same as figure 5.1a and figure 5.3) for MMD-GANs on the 2D and 3D ring datasets. The original space and the projected space during training are visualized in figure 5.C.1 It is clear that the projected spaces are quite different between MMD-GAN and our method. Unlike GRAM-nets, in the projected space, the generated samples do not overlap with the data samples.

5.C.2 Stabilization techniques

We also evaluate the effect of the various stabilization techniques used for training, namely the autoencoder penalty (AE) and the feasible set reduction (FSR) techniques from (Li et al., 2017) on the Cifar10 data over two settings of the batch size. Table 5.C.1 shows the results. The performance of MMD-GAN training clearly relies heavily on FSR. This penalty not only stabilizes the critic but it can also provides additional learning signal to the generator. Because these penalties are important to the performance of

Table 5.C.1: Performance of MMD-GAN (Inception scores; larger is better) for MMD-GAN with and without additional penalty terms: feasible set reduction (FSR) and the autoencoding loss (AE). The full MMD-GAN method is MMD+FSR+AE.

Batch Size	**MMD+FSR+AE**[7]	**MMD+FSR**	**MMD+AE**	**MMD**
64	5.35 ± 0.05	5.40 ± 0.04	3.26 ± 0.03	3.51 ± 0.03
300	5.43 ± 0.03	5.15 ± 0.06	3.68 ± 0.22	3.87 ± 0.03

MMD-GANs, it requires tuning several weighting parameters, which need to be set carefully for successful training.

We would like to re-emphasize the stability of GRAM-nets with respect to different settings of network sizes, noise dimension, projected dimension, learning rate, batch size without relying on regularization terms with additional hyperparameters.

5.D Architecture

For the generator used in section 5.4, we used the following DCGAN architecture,

Op	**Input**	**Output**	**Filter**	**Stride**	**Padding**
Linear	128	2048	-	-	-
Reshape	2048	(-1,4,4,128)	-	-	-
Conv2D_transpose	128	64	4	2	SAME
Conv2D_transpose	64	32	4	2	SAME
Conv2D_transpose	32	3	4	2	SAME

Table 5.D.1: DCGAN generator architecture for Cifar10.

We used two different architectures for the experiments on Cifar10 dataset. Table 5.D.2 shows the standard DCGAN discriminator that was used. Table 5.D.3 shows the architecture of the weaker DCGAN discriminator architecture that was also used for Cifar10 experiments. While leaky-ReLU was used as non-linearity in the discriminator, ReLU was used in the generator, except for the last layer, where it was tanh. Batchnorm was

Op	Input	Output	Filter	Stride	Padding
Conv2D	3	32	4	2	SAME
Conv2D	32	64	4	2	SAME
Conv2D	64	128	4	2	SAME
Flatten	128	2048	-	-	-
Linear	2048	128	-	-	-

Table 5.D.2: DCGAN discriminator architecture for Cifar10.

Op	Input	Output	Filter	Stride	Padding
Conv2D	3	32	4	2	SAME
Conv2D	32	32	4	2	SAME
Conv2D	32	64	4	2	SAME
Flatten	64	1024	-	-	-
Linear	1024	128	-	-	-

Table 5.D.3: Shallow DCGAN discriminator architecture.

used in both the generator and the discriminator.

5.E Inception score

Inception score (IS) is another evaluation metric for quantifying the sample quality in GANs. Compared FID, IS is not very robust to noise and cannot account for mode dropping. In addition to the FID scores that we provide in the paper, here we also report IS for all the methods on CIFAR10 for completeness since the MMD-GAN paper used it as their evaluation criteria.

Table 5.E.1: Inception Scores for MMD-GAN, GAN, GRAM-net and MMD-nets on CIFAR10 for three random initializations.

	MMD-GAN	**GAN**	**GRAM-net**
Inception Score	5.35 ± 0.12	5.17 ± 0.13	$\mathbf{5.73 \pm 0.10}$
	5.21 ± 0.14	4.94 ± 0.15	$\mathbf{5.44 \pm 0.12}$
	5.31 ± 0.10	5.27 ± 0.05	$\mathbf{5.45 \pm 0.18}$

Chapter 6

A Bayesian-Symbolic Approach to Physics Learning in Intuitive Physics

6.1 Introduction

Imagine a ball rolling down a ramp. If asked to predict the trajectory of the ball, most of us will find it fairly easy to make a reasonable prediction. Not only that, simply by observing a single trajectory people can make reasonable guesses about the material and weight of the ball and the ramp. It is astonishing that while the exact answers to any of these prediction and reasoning tasks requires an in-depth knowledge of Newtonian mechanics and solving of some intricate equations, yet an average human can perform such tasks without any formal training in physics. Studies suggest that from early age humans come to understand physical interactions with very limited supervision, and can efficiently reason and plan actions in common sense tasks, even in absence of complete information (Spelke, 2000; Battaglia et al., 2013). For example, with limited data, 4 or 5 years old children are capable of learning the physical laws behind magnetism (Bonawitz et al., 2019). Physical reasoning is considered a core domain of human common-sense knowledge (Spelke and Kinzler, 2007). Recent studies suggest that the ability to efficiently learn physical properties and interactions with limited supervision is driven by a noisy model of Newtonian dynamics, referred to as the *intuitive physics engine* (IPE; Bates et al., 2015; Gerstenberg et al., 2015; Sanborn et al., 2013; Lake et al., 2017; Battaglia et al., 2013). This has led to a surge in research aimed at developing agents with an IPE, or a model of the environment dynamics (Amos et al., 2018; Chang et al., 2016; Grzeszczuk and Animator, 1998; Fragkiadaki et al., 2015; Battaglia et al., 2016;

Watters et al., 2017; Sanchez-Gonzalez et al., 2019; Ehrhardt et al., 2017; Kipf et al., 2018; Seo et al., 2019; Baradel et al., 2020). These efforts have created methods that either trade-off data-efficiency, by using deep neural networks (NNs), for high predictive accuracy (Breen et al., 2019; Battaglia et al., 2016; Sanchez-Gonzalez et al., 2019) or trade-off flexibility to learn from data for data-efficiency by using symbolic methods (Ullman et al., 2018; Smith et al., 2019; Sanborn et al., 2013; Bramley et al., 2018).

Inspired by the highly data-efficient ability of humans to learn and reason about their physical environment with incomplete information, we present Bayesian-symbolic physics (BSP), a Bayesian-symbolic model with an expectation-maximization (EM) algorithm that combines the sample efficiency of symbolic methods with the accuracy and generalization of data-driven approaches, using statistical inference of unobserved object properties and symbolic learning of physical force laws. In BSP, we model the evolution of the environment's dynamics over time as a generative program of entities interacting under Newtonian mechanics. As a probabilistic method, BSP treats the properties of entities, such as mass and charge, as random variables. Since Newtonian force laws are functions of these properties, in BSP we replace data-hungry NNs with symbolic regression (SR) to learn explicit force expressions, and then evolve them deterministically using equations of motion. A naive SR implementation here is not enough though due to two issues. One is that if it operates on a vanilla grammar that does not constrain the search space over force-laws, it can potentially have worse data-efficiency than a NN. Therefore, we introduce a *grammar of Newtonian physics* that leverages *dimensional analysis* to induce a physical unit system over the search space and impose physics-based constraints on the production rules. This prunes physically meaningless laws, therefore, drastically speeding up SR. Another issue is that the symbolic force expressions usually contain global constants, e.g. the gravitational constant, to learn, and common ways to deal with this challenge turn out to be inefficient especially in an EM setup. We tackle this challenge by using SR in a *bilevel optimization* framework in which a lower-level gradient based optimization step is used to optimize the constant. In short, our three main contributions are:

- We introduce a Bayesian-symbolic model for physical dynamics and an EM based algorithm, which combines approximate inference methods and SR, for maximum likelihood learning.
- We introduce a grammar of Newtonian physics that appropriately constrains SR for data-efficient learning, based on priors from dimensional analysis and physics-based

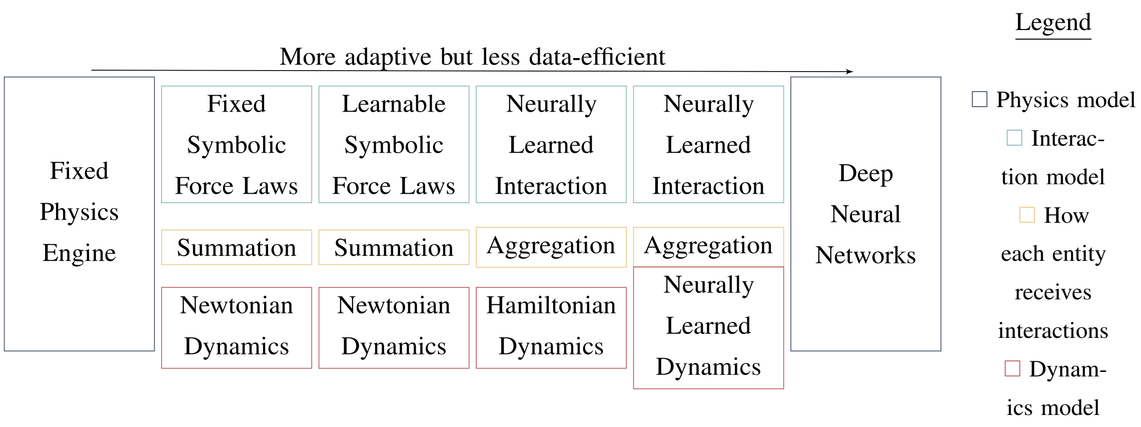

Legend
More adaptive but less data-efficient
Fixed Physics Engine
Fixed Symbolic Force Laws
Learnable Symbolic Force Laws
Neurally Learned Interaction
Neurally Learned Interaction
Deep Neural Networks
Summation
Summation
Aggregation
Aggregation
Newtonian Dynamics
Newtonian Dynamics
Hamiltonian Dynamics
Neurally Learned Dynamics
Physics model
Interaction model
How each entity receives interactions
Dynamics model

Figure 6.1: From left to right are rule-based to purely data-driven models of physics. Examples for each column are (1) Smith et al. (2019), (2) Ullman et al. (2018), (3) BSP (Ours), (4) (H)OGN Sanchez-Gonzalez et al. (2019), (5) IN Battaglia et al. (2016) and (6) Breen et al. (2019).

constraints.

- Empirically, we show that BSP reaches human-like data-efficiency, often requiring just 1 to 5 synthetic scenes to learn the underlying force laws—much more data efficient than the closest neural alternatives. We then illustrate how BSP can discover physical laws from real-world common sense scenes from Wu et al. (2016). Finally, we study BSP on tasks previously used to study human physical reasoning in Ullman et al. (2018) and discuss the similarity and differences with human results.

6.2 Related work

Many symbolic and data driven models of learning and reasoning about physics can be broken down into smaller components that are either learned or fixed. In figure 6.1, we compare some of the closely-related recent work on physics learning. Starting on the right end, we have fully learned, deep NN approaches such as that used by Breen et al. (2019). This approach does not use any prior knowledge about physics, and learns to predict dynamics in a purely data-driven way. In the middle are hybrid models that introduce some prior knowledge about physical interactions or dynamics, in their NN-based prediction models. These include interaction networks (INs; Battaglia et al., 2016), ODE graph networks (OGNs), and Hamiltonian ODE graph networks (HOGNs; Sanchez-Gonzalez et al., 2019). Since these middle approaches use deep NNs, they tend to have very good predictive accuracy, yet poor sample complexity, requiring orders of magnitude more data to train than humans (Ullman et al., 2018; Battaglia et al., 2016; Sanchez-Gonzalez et al., 2019). On the other end of the spectrum (left) are fully symbolic, rule-based physics models and engines (Smith et al., 2019; Allen et al., 2019; Wu et al., 2015; Ullman et al., 2018). While these methods are suitable for reasoning tasks, they lack the flexibility of data-driven, learned models as they cannot generalize or adapt to changes in the environment that their fixed physics engine cannot simulate. For example, inference can fail on physically implausible scenes, and may require additional workarounds such as 'low probability events' outside the dynamics (Smith et al., 2019).

Symbolic regression has been used for general physics learning in prior research, ranging from Schmidt and Lipson (2009)'s work on discovering force laws from experimental data, to the more recent work of Cranmer et al. (2020) on distilling symbolic forces from INs using genetic algorithms. More recently, Udrescu and Tegmark (2020) proposed AI Feynman, which recursively simplifies the SR problem using dimensional analysis and symmetries inferred by neural networks, to discover the underlying physics equations of the data. The focus of these types of work has been to discover underlying laws and equations based on *direct* input-output data. The focus of BSP, on the other hand, is on physics learning based on *indirect* signals from the environment; this is a task of interest in both intuitive physics studies with humans, and for human-like AI. Further, most symbolic approaches learn physics while assuming all properties in a system are known, which renders them inapplicable to environments with incomplete information. Some neural approaches focus on addressing such limitations in an end-to-end fashion (Zheng et al., 2018; Veerapaneni et al., 2020; Janner et al., 2019).

6.3 Bayesian-symbolic physics

BSP represents the physical environment using a generative model that evolves under Newtonian dynamics (section 6.3.1). In this model, physical laws are treated as learnable symbolic expressions and learned by symbolic regression and a specialized grammar of Newtonian physics that confines the search space and prevent the model from learning physically meaningless laws (section 6.3.2). BSP does not require all properties of the entities to be fully observed. It models these properties as latent random variables and infers them using Bayesian learning. To fit BSP on data, with incomplete information, we propose an EM algorithm that iterates between Bayesian inference of the latent properties, and SR which gives maximum likelihood estimation of the force expressions (section 6.3.3).

6.3.1 Generative model of the environment

In BSP's generative model, we represent each entity $i \in \{1 \ldots N\}$ by a vector of intrinsic physical properties z^i (such as mass, charge, and shape), and a time dependent state vector $\mathbf{s}_t^i = (\mathbf{p}_t^i, \mathbf{v}_t^i)$ which describes the evolution of its position $\mathbf{p}_t^i \in \mathbb{R}^d$ and velocity $\mathbf{v}_t^i \in \mathbb{R}^d$ under Newtonian dynamics. Here, d refer to the dimensionality of the environment, and is typically 2 or 3. Let $\{\tau^i\}_{i=1}^N$ be the set of observed trajectories from an environment with N entities, where $\tau^i = \mathbf{p}_{1:T}^i := (\mathbf{p}_1^i, \ldots, \mathbf{p}_T^i)$. Then, together with the prior on z, for an observed trajectory data, $\mathcal{D}$, the generative model of BSP defines a joint probability distribution $p(\mathcal{D}, z; F)$ over $\mathcal{D}$ and latent properties z, given the force function F.[1] The state transition of an entity in a Newtonian system depends on its properties and current state as well as its interaction with other entities. So, in BSP the force on entity i at time t is defined as $\mathbf{f}_t^i = \sum_{j=1}^N F(z^i, \mathbf{s}_t^i, z^j, \mathbf{s}_t^j)$, where $F(z^i, \mathbf{s}_t^i, z^j, \mathbf{s}_t^j)$ is the interaction force between entities i and j. Then, the trajectory τ_i of entity i is generated by a transition function $\mathbb{T}$ that consumes the current state and the resultant force to compute $\mathbf{s}_{t+1}^i = \mathbb{T}\left(\mathbf{s}_t^i, \mathbf{f}_t^i\right)$. Similar to Sanchez-Gonzalez et al. (2019), we use numerical integration to simulate the Newtonian dynamics inside $\mathbb{T}$. Specifically, we choose the Euler integrator and expand $\mathbb{T}$ as

$$\mathbf{a}_t^i = \mathbf{f}_t^i / m^i, \quad \mathbf{v}_{t+1}^i = \mathbf{v}_t^i + \mathbf{a}_t \varepsilon, \quad \mathbf{p}_{t+1}^i = \mathbf{p}_t^i + \mathbf{v}_{t+1}^i \varepsilon, \tag{6.1}$$

where m^i is the mass of the recipient of the force $\mathbf{f}_t^i$ and ε is the step size of the Euler integrator. Finally, we add Gaussian noise to each trajectory $\{\tau^i\}_{i=1}^N$, that is, $\mathcal{D} := \{\tilde{\tau}^i\}_{i=1}^N$ where $\tilde{\tau}^i := (\tilde{\mathbf{p}}_1^i, \ldots, \tilde{\mathbf{p}}_T^i)$, $\tilde{\mathbf{p}}_t^i \sim \mathcal{N}(\mathbf{p}_t^i, \sigma^2)$ and σ is the noise level. See appendix 6.A.1 for the details of the complete generative process and illustrative examples.

6.3.2 A grammar of Newtonian physics

In order to attain good data efficiency, we choose to learn the pairwise force $F(z^i, \mathbf{s}^i, z^j, \mathbf{s}^j)$ between entities i and j using symbolic search. This approach can be inefficient if

[1] As physical dynamics are typically sensitive to *initial states*, we assume the noise-free initial states are given either as part of the data $\mathcal{D}$, or can be accurately estimated in a pre-processing step, e.g. by using consecutive positions, thus are omitted in the notation.

$$\begin{tikzcd}
\boxed{\begin{array}{c} m_i+\mu_j,\ m_i-\mu_j, \\ m_i\times\mu_j,\ m_i\div\mu_j, \\ \|\mathbf{p}_i\|_2,\ \|\mathbf{p}_i-\mathbf{p}_j\|_2\end{array}} \arrow[r, "\text{Dimensional}", "\text{analysis}"'] & \boxed{\begin{array}{c} \cancel{m_i+\mu_j},\ \cancel{m_i-\mu_j}, \\ m_i\times\mu_j,\ m_i\div\mu_j, \\ \|\mathbf{p}_i\|_2,\ \|\mathbf{p}_i-\mathbf{p}_j\|_2\end{array}} \arrow[r, "\text{Translation}", "\text{invariance}"'] & \boxed{\begin{array}{c} \cancel{m_i+\mu_j},\ \cancel{m_i-\mu_j}, \\ m_i\times\mu_j,\ m_i\div\mu_j, \\ \cancel{\|\mathbf{p}_i\|_2},\ \|\mathbf{p}_i-\mathbf{p}_j\|_2\end{array}}
\end{tikzcd}$$

Figure 6.1: Illustration of how the dimensional analysis and translation invariance priors help constrain the search space. Each box contains a subset of valid and illegal (stroked) sub-expressions.

the search space of possible functions is too large, or inaccurate if the search space is too small. So, we constrain the function F to be a member of a context-free language with a grammar $\mathcal{G}$, which we call *the grammar of Newtonian physics*. We design the grammar to be expressive enough to represent a large variety of potential force laws, while incorporating some simple, general constraints to improve the efficiency of symbolic search. Here we describe $\mathcal{G}$ informally; for the formal description, see figure 6.A.2 (appendix 6.A.2).

We consider the following terminal nodes in $\mathcal{G}$: the masses m_i, m_j of the entities, their friction coefficients μ_i, μ_j, shapes s_i, s_j[2], positions $\mathbf{p}_i, \mathbf{p}_j$, velocities $\mathbf{v}_i, \mathbf{v}_j$ the contact point $\mathbf{c}$ i.e. the position (if any) at which they touch, and finally a set of K learnable constants $\{c_k\}_{k=1}^K$; the units of these terminals will be discussed in the next paragraph. In cases of no contact, $\mathbf{c}$ is set as the middle position of the two objects, i.e. $\mathbf{c} = \left(\mathbf{p}_i + \mathbf{p}_j\right)/2$. We include the operators: $(\cdot)^2$ (square), $+$, $-$, $\times$, $\div$, $\|\cdot\|_2$ (L2-norm), normalise$(\cdot)$ and project$(\cdot,\cdot)$, which projects a vector onto the unit ball.[3] The grammar also allows forces to be conditioned on a Boolean expression, in order to support *conditional forces* that only apply when a condition is true, e.g., when two objects collide. We provide $\mathcal{G}$ two primitive functions that encode the output of the perception system: doesCollide for collision detection and isOn to check if an entity is on a surface. These functions output integers 0 or 1. A rule in the grammar then allows a force expression to be multiplied by a conditional, so that BSP can learn expressions that represent when a conditional force should be applied.

Naively supporting all possible expressions for any combination of terminals would make SR highly inefficient, and even lead to *physically impossible* force laws. Therefore, we introduce two simple and general types of prior knowledge inspired by physics: dimensional analysis and translation invariance. Figure 6.1 shows examples of expressions that are excluded by each. First, inspired by dimensional analysis in natural

[2]Each type of objects such as discs, rectangles, etc. have their shapes with corresponding parameters, e.g. radius for discs.

[3]In our work we consider maximally three forces to be learn in the same time, thus setting $K = 3$; more learnable constants and/or entity properties can be easily added to the grammar if needed.

sciences, where the relations between different units of measurement are tracked (Brescia, 2012), we built the concept of *units of measurement* into the non-terminals of $\mathcal{G}$. The units we consider are kilogram (Kg) for mass, meter ($Meter$) for distance, and meter per second ($MeterSec$) for speed. With this unit system in place, we only allow addition and subtraction of symbols with the *same* units, avoiding physically impossible sub-expressions such as $Kg - Meter$.[4] Importantly, this can lead to force laws with unit Newton (N).[5] Second, the grammar ensures that all force laws are *translation-invariant*, that is, independent on the choice of the origin of the reference frame. To do this, the grammar forbids the direct use of *absolute* positions $\mathbf{p}_i$, $\mathbf{p}_j$ and $\mathbf{c}$ and only allows their differences to be used expressions.[6]

Finally, some care is needed to ensure the grammar is unambiguous. For example, if we used a rule like $Coeff \rightarrow Coeff \times Coeff$, then the grammar could generate many expressions that redundantly represent the same function. This would make search much more expensive. Instead, we rewrite this rule in an equivalent right-branching way, e.g., $Coeff \rightarrow BaseCoeff \times Coeff$. This significantly reduces the search space without changing the expressivity of the grammar. Overall, although the grammar puts basic constraints on plausible physical laws, it is still expressive: there are more than *7 million* possible trees up to depth 6 while even the expression of universal gravitation has a depth of 7; the number of expressions up to depth 7 in $\mathcal{G}$ is intractable to count.

6.3.3 Learning algorithm

Following the EM approach, our learning method alternates between an E-step, where object property distributions are estimated given the current forces via approximate inference, and an M-step step, where forces are learned given object property distributions via SR (section 6.3.3.1). For the E-step, we consider two standard inference options: importance sampling (for any prior), and Hamiltonian Monte Carlo otherwise (for continuous priors only). Appendix 6.A.4 discusses some details on applying them in BSP. Note appendix 6.A.3 also provides all pseudo-code for algorithms introduced in

[4]Constraining the grammar using units shares a similar spirit to type inference in type-directed synthesis (Feser et al., 2015; Osera and Zdancewic, 2015), which are crucial to improve search by avoiding illegal programs.

[5]When forming the unit N, constants can in fact has arbitrary units. But if there is any sub-expression like $Kg - Meter$, which is disallowed by our grammar, the final expression would not have any proper unit.

[6]This is consistent with how such variables are pre-processed in neural network approaches (Breen et al., 2019). Usually the mean of a pair of positions are subtracted from the pair to make them translation-invariant.

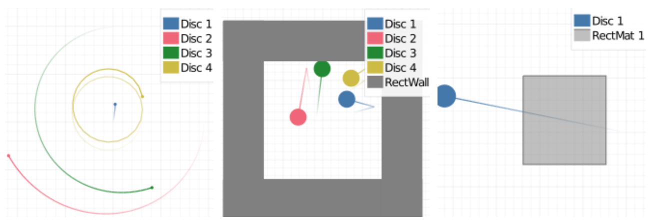
this section.

Implementation In our work, we implement the generative models as probabilistic programs using the Turing probabilistic programming language (Ge et al., 2018) in Julia. As such the E-step is simply done by Turing's built-in samplers. For the M-step, we use the cross-entropy implementation from the ExprOptimization.jl package which allows users to define grammars with intuitive syntax.

6.3.3.1 Symbolic regression with bilevel optimization for learnable constants

Symbolic regression (SR) is a function approximator that searches over a space of mathematical expressions defined by a context free grammar (CGF) (Koza, 1994). In our work, we use the cross-entropy method for SR. The method starts with a Probabilistic context-free grammars (PCFG) that assumes a uniform distribution over the production rules (PCFGs extend CFGs by assigning each production rule a probability). At each successive iteration, it samples n trees (up to depth d) from the current PCFG, evaluates their fitness by a loss function $\mathcal{L}$, and uses the top-k trees to fit a PCFG via maximum likelihood for the next iteration. This process returns the learned force law at the end of the training. More formally, to learn force laws, we need to find an expression $e \in L(\mathcal{G})$, where $L(\mathcal{G})$ is the language generated by $\mathcal{G}$, and values for the learnable constants $c := \{c_i\}_{k=1}^{3}$ that define the force function $\mathbf{f}_{e,c}$. The loss used by the cross-entropy method involves computing the log-likelihood of the generative model. As the observed trajectory is generated sequentially given an initial state, the computation of the log-likelihood term cannot be parallelized, and can be computationally expensive in practice. Therefore, following Battaglia et al. (2016) and Sanchez-Gonzalez et al. (2019), we use a *vectorized* version of the log-likelihood that basically performs simulation in each time stamp in parallel $LL(e,c;z,\mathcal{D}) = \sum_{i=1}^{N}\sum_{t=1}^{T-1} \log \mathcal{N}(\tilde{\mathbf{p}}_{t+1}^{i}; \mathbb{T}\left(\mathbf{s}_t^i, \mathbf{f}_{e,c,t}^{i}\right), \sigma)$ where $\mathbb{T}$ is expanded following equation 6.1 and $\tilde{\mathbf{s}}_t^i := (\tilde{\mathbf{p}}_t^i, \tilde{\mathbf{v}}_t^i)$. Clearly, LL differs from its corresponding sequential likelihood, as the input for the integrator contains noise at each step. However, similar to previous work, we found it is not an issue when learning forces by regression.

In order to prevent overfitting by finding over-complex expressions, we add a regularization term—weighted log-probability under a uniform PCFG prior of $\mathcal{G}$—to the negative log-likelihood; to arrive at our final loss per trajectory $\mathcal{L}(e,c;z,\mathcal{D}) = -LL(e,c;z,\mathcal{D}) + \lambda \log \mathcal{P}_0(e)$. Here $\mathcal{P}_0$ is the uniform PCFG of $\mathcal{G}$, and λ is the hyper-



(a) NBODY (b) BOUNCE (c) MAT

Figure 6.1: Example scenes from SYNTH. Entities in gray are static.

parameter that controls the regularization. The loss for multiple trajectories is just a summation of $\mathcal{L}$ over individual trajectories. The continuous constants c require care as they can take any value. To handle this, we use *bilevel optimization* (Dempe, 2002), where the upper-level is the original symbolic regression problem, and the lower-level is an extra optimization for constants. This means we optimize the constants before computing the loss of each candidate tree within the cross-entropy iterations. The defined loss for each expression e in SR is then $\mathcal{L}(e;z,\mathcal{D}) = \mathcal{L}(e, \arg\min_c \mathcal{L}(e,c);z,\mathcal{D})$. In BSP, we use the L-BFGS optimizer to solve the lower-level optimization. Our way of handling learnable constants is related to other SR methods and bilevel optimization. Traditionally, constants are either randomly generated from a predefined, fixed integer set or a continuous interval, or for evolutionary algorithms, they can be mutated and combined during evolution to produce constants that fit better; such constants are often referred as *ephemeral constants* (Davidson et al., 2001). Compared to these methods, the benefit of our formulation is that the evaluation of each tree candidate depends on the symbolic form *only* as the constants are *optimized-away*, making the search more efficient. Note that although the literature has not explicitly considered our way of constant learning as bilevel optimization problem, similar strategies are also used in (Cerny et al., 2008; Kommenda et al., 2013; Quade et al., 2016). In contrast to recent use of bilevel optimization in meta-learning, e.g. (Finn et al., 2017), our method is simpler: As our upper-level optimization is gradient-free, we do not need to pass gradient from the lower-level to the upper-level.

6.4 Experiment: Learning force laws in fully observed environment

In this section, we evaluate the BSP in a data-limited setting when the properties are fully observed.

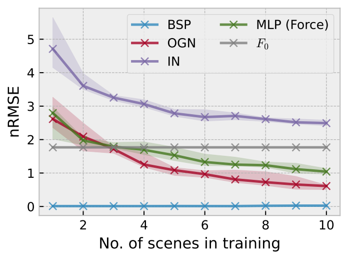
Synthetic datasets (SYNTH). We created three synthetic datasets for controlled evaluation: NBODY (n-body simulations with 4 bodies), BOUNCE (bouncing balls) and MAT (mats with friction); see figure 6.1 for an illustration. NBODY (n-body simulation with 4 bodies) is populated by placing a heavy body with large mass and no velocity at $[0,0]$, and three other bodies at random positions with random velocities such that, they orbit the heavy body in the middle in the absence of the other two bodies. The gravitational constant is set such that the system is stable for the duration of the simulation. The ground truth force to learn is the gravitational force between bodies. BOUNCE is generated by simulating *elastic collisions* between balls in a box. The ground truth force to learn is the collision resolution force. MAT simulates friction-based interaction between discs and a mat. We populate this dataset by rolling discs over mats and applying a friction force when they come into contact. We randomized the initial states of the discs as well as the sizes, friction coefficients, and positions of the mats. The ground truth force to learn is the force of friction.

All scenes are simulated using a physics engine with a time-discretization of 0.02, for 50 frames. We generate 100 scenes per dataset, and hold-out 20 of them for testing. Appendix 6.B.1 provides the ground truth force expressions used to generate each dataset under our grammar.

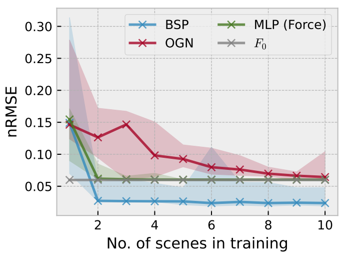
6.4.1 Data-efficiency: Symbolic vs neural

Baselines For the experiments in this section we use four different neural baselines: (i) A specialized instance of the OGN model (Sanchez-Gonzalez et al., 2019) that only outputs the partial derivative of the velocity variable, unlike the original model that also outputs the partial derivative of the position variable. This is because under Newtonian dynamics, the partial derivative of the position variable is simply the velocity. (ii) An Interaction Network (IN) (Battaglia et al., 2016) (iii) A multi-layer perceptron-based force model (MLP (Force)) that directly outputs the force, and (iv) A multi-layer perceptron-based position model (MLP (Position)) that outputs the next position. See appendix 6.B.2 for details of the neural architecture, training and parameterization setup for all the baselines. Lastly, as a reference, we also include the performance of a zero-force baseline (F_0), which corresponds to the constant velocity baseline in (Battaglia et al., 2016). Note that all neural baselines as well as BSP are provided with symbolic representations for fair comparisons.

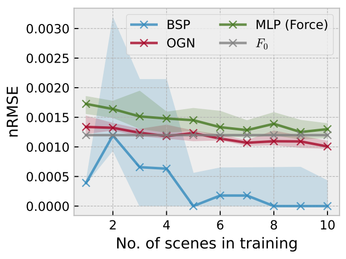
In order to compare the symbolic M-step of BSP against the neural baselines in terms



(a) NBODY



(b) BOUNCE



(c) MAT

Figure 6.2: Comparison of neural baselines and BSP, using predictive error on held out scenes given varying number of training scenes. Some baselines are not displayed due to very poor performance; see figure 6.C.1 in appendix 6.C for the version with all methods displayed.

of data-efficiency, we report the per-frame prediction accuracy on held-out datasets, as a function of the amount of training data. We use *noise-free* trajectories in this evaluation. Since the neural baselines cannot be trained if the properties are not fully observed, we provide all properties as observed data. For each dataset, we hold out 20 scenes for evaluation. We randomly shuffle the remaining 80 scenes, and use the first k scenes to fit the models. Because an average person can perform physical learning task similar to the ones we use with fewer than 5 scenes (Ullman et al., 2018), we only vary k from 1 to 10 in our experiments. We use the normalized root mean squared error (nRMSE), per frame per entity, between the predicted location of the entity and its actually location, as the performance metric. We repeat each of the experiments five times with different training set. These results are shown in figure 6.2, where the line plots are median values and the error bars are the 25% and 75% quantiles. Note that the ground truth force F^* has an RMSE of 0 per frame. As can be seen, for most values of k across of the three datasets, the symbolic M-step of BSP is more data-efficient than the neural baselines. The exceptions are in the BOUNCE dataset for $k = 1$ and in the MAT dataset for $k = 2, 3, 4$. This is likely due to specific bad local minima that may exist in the limited training data.

For NBODY and MAT, BSP can find the ground truth force function with 1 scene and 10 scenes respectively. BOUNCE is the only case where our method fails to find the true law within 10 scenes, we include the typical inferred force law in appendix 6.B.3.1 as well the predicated trajectories of some selected scenes for inspection. In appendix 6.B.3.2, we also demonstrate that this inferred force law closely approximates the true force law and so can generalize to other scenes.

For BOUNCE, the neural baselines cannot reach the performance of F_0 even after

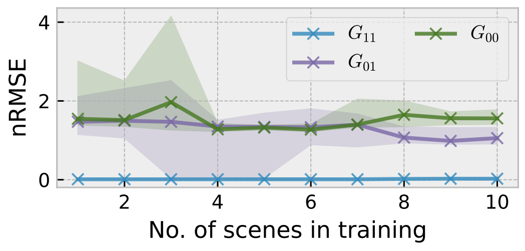
10 scenes for training. This is a known issue with neural network approaches when learning collisions, as the inherently sparse nature of the collision interaction does not provide enough training signal (Battaglia et al., 2016). The order of performance between neural baselines also indicates our discussion around figure 6.1. Object-centric modelling (OGN and IN) tends to have better performance than the rest by decomposing the transition into interaction and dynamics. Predefined dynamics with numerical integration (OGN and MLP force) have better performance than their correspondences with learned dynamics (IN and MLP position) by a notion of "force" (as the Euler integrator is used in this case).

In short, the symbolic regression proper priors that constrains the search space in BSP leads to significantly better performance in terms of data efficiency across the three datasets studied. Together with the performance rank of neural models, our experiments shows how different levels and forms of prior help with data-efficient learning.

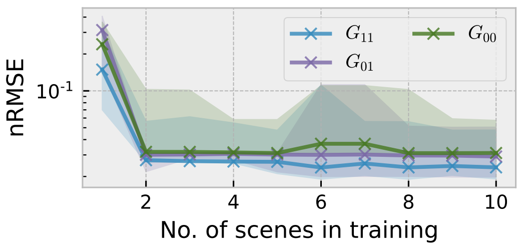
6.4.2 Ablation study of priors in the BSP grammar

To demonstrate the impact of the grammar of Newtonian physics on the overall data efficiency of BSP, we consider two ablations of our grammar: (i) $\mathcal{G}_{01}$, which is $\mathcal{G}$ without the dimensional analysis prior and $\mathcal{G}_{00}$, which is $\mathcal{G}$ without both priors in (i) and (ii); we also refer to $\mathcal{G}$ as $\mathcal{G}_{11}$ in this section. For reference, for a maximum depth of 5, $\mathcal{G}_{00}$ contains 8,593,200 expressions, $\mathcal{G}_{01}$ contains 7,935,408 expression and $\mathcal{G}_{11}$ contains 75,816. For a maximum depth of 6, $\mathcal{G}_{11}$ contains 771,120, and the number for other variants is intractable.

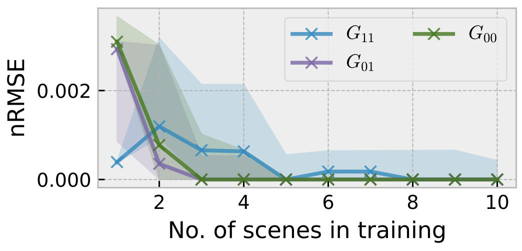
We repeat the experiment from section 6.4.1 for all three grammar variants, and report the results in figure 6.3. As can be seen, both priors in BSP's grammar contribute to data efficiency of its M-step while dimensional analysis has more impact. This is aligned with the analysis of the number of expressions per grammar above, showing that data efficiency improves as the number of possible expressions decreases. There is also a case in which the priors do not show advantages: MAT, in which the friction law is simple enough (the shallowest expression among all) thus easy to find even without priors.



(a) NBODY



(b) BOUNCE (y-axis in log scale)



(c) MAT

Figure 6.3: Ablation study of priors using predictive error on held out scenes given varying number of training scenes. Comparison between $\mathcal{G}_{11}$ and $\mathcal{G}_{01}$ shows the effect of the dimensional analysis prior and Comparison between $\mathcal{G}_{01}$ and $\mathcal{G}_{00}$ shows the effect of the translation invariance prior.

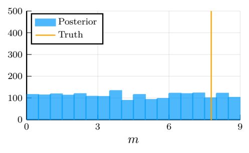
6.5 Experiment: Learning force laws in partially observed environments

We now evaluate BSP's performance in environments with some unknown intrinsic entity properties.[7] We do not consider the neural baselines in this section, as they did not show competitive performance compared to BSP, even in the fully observed environment.

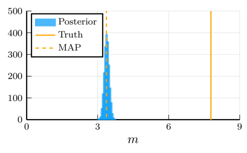
6.5.1 EM performance on SYNTH

We first demonstrate BSP's ability to jointly learn and reason about the environment by recovering the true force law when some properties are unobserved. As an illustrative example, we use three scenes from the NBODY dataset (with four entities per scene), such that if the true masses are given, the M-step can successfully learn the true force law. We assume that the mass of the heavy entity is known but the masses of other the three,

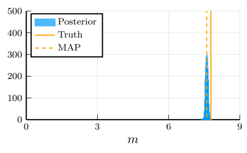
[7]It is worth to mention that there is an identifiability issue for joint reasoning-learning tasks with limited data, which we elaborate in appendix 6.A.5.



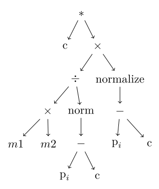
(a) $i = 0$: $F_0 = 0$



(b) $i = 1$: $F_1 = F^{\dagger}$



(c) $i = 3$: $F_3 \approx F^*$



(d) Expression tree for F_3

Figure 6.1: Results of the EM algorithm on NBODY. Figure 6.1a to figure 6.1c shows the posterior of mass for Entity 1 in Scene 1 with the corresponding force function for different EM iteration i. In figure 6.1b, the force function $F^{\dagger} = 239.99 \frac{m_i m_j}{\|\mathbf{p}_i - \mathbf{p}_j\|_2} \frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|_2}$. The constant in figure 6.1d is $c = 2.04 \times 10^3$.

	NBODY			BOUNCE			MAT		
	Median	25% Q	75% Q	Median	25% Q	75% Q	Median	25% Q	75% Q
BSP	8.05e-1	7.83e-1	1.07e0	3.16e-2	3.11e-2	3.94e-2	5.39e-4	3.58e-4	7.63e-4
F_0		1.77e0			5.98e-2			1.20e-3	

Table 6.1: Test predictive performance (nRMSE) on partially observed SYNTH scenarios (using 5 random scenes for training and from 5 different runs). Note BSP consistently beats the constant baseline.

lighter entities are unknown with a uniform prior $\mathcal{U}(0.02, 9)$. We use the EM algorithm to fit the same generative model that simulates the data using BSP. Figure 6.1 shows the posterior distribution over mass and the force function at initialization (figure 6.1a), middle (figure 6.1b), and convergence (figure 6.1c). In this run, after 3 iterations, our algorithm successfully recovers the true force function. We repeat this experiment ten times with randomly sampled scenes. For eight of them, BSP successfully recovers the true force law. Appendix 6.D.1.1 provides another demonstrative example on MAT.

For a quantitative analysis, we run the EM algorithm on each of the three scenarios from SYNTH with 5 random scenes repeated 5 times. In table 6.1 we report nRMSE on a fixed testing set of 20 scenes. Notice that the performance has a large variance due to the fact that not all randomly selected 5-scene subsets provide enough training signal. In all three scenarios, BSP consistently outperforms the zero force baseline and can successfully recover the true force law in some random subsets.

	Mode	25% Q	75% Q
BSP	4.30e-2	3.45e-2	5.09e-2
F_0	5.05e-2	4.61e-2	5.49e-2
BSP	9.22e-3	8.25e-3	9.87e-3
F_0	2.19e-2	2.17e-2	2.21e-2

Table 6.2: Test predictive performance (nRMSE) for FALL (top) and SPRING (bottom)

6.5.2 Real world data: Physics 101

While SYNTH benchmark is interesting, a strength of our approach is the ability to generalize to real world data. To demonstrate this, we use the PHYS101 dataset (Wu et al., 2016), a dataset of real world physical scenes. We consider two scenarios, FALL and SPRING (shown in figure 6.2). As BSP works on symbolic inputs, we pre-process raw videos using standard tracking algorithms from OpenCV to extract observations in numerical form; see the appendix for pre-processing details.

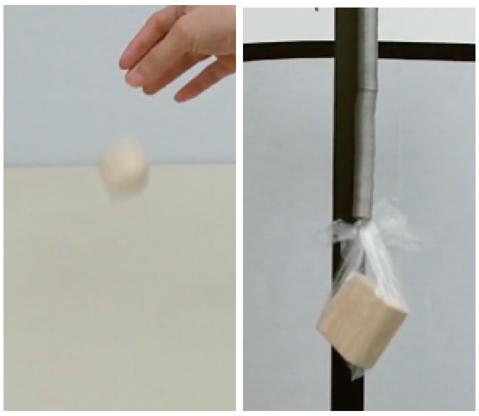


Figure 6.2: Example frames for FALL (left) and SPRING (right)

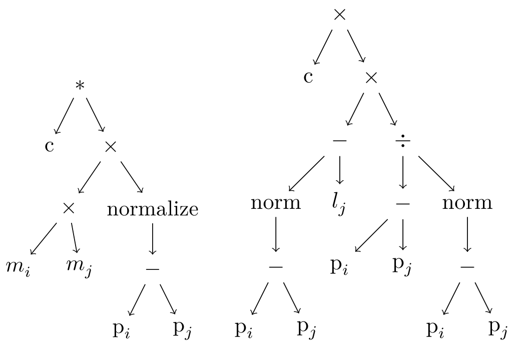


Figure 6.3: Learned force expression for FALL (left) and SPRING (right)

FALL We first train BSP on a single scene from FALL, where an object is dropped on a table. A typical force expression that BSP discovers is $c \times m_i \times m_j \times \text{normalize}(\mathbf{p}_i - \mathbf{p}_j)$, as shown in figure 6.3, suggesting that BSP is able to learn the correct form of gravitational force law. Here, the direction of $\mathbf{p}_i - \mathbf{p}_j$ points towards the table and m_j is the mass of the table, which together with c, serves as the constant g in $F_g = m_i g$. In another solution frequently found by BSP, it learns the direction as $\text{normalize}(\mathbf{v}_i)$ as the velocity is always downwards. BSP can also learn global forces directly if constant vectors $[1,0]$ and $[0,1]$ are provided, which is done in the next section.

SPRING After learning the gravity from FALL, in SPRING, assuming that the original length of the spring is known, we train BSP on a single scene to evaluate if it can learn the Hooke's law $F = kx$. Here k is the tensor coefficient, and x is the displacement of the spring. An example force law that BSP learns in this case is $c \times (\text{norm}(\mathbf{p}_i - \mathbf{p}_j) - l_j) \times (\mathbf{p}_i - \mathbf{p}_j) \div \text{norm}(\mathbf{p}_i - \mathbf{p}_j)$, as shown in figure 6.3, clearly suggesting that BSP can learn Hooke's law.

Finally, to quantitatively evaluate BSP's performance on PHYS101, we select a fixed set of 4 scenes from each scenario and use 2 for training and 2 for testing, repeating for all the permutations of the 4 scenes. The aggregated test performance in terms of normalised RMSE is given in table 6.2. BSP only slightly outperforms compared to the zero force baseline on the FALL. This is because the FALL contains only limited number of frames (< 10) and the trajectories do not diverge too far away from what zero force would predict. For SPRING, BSP outperforms the baseline significantly, closely estimating with ground truth time period of the harmonic motion. We also provide qualitative evaluation via a time-series plot of the change of the block's vertical position with time in the appendix.

6.5.3 Does BSP perform similarly to humans?

In this section, we compare BSP's performance against humans' on the experiment done in (Ullman et al., 2018). For this purpose, we use the ULLMAN dataset from this study, which consists of 60 videos in which a set of discs interact with each other and mats within a bounded area, as exemplified in figure 6.4. While similar to SYNTH, ULLMAN has a lot more diversity in the scenes (stimulus) and reasoning tasks. The force laws in ULLMAN are similar to those in SYNTH but they have different constants and the scenes are generated from a completely different simulator. In the original experiment, each participant is presented with 5 videos. Each of the videos is from a different "world", such that the object properties (for each colour) and force laws are different in every video. For each video, the participant is asked 13 multiple choice questions related to the mass of discs ("Mass"), roughness of mats ("Friction") and types of global ("Global") and pairwise forces ("Pairwise"). For example, "How massive are red objects?" where the options to choose from are "Light", "Medium" and "Heavy". Please refer to appendix 6.D.3 for the complete set of questions and options.

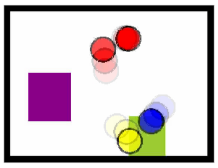


Figure 6.4: Example from ULLMAN

	Human	BSP	Chance
Mass	43%	40%	33%
Friction	44%	39%	33%
Global	68%	55%	20%
Pairwise	62%	50%	33%

Table 6.3: Accuracy per question category

To be consistent with the setup in (Ullman et al., 2018), we assume that the friction and collision forces are known apriori. Thus, the goal is to apply BSP on a **single** scene and infer the properties by learning the expressions for the residual global and pairwise force. The properties to infer are the mass for the discs, the friction coefficients for the mats and a latent property q that controls the pairwise interaction. To accommodate for the global force, we added two constant vectors $[1,0]$ and $[0,1]$ to the grammar; properties related to the known forces are also removed from the grammar. In comparison to models studied in Ullman et al. (2018), BSP aims to learn the force expressions explicitly, rather than inferring binary variables to turn on/off predefined force components. Appendix 6.D.3 provides details on the learning tasks and setup of BSP. We perform 3 runs of BSP on each of the 60 scenes and use the learning results to answer the same set of 13 questions presented to participants in (Ullman et al., 2018).

Table 6.3 summarises the accuracy for humans and BSP on the four question categories. As it can be seen, BSP's performance is worse than that of humans' but convincingly better than chance. Considering the difficulty of inferring 9 properties and learning the targeted force law using only 1 scene, this may not be surprising. There are intriguing similarities between the answers given by BSP and human participants. Both display the same relative order of accuracy across question types "Global" > "Pairwise" > "Friction" > "Mass" while BSP's performance is still inferior to humans'. We hypothesize that humans may have much prior experience with similar physical scenes to answer these questions or they may answer these questions in a different way than explicitly learning the forces. Fully addressing the similarity and difference between BSP and humans requires more analysis that is out of the scope of this paper.

Appendix

6.A Technical details

6.A.1 The complete generative process

Section 6.3 describes the top-down generative model piece by piece. To improve the clarity the presentation, we provide the complete generative process of the observation given force function F, which corresponds to the E-step in our method, as a probabilistic program in algorithm 6.1. In this probabilistic program, we use the keyword ASSUME and OBSERVE for sampling latent variables and observations separately, following the notations from Wood et al. (2014). We also provide an example of the generative process of a three-body problem in a more intuitive manner in figure 6.A.1.

6.A.2 Grammar for Newtonian physical laws: The complete form

The complete grammar following section 6.3.2 is given in figure 6.A.2.

6.A.3 Algorithmic description for the learning method

We provide a complete description of EM algorithm in algorithm 6.2.

We provide a complete description of the cross-entropy method with learnable constants in algorithm 6.3.

6.A.4 Reasoning about unknown properties

Importance sampling In cases where the prior distribution is discrete, the inference is done by importance sampling (IS) which produces a set of weighted samples. As the number of samples to accurately estimate the marginal log-likelihood in the M-step can

Algorithm 6.1 Complete generative process given force laws

```
                                                        ▷ Sample latent variables
 1: for $i = 1, \dots, N$ do
 2:     ASSUME $z^i$ from prior for entity $i$
 3:     if initial state is not given then ASSUME $\mathbf{p}_0^i$ from prior for entity $i$ ASSUME $\mathbf{v}_0^i$
    from prior for entity $i$
 4:     end if
 5:     Set $\mathbf{s}_0^i = (\mathbf{p}_0^i, \mathbf{v}_0^i)$
 6: end for
 7: for $t = 1, \dots, T$ do
 8:     for $i = 1, \dots, N$ do                            ▷ Compute force and acceleration
 9:         for $j = 1, \dots, N$ do
10:             Compute $\mathbf{f}_t^{i,j} = F(z^i, \mathbf{s}_{t-1}^i, z^j, \mathbf{s}_{t-1}^i)$
11:         end for
12:         Compute $\mathbf{f}_t^i = \sum_{j=1}^N \mathbf{f}_t^{i,j}$
13:         Compute $\mathbf{a}_t^i = \mathbf{f}_t^i / m^i$                       ▷ Euler's integration
14:         Update $\mathbf{v}_t^i = \mathbf{v}_{t-1}^i + \mathbf{a}_t \varepsilon$
15:         Update $\mathbf{p}_t^i = \mathbf{p}_{t-1}^i + \mathbf{v}_t^i \varepsilon$
16:         Set $\mathbf{s}_t^i = (\mathbf{p}_t^i, \mathbf{v}_t^i)$                    ▷ Sample observations
17:         OBSERVE $\tilde{\mathbf{p}}_t^i$ from $\mathcal{N}(\mathbf{p}_t^i, \sigma^2)$
18:     end for
19: end for
```

be large and this would induce a large computational cost in the M-step, IS is followed by a re-sampling step to select only a small set of k weighted samples in the M-step $\{(\omega_1, z_1), \dots, (\omega_k, z_k)\}$, where the weights are re-normalized.

Hamiltonian Monte Carlo Since for a fixed F, the generative model in BSP is end-to-end, piecewise differentiable with respect to properties, we can use Hamiltonian Monte Carlo (HMC; Duane et al., 1987; Neal, 2011) for inference. In order to draw k samples from the posterior *robustly* in the E-step, we first run $k + k'$ independent HMC chains by the no-U-turn sampler (NUTS; Hoffman and Gelman, 2014) for a reasonably large number of iterations, where k' is a hyper-parameter. After this, we remove k' chains with the smallest effective sample size (ESS). This reduces the chance of using samples from chains that mixed poorly or got stuck in bad region due to random

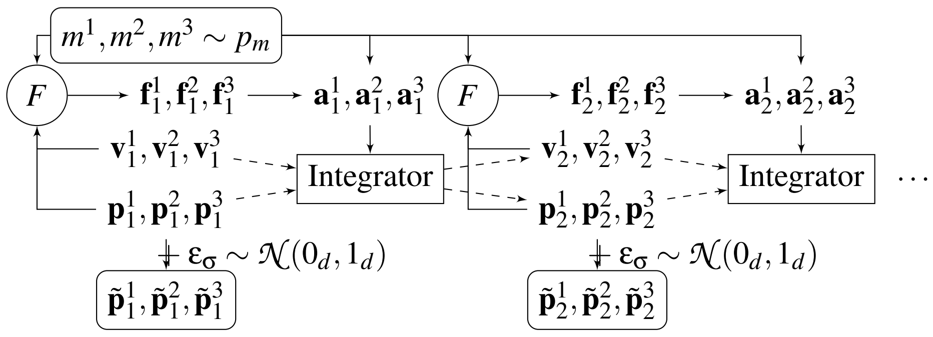

$m^1, m^2, m^3 \sim p_m$
F
$\mathbf{f}_1^1, \mathbf{f}_1^2, \mathbf{f}_1^3$
$\mathbf{a}_1^1, \mathbf{a}_1^2, \mathbf{a}_1^3$
F
$\mathbf{f}_2^1, \mathbf{f}_2^2, \mathbf{f}_2^3$
$\mathbf{a}_2^1, \mathbf{a}_2^2, \mathbf{a}_2^3$
$\mathbf{v}_1^1, \mathbf{v}_1^2, \mathbf{v}_1^3$
Integrator
$\mathbf{v}_2^1, \mathbf{v}_2^2, \mathbf{v}_2^3$
Integrator
$\cdots$
$\mathbf{p}_1^1, \mathbf{p}_1^2, \mathbf{p}_1^3$
$\mathbf{p}_2^1, \mathbf{p}_2^2, \mathbf{p}_2^3$
$+ \varepsilon_\sigma \sim \mathcal{N}(0_d, 1_d)$
$+ \varepsilon_\sigma \sim \mathcal{N}(0_d, 1_d)$
$\tilde{\mathbf{p}}_1^1, \tilde{\mathbf{p}}_1^2, \tilde{\mathbf{p}}_1^3$
$\tilde{\mathbf{p}}_2^1, \tilde{\mathbf{p}}_2^2, \tilde{\mathbf{p}}_2^3$

Figure 6.A.1: The generation of an observed trajectory: a three-body example with unknown mass. Circles are the learnable force function, rectangles are fixed functions, rounded rectangles are random variables and others are deterministic variables.

initialization. Finally, we pick the last sample from each chain as the samples returned by the E-step $\{z_1, \ldots, z_k\}$. To be consistent with the samples from IS, we also assign equal weights $\omega_i = 1/k$ to all samples.

6.A.5 Identifiability in reasoning and learning tasks

It is worth mentioning the fact that reasoning and learning tasks which BSP targets are not necessarily *identifiable*, especially when data is very limited or when force laws and object properties are jointly learned. When the data is limited, a certain level of *diversity of attribute values* in the data has to be provided so that their impact on the force law will be observed. For example, consider a dataset with multiple scenes of a 2-body simulation with the same 2 entities and random initialization of position and velocities. In this setup, no matter how many scenes are given, the actual gravitational force is not identifiable because the product of mass is a constant for all scenes. This can be resolved by introducing more entities in the same scene, or more scenes with entities that have different attributes. In the case of joint reasoning and learning, the interplay between attribute units and learnable constants in the force law could potentially create ambiguity. For example, if a force law acts on an attribute linearly, the learning algorithm is free to scale up the constant in the force law and scale down the attribute value accordingly to reach the same results. This can actually be seen by the fact that constants in force laws have their own units, e.g. the gravitational constant G has a unit of $\mathrm{m}^3\mathrm{kg}^{-1}\mathrm{s}^{-2}$. Scaling the constant and the attribute accordingly can be seen as a unit change. Such ambiguity between properties and force laws is also the reason why one might not want to be Bayesian on force law, because there would be a mode switching problem in posterior

$$\begin{aligned}
\mathit{Constant} &\rightarrow c_1 \mid c_2 \mid c_3 \\
\mathit{Unitless} &\rightarrow \mu_1 \mid \mu_2 \\
&\quad \mid \mu_1 - \mu_2 \mid \mu_1 + \mu_2 \\
\mathit{Kg} &\rightarrow m_i \mid m_j \\
&\quad \mid m_i - m_j \mid m_j + m_i \\
\mathit{KgSq} &\rightarrow m_i \times m_j \mid (\mathit{Kg})^2 \\
\mathit{MeterVec} &\rightarrow \mathbf{p}_i - \mathbf{p}_j \\
&\quad \mid \mathbf{p}_i - \mathbf{c} \mid \mathbf{p}_j - \mathbf{c} \\
\mathit{MeterSecVec} &\rightarrow \mathbf{v}_i \mid \mathbf{v}_j \mid \mathbf{v}_i - \mathbf{v}_j \\
\mathit{Meter} &\rightarrow \|\mathit{MeterVec}\|_2 \mid l_j \\
\mathit{MeterSq} &\rightarrow (\mathit{Meter})^2 \\
\mathit{MeterSec} &\rightarrow \|\mathit{MeterSecVec}\|_2 \\
\mathit{MeterSecSq} &\rightarrow (\mathit{MeterSec})^2
\end{aligned}$$

$$\begin{aligned}
\mathit{TransInvVec} &\rightarrow \mathit{MeterVec} \mid \mathit{MeterSecVec} \\
\mathit{UnitlessVec} &\rightarrow \text{normalize}(\mathit{TransInvVec}) \mid \mathit{MeterVec} \div \mathit{Meter} \\
&\quad \mid \mathit{MeterSecVec} \div \mathit{MeterSec} \\
\mathit{Meter} &\rightarrow \text{project}(\mathit{MeterVec}, \mathit{UnitlessVec}) \\
\mathit{MeterSec} &\rightarrow \text{project}(\mathit{MeterSecVec}, \mathit{UnitlessVec}) \\
\mathit{BaseCoeff} &\rightarrow \mathit{Unitless} \mid \mathit{Kg} \mid \mathit{KgSq} \mid \mathit{KgSq} \div \mathit{Kg} \mid \mathit{Meter} \\
&\quad \mid \mathit{MeterSq} \mid \mathit{Meter} - \mathit{Meter} \mid \mathit{Meter} + \mathit{Meter} \\
&\quad \mid \mathit{MeterSec} \mid \mathit{MeterSecSq} + \mathit{MeterSecSq} \\
&\quad \mid \mathit{MeterSecSq} \mid \mathit{MeterSecSq} - \mathit{MeterSecSq} \\
\mathit{Coeff} &\rightarrow \mathit{BaseCoeff} \mid \mathit{BaseCoeff} \times \mathit{BaseCoeff} \\
&\quad \mid \mathit{BaseCoeff} \div \mathit{BaseCoeff} \\
\mathit{BaseForce} &\rightarrow \mathit{Constant} \times \mathit{Coeff} \times \mathit{UnitlessVec} \\
\mathit{Bool} &\rightarrow \text{isOn}(\mathbf{p}_i, s_i, \mathbf{p}_j, s_j) \mid \text{doesCollide}(\mathbf{p}_i, s_i, \mathbf{p}_j, s_j) \\
\mathit{Force} &\rightarrow \mathit{BaseForce} \mid \mathit{BaseForce} \times \mathit{Bool} \mid \mathit{Force} + \mathit{BaseForce}
\end{aligned}$$

Figure 6.A.2: A grammar of Newtonian physical laws

sampling.

6.B Experimental Details for Section 6.4

All experiments are performed on CPU using two servers. One has Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz and the other has Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz. The two servers has 24 + 32 = 56 cores in total to help run experiments in parallel.

6.B.1 Ground truth forces

The symbolic trees of ground truth forces that are used to generate the datasets that are used in section 6.4 are given in figure 6.B.1.

Algorithm 6.2 Expectation–maximization for Bayesian-symbolic physics

Input: Dataset $\mathcal{D}$, grammar $\mathcal{G}$, number of EM iterations m, sample size k and extra chains k' in E-step, number of repeats in M-step r

Output: A force function F and k samples of latent properties $\{z_1, \dots, z_k\}$

```
1: Initialize the force function $F_0$ as constantly zero
2: for $i = 1, \dots, m$ do
3:     Get $k$ weighted posterior samples $\{(\omega_1, z_1), \dots, (\omega_k, z_k)\}$ by IS or HMC    ▷ E-step
4:     Define current loss function $\mathcal{L}_i(e, c) = \sum_{i=1}^{k} \omega_i \mathcal{L}(e, c; z_i, \mathcal{D})$    ▷ M-step starts
5:     Get candidates $\mathcal{C} = \{(t_1^*, c_1^*), \dots, (t_r^*, c_r^*)\}$ by algorithm 6.3 with $\mathcal{L}_i$ for $r$ repetitions
6:     Find $(t^*, c^*)$ from $\mathcal{C}$ with the best loss and set $F_i = \text{getF}(t^*, c^*, \mathcal{G})$    ▷ Update force
7: end forreturn $F = F_m$ and $\{z_1, \dots, z_k\} \sim p(z \mid \mathcal{D}; F_m)$
```

6.B.2 Neural baselines

We now describe the neural baselines. Notation-wise, we use d_{in} to denote the total dimension of properties and state (position and velocity) for each entity, and use d_{out} for the dimension of position/velocity/force dimension (2 in our case). The corresponding implementation can be found in `src/network.jl` in our source code, and the hyper-parameters (network sizes and training) can be found in `scripts/runexp.jl`, which we also summarise below.

OGN For the OGN baseline, we use an MLP of $d_{\text{in}} \to 100 \to 50$ as the node model and a MLP of $(50 + 50) \to 100 \to 100 \to 100 \to d_{\text{out}}$ as the edge model; the activation function is the rectified linear unit (ReLU) for both models. For training, we use the ADAM optimizer (Kingma and Ba, 2014) with a learning rate of 2×10^{-3} for 2,000 epochs.

In addition, for OGN, we found that we need provide additional prior knowledge on how forces are related to the mass and acceleration by parameterising them as $F_e(\cdot) = m a_\theta(\cdot)$, where θ is NN parameters, otherwise they fail to learn. This parameterization is fact consistent with (Sanchez-Gonzalez et al., 2019) in which NNs output partial derivatives of the Hamiltonian system.

Algorithm 6.3 Cross-entropy method with learnable constants

Input: Grammar $\mathcal{G}$ with learnable constants c, loss function $\mathcal{L}$, total population number n, selected population number k, number of iterations m and maximum tree depth d

Output: An expression tree e^* with optimized constants c^*

```
 1: initialize a PCFG $\mathcal{P}_0$ for $\mathcal{G}$ uniformaly
 2: for $i = 1, \dots, m$ do
 3:     Initialize an empty candidate set $\mathcal{C}$
 4:     for $j = 1, \dots, n$ do
 5:         Sample an expression $e_j \sim \mathcal{P}_{i-1}, e_{i-1}$ with a maximum depth of $d$
 6:         Solve $c_j^* = \arg\min_c \mathcal{L}(e_j, c)$ by L-BFGS        ▷ Lower-level optimization
 7:         Compute the loss of the sampled tree $\ell_j = \mathcal{L}(e_j, c_j^*)$ and add $(e_j, \ell_j)$ to $\mathcal{C}$
 8:     end for
 9:     if $i < m$ then
10:         Fit a PCFG $\mathcal{P}_i$ on trees from $\mathcal{C}$ with the top-$k$ fitness via maximum-
    likelihood
11:     end if
12: end forreturn the best expression tree $e^*$ from $\mathcal{C}$ and the corresponding constant
    as $c^*$
```

IN The node model for IN is same as that of OGN. The edge model for IN is same as that of OGN except the output dimension of the last layer is 50. There is an extra network for transition, a MLP of shape $N \times 50 \to 100 \to 100 \to 100 \to 2 \times N \times d_{\text{out}}$ (with ReLU activations), that takes the concatenation of embeddings for N entities and outputs the change of the next state (position and velocity) of the whole system. The training uses the ADAM optimizer that of OGN but with a learning rate of 1×10^{-3} for 400 epochs.

MLP force The MLP (force) baselines has a neural network that inputs the states of a pair of entities and outputs the force from one of them applies to the other. The network is an MLP of shape $2 \times d_{\text{in}} \to 100 \to 100 \to 100 \to d_{\text{out}}$ (with ReLU activations). The training uses the ADAM optimizer that of OGN but with a learning rate of 1×10^{-3} for 2,000 epochs.

MLP position The MLP (force) baselines has a neural network that inputs the state and properties of the whole system (as the concatenation of N entities) and outputs the change of the next state (position and velocity) of the whole system. The network is a MLP of shape $N \times d_{\text{in}} \rightarrow 100 \rightarrow 100 \rightarrow 100 \rightarrow 2 \times N \times d_{\text{out}}$ (with ReLU activations). The training uses the ADAM optimizer that of OGN but with a learning rate of 1×10^{-3} for 400 epochs.

6.B.3 A close look at the approximated bounce law

6.B.3.1 The learned bounce law

As mentioned in section 6.4 and discussed in section 6.B.3.2, the only case in which BSP fails to infer the true law (within 10 scenes) is of special interest and requires for further inspection. A typical approximate law learned in section 6.4 is shown in figure 6.B.2; see section 6.B.3.2 for discussion on how this law differs from the true one. To highlight, there are basically two mismatches between the true law and the learned law. First, there is no projection operation that correctly calculates the effect of speed. Second, the mass-based coefficient is missing. To assist inspection, we also provide some visualizations in figure 6.B.3 using initial conditions from the training set for inspection. The corresponding animations can be found in the supplementary material (the `suupl/bounce_inspection` folder; see `bsp.xuk.ai`).

6.B.3.2 Generalization in new scenes

It is worth checking how the laws learned in section 6.4 generalize to new scenes beyond the training data. In cases where the true law is successfully recovered, the expression will generalize to novel scenes undoubtedly. Therefore, it is more interesting to inspect the generalization ability of an approximate law, that is a law which is not completely equivalent to the true law but is close. The emerged law for the BOUNCE dataset is such an example as mentioned earlier. It has an expression of $F^{\dagger} = c \left\|\mathbf{v}_i - \mathbf{v}_j\right\|_2 \frac{\mathbf{p}_i - \mathbf{c}}{\left\|\mathbf{p}_i - \mathbf{c}\right\|_2} \text{doesCollide}(\mathbf{p}_i, s_i, \mathbf{p}_j, s_j)$; see figure 6.B.2 in appendix 6.B.3.1 for the actual tree. Although it is not identical to the true law, it is still a good approximation: it takes into accounts the velocity difference into consideration and finds the correct force direction. We now consider applying this law to a completely new scene: a vertical-view world where the gravity is pointing in downward direction. Figure 6.B.4 shows the predicted trajectory with true and the approximate law with two different initial conditions. As it can be seen, the approximate law successfully generalize this novel world. For the first condition, the projection is very close to the true one, while for the second condition, the concept of bounce is also correctly transferred. The corresponding animations for these plots can also be found in the supplementary material for further inspections (the `suppl/generalization` folder; see `bsp.xuk.ai`).

6.C Figure 6.2 with all methods displayed

Figure 6.2 omits some poor results for better visualization. The corresponding plots with all methods displayed are shown in figure 6.C.1.

6.D Experimental details for Section 6.5

6.D.1 SYNTH

Hyper-parameters We refer readers to `scripts/runexp.jl` of our source code for hyper-parameters used in the quantitative experiments (table 6.1). For the rest, in the E-step, we use $k = 3$ and $k' = 2$ and the hyper-parameters for NUTS are: 150 adaptation steps, 150 HMC iterations, a maximum tree depth of 4 and a target acceptance ratio of 0.75. In the M-step, we repeat $r = 2$ runs and the hyper-parameters for the cross-entropy method are: 800 total populations, 400 selected populations, 4 iterations and a maximum depth of 10. The weighting parameter for the PCFG prior are 1 for the NBODY and BOUNCE datasets and 1×10^{-4} for the MAT dataset.

6.D.1.1 An extra demonstration of EM on MAT

As another example, we use five scenes from the (noisy) MAT dataset. We assume that the only unknown is the friction coefficient of the mat with a truncated Gaussian prior $\mathcal{T}\text{runcated}(\mathcal{N}(\mu_0, 2^2), 0, 5)$ (truncated between 0 and 5), where μ_0 is the true coefficient, Note that the variance 2^2 is large enough to be uncertain, justifying a fair choice of the prior. Similarly, we use the EM algorithm to fit the same generative model that simulates the data using BSP. figure 6.D.1 shows the posterior distribution over mass and the force function at initialization (6.D.1a), middle (6.D.1b) and convergence (6.D.1c) of the algorithm. Compared the expression at convergence with the true law, the algorithm learns $\mathbf{v}_i - \mathbf{v}_j$ instead of $\mathbf{v}_i$ as the mat velocity is zero, i.e. $\mathbf{v}_j = 0$, in all scenes,

6.D.2 PHYS101

Pre-processing of videos We use an open-source implementation of standard tracking algorithms from OpenCV to track the entities. The code is available at `github.com/bikz05/object-tracker`. To use the tracker, we manually select a bounding box of the entity of interest and run the tracking algorithm. For FALL, it is done for the falling object; for SPRING, this is done for both the hanging object and the spring joint.

For FALL, we found that the tracking algorithms can fail if the object is too fast. In such cases, there are only limited frames (< 10) to track thus we manually annotate these frames to get the trajectory of the falling object.

The processed data can be found in `data/phys101/processed/` of the supplementary material; see `bsp.xuk.ai`.

Qualitative evaluation To qualitatively see how well this learned force from SPRING performs at prediction, we also show that how the vertical coordination of the object position changes over time in figure 6.D.2. As it can be seen, the learned force produces prediction that matches the periodicity quite well with some small deviation from the amplitude.

6.D.3 ULLMAN

Pre-processing of visual stimulus We preprocess the videos from ULLMAN by derendering the objects to symbolic forms, i.e. position trajectories of all entities. This is done by template matching of the discs and mats. We manually crop the video frames to obtain templates for discs with three different colours and mats with three different colours, and match the location of each of them for each frame. The code for this preprocessing step can be found in `scripts/preprocess-ullman.py` of the source code and the processed data can be found in `data/ullman/processed/` of the supplementary material.

On reverse-engineering the collision and friction forces from stimulus As the ULLMAN data is generated by an unknown simulator, the ground truth forces are not directly accessible. Therefore, we need to "reverse engineer" these forces so that we can provide them to BSP a prior, which is consistent with the setup of human study in Ullman et al. (2018). We assume the ground truth forces for friction and collision have their pre-defined expressions, similar to those used for the simulator for SYNTH. However, each of these expressions also contains a constant that is unknown for the actual simulation of the ULLMAN data. To this end, we use World 1 from the ULLMAN data to fit these constants because World 1 contains only friction and collision. For the rest of experiments of BSP, we assume these "reverse engineered" forces are given and BSP only needs to learn the residual, as detailed next. Note that this reverse engineering step may introduce systematic bias to the rest of learning as well if there is a mismatch between the actual ground truth. In some of our exploratory analysis on the mass inference results from BSP, we unexpectedly found that BSP can confuse heavy objects with light objects. This is different from the pattern of confusion that the human subjects display. We hypothesize this is due to the potential mismatch between the ground truth collision and friction forces and the reverse-engineered forces that we provide to BSP.

Details for the learning task As there are three discs and three mats, the number of properties to infer is nine in total. The residual force to learn has the form: $C_1 \frac{f(q_1,q_2)}{\|\mathbf{p}_1-\mathbf{p}_2\|_2^2}\boldsymbol{u}+C_2\boldsymbol{u}_C$, where C_1 and C_2 are constants, f itself is an expression of how the sign of the pairwise force depends on q_i and q_j, $\boldsymbol{u}$ is the direction of the pairwise force and $\boldsymbol{u}_C$ is the direction of the global force (up, down, left or right).

Questions and options presented to participants Participants are asked for a set of questions that would not reveal personally identifiable information.

1. Mass related questions (3)

- How massive are [red] objects?
- How massive are [yellow] objects?
- How massive are [blue] objects?

Options are “Light”, “Medium” and “Heavy”.

2. Friction coefficient related questions (3)

- How rough are [green] patches?
- How rough are [purple] patches?
- How rough are [brown] patches?

Options are “As smooth as the table-top”, “A little rough” and “Very rough”.

3. Pairwise force related questions (6)

- How do [red] and [red] objects interact?
- How do [red] and [yellow] objects interact?
- How do [red] and [blue] objects interact?
- How do [yellow] and [yellow] objects interact?
- How do [yellow] and [blue] objects interact?
- How do [blue] and [blue] objects interact?

Options are “Attract”, “Repel” and “None”.

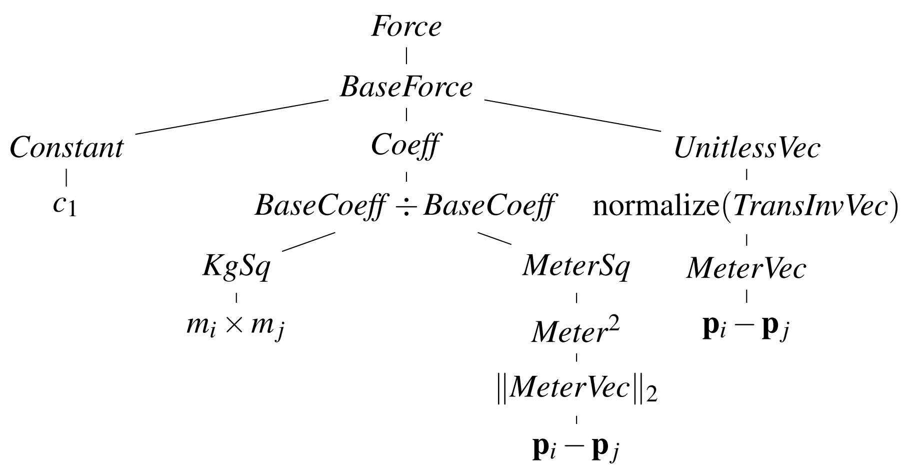
4. Global force related questions (1)

- Is a global force pulling the objects, and if so in what direction is it pulling?

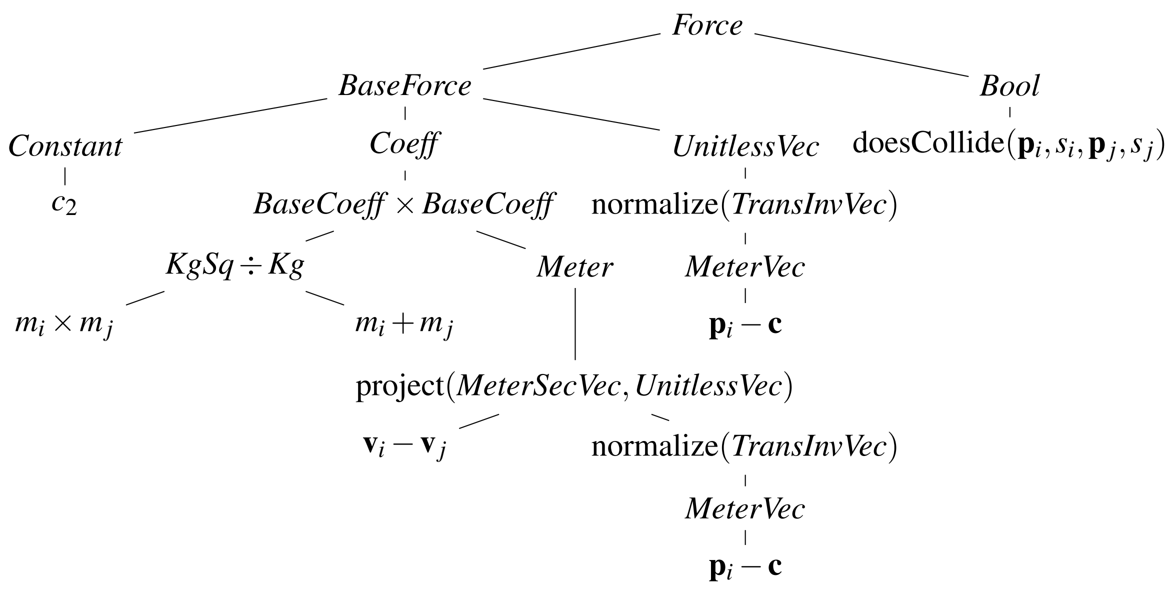
Options are “Yes, it pulls North”, “Yes, it pulls South”, “Yes, it pulls East”, “Yes, it pulls West” and “No global force”.

6.E Reproducibility

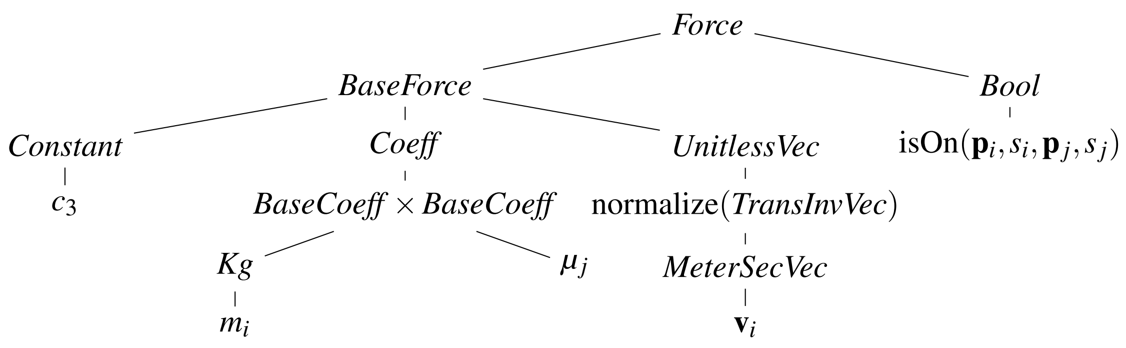
The source code of experiments and the synthetic or processed data can be found at `https://code.xuk.ai/bsp/`.



(a) Gravitational force, $c_1\ m_i m_j\ \frac{1}{\|\mathbf{p}_i - \mathbf{p}_j\|_2}\ \frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|_2}$, (depth 8); $c_2 = 2/\varepsilon = 100$ where $\varepsilon = 2 \times 10^{-2}$, (depth 7); $c_1 = 8.17 \times 10^3$


Force
BaseForce
Bool
Constant
Coeff
UnitlessVec
doesCollide($\mathbf{p}_i, s_i, \mathbf{p}_j, s_j$)
c_2
BaseCoeff × *BaseCoeff*
normalize(*TransInvVec*)
KgSq ÷ *Kg*
Meter
MeterVec
$m_i \times m_j$
$m_i + m_j$
$\mathbf{p}_i - \mathbf{c}$
project(*MeterSecVec*, *UnitlessVec*)
$\mathbf{v}_i - \mathbf{v}_j$
normalize(*TransInvVec*)
MeterVec
$\mathbf{p}_i - \mathbf{c}$

(b) Collision force, $c_2\ \frac{m_i m_j}{m_i + m_j}$ project$(\mathbf{v}_i - \mathbf{v}_j, \frac{\mathbf{p}_i - \mathbf{c}}{\|\mathbf{p}_i - \mathbf{c}\|_2})\ \frac{\mathbf{p}_i - \mathbf{c}}{\|\mathbf{p}_i - \mathbf{c}\|_2}$ doesCollide$(\mathbf{p}_i, s_i, \mathbf{p}_j, s_j)$, (depth 8); $c_2 = 2/\varepsilon = 100$ where $\varepsilon = 2 \times 10^{-2}$ is the step size of the integrator



(c) Friction force, $c_3\ m_i \mu_j\ \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|_2}$ isOn$(\mathbf{p}_i, s_i, \mathbf{p}_j, s_j)$, (depth 5); $c_3 = 9.8$

Figure 6.B.1: Expression trees (under $\mathcal{G}$) of true force laws that generates the datasets used in section 6.4.

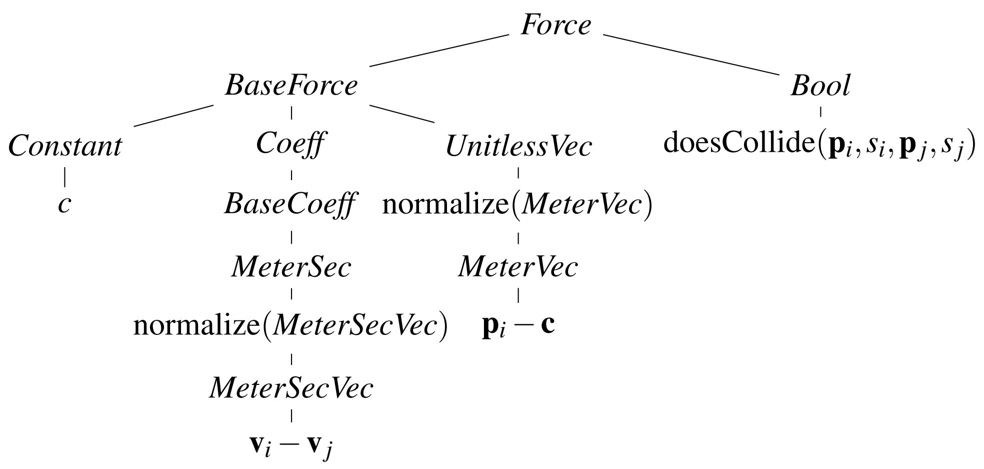
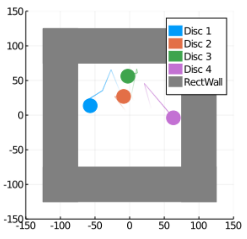
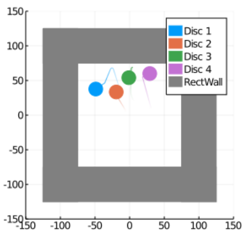
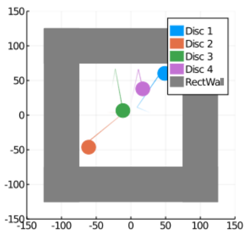
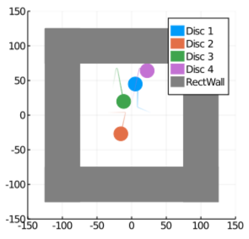


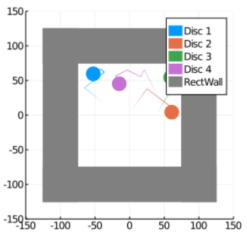
Figure 6.B.2: Approximate bounce law, $c\,\|\mathbf{v}_i - \mathbf{v}_j\|_2 \frac{\mathbf{p}_i - \mathbf{c}}{\|\mathbf{p}_i - \mathbf{c}\|_2}$ doesCollide$(\mathbf{p}_i, s_i, \mathbf{p}_j, s_j)$, learned by BSP under our grammar; $c = 130.22$

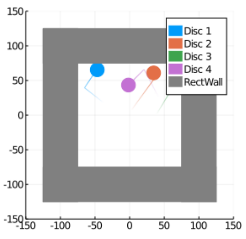

(a) Scene 1 (true)

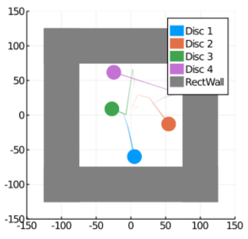

(b) Scene 1 (learned)


(c) Scene 2 (true)


(d) Scene 2 (learned)


(e) Scene 3 (true)


(f) Scene 3 (learned)


(g) Scene 4 (true)

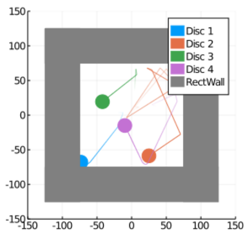
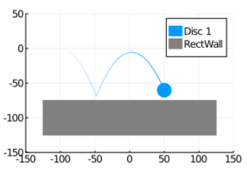
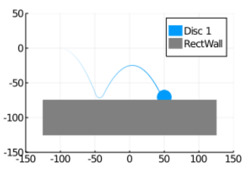
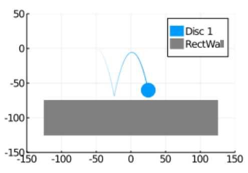

(h) Scene 4 (learned)

Figure 6.B.3: Predicated trajectories of the true bounce law and the learned bounce law.


(a) Condition 1 (true)


(b) Condition 1 (learned)


(c) Condition 2 (true)

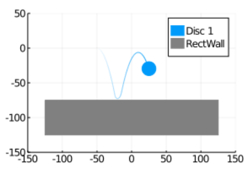
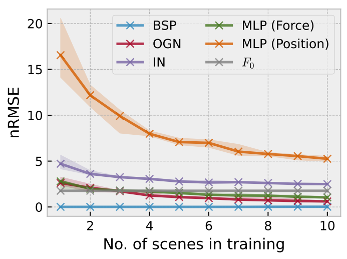
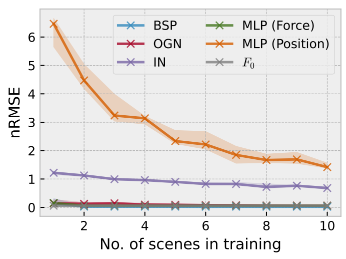

(d) Condition 2 (learned)

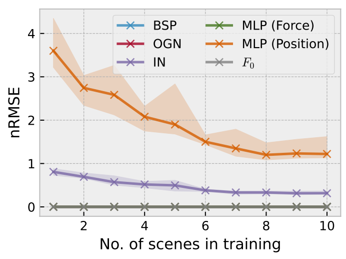
Figure 6.B.4: Generalization of the approximate bounce law in a vertical world with downward gravity.



(a) NBODY

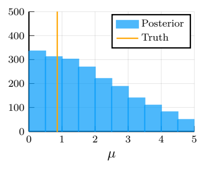


(b) BOUNCE

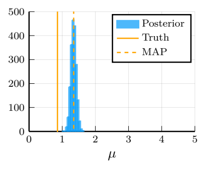


(c) MAT

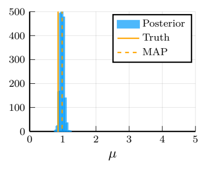
Figure 6.C.1: Comparison of neural baselines and BSP, using predictive error on held out scenes given varying number of training scenes. Some baselines are not displayed due to very poor performance.



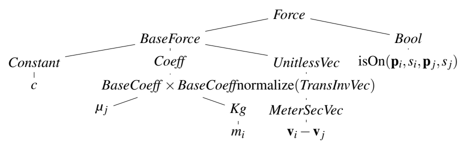
(a) $i = 0$: $F_0 = 0$



(b) $i = 1$: $F_1 = F^\dagger$



(c) $i = 4$: $F_4 \approx F^*$



(d) Expression tree for F_4

Figure 6.D.1: Results of the EM algorithm on MAT. figure 6.D.1a to figure 6.D.1c shows the posterior of friction coefficient in Scene 2 with the corresponding force function during EM. In figure 6.D.1b, the force function $F^\dagger = -22.99\, \mu_j m_i \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|_2}$ isOn$(\mathbf{p}_i, s_i, \mathbf{p}_j, s_j)$. The constant in figure 6.D.1d is $c = -8.605$.

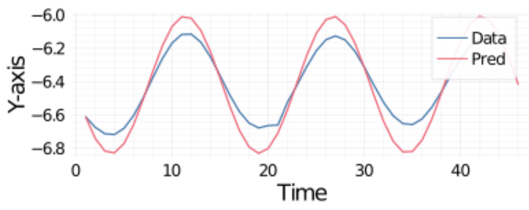


Figure 6.D.2: Prediction of the vertical position

Chapter 7

Free and Open-source Software for Generative Modelling

7.1 The TURING probabilistic programming ecosystem

7.1.1 A brief introduction to probabilistic programming

When applying generative modelling in practice or to a new application, it is usually necessary first to derive the inference method, e.g. in the form of VI or MCMC algorithms, and then implement it in application-specific code, which can be challenging to practitioners due to the level of mathematics involved and the engineering efforts to ensure the implementation is bug-free. What is more challenging is that in Bayesian analysis, building models from data is often an iterative process, where a model is proposed, fit to data and modified depending on the inference results. These technical difficulties discourage researchers from specific domains (e.g. biology) who are not experts of generative modelling. In contrast, deep learning methods have benefited enormously from easy-to-use frameworks based on automatic differentiation that implement end-to-end optimisation. There is a real potential for automated probabilistic inference methods (in conjunction with existing automated optimisation systems) to make generative modelling more accessible.

Probabilistic programming languages (PPLs; Murray, 2013; Goodman and Stuhlmüller, 2014; Wood et al., 2014; Carpenter et al., 2017; Minka et al., 2018; Bingham et al., 2019; Cusumano-Towner et al., 2019) aim to fill this gap by providing a very flexible framework for defining probabilistic models and automating the model learning process using generic inference engines. This frees researchers from writing complex models by hand and enables them to focus on designing a suitable model using their insight and expert knowledge, and accelerates the iterative process of model modification. Moreover, probabilistic programming languages make it possible to implement and publish novel learning and inference algorithms in the form of generic inference engines. This enables fair direct comparison between new and existing learning and inference algorithms on the same set of problems, something that is sorely needed by the scientific community. Furthermore, modern PPLs open further opportunities to integrate different paradigms such as deep learning, differential equations, symbolic methods, etc. into a single framework, which opens a lot of opportunities to combine the strength of different approaches.

7.1.2 The TURING probabilistic programming language

TURING.JL is a Julia library for general-purpose probabilistic programming. It allows the user to write models using standard Julia syntax, and provides a wide range of sampling-based inference methods for solving problems across probabilistic machine learning and Bayesian statistics. Compared to other probabilistic programming languages, Turing has a special focus on modularity, and decouples the modelling language (i.e. the compiler) and inference methods. This modular design, together with the use of a high-level numerical language Julia, makes Turing particularly extensible: new model families and inference methods can be easily added. More importantly, infrastructure-level functionality such automatic differentiation (AD), hardware acceleration (such as CUDA support) is also supported in a Turing-agnostic way thus users can plug any other libraries. For example, users can use DIFFERENTIALEQUATIONS.JL (or DIFFEQS.JL) for differential equations and efficient solvers in Turing's probabilistic program and use HMC samplers which would require to differentiate through the solvers to compute gradients.

"Hello, world!" in TURING Below is a simple Gaussian model with unknown mean and variance written in TURING.

```julia
@model GaussUnknown(xs) = begin
    # Latent variables
    v ~ InverseGamma(2,3)
    m ~ Normal(0, sqrt(v))
    # Observations
    for i in eachindex(xs)
```

```
        xs[i] ~ Normal(m, sqrt(v))
    end
end

chain = sample(GaussUnknown([1.5, 2.0]), NUTS(), 2_000)
```

The model `GaussUnknown` is defined by `@model` macro as an generative process with intuitive, math-like notations. The argument `xs` of the model is assumed to be potential observations that can be passed in later and any other variables on the left-hand side of `~` are assumed to be latent variables. After being defined, users can conditon the generative process on observations by `GaussUnknown([1.5, 2.0])`. This conditioned model can then be used in the `sample` function to draw samples using the specific samplers. In this example, we draw $2,000$ samples using `NUTS`.

A short history of TURING The first version of TURING was designed and developed by Hong Ge and Zoubin Ghahramani in the Computational and Biological Learning Lab at the University of Cambridge. Originally TURING only supports simulation-based inference methods such as importance sampling, sequential Monte Carlo and particle filtering. I started working on TURING to implement Hamiltonian Monte Carlo sampler as my master project, during which the modelling language and the compiler, trace structure (how random variables are stored and fetched in probabilistic programs) and automatic differentiation are redesigned or added. After that, Hong and I designed and implemented the compositional inference interface, as introduced in Ge et al. (2018).

I will not expand on details of TURING as this is not the focus of this chapter; for that please refer to Ge et al. (2018) or `https://turing.ml`. Instead, I will discuss one major practical difference between TURING and other PPLs, which is how different functionalities are implemented as modular packages that, together with non-Turing packages, forms a probabilistic programming ecosystem.

7.1.3 From a framework to an ecosystem

As the time when Ge et al. (2018) was published, TURING was still a standalone library that contains all its functionality, just like all PPLs are implemented. However, during the integration of Turing with many other libraries, e.g. the differential equation package mentioned earlier, it was found that an integrated PPL system is hard to debug against a third-party library. For example, it is ideal to debug the inference algorithm outside the modelling language to avoid potential issue from the compiler. The TURING team soon realised the importance of modularising each functionality, even into separate packages. As the first step, I extracted all codes related to HMC from TURING.JL and put them into a separate package called ADVANCEDHMC.JL with some important refactoring and redesign. Such separation also enforces us, the developers and the researchers, to design inter-package abstraction that works well in practice and allows fundamental research innovations. Since then, different components of TURING.JL has been moved to separate packages, e.g. the compiler is now in DYNAMICPPL.JL and the chain analysis is in MCMCCHAINS.JL. One interesting outcome is that these components are no longer TURING-specific—other PPLs in Julia have also been using the inference libraries developed by the TURING team, which allows their focus on the programming language side without maintaining the inference code. Another important outcome is that new research in PPLs can also be done on specific packages without touching unnecessary parts. For example, work has been done on static analysis on probabilistic programs to extract variable dependencies, which can be later used in inference algorithms to improve efficiency. This feature is implemented in IRTRACKER.JL (IR is short for intermediate representation; Gabler et al., 2019) and made available to TURING as a "plugin".

Figure 7.1 shows a diagram of how different packages from TURING and other libraries form a collaborating ecosystem that allows different levels of interoperation. A few highlights are as below

- Shared infrastructure support: As mentioned, TURING.JL can use AD (forward-mode AD via FORWARDDIFF.JL, tape-based reverse-mode AD via REVERSEDIFF.JL or TRACKER.JL, AD based on source-code transformation via ZYGOTE.JL, etc) and CUDA (CUDA.JL) together with other packages. It is made possible by the language features from Julia together with generically implemented AD and hardware support. Compared with frameworks like TensorFlow, PyTorch and JAX which have their own numerical libraries and the corresponding AD systems, the numerical functionality are well supported at language level by Julia (linear algebra, statistics, etc) and AD can be implemented to support any program written in Julia thanks to the multiple dispatch paradigm that Julia takes for polymorphism. From the user point of view, they usually do not need to think about the underlying AD system and just need to write normal Julia code. Also, because numeric computations is decoupled from AD systems, users are provided with the options to switch between different AD backends, which allows them to

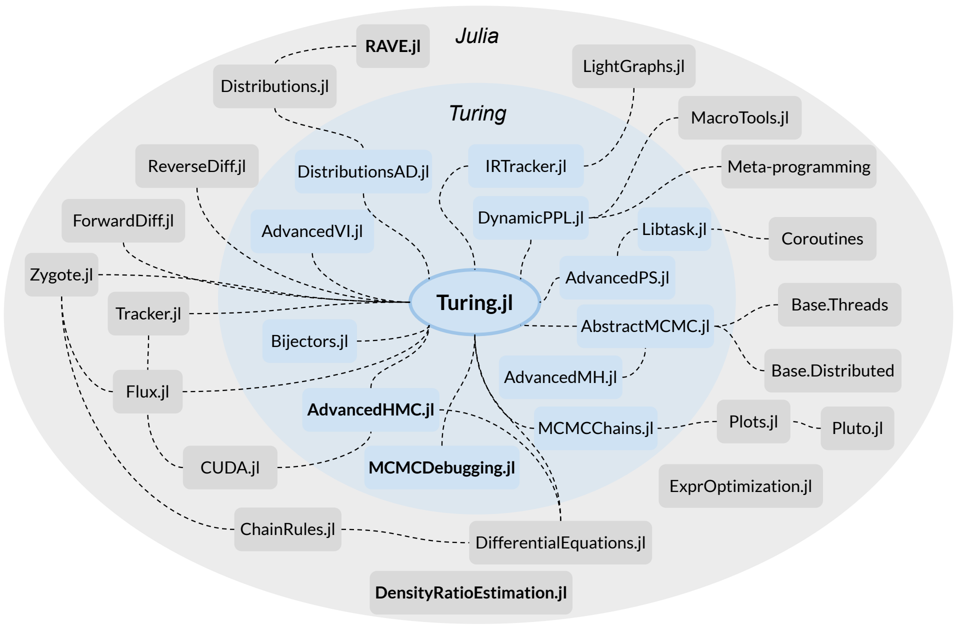


Figure 7.1: The TURING probabilistic programming ecosystem in Julia. Blue area covers packages developed and maintained by the Turing team, forming the TURING ecosystem. Broader region covers other packages in Julia. Packages with name in bold are actively maintained by me.

take advantages of each AD paradigms.

- Bayesian deep learning: Neural networks implemented via FLUX.JL can be plugged into the probabilistic program in TURING, thus implementing Bayesian deep learning. This is very convenient feature to obtain model uncertainty for small neural networks but such combination does not address the computation challenge Bayesian deep learning faces when NNs are large. However, such interoperation can be used to implement common-ground benchmark to accelerate new efficient inference methods for Bayesian neural networks.

- Bayesian differential equations: Differential equations are useful tools to represent physical quantities, dynamics, etc. They usually consist some unknown parameters and one way to estimate them is via Bayesian inference as to characterise the uncertainty in these parameters. The way to implement this in Julia is a simple inter-operation between TURING.JL and DIFFERENTIALEQUATIONS.JL.

Note that the support of differential equations is not unique for TURING and it is also supported in STAN. However, the advantage of the TURING.JL or Julia is that any new functionality in DIFFERENTIALEQUATIONS.JL, which is maintained by researchers specialised in differential equations, will be available in TURING by default. However, for other PPL teams like STAN, they would have to maintain their own differential equation implementations, which is non-trivial on its own. In fact, DIFFERENTIALEQUATIONS.JL is the differential equation library that has the largest number of solvers.

- MCMC debugging: Not only the efforts from the user side can be automated but also the development efforts. One example is debugging new MCMC samplers. MCMCDEBUGGING.JL implements the Geweke test (Geweke, 2004) to diagnose the correctness of MCMC samplers through models defined in TURING.
- Bayesian-symbolic programming: The Bayesian-symbolic framework introduced in chapter 6 is implemented by TURING.JL and EXPROPTIMIZATION.JL, which is a library that provides an intuitive syntax to define grammar through production rules. Each of them automates Bayesian inference and symbolic regression and can be easily used together. Note that TURING.JL is naturally capable of differentiable through the expression generated by EXPROPTIMIZATION.JL to run HMC, which is non-trivial for other alternatives unless specifically developed to support.

As AHMC is the start point of the transformation from a single library into an ecosystem and the most popular functionality/package within the TURING ecosystem, I will expand on the design and performance of AHMC in the next section.

7.2 ADVANCEDHMC.JL: An efficient and user-friendly HMC implementation

Hamiltonian Monte Carlo (HMC) is an efficient Markov chain Monte Carlo (MCMC) algorithm which avoids random walks by simulating Hamiltonian dynamics to make proposals (Duane et al., 1987; Neal, 2011). Due to the statistical efficiency of HMC, it has been widely applied to fields including physics (Duane et al., 1987), differential equations (Kramer et al., 2014), social science (Jackman, 2009) and Bayesian inference (e.g. Bayesian neural networks; Neal, 2012). The No-U-Turn Sampler (NUTS; Hoffman and Gelman, 2014) is an extension of the HMC sampler which automatically tunes two key parameters, the leapfrog step size and integration time (aka trajectory length), which used to require manual adjustments through extensive preliminary runs. Together with a robust implementation in the STAN probabilistic programming language (PPL), NUTS has become the default choice for HMC sampling for many probabilistic modelling practitioners (Carpenter et al., 2017).

Although the integrated implementation of NUTS in STAN makes Bayesian inference easy for domain experts relying on the STAN language, it is desirable to have a high quality, standalone NUTS implementation in a high-level language, e.g. for research on HMC algorithms, reproducible comparisons and real-world approximate inference applications. To this end, we introduce ADVANCEDHMC.JL (AHMC), a robust, modular and efficient implementation of STAN's NUTS and several other commonly used HMC variants in Julia.[1]

7.2.1 A modular Hamiltonian Monte Carlo implementation

The idea behind AHMC is to allow users to easily construct HMC samplers in a similar way that how HMC is mathematically defined. This idea is implemented through defining abstract types over different components of the HMC algorithm (such as integrators, trajectory samplers, adaptors) and defining concrete behaviour of the function for different concrete types of each component or for a specific combination of a few concrete types, through *multiple dispatch*.[2] Compared to a potential solution using object-oriented programming, this paradigm maximally decouples the implementation and allows reuse of codes for different components, which gives a combinatorial number of valid HMC samplers. Importantly, it is well aligned with how HMC is mathematically defined for different types of HMC algorithms.[3]

[1]Code is available at `https://github.com/TuringLang/AdvancedHMC.jl`.

[2]In programming languages, multiple dispatch, also called multimtehods, is a paradigm of polymorphism in which specific implementations of a a function can be dispatched based on the concrete types of augments at run-time (Bansal, 2010). This feature is supported in a range of programming languages such as C#, Groovy, Common Lisp and Julia. It is in fact a core design of Julia and Julia heavily uses it in its own standard library (Bezanson et al., 2017).

[3]In fact, function definitions in mathematics also implements multiple dispatch. Recall how the addition operation is defined differently for different types of numbers such as integer and real. In fact, we do have a type system for numbers and then define addition for each type, such as integer, real and complex, i.e. we have the same function called addition but multiple implementations/definitions of it based on the actual number types it acts on.

As types (abstract or concrete) can be seen as sets of entities following the same semantics, we will now describe what kinds of HMC samplers are supported in AHMC as sets. AHMC supports various HMC algorithms in the set below resulted from a Cartesian product of a set of HMC trajectories and a set of adaptors:

$$(\textsc{StaticTrajectory} \cup \textsc{DynamicTrajectory}) \times \textsc{Adaptor}.$$

Here StaticTrajectory contains a set of concrete types for HMC with fixed-length trajectory length, which contains HMC with fixed step size and step numbers and HMC with fixed total trajectory length. DynamicTrajectory is more involved: It is a set of HMC with adaptive trajectory length, which is defined as a Cartesian product of four sets of different HMC components:

$$\textsc{Metric} \times \textsc{Integrator} \times \textsc{TrajectorySampler} \times \textsc{TerminationCriterion},$$

where

$$\begin{aligned}
\textsc{Metric} &= \{\textsc{UnitEuclidean}, \textsc{DiagEuclidean}, \textsc{DenseEuclidean}\} \\
\textsc{Integrator} &= \{\textsc{Leapfrog}, \textsc{JitteredLeapfrog}, \textsc{TemperedLeapfrog}\} \\
\textsc{TrajectorySampler} &= \{\textsc{Slice}, \textsc{Multinomial}\} \\
\textsc{TerminationCriterion} &= \{\textsc{ClassicNoUTurn}, \textsc{GeneralisedNoUTurn}\}
\end{aligned}$$

Finally, Adaptor consists of any BaseAdaptor or any composition of two or more BaseAdaptor, where $\textsc{BaseAdaptor} \in \{\textsc{Preconditioner}, \textsc{NesterovDualAveraging}\}$. Preconditioner behaves differently based on the choice of metric spaces (Metric). A special composition called StanHMCAdaptor is provided to compose Stan's windowed adaptor, which has been shown to be robust in practice (Carpenter et al., 2017).

7.2.1.1 Example code of building Stan's NUTS using AdvancedHMC.jl

The code snippet below illustrates how to use AHMC to construct a no-U-turn sampler, given the target log density function and its gradient, logdensity_f and grad_f respectively.

```julia
using AdvancedHMC
n_samples, n_adapts, target = 10_000, 2_000, 0.8 # set up sampling parameters
q0 = randn(D) # draw a random starting point
### Building up NUTS
metric = DiagEuclideanMetric(D) # diagonal Euclidean metric space
```

```
 6 h = Hamiltonian(metric, logdensity_f, grad_f) # Hamiltonian on the target distribution
 7 eps_init = find_good_eps(h, q0) # initial step size
 8 int = Leapfrog(eps_init) # leapfrog integrator
 9 traj = NUTS{MultinomialTS,GeneralisedNoUTurn}(int) # multi. sampling with gen. no U-turn
10 adaptor = StanHMCAdaptor( # Stan's windowed adaptor
11     Preconditioner(metric), NesterovDualAveraging(target, eps_init)
12 )
13 samples, stats = sample(h, traj, q0, n_samples, adaptor, n_adapts) # draw samples
```

Here LOGDENSITY_F and GRAD_F are functions of the target distributions's log density and its gradient, which are functions independent of AHMC and can be, e.g. derived from different PPLs or defined by normalising flows (Rezende and Mohamed, 2015; Dinh et al., 2016; Papamakarios et al., 2017). The gradient function can also be defined programmingly using any AD packages in Julia. We will show an example of such models in the next section.

7.2.1.2 AD and GPU support for ADVANCEDHMC.JL via CUDA.JL

In order to run HMC on CUDA, one only needs to change Line 3 of the demo code from Q0 = RANDN(D) to Q0 = CUARRAY(RANDN(D)), assuming LOGDENSITY_F and GRAD_F in Line 6 are GPU friendly, which is how CUDA.JL could be used with ADVANCEDHMC.JL to run HMC on GPUs. An example using GPU accelerated AHMC to draw samples from a normalising flow, named FLOW_MODEL, trained on MNIST (LeCun, 1998) is given below. The function LOGPDF(M, X) is used to compute the log density for model M on data batch X.

```
logdensity_f(x) = logpdf(flow_model, reshape(x, 784, 1))[1]
# Define gradient function via reverse AD
function grad_f(x)
    val, back = Tracker.forward(logdensity_f, x)
    grad = back(1)
    return (Tracker.data(val), Float32.(Tracker.data(grad[1][:,1])))
end
```

Here TRACKER is an automatic differentiation (AD) library which implements reverse-mode AD.

How does it work? All arrays in AHMC are abstractly typed, meaning that the concrete type is deduced at compile time from Q0. That is to say, if Q0 is defined on the GPU, i.e. it is a CUARRAY, all internal arrays in HMC will be too. CUDA.JL has a wide coverage of operations defined on GPUs, ranging from standard array operations to special statistics functions. It also supports compiling Julia code to CUDA code at run-time and its automatic conversion can be as fast as manually optimised CUDA codes, if not faster. That is to say, if a function is written in pure Julia, it probably supports GPUs acceleration without much housekeeping.[4]

7.2.2 Related work

A summary of the related work on HMC/NUTS implementations is given in table 7.1. We want to emphasis that there exists a Python API of AHMC implemented by an independent team available at `https://github.com/salilab/hmc`.

7.2.3 Evaluations

To compare the NUTS implementation between AHMC and STAN, we consider five models from MCMCBENCHMARKS.JL, a package for benchmarking MCMC libraries in Julia.[5] The five models are

1. A *Gaussian model* (Gaussian), a simple two parameter Gaussian distribution

$$\begin{aligned} \mu &\sim \mathcal{N}(0,1) \\ \sigma &\sim \mathcal{Truncated}(\mathcal{Cauchy}(0,5),0,\infty) \\ y_n &\sim \mathcal{N}(\mu,\sigma)\ (n=1,\ldots,N) \end{aligned}$$

2. The *signal detection model* (SDT), a model used in psychophysics and signal processing, which decomposes performance in terms of discriminability and bias (Green et al., 1966)

$$\begin{aligned} d &\sim \mathcal{N}(0,\frac{1}{\sqrt{2}}) \\ c &\sim \mathcal{N}(0,\frac{1}{\sqrt{2}}) \\ x &\sim \mathrm{SDT}(d,c) \end{aligned}$$

[4]With this being said, not all native Julia codes are performant on GPUs. For example, one would be careful of not using scalar operations as much because they are slow on GPUs.

[5]Available at `https://github.com/StatisticalRethinkingJulia/MCMCBenchmarks.jl`.

3. A Bayesian *linear regression* model (BLR) with a truncated Cauchy prior on the weights

$$\begin{aligned} B_d &\sim \mathcal{N}(0, 10) \\ \sigma &\sim \mathcal{Truncated}(\mathcal{Cauchy}(0, 5), 0, \infty) \\ y_n &\sim \mathcal{N}(\mu_n, \sigma) \end{aligned}$$

   where $\mu = B_0 + \boldsymbol{B}^T \boldsymbol{X}, d = 1, \ldots, D$ and $n = 1, \ldots, N$.

4. A *hierarchical Poisson regression* (HPR) model

$$\begin{aligned} a_0 &\sim \mathcal{N}(0, 10) \\ a_1 &\sim \mathcal{N}(0, 1) \\ b_\sigma &\sim \mathcal{Truncated}(\mathcal{Cauchy}(0, 1), 0, \infty) \\ b_d &\sim \mathcal{N}(0, b_\sigma) \\ y_n &\sim \mathcal{Poisson}(\log \lambda_n) \end{aligned}$$

   where $\log \lambda_n = a_0 + b_{z_n} + a_1 x_n, d = 1, \ldots, N_b$ and $n = 1, \ldots, N$.

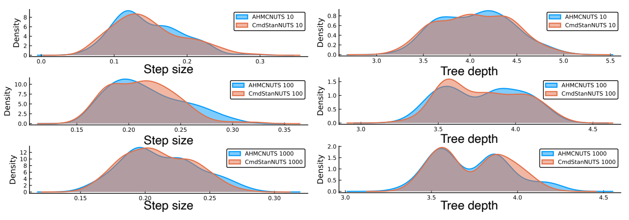
5. The *linear ballistic accumulator model* (LBA), a cognitive model of perception and simple decision making (Brown and Heathcote, 2008)

$$\begin{aligned} \tau &\sim \mathcal{Truncated}(\mathcal{N}(0.4, 0.1), 0, mn) \\ A &\sim \mathcal{Truncated}(\mathcal{N}(0.8, 0.4), 0, \infty) \\ k &\sim \mathcal{Truncated}(\mathcal{N}(0.2, 0.3), 0, \infty) \\ \nu_d &\sim \mathcal{Truncated}(\mathcal{N}(0, 3), 0, \infty) \\ x_n &\sim \text{LBA}(\boldsymbol{\nu}, \tau, A, k) \end{aligned}$$

   where $mn = \min_i x_{i,2}, d = 1, \ldots, N_c$ and $n = 1, \ldots, N$.

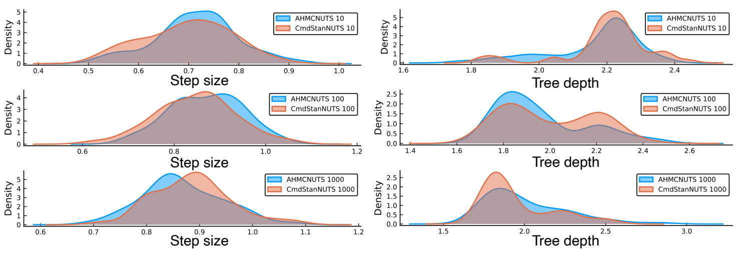
7.2.3.1 Statistical property of simulated trajectories

To compare the statistical property between STAN and AHMC, we run multiple chains of NUTS with target acceptance rate 0.8 for 2,000 steps with 1,000 for adaptation, where the warm-up samples are dropped. Each model is benchmarked with multiple data sizes N. We compare the simulated trajectories by the distributions of step size and tree depth and the average effective sample size (ESS) for all variables. The results for



	N	ESS			
		b_0	σ	b_1	b_2
STAN	10	413.939	266.476	381.219	423.441
AHMC	10	354.946	263.894	411.769	399.420
STAN	100	621.796	729.812	465.990	608.608
AHMC	100	473.005	734.189	606.996	621.543
STAN	1000	668.524	789.987	464.459	648.201
AHMC	1000	485.988	786.577	676.097	689.344

Figure 7.1: BLR (50 runs). For the density plots, blue is for AHMC and orange is for STAN and each row is for a different size N, corresponding to the table on the right.



	N	ESS	
		μ	σ
STAN	10	513.163	466.577
AHMC	10	503.535	447.722
STAN	100	786.531	782.231
AHMC	100	786.531	796.628
STAN	1000	864.010	876.660
AHMC	1000	832.255	844.452

Figure 7.2: Gaussian (50 runs); left to right: step size, tree depth, ESS

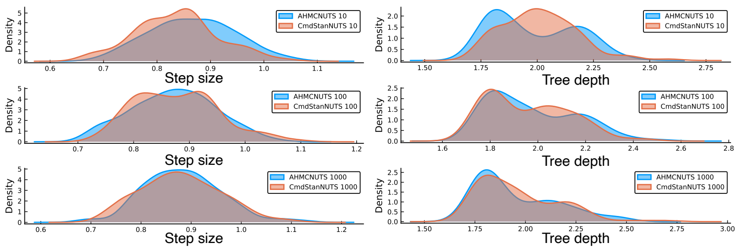
the five benchmark models are shown in figure 7.1-7.5. As an example for discussion, it can been seen from figure 7.3 that the distributions of step size and tree depth are similar and the average ESS for all variables are close, indicating AHMC's NUTS is statistically similar to the implementation in STAN. Conclusions from the results of the other four models remain similar.

7.2.3.2 Computational efficiency via running time

All the benchmark models used in this paper are implemented in TURING (Ge et al., 2018), a universal PPL in Julia that uses AHMC as its HMC backend. Below is the code snippet of running NUTS using TURING for the BLR model.

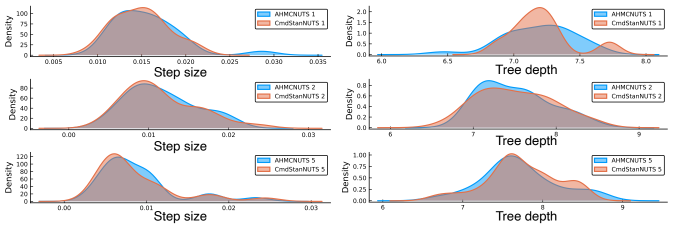
```julia
@model LR(x, y, Nd, Nc) = begin
    B ~ MvNormal(zeros(Nc), 10)
    B0 ~ Normal(0, 10)
    sigma ~ Truncated(Cauchy(0, 5), 0, Inf)
    mu = B0 .+ x * B
    y ~ MvNormal(mu, sigma)
end
x, y, Nd, Nc = ... # load data
chain = sample(LR(x, y, Nd, Nc), NUTS(2_000, 1_000, 0.8))
```

The average time used to run the five benchmark models for multiple times using



	N	ESS	
		d	c
Stan	10	710.762	703.327
AHMC	10	802.236	815.929
Stan	100	820.741	823.152
AHMC	100	814.308	846.357
Stan	1000	844.478	872.961
AHMC	1000	829.792	859.018

Figure 7.3: SDT (100 runs); left to right: step size, tree depth, ESS



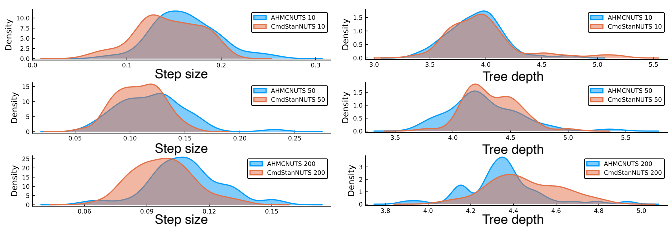
	N	ESS		
		a_0	a_1	b_σ
Stan	10	221.485	215.013	266.900
AHMC	10	216.491	214.459	258.638
Stan	20	208.286	207.041	241.080
AHMC	20	206.458	200.469	236.546
Stan	50	172.484	172.982	216.586
AHMC	50	200.755	201.548	247.384

Figure 7.4: HPR (25 runs); left to right: step size, tree depth, ESS (of some variables)

Stan and using AHMC via Turing are reported in table 7.2. We see that AHMC has comparable performance for all models except for HPR, which could be the result of the difference in the implementation of the probability mass function for the Poisson distribution. Also note that AHMC scales better for LBA models while Stan scales better for the rest of the models. This is likely due to the fact that the LBA model relies on a manually implemented distribution which is highly optimised by Julia's compiler.

7.2.4 Conclusion

We have introduced AdvancedHMC.jl, a new Julia package that provides robust, modular and efficient implementations of advanced HMC algorithms. We also highlight the modularity of the package and compare the implemented NUTS with Stan, both statistically and computationally. Overall, we hope that AHMC can serve as a robust baseline for future research on HMC algorithms and can enable the development of new PPLs which take advantage of the decoupling of inference algorithms from the actual modelling language.



	N	ESS			
		τ	A	ν_1	ν_2
STAN	10	226.463	282.656	305.614	276.557
AHMC	10	340.722	304.523	337.610	336.357
STAN	50	212.838	238.003	24.009	232.667
AHMC	50	248.249	238.979	248.331	255.421
STAN	200	244.926	264.967	268.793	270.36
AHMC	200	256.638	263.098	270.978	266.769

Figure 7.5: LBA (50 runs); left to right: step size, tree depth, ESS (of some variables)

Table 7.1: Comparison of different HMC/NUTS implementations. TFP.MCMC refers to TENSORFLOW.PROBABILITY's MCMC library. DYNAMICHMC is another high-quality HMC implementation in Julia. Windowed adaption is a method for joint adaption of leapfrog integration step-size and mass matrix. Windowed adaption was proposed by the Stan team (Carpenter et al., 2017), and has demonstrated remarkable robustness in a wide range of Bayesian inference problems. Partial support for GPU means the log density function can be accelerated by GPU, but the HMC sampler itself runs on CPU. Slice and Multinomial are two methods for sampling from dynamic Hamiltonian trajectories, e.g. those generated by the No-U-Turn algorithm (see e.g. Betancourt (2017) for details). Tempered leapfrog integrator improves convergence when the target distribution has multiple modes by performing tempering within Hamiltonian trajectories (Neal, 2011). Coupled multinomial is supported through COUPLEDHMC.JL.

	STAN	AHMC	PYRO	TFP.MCMC	PYMC3	DYNAMICHMC
Adaption						
Step-size	✓	✓	✓	✓	✓	✓
Mass matrix	✓	✓	✓	✓	✓	✓
Windowed adaption	✓	✓	✓	✗	✗	✓
Hamiltonian trajectory						
Classic No-U-Turn	✓	✓	✗	✓	✗	✗
No-U-Turn	✓	✓	✓	✓	✓	✓
Strict No-U-Turn	✓	✓	✗	✗	✓	✗
Fixed trajectory length	✓	✓	✓	✓	✓	✗
Trajectory sampler						
Slice	✓	✓	✓	✓	✗	✗
Multinomial	✓	✓	✓	✓	✓	✓
Coupled multinomial	✗	✓	✗	✗	✗	✗
Integrator						
Leapfrog	✓	✓	✓	✓	✓	✓
Jittered Leapfrog	✓	✓	✗	✗	✗	✗
Tempered Leapfrog	✗	✓	✗	✗	✗	✗
DIFFEQS.JL integrators	partial	✓	✗	✗	✗	✗
GPU support	partial	✓	✓	✓	partial	partial

0	Gaussian [2]		SDT [3]		LR [2]		HPR [1]		LBA [2]	
	N	seconds	*N*	seconds	*N*	seconds	*N*	seconds	*N*	seconds
STAN	10	0.8039	10	0.7759	10	0.8669	10	2.4870	10	1.9179
AHMC	10	0.3361	10	0.3285	10	1.1356	10	19.4587	10	2.6906
STAN	100	0.7561	100	0.7261	100	0.9824	20	3.5025	50	7.8471
AHMC	100	0.3303	100	0.3201	100	1.3202	20	28.2982	50	11.0270
STAN	1000	0.7614	1000	0.7089	1000	2.2600	50	5.8954	200	31.3762
AHMC	1000	0.5081	1000	0.3179	1000	3.8326	50	40.0322	200	33.6125

Table 7.2: Computational efficiency for five models using [1] 25 runs, [2] 50 runs or [3] 100 runs. For AHMC, forward-mode AD is used for computing gradient. AHMC can be used together with different AD backends/packages, e.g. FORWARDDIFF.JL's forward-mode AD, TRACKER.JL's reverse-mode AD and ZYGOTE.JL's source-to-source AD.

Chapter 8

Conclusion and Future Work

In this thesis, I present four pieces of my original work that improves the robustness and efficiency of algorithms for generative modelling as well as my work and contributions to free and open-source software for generative modelling.

In chapter 3, two novel couplings for multinomial HMC—a HMC variant that is widely used in modern, standard HMC libraries—are introduced. Theoretical analysis is provided on the validity of the proposed methods. Simulations are performed to demonstrate their advantages over existing methods in terms of meeting time and robustness to HMC parameters. The improved efficiency and robustness is an important step towards practical use of coupled HMC and unbiased MCMC as it is the first time that coupled HMC shows comparable efficiency as non-coupled HMC while providing unbiased estimates. I hope this work will help advance the research on searching more efficient coupled HMC methods, and a wider use of coupled HMC for probabilistic modelling in practice.

In chapter 4, I introduce RAVE, a novel inference method for BNP models that provides a probabilistic way to perform amortised variational inference without the need of explicitly truncating the maximum number of inferred posterior features as is common in previous work on variational inference for BNP. It is clearly demonstrated that the inference method outperforms both structured and mean-filed amortised variational inference methods that resort to explicit truncation of the posterior features. This work paves the way for using truly non-parametric Bayesian methods in specific research directions such as continual learning, scene understanding, etc.

In chapter 5, a new algorithm for training neural samplers called GRAM is introduced. It is demonstrated that while MMD-nets in their original formulation fail to generate high dimensional images in good quality, their performance can be greatly improved by training them under a low dimensional mapping. Unlike the alternative adversarial (MMD-GAN) for learning such a mapping, GRAM learns this mapping while avoiding the saddle-point optimization by being trained to match density ratios of the input and the projected pair of distributions. This leads to a sizeable improvement in stability and generative quality, that is at par with or better than adversarial generative methods. GRA M makes it easier to use neural samplers in complex systems because it does not require careful hyper-parameter tuning for successful training.

Chapter 6 introduces BSP, a Bayesian approach to learning symbolic physics which, to our knowledge, is the first to combine symbolic learning of physical force laws and statistical learning of unobserved attributes. The work enables data-efficient symbolic physics learning from partially-observed trajectory data and paves the way for using learnable IPEs in intuitive physics by providing a computational framework to study how humans' iterative reasoning-learning is mentally performed.

Lastly, chapter 7 summarises the development of the open-source ecosystem around TURING.JL, especially ADVANCEDHMC.JL (AHMC) an open-source library that implements CMHMC along with other popular HMC methods in a modular and efficient manner. Importantly, such ecosystem is designed for reusable components for developing feature-rich PPLs and interoperation between different Julia packages. The former has influenced other PPLs in Julia and the latter has made it easy to Bayesian inference together with techniques such as deep learning, differential equations, symbolic regression, etc.

8.1 Future work

8.1.1 Advanced coupled HMC and a vision of a new Bayesian inference workflow

Couplings for more advanced HMC variants One future work of chapter 3 is extending the coupling methods to more advanced HMC variants, e.g. the no-U-turn algorithm (Hoffman and Gelman, 2014), which is the default sampler in many probabilistic programming languages because of its ability of self-tuning hyper-parameters.

Learning the initial distribution π_0 The meeting time and overall efficiency of the coupled HMC sampler is dependent on the choice of the initial distribution π_0. An interesting research direction is to learn such a π_0 that is optimised for coupled MCMC estimators. One potential way to achieve so it to adapt the recent developed contrastive divergence for combining MCMC and VI (Ruiz and Titsias, 2019) that learns an optimised variational distribution as the initial distribution for a MCMC sampler. Applying the same idea to coupled MCMC can potentially make unbiased MCMC estimators more efficient and reliable to use.

A new Bayesian inference workflow As coupled multinomial HMC has shown comparable efficiency of unbiased MCMC as to normal MCMC estimation, it would be interesting to support it as a standard inference method in PPLs. Interestingly, this would also require a change in the common abstraction of how Bayesian inference is performed. For most of the existing PPLs, the inference is done through a `sample` function, which obtains approximate samples for the target model, and the approximate samples usually consumed manually by users to perform Monte Carlo estimation. However, for unbiased estimation, it would mean to augment this interface via a function called `estimate` and update `sample` accordingly:

- `sample` should return *typed samples* which are samples with additional information, if available, that could be consumed by `estimate` later to perform unbiased estimation. For unbiased MCMC, it would be coupled samples; for particle-based methods (like importance sampling), it would be weighted samples.
- `estimate` then takes typed samples and a function of interest h to perform the corresponding unbiased estimation for each type; for normal MCMC, this simply reduced to normal MC estimation, which can be biased. It could also be interesting to support `estimate` on models directly so `sample` is automatically performed.

8.1.2 Amortised feature complexity via RAVE

One limitation of the current way of using RAVE in VAEs is that the feature complexity is same across all data points. However, one could also learn an inference network that learns the dimension of representation for a given data point, essentially amortising the feature complexity. This can be useful for problem in which a data point is a scene that can contain as many objects as possible so the representation of the scene could also be unbounded.

Other directions for future work include extending RAVE to other non-parametric models.

8.1.3 GRAM for representation learning

Intuitively speaking, the projection network in GRAM has to learn an informative feature space in order to preserve the density ratios. Such feature space, during training or at convergence, may be useful in general as representations of the data. Thus, one interesting direction to explore is to facilitate the principle of density-ratio matching in the context of representation learning, potentially together with contrastive learning.

8.1.4 Improving neuro-symbolic generative modelling

The current setup of BSP relies on two main assumptions. First, we assume it has access to the grammar of Newtonian physics. However, how this knowledge is acquired, e.g. through evolution, is not addressed. Second, BSP works on symbolic inputs and assumes perceptual modules are given. This requires extra pre-processing steps when applying BSP to perceptual data, e.g. videos from PHYS101 and ULLMAN. The performance of BSP also depends on the quality of the pre-processing step. It will be interesting to address these two limitations by formulating a computational framework in which the grammar and the perceptual modules can be learned, either in separate phases or jointly with symbolic force learning.

Symbolic prior learning The learning of grammar is essentially the learning of the prior in a symbolic form. The prior should be learned in a way such that post-learning with the prior is data-efficient, as it is shown by BSP. This direction is closely related to active research in deep learning include pre-training and meta-learning.

Foundations of neuro-symbolic generative modelling Currently the learning of neuro-symbolic generative models relies on adapting existing generic learning methods. However, some methods could be ill-defined when a symbolic component is used in the generative process. For example, the support of the model distribution could have a mismatch with the data distribution if the symbolic part is wrong during learning.

Proper measure-theoretical research to better understand and characterise the learning of neuro-symbolic generative models should be conducted to consolidate the foundation of neuro-symbolic generative modelling.

8.1.5 Towards a probabilistic symbolic programming ecosystem

As mentioned in chapter 6 and chapter 7, BSP is implemented using TURING.JL (for the generative process and the E-step) together EXPROPTIMIZATION.JL (for the symbolic regression in the M-step). This idea could be extended with more inference and symbolic learning methods, together with enhanced modelling language support to allow users to easily define probabilistic programs in which some functions follow a particular grammar. Some efforts such as Dai et al. (2019); Manhaeve et al. (2021) have been made to propose similar ideas but not yet any practical framework has been developed to fulfil this mission. This paradigm could be useful in areas where domain-specific knowledge can be provided in a symbolic form such as physics and chemistry.

Bibliography

Abbasnejad, M. E., Dick, A., and van den Hengel, A. (2017). Infinite variational autoencoder for semi-supervised learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 781–790. IEEE.

Adams, R., Wallach, H., and Ghahramani, Z. (2010). Learning the structure of deep sparse graphical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 1–8.

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

Allen, K. R., Smith, K. A., and Tenenbaum, J. B. (2019). The tools challenge: Rapid trial-and-error learning in physical problem solving. *arXiv preprint arXiv:1907.09620*.

Amos, B., Dinh, L., Cabi, S., Rothörl, T., Colmenarejo, S. G., Muldal, A., Erez, T., Tassa, Y., de Freitas, N., and Denil, M. (2018). Learning awareness models. *arXiv preprint arXiv:1804.06318*.

Antoniak, C. E. (1974). Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The annals of statistics*, pages 1152–1174.

Arjovsky, M., Chintala, S., and Bottou, L. (2017a). Wasserstein GAN. *arXiv preprint arXiv:1701.07875*.

Arjovsky, M., Chintala, S., and Bottou, L. (2017b). Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

Asuncion, A. and Newman, D. (2007). UCI machine learning repository.

Bansal, S. (2010). Multiple polymorphic arguments in single dispatch object oriented languages. In *International Conference on Contemporary Computing*, pages 260–271. Springer.

Baradel, F., Neverova, N., Mille, J., Mori, G., and Wolf, C. (2020). Cophy: Counterfactual learning of physical dynamics. In *International Conference on Learning Representations*.

Bates, C., Battaglia, P. W., Yildirim, I., and Tenenbaum, J. B. (2015). Humans predict liquid dynamics using probabilistic simulation. In *CogSci*.

Battaglia, P. W., Hamrick, J. B., and Tenenbaum, J. B. (2013). Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D., and Kavukcuoglu, K. (2016). Interaction networks for learning about objects, relations and physics. *arXiv:1612.00222 [cs]*.

Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*.

Betancourt, M. (2018). A conceptual introduction to Hamiltonian Monte Carlo. *arXiv:1701.02434 [stat]*. arXiv: 1701.02434.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978.

Bińkowski, M., Sutherland, D. J., Arbel, M., and Gretton, A. (2018). Demystifying MMD GANs. *arXiv preprint arXiv:1801.01401*.

Bishop, C. M. (2013). Model-based machine learning. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1984):20120222.

Biswas, N., Jacob, P. E., and Vanetti, P. (2019). Estimating convergence of Markov chains with l-lag couplings. In *Advances in Neural Information Processing Systems*, volume 32.

Blei, D. M. and Jordan, M. I. (2004). Variational methods for the Dirichlet process. In *International Conference in Machine Learning (ICML)*.

Bogachev, V. I. (2007). *Measure theory*, volume 1. Springer Science & Business Media.

Bonawitz, E., Ullman, T. D., Bridgers, S., Gopnik, A., and Tenenbaum, J. B. (2019). Sticking to the evidence? a behavioral and computational case study of micro-theory change in the domain of magnetism. *Cognitive Science*, 43(8):e12765.

Bonneel, N., Van De Panne, M., Paris, S., and Heidrich, W. (2011). Displacement interpolation using Lagrangian mass transport. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pages 1–12.

Bou-Rabee, N., Eberle, A., and Zimmer, R. (2020). Coupling and convergence for Hamiltonian Monte Carlo. *Annals of applied probability: an official journal of the Institute of Mathematical Statistics*, 30(3):1209–1250.

Bramley, N. R., Gerstenberg, T., Tenenbaum, J. B., and Gureckis, T. M. (2018). Intuitive experimentation in the physical world. *Cognitive psychology*, 105:9–38.

Breen, P. G., Foley, C. N., Boekholt, T., and Zwart, S. P. (2019). Newton vs the machine: solving the chaotic three-body problem using deep neural networks. *arXiv:1910.07291 [astro-ph, physics:physics]*.

Brescia, F. (2012). *Fundamentals of Chemistry: A Modern Introduction (1966)*. Elsevier.

Brown, S. D. and Heathcote, A. (2008). The simplest complete model of choice response time: Linear ballistic accumulation. *Cognitive psychology*, 57(3):153–178.

Bryant, M. and Sudderth, E. B. (2012). Truly nonparametric online variational inference for hierarchical Dirichlet processes. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 2699–2707.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1).

Carter, L. L. and Cashwell, E. D. (1975). Particle-transport simulation with the Monte Carlo method. Technical report, Los Alamos Scientific Lab.

Cerny, B. M., Nelson, P. C., and Zhou, C. (2008). Using differential evolution for symbolic regression and numerical constant creation. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1195–1202, Atlanta, GA, USA. Association for Computing Machinery.

Chang, M. B., Ullman, T., Torralba, A., and Tenenbaum, J. B. (2016). A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*.

Chatzis, S. P. (2014). Indian buffet process deep generative models. *arXiv preprint arXiv:1402.3427*.

Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691.

Chen, Z. and Vempala, S. S. (2019). Optimal convergence rate of Hamiltonian Monte Carlo for strongly logconcave distributions. *arXiv:1905.02313 [cs, stat]*.

Cox, R. T. (1963). The algebra of probable inference. *American Journal of Physics*, 31(1):66–67.

Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. (2020). Discovering symbolic models from deep learning with inductive biases. *arXiv:2006.11287 [astro-ph, physics:physics, stat]*.

Cusumano-Towner, M. F., Saad, F. A., Lew, A. K., and Mansinghka, V. K. (2019). Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th acm sigplan conference on programming language design and implementation*, pages 221–236.

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300.

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

Dai, W.-Z., Xu, Q., Yu, Y., and Zhou, Z.-H. (2019). Bridging machine learning and logical reasoning by abductive learning.

Davidson, J. W., Savic, D. A., and Walters, G. A. (2001). Symbolic and numerical regression: Experiments and applications. In John, R. and Birkenhead, R., editors, *Developments in Soft Computing*, Advances in Soft Computing, pages 175–182, Heidelberg. Physica-Verlag HD.

De Finetti, B. (1937). La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pages 1–68.

Dempe, S. (2002). *Foundations of bilevel programming*. Springer Science & Business Media.

Devroye, L. (1990). Coupled samples in simulation. *Operations Research*, 38(1):115–126.

Dinh, L. and Dumoulin, V. (2016). Training neural Bayesian nets.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*.

Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

Doshi-Velez, F., Miller, K., Van Gael, J., and Teh, Y. W. (2009). Variational inference for the Indian buffet process. In *Artificial Intelligence and Statistics*, pages 137–144.

Du, Y. and Mordatch, I. (2019). Implicit generation and modeling with energy based models.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222.

Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., and Courville, A. (2016). Adversarially learned inference. *arXiv preprint arXiv:1606.00704*.

Dziugaite, G. K., Roy, D. M., and Ghahramani, Z. (2015). Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*.

Ehrhardt, S., Monszpart, A., Mitra, N. J., and Vedaldi, A. (2017). Learning a physical long-term predictor. *arXiv preprint arXiv:1703.00247*.

Feser, J. K., Chaudhuri, S., and Dillig, I. (2015). Synthesizing data structure transformations from input-output examples. *ACM SIGPLAN Notices*, 50(6):229–239.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv:1703.03400 [cs]*.

Fox, E. B., Hughes, M. C., Sudderth, E. B., Jordan, M. I., et al. (2014). Joint modeling of multiple time series via the beta process with application to motion capture segmentation. *The Annals of Applied Statistics*, 8(3):1281–1313.

Fragkiadaki, K., Agrawal, P., Levine, S., and Malik, J. (2015). Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*.

Gabler, P., Trapp, M., Ge, H., and Pernkopf, F. (2019). Graph tracking in dynamic probabilistic programs via source transformations.

Gal, Y. (2016). Uncertainty in deep learning.

Gao, B. and Pavel, L. (2018). On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv:1704.00805 [cs, math]*.

Ge, H., Xu, K., and Ghahramani, Z. (2018). Turing: A language for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics*.

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

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(6):721–741.

Georgoulas, A., Hillston, J., and Sanguinetti, G. (2017). Unbiased Bayesian inference for population markov jump processes via random truncations. *Statistics and computing*, 27(4):991–1002.

Gershman, S. and Blei, D. (2012). A tutorial on Bayesian nonparametric models. *Journal of Mathematical Psychology*, 56:1–12.

Gershman, S. and Goodman, N. (2014). Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36.

Gerstenberg, T., Goodman, N. D., Lagnado, D. A., and Tenenbaum, J. B. (2015). How, whether, why: Causal judgments as counterfactual contrasts. In *CogSci*.

Geweke, J. (2004). Getting it right: Joint distribution tests of posterior simulators. *Journal of the American Statistical Association*, 99(467):799–804.

Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459.

Girolami, M. and Calderhead, B. (2011). Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Glynn, P. W. and Rhee, C.-H. (2014). Exact estimation for Markov chain equilibrium expectations. *Journal of Applied Probability*, 51A:377–389.

Glynn, P. W. and Whitt, W. (1992). The asymptotic efficiency of simulation estimators. *Operations Research*, 40(3):505–520.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Goodman, J. and Weare, J. (2010). Ensemble samplers with affine invariance. *Communications in applied mathematics and computational science*, 5(1):65–80.

Goodman, N. D. and Stuhlmüller, A. (2014). The Design and Implementation of Probabilistic Programming Languages. `http://dippl.org`. Accessed: 2021-8-21.

Görür, D. and Rasmussen, C. E. (2010). Dirichlet process Gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25(4):653–664.

Green, D. M., Swets, J. A., et al. (1966). *Signal detection theory and psychophysics*, volume 1. Wiley New York.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773.

Griffiths, T. L. and Ghahramani, Z. (2011). The Indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12(Apr):1185–1224.

Grzeszczuk, N. and Animator, T. D. H. G. N. (1998). Fast neural network emulation and control of physics-based models. *Proc. ACM SIGGRAPH '98 (New York, 1998).–ACM Press*, pages 9–20.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings.

Hairer, E., Lubich, C., and Wanner, G. (2006). *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Number 31 in Springer series in computational mathematics. Springer, Berlin ; New York, 2nd ed edition.

Heng, J. and Jacob, P. E. (2019). Unbiased Hamiltonian Monte Carlo with couplings. *Biometrika*, 106(2):287–302.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 6626–6637.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

Ho, J., Lohn, E., and Abbeel, P. (2020). Compression with flows via local bits-back coding. *arXiv:1905.08500 [cs, math, stat]*.

Hoffman, M. and Blei, D. (2015). Stochastic structured variational inference. In *Artificial Intelligence and Statistics*, pages 361–369.

Hoffman, M. D. and Gelman, A. (2014). The no-U-turn sampler: Adaptively setting path Lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623. arXiv: 1111.4246.

Hu, Z., Qirong, H., Dubey, A., and Xing, E. (2015). Large-scale distributed dependent nonparametric trees. In *International Conference on Machine Learning*, pages 1651–1659.

Huang, J., Gretton, A., Borgwardt, K., Schölkopf, B., and Smola, A. (2006). Correcting sample selection bias by unlabeled data. *Advances in Neural Information Processing Systems*, 19:601–608.

Hughes, M., Kim, D. I., and Sudderth, E. (2015). Reliable and scalable variational inference for the hierarchical Dirichlet process. In *Conference on Artificial Intelligence and Statistics*, volume 38, pages 370–378.

Hughes, M. C. and Sudderth, E. (2013). Memoized online variational inference for Dirichlet process mixture models. In *Advances in Neural Information Processing Systems*, pages 1133–1141.

Hurwitz, C., Kudryashova, N., Onken, A., and Hennig, M. H. (2021). Building population models for large-scale neural recordings: Opportunities and pitfalls. *Current Opinion in Neurobiology*, 70:64–73.

Jackman, S. (2009). *Bayesian analysis for the social sciences*, volume 846. John Wiley & Sons.

Jacob, P. E. (2020). Couplings and monte carlo.

Jacob, P. E., Lindsten, F., and Schön, T. B. (2016). Coupling of particle filters. *CoRR*.

Jacob, P. E., Lindsten, F., and Schön, T. B. (2019a). Smoothing with couplings of conditional particle filters. *Journal of the American Statistical Association*, pages 1–20.

Jacob, P. E., O'Leary, J., and Atchadé, Y. F. (2019b). Unbiased Markov chain Monte Carlo with couplings. *arXiv:1708.03625 [stat]*.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Janner, M., Levine, S., Freeman, W. T., Tenenbaum, J. B., Finn, C., and Wu, J. (2019). Reasoning about physical interactions with object-oriented prediction and planning. *arXiv:1812.10972 [cs, stat]*.

Jaynes, E. T. (2003). *Probability theory: The logic of science*. Cambridge university press.

Johnson, V. E. (1996). Studying convergence of Markov chain Monte Carlo algorithms using coupled sample paths. *Journal of the American Statistical Association*, 91(433):154–166.

Johnson, V. E. (1998). A coupling-regeneration scheme for diagnosing convergence in Markov chain Monte Carlo algorithms. *Journal of the American Statistical Association*, 93(441):238–248.

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

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *arXiv:1312.6114 [cs, stat]*.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. (2018). Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*.

Kommenda, M., Kronberger, G., Winkler, S., Affenzeller, M., and Wagner, S. (2013). Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, GECCO '13 Companion, pages 1121–1128, Amsterdam, The Netherlands. Association for Computing Machinery.

Korattikara, A., Chen, Y., and Welling, M. (2014). Austerity in MCMC Land: Cutting the Metropolis-Hastings Budget. In *International Conference on Machine Learning*, pages 181–189. PMLR.

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112.

Kramer, A., Calderhead, B., and Radde, N. (2014). Hamiltonian Monte Carlo methods for efficient parameter estimation in steady state dynamical systems. *BMC bioinformatics*, 15(1):253.

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., and Blei, D. M. (2017). Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, 40.

Larranaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I., Lozano, J. A., Armananzas, R., Santafé, G., Pérez, A., et al. (2006). Machine learning in bioinformatics. *Briefings in bioinformatics*, 7(1):86–112.

LeCun, Y. (1998). The MNIST database of handwritten digits. *online:* `http://yann.lecun.com/exdb/mnist/`.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, C.-L., Chang, W.-C., Cheng, Y., Yang, Y., and Póczos, B. (2017). MMD GAN: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2200–2210.

Li, C.-L., Chang, W.-C., Mroueh, Y., Yang, Y., and Póczos, B. (2019a). Implicit kernel learning. *arXiv preprint arXiv:1902.10214*.

Li, Y., Bradshaw, J., and Sharma, Y. (2019b). Are generative classifiers more robust to adversarial attacks? *arXiv:1802.06552 [cs, stat]*.

Li, Y., Swersky, K., and Zemel, R. (2015). Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727.

Lin, Z., Jain, A., Wang, C., Fanti, G., and Sekar, V. (2020). Using GANs for sharing networked time series data: Challenges, initial promise, and open questions. *arXiv:1909.13403 [cs, stat]*.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2017). Are gans created equal? a large-scale study. *arXiv preprint arXiv:1711.10337*.

Lux, I. and Koblinger, L. (1991). *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*. CRC press.

Lyne, A.-M., Girolami, M., Atchadé, Y., Strathmann, H., Simpson, D., et al. (2015). On Russian roulette estimates for Bayesian inference with doubly-intractable likelihoods. *Statistical science*, 30(4):443–467.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Mangoubi, O. and Smith, A. (2017). Rapid mixing of Hamiltonian Monte Carlo on strongly log-concave distributions. *arXiv:1708.07114 [math, stat]*.

Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., and De Raedt, L. (2021). Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504.

Mescheder, L., Nowozin, S., and Geiger, A. (2017a). Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. In *International Conference on Machine Learning*, pages 2391–2400. PMLR.

Mescheder, L., Nowozin, S., and Geiger, A. (2017b). The numerics of GANs. In *Advances in Neural Information Processing Systems*, pages 1825–1835.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

Meyn, S. P. and Tweedie, R. L. (2012). *Markov chains and stochastic stability*. Springer Science & Business Media.

Miao, Y., Grefenstette, E., and Blunsom, P. (2017). Discovering discrete latent topics with neural variational inference. In *International Conference in Machine Learning (ICML)*.

Minka, T., Winn, J., Guiver, J., Zaykov, Y., Fabian, D., and Bronskill, J. (2018). /Infer.NET 0.3. Microsoft Research Cambridge. http://dotnet.github.io/infer.

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 1791–1799. PMLR.

Müller, P. and Quintana, F. A. (2004). Nonparametric Bayesian data analysis. *Statistical science*, pages 95–110.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Murray, L. M. (2013). Bayesian state-space modelling on high-performance hardware using libbi. *arXiv preprint arXiv:1306.3277*.

Møller, J., Syversveen, A. R., and Waagepetersen, R. P. (1998). Log Gaussian Cox processes. *Scandinavian Journal of Statistics*, 25(3):451–482.

Nalisnick, E. and Smyth, P. (2017). Stick-breaking variational autoencoders. In *International Conference on Learning Representations (ICLR)*.

Neal, R. M. (2011). MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2.

Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Neal, R. M. (2017). Circularly-coupled Markov chain sampling. *arXiv:1711.04399 [stat]*.

Neiswanger, W., Wang, C., and Xing, E. (2013). Asymptotically exact, embarrassingly parallel mcmc. *arXiv preprint arXiv:1311.4780*.

Nowozin, S., Cseke, B., and Tomioka, R. (2016). f-GAN: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pages 271–279.

Nuesken, N. and Pavliotis, G. A. (2018). Constructing sampling schemes via coupling: Markov semigroups and optimal transport. *arXiv:1806.11026 [math]*.

Orbanz, P. and Teh, Y. W. (2010). Bayesian nonparametric models. In *Encyclopedia of Machine Learning*. Springer.

Osera, P.-M. and Zdancewic, S. (2015). Type-and-example-directed program synthesis. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '15, pages 619–630, New York, NY, USA. Association for Computing Machinery.

O'Leary, J., Wang, G., and Jacob, P. E. (2020). Maximal couplings of the metropolis–hastings algorithm. *arXiv preprint arXiv:2010.08573*, 1(2):4.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2019). Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*.

Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347.

Peng, S., Wang, G., and Xie, D. (2016). Social influence analysis in social networking big data: Opportunities and challenges. *IEEE network*, 31(1):11–17.

Pharr, M., Jakob, W., and Humphreys, G. (2016). *Physically based rendering: From theory to implementation*. Morgan Kaufmann.

Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195.

Plummer, M., Best, N., Cowles, K., and Vines, K. (2006). CODA: convergence diagnosis and output analysis for MCMC. *R News*, 6(1):7–11. Number: 1.

Qiu, Y., Zhang, L., and Wang, X. (2019). Unbiased contrastive divergence algorithm for training energy-based latent variable models. In *International Conference on Learning Representations*.

Quade, M., Abel, M., Shafi, K., Niven, R. K., and Noack, B. R. (2016). Prediction of dynamical systems by symbolic regression. *Physical Review E*, 94(1):012214.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Ranganath, R., Gerrish, S., and Blei, D. (2014). Black box variational inference. In *Artificial Intelligence and Statistics*.

Rezende, D. and Mohamed, S. (2015). Variational Inference with Normalizing Flows. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France. PMLR.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*.

Rhee, C.-h. and Glynn, P. W. (2012). A new approach to unbiased estimation for sde's. *arXiv:1207.2452 [math, q-fin]*.

Robbins, H. and Monro, S. (1985). A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer.

Rosenthal, J. S. (1997). Faithful couplings of Markov chains: now equals forever. *Advances in Applied Mathematics*, 18(3):372–381.

Roth, K., Lucchi, A., Nowozin, S., and Hofmann, T. (2017). Stabilizing training of generative adversarial networks through regularization. In *Advances in neural information processing systems*, pages 2018–2028.

Rowland, M., Choromanski, K. M., Chalus, F., Pacchiano, A., Sarlos, T., Turner, R. E., and Weller, A. (2018). Geometrically coupled Monte Carlo sampling. In *Advances in Neural Information Processing Systems 31*, pages 195–206.

Ruiz, F. and Titsias, M. (2019). A contrastive divergence for combining variational inference and mcmc. In *International Conference on Machine Learning*, pages 5537–5545. PMLR.

Ruiz, F. R., AUEB, M. T. R., and Blei, D. (2016). The generalized reparameterization gradient. In *Advances in neural information processing systems*, pages 460–468.

Sanborn, A. N., Mansinghka, V. K., and Griffiths, T. L. (2013). Reconciling intuitive physics and newtonian mechanics for colliding objects. *Psychological review*, 120(2):411.

Sanchez-Gonzalez, A., Bapst, V., Cranmer, K., and Battaglia, P. (2019). Hamiltonian graph networks with ode integrators. *arXiv:1909.12790 [physics]*.

Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85.

Seo, S., Meng, C., and Liu, Y. (2019). Physics-aware difference graph networks for sparsely-observed dynamics. In *International Conference on Learning Representations*.

Singh, R., Ling, J., and Doshi-Velez, F. (2017). Structured variational autoencoders for the beta-Bernoulli process.

Smith, K., Mei, L., Yao, S., Wu, J., Spelke, E., Tenenbaum, J., and Ullman, T. (2019). Modeling expectation violation in intuitive physics with coarse probabilistic object representations. In *Advances in Neural Information Processing Systems*, pages 8983–8993.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR.

Spelke, E. S. (2000). Core knowledge. *American psychologist*, 55(11):1233.

Spelke, E. S. and Kinzler, K. D. (2007). Core knowledge. *Developmental science*, 10(1):89–96.

Sriperumbudur, B. K., Fukumizu, K., Gretton, A., Schölkopf, B., and Lanckriet, G. R. (2009). On integral probability metrics,\phi-divergences and binary classification. *arXiv preprint arXiv:0901.2698*.

Srivastava, A., Valkoz, L., Russell, C., Gutmann, M. U., and Sutton, C. (2017). VEEGAN: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems*, pages 3310–3320.

Srivastava, A., Xu, K., Gutmann, M. U., and Sutton, C. (2019). Generative ratio matching networks. In *International Conference on Learning Representations*.

Sugiyama, M., Suzuki, T., and Kanamori, T. (2012). *Density ratio estimation in machine learning*. Cambridge University Press.

Sugiyama, M., Yamada, M., von Bünau, P., Suzuki, T., Kanamori, T., and Kawanabe, M. (2011). Direct density-ratio estimation with dimensionality reduction via least-squares hetero-distributional subspace search. *Neural Networks*, 24 2:183–98.

Teh, Y. W., Grür, D., and Ghahramani, Z. (2007). Stick-breaking construction for the Indian buffet process. In *Artificial Intelligence and Statistics*, pages 556–563.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2005). Sharing clusters among related groups: Hierarchical Dirichlet processes. In *Advances in neural information processing systems*, pages 1385–1392.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581.

Teh, Y. W., Welling, M., Osindero, S., and Hinton, G. E. (2003). Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4(Dec):1235–1260.

Thorisson, H. (2000). *Coupling, Stationarity, and Regeneration*. Probability and Its Applications. Springer New York.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622.

Tran, D., Ranganath, R., and Blei, D. M. (2017). Hierarchical implicit models and likelihood-free variational inference. *arXiv preprint arXiv:1702.08896*.

Udrescu, S.-M. and Tegmark, M. (2020). Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631.

Ullman, T. D., Stuhlmüller, A., Goodman, N. D., and Tenenbaum, J. B. (2018). Learning physical parameters from dynamic scenes. *Cognitive Psychology*, 104:57–82.

Valera, I., Pradier, M. F., Lomeli, M., and Ghahramani, Z. (2018). General latent feature models for heterogeneous datasets. *arXiv:1706.03779 [stat]*.

van den Oord, A., Vinyals, O., et al. (2017). Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6297–6306.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J., and Levine, S. (2020). Entity abstraction in visual model-based reinforcement learning. pages 1439–1456.

Villani, C. (2003). *Topics in optimal transportation*. American Mathematical Soc.

Walker, S. G., Damien, P., Laud, P. W., and Smith, A. F. M. (1999). Bayesian nonparametric inference for random distributions and related functions. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 61(3):485–527.

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. (2017). Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547.

Webb, S., Golinski, A., Zinkov, R., Siddharth, N., Rainforth, T., Teh, Y. W., and Wood, F. (2017). Faithful inversion of generative models for effective amortized inference. *arXiv preprint arXiv:1712.00287*.

Wei, C. and Murray, I. (2016). Markov chain truncation for doubly-intractable inference. *arXiv preprint arXiv:1610.05672*.

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer.

Winn, J., Bishop, C. M., Diethe, T., Guiver, J., and Zaykov, Y. (2020). Model-based machine learning.

Wood, F., Meent, J. W., and Mansinghka, V. (2014). A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032.

Wu, A., Nowozin, S., Meeds, E., Turner, R. E., Hernandez-Lobato, J. M., and Gaunt, A. L. (2018). Deterministic variational inference for robust bayesian neural networks. *arXiv preprint arXiv:1810.03958*.

Wu, J., Lim, J., Zhang, H., Tenenbaum, J., and Freeman, W. (2016). Physics 101: Learning physical object properties from unlabeled videos. In *Procedings of the British Machine Vision Conference 2016*, pages 39.1–39.12, York, UK. British Machine Vision Association.

Wu, J., Yildirim, I., Lim, J. J., Freeman, B., and Tenenbaum, J. (2015). Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Advances in neural information processing systems*, pages 127–135.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747 [cs]*.

Xie, J., Lu, Y., Zhu, S.-C., and Wu, Y. (2016). A theory of generative convnet. In *International Conference on Machine Learning*, pages 2635–2644.

Xu, K., Fjelde, T. E., Sutton, C., and Ge, H. (2021a). Couplings for multinomial Hamiltonian Monte Carlo. In *International Conference on Artificial Intelligence and Statistics*.

Xu, K., Ge, H., Tebbutt, W., Tarek, M., Trapp, M., and Ghahramani, Z. (2020). AdvancedHMC.jl: A robust, modular and effcient implementation of advanced HMC algorithms. In *Symposium on Advances in Approximate Bayesian Inference*.

Xu, K., Srivastava, A., Gutfreund, D., Sosa, F. A., Tomer, U., Tenenbaumm, J. B., and Sutton, C. (2021b). A Bayesian-symbolic approach to reasoning and learning in intuitive physics. In *Advances in Neural Information Processing Systems*.

Xu, K., Srivastava, A., and Sutton, C. (2019). Variational Russian roulette for deep Bayesian nonparametrics. In *International Conference on Machine Learning*.

Yeung, S., Kannan, A., Dauphin, Y., and Fei-Fei, L. (2017). Tackling over-pruning in variational autoencoders. *arXiv preprint arXiv:1706.03643*.

Zheng, D., Luo, V., Wu, J., and Tenenbaum, J. B. (2018). Unsupervised learning of latent physical properties using perception-prediction networks. *arXiv:1807.09244 [cs, stat]*.

Index