Scalable and Interpretable Learning with Probabilistic Models for Knowledge Discovery

Sulin Liu

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Electrical and Computer Engineering
Adviser: Ryan P. Adams and Peter J. Ramadge

September 2023

Abstract

Novel machine learning methods are at the heart of a transformation underway in science and engineering. Probabilistic models have served as the foundation learning models for knowledge discovery. As surrogate models, they enable efficient black-box optimization or active learning of the system behavior under limited budget to evaluate/query the complex system. Another important use case is to use probabilistic models as generative models to generate *de novo* designs with desired properties or samples from the equilibrium distribution of physical systems. However, to fully unleash the potential of probabilistic models for knowledge discovery, it is imperative to develop models that are scalable to growing data size and complexity while remaining interpretable to domain experts.

In this dissertation, I start with the development of a novel approach that addresses the sparse solution identification problem in Bayesian optimization with probabilistic surrogate models. The discovery of sparse solutions not only enhances the interpretability of the solutions to humans for understanding the system behavior but also facilitates easier streamlined deployment and maintenance with a reduced number of parameters.

Next, I present a novel approach utilizing deep learning to enhance the scalability of Gaussian process inference. Gaussian processes are extensively used as probabilistic surrogate models in knowledge discovery, but their practical usage is hindered by the high cost associated with the identification of kernel hyperparameters in GP regression, which involves costly marginal likelihoods. I show how to side-step the need for expensive marginal likelihoods by employing "amortized" inference over hyperparameters. This is achieved through training a single neural network, which consumes a set of data and produces an estimate of the kernel function, useful across different tasks.

Finally, I introduce *marginalization models*, a new family of generative models for high-dimensional discrete data which is ubiquitous in science discovery. Marginalization models offer scalable and flexible generative modeling with tractable likelihoods through explicit modeling of all induced marginal distributions with neural networks. Direct modeling of the marginals enables efficient marginal inference and scalable training of any-order generative models for sampling from a given (unnormalized) probability function, which overcomes major limitations of previous methods with exact likelihoods.

Acknowledgements

Throughout the past six years of my PhD, I am very grateful to have met and built relationships with many amazing people, which has made this journey such a wonderful and fulfilling one.

I am particularly fortunate to be advised by Ryan Adams and Peter Ramadge. It is a very rewarding experience to work closely with them. I feel grateful to have them as my mentors and it is not an overstatement to say that they have greatly shaped my way of thinking and taste of research.

Ryan has largely expanded my view of the machine learning field that I have never imagined before, both in terms of technicality and broader impact. But perhaps most importantly, he has taught me how to think, deeply and out of the box. I am constantly "wow"ed by his ability to come up with a totally different perspective to look at a problem, which often leads to a much more interesting and elegant approach. Beyond that, I have learned much from him about how to communicate ideas and collaborate, especially with people from different backgrounds. He has also been a great role model for me in terms of always stepping out of comfort zone and exploring new knowledge.

Peter has been incredibly supportive and encouraging throughout my PhD. He has provided me the freedom to explore new ideas and directions and has always been there to help me when I needed. Many times when I was stuck on a problem, he was able to provide great intellectual insights and ask the fundamental questions that helped me think deeply about the problem and move forward. He has also trained me to communicate and present ideas in a clear and concise manner, although there is still a long way to go to reach his level. I have had the great pleasure to work as a teaching assistant in his class multiple times. I am deeply inspired by his passion and dedication to teaching, from preparing the material to delivering the lectures. I strive to live up to his standards and become a better teacher.

Besides my advisors, I am incredibly grateful to have worked with Ben Letham, Qing Feng, David Eriksson, and Eytan Bakshy during my internship at Facebook (now Meta) research. Ben and Eytan have been the best internship mentors I could ask for and I have learned a lot from defining practically interesting research problems to making an impact through research-to-product conversion. It is also such a pleasant collaborating experience with Qing and David, without whom it would not be possible to turn the internship project into the final paper. I have also gained a lot of knowledge and understanding of Bayesian optimization in practice through working closely with the amazing people on the team.

I am incredibly grateful to have met and collaborated with many other amazing people at Prince-

ton: Jaime Fernández Fisac, Adji Bousso Dieng, Athindran Ramesh Kumar, Xingyuan Sun, Hossein Valavi, Daniel Lwanzo Paluku, and Liyi Zhang. The collaborations not only led to research outcomes but also rewarded me with great friendships and enriched my understanding of various research topics. I also wish to express gratitude to Jonathan Pillow, Tom Griffiths and Adji for serving on my thesis committee and providing valuable feedback. They have definitely helped me think more deeply about my research and broaden my view of the field.

I would also like to thank Sinno Jialin Pan, Loong Fah Cheong and Teng Joon Lim, who have mentored me and built my initial foundation for machine learning research. Without their mentorship and support, I could not have reached where I am.

During the past year, I have also had the great pleasure to work with and talk to amazing folks with material science backgrounds under the NSF ID4 institute, including but not limited to: Eric Toberer, Elif Ertekin, Steven Lopez, Drew Novick, Angela Pak, Rees Chang, and Noah Shinn. I am grateful for the opportunity to learn physics and material science from them and discuss how machine learning can accelerate science discovery, which has led to promising on-going research collaborations. And more importantly, the discussions and interactions with them have kindled my profound interest in developing better machine-learning frameworks in the material science/chemical physics setting and I am excited to continue working in this area in the future.

I am grateful to the lab mates I have met over the years in LIPS: Diana, Deniz, Xingyuan, Tianju, Jad, Ari, Alex Beatson, Jordan, Geoffrey, Sam, Josh, Yaniv, Gregory, David, Daniel, Wenda, Olga, Nick, Yucen, Alex Guerra, Mehran, Jose, Eder, Jenny, Jin, Kathryn and more. I will miss the retreats, game sessions, self-organized work sessions and conversations over research and (more often) many other random fun things. Within Ramadge's group, I have met some of the most supportive peers: Jonathan, Hossein, Hejia, Jimmy, Ting-Han, Athindran and Arnab. I really enjoyed the time we spent together in the office and the chats we had. I would also like to extend my gratitude to the Princeton ECE and CSML staff who have provided much help and make graduate school feel like home, especially Andrea, Lisa, Colleen, Roelie, Jean, Kate, and Chris.

I wish to thank the wonderful friends in Princeton who have made this PhD such a memorable and enjoyable one: Fei, Tianran, Bonan, Yuchen, Ping, Pengning, Hongyang, Hongxu, Qi, Xiaoliang, Sammy, and many others. I will remember the many late-night poker/cards/drinks and hotpot/holiday parties. And I am fortunate to witness some of your important life milestones. Many thanks go to the friends outside of Princeton for much support and memorable reunion trips: Yaodong, Yunxiang, Mengchen, Xudong, Shuoguang, Yifan, Can, and many others.

My parents, Hongxing and Yan, have supported me unconditionally throughout my life. This

PhD is no different, despite being physically separated due to COVID. I am grateful for the opportunities they have provided me and the freedom they have given me to pursue my career path.

Finally, the best thing that happened during my PhD is to meet Amy at Princeton. She has been supportive through the ups and downs of my PhD. She's celebrated my successes, comforted me during the hard times, helped me think through problems, and encouraged me to pursue my interests. This thesis would not have been possible without her.

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Probabilistic models, as a principled machine learning approach for modeling data distributions, have recently started to play an instrumental role in fostering scientific exploration and discovery.

Surrogate models serve as valuable tools in diverse domains spanning science, engineering, robotics, and many other fields, wherein they model intricate system behavior. Exploiting the uncertainty quantification offered by probabilistic surrogate models, automatic algorithms can be designed to efficiently accomplish the objectives of the given use case via actively interacting with the system.

One major use case is optimization, exemplified when determining the optimal material composition for battery cathodes via experimental testing. In such cases, Bayesian optimization (Shahriari et al., 2015b) is employed, using a probabilistic model to understand and iteratively fine-tune the relationship between composition and performance informed by experimental outcomes. Simultaneously, the subsequent experimental condition is chosen strategically, based on the surrogate model's uncertainty quantification, balancing the exploration of novel compositions with the exploitation of well-performing known compositions, thus expediting the discovery of the optimum composition.

Active learning presents another primary use case, such as in training a surrogate model to accurately simulate molecular dynamics (Vandermause et al., 2020). The process begins with an initial probabilistic model based on limited data, which is then systematically enhanced through actively querying the system for additional labeled data. The selection of the most informative samples for labeling is guided by the surrogate model's inherent uncertainty estimation, culminating in an accurate model with minimal labeling effort.

In addition to surrogate models, probabilistic generative models have witnessed remarkable advancements in modeling complex data distributions across various domains, including natural

language modeling (Brown et al., 2020), image generation (Song and Ermon, 2019; Ho et al., 2020), audio synthesis (Huang et al., 2018), and scientific discovery applications (Wang et al., 2022; Schneuing et al., 2022). When it comes to training generative models for scientific discovery, there are two primary settings. The first setting is maximum likelihood training, where the goal is to train a generative model to maximize the likelihood of the training data. This setting is commonly employed in tasks like image generation, natural language modeling, and drug design, where the objective is to generate data that closely resembles the training data distribution. The second setting is distribution matching, aiming to align the generative distribution with a target density. This setting is less explored for images and language but is frequently employed in applications such as sampling lattice models and estimating equilibrium properties of molecules or materials, where it is necessary to generate samples from the thermodynamic equilibrium distribution of the physical system.

In this dissertation, I present novel approaches to address the interpretability and scalability challenges of probabilistic models in the context of knowledge discovery. Before delving into the details of the proposed approaches, I provide a brief overview of the existing literature for both surrogate and generative models.

The rest of this chapter is organized as follows: Section 1.1.1 first gives a brief overview of the Gaussian process, which is a popular probabilistic surrogate model used in scientific discovery. Then I review the basic methodological aspects of Bayesian optimization in Section 1.1.2. Section 1.2 provides a brief overview of the existing literature on generative models with a particular focus on the applications in scientific discovery. Finally, I conclude this chapter with an outline of the overall dissertation in Section 1.3.

1.1 Probabilistic Surrogate Models

1.1.1 Gaussian Processes

The Gaussian process defines a nonparametric distribution over functions of the forms $f(\cdot) : \mathcal{X} \to \mathbb{R}$. $\mathcal{X}$ is the domain of the function f, which is usually a subset of $\mathbb{R}^D$ for some D. The output space is assumed to be single output with the range being $\mathbb{R}$, although an extension to m-dimensional multiple output $\mathbb{R}^m$ is straightforward.

The Gaussian process model specifies a joint Gaussian distribution over the outputs of the function at any finite set of points in the domain. More precisely, for any finite set of points in $\mathcal{X}$, $\mathbf{X} := \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the corresponding function values $\mathbf{f} := (f(\mathbf{x}_1), \cdots, f(\mathbf{x}_N))^\mathsf{T}$ follow a multi-

variate Gaussian distribution: $\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K_{XX}})$, where $\boldsymbol{\mu}$ is an N-dimensional mean vector and $\mathbf{K_{XX}}$ is an $N \times N$ positive-definite covariance matrix. The parameters of the mean and covariance are determined by a mean function $\mu(\cdot) : \mathcal{X} \to \mathbb{R}$ and a positive-definite covariance function $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:

$$[\boldsymbol{\mu}]_n = \mu(\mathbf{x}_n), \tag{1.1}$$

$$[\mathbf{K_{XX}}]_{n,n'} = k(\mathbf{x}_n, \mathbf{x}_{n'}) \tag{1.2}$$

Hence a Gaussian process is usually denoted by $f(\cdot) \sim \mathcal{GP}(\mu(\cdot), k(\cdot, \cdot))$. The mean function $\mu(\cdot)$ is usually set to zero, and the covariance function $k(\cdot, \cdot)$ is often parameterized by the choice of the function and its associated hyperparameters θ. In this case, we write $k_\theta(\cdot, \cdot)$ to indicate the dependence of the covariance function on the hyperparameters.

For a training dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, each y_n is commonly assumed to be generated by adding an i.i.d. zero-mean Gaussian noise to $f(\mathbf{x}_n)$, i.e., $y_n = f(\mathbf{x}_n) + \epsilon_n$, where $\epsilon_n \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Denote $\mathbf{y} := [y_1, \cdots, y_N]^\top \in \mathbb{R}^{N \times 1}$. For new data input $\widetilde{\mathbf{X}} := \{\widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_{N'}\}$ of size N', the Gaussianity of the prior and likelihoods make it possible to compute the predictive distribution in closed form:

$$\widetilde{\mathbf{f}} | \widetilde{\mathbf{X}}, \mathcal{D} \sim \mathcal{N}\left(\widetilde{\boldsymbol{\mu}}, \mathbf{K}_{\widetilde{\mathbf{f}}}\right),$$

$$\widetilde{\boldsymbol{\mu}} = \mathbf{K}_{\widetilde{\mathbf{X}}\mathbf{X}} \left(\mathbf{K_{XX}} + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} \mathbf{y}, \quad \mathbf{K}_{\widetilde{\mathbf{f}}} = \mathbf{K}_{\widetilde{\mathbf{X}}\widetilde{\mathbf{X}}} - \mathbf{K}_{\widetilde{\mathbf{X}}\mathbf{X}} \left(\mathbf{K_{XX}} + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} \mathbf{K}_{\mathbf{X}\widetilde{\mathbf{X}}},$$

where $\mathbf{K}_{\mathbf{X}\widetilde{\mathbf{X}}} \in \mathbb{R}^{N \times N'}$ with $[\mathbf{K}_{\mathbf{X}\widetilde{\mathbf{X}}}]_{n,n'} = k(\mathbf{x}_n, \widetilde{\mathbf{x}}_{n'})$.

Choice of kernel function

The choice of kernel function is crucial to Gaussian process generalization, as different kernel functions impose various model assumptions, e.g., smoothness, periodicity, etc. (See Chapter 4 of Rasmussen and Williams (2006) for an extensive discussion.) If the problem has a known structure, one can sometimes choose a kernel to capture it. Otherwise, kernel learning must be performed by defining an expressive space of kernel functions and selecting the best one through optimization (Wilson and Adams, 2013; Wilson et al., 2016; Sun et al., 2018) or search (Duvenaud et al., 2013; Lloyd et al., 2014).

Here we give a few examples of popular kernel functions. The *Radial Basis Function* (RBF) kernel, also known as the "squared exponential" kernel, is given by

$$k_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2l^2}\right),$$

where σ_f^2 is the signal variance and l is the lengthscale.

The *Matérn* kernel is given by

$$k_{\text{Matérn}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{\|\mathbf{x} - \mathbf{x}'\|}{l} \right)^{\nu} K_\nu \left(\sqrt{2\nu} \frac{\|\mathbf{x} - \mathbf{x}'\|}{l} \right),$$

where K_ν is the modified Bessel function of the second kind and ν is a hyperparameter that controls the smoothness of the function. When ν is set to $1/2$, the Matérn kernel is identical to the absolute exponential kernel, i.e.

$$k_{\text{Matérn}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{l} \right) \quad \nu = 1/2.$$

$\nu = 3/2$ and $\nu = 5/2$ correspond to the Matérn 3/2 and Matérn 5/2 kernels,

$$k_{\text{Matérn}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \left(1 + \frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{l} \right) \exp\left(-\frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{l} \right) \quad \nu = 3/2,$$

$$k_{\text{Matérn}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \left(1 + \frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|}{l} + \frac{5\|\mathbf{x} - \mathbf{x}'\|^2}{3l} \right) \exp\left(-\frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|}{l} \right) \quad \nu = 5/2,$$

which are popular choices for modeling functions that are not infinitely differentiable (assumed in RBF kernel) but at least once or twice differentiable.

The *Exp-Sine-Squared* kernel, also known as "periodic" kernel, is given by

$$k_{\text{Exp-Sine-Squared}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{2\sin^2(\pi\|\mathbf{x} - \mathbf{x}'\|/p)}{l^2} \right),$$

where p is the periodicity and l is the lengthscale.

To illustrate how different kernels model different function properties, we show examples of functions drawn from GPs with the kernels discussed above in Figure 1.1.

To model functions with multiple desired structures, it is possible to combine different kernels to form a composite kernel. The standard ways to build composite kernels are by adding or multiplying two or more kernels. For example, multiplying a linear kernel and a periodic kernel can be used to model a periodic function with increasing/decreasing amplitude. Adding is often used to combine kernels for individual dimensions, e.g., a sum of RBF kernels with different lengthscales can be used to model functions with different smoothness in different dimensions. Chapter 2 of Duvenaud (2014) gives a comprehensive guide on how to express structure with composite kernels.

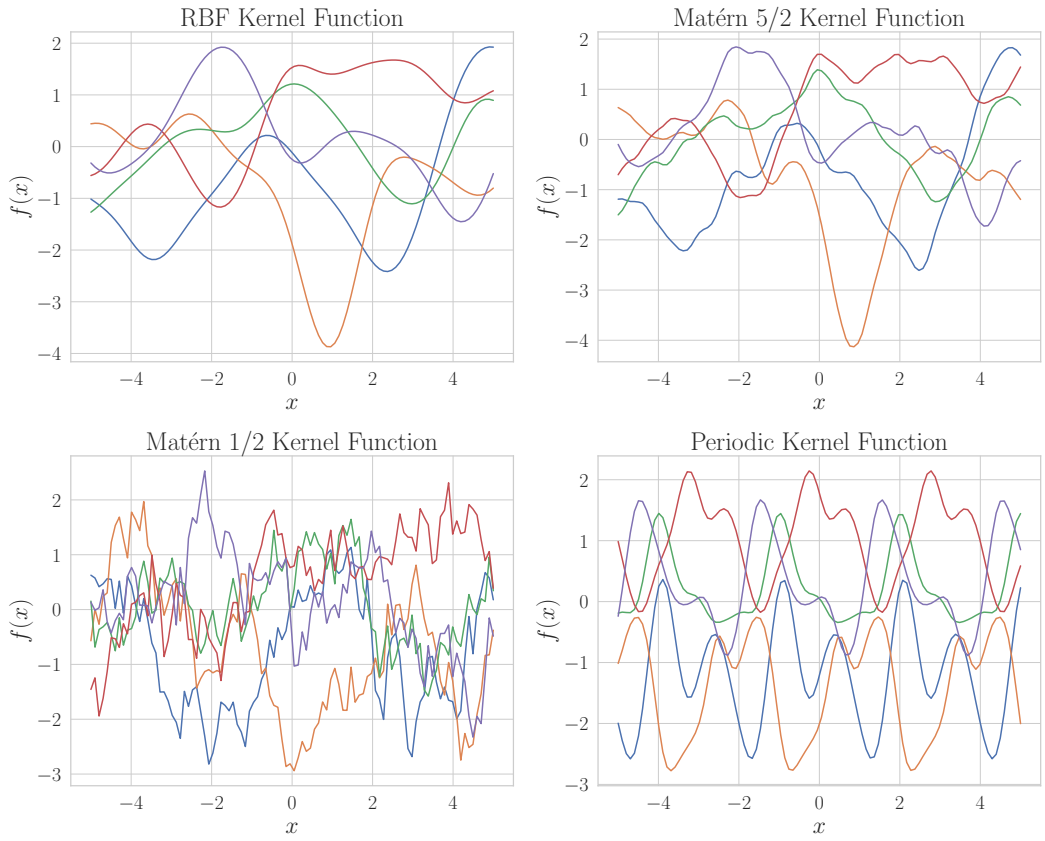


Figure 1.1: Examples of functions drawn from GPs with different kernels. **Top left**: Radial Basis Function (RBF) kernel, lengthscale $l = 1$. **Top right**: Matérn 5/2 kernel, lengthscale $l = 1$. **Bottom left**: Matérn 1/2 kernel, lengthscale $l = 1$. **Bottom right**: Exp-Sine-Squared kernel, lengthscale$l = 1$, periodicity $p = 3$.

Hyperparameter inference and the associated scalability challenge

Beyond the particular choice of kernel, it is also common for the covariance function to have so-called *hyperparameters* θ that govern its specific structure, and the parameterized kernel function is written as $k_\theta(\cdot,\cdot)$. Although a fully-Bayesian treatment is possible (Neal, 1999; Murray and Adams, 2010; Filippone and Girolami, 2014; Murray and Graham, 2016), the most common approach to determining hyperparameters is to use empirical Bayes and maximize the log marginal likelihood (evidence) with respect to the hyperparameter θ, i.e., perform type II maximum likelihood (Berger, 2013; MacKay, 1992). The log MLL for observed data $\{\mathbf{X}, \mathbf{y}\}$ is given by:

$$\log p(\mathbf{y}|\mathbf{X},\theta) = -\frac{1}{2}\mathbf{y}^\top \left(\mathbf{K}_{\mathbf{XX}}(\theta) + \sigma_\epsilon^2\mathbf{I}\right)^{-1}\mathbf{y} - \frac{1}{2}\log\left|\mathbf{K}_{\mathbf{XX}}(\theta) + \sigma_\epsilon^2\mathbf{I}\right| - \frac{N}{2}\log 2\pi\,, \tag{1.3}$$

where we write $\mathbf{K}_{\mathbf{XX}}(\theta)$ to indicate the dependence of the Gram matrix on the hyperparameters.

To solve the above optimization problem, quasi-Newton methods such as L-BFGS (Liu and Nocedal, 1989) or nonlinear conjugate gradient (Hestenes et al., 1952; Fletcher and Reeves, 1964) are usually used. These iterative optimization methods involve taking the gradient of the objective several times for each optimization step. As the gradient of Equation (1.3) scales as $\mathcal{O}(N^3)$, this optimization becomes prohibitively expensive on large-scale problems, dominating the computational cost of using GP. Moreover, the non-concavity of the objective in Equation (1.3) makes it difficult to ensure convergence to a good maximum.

To address the scaling issue, a low-rank approximation to the kernel matrix is often used either by subsampling the data or via virtual "inducing" points (Smola and Bartlett, 2001; Williams and Seeger, 2001; Csató and Opper, 2002; Quiñonero-Candela and Rasmussen, 2005; Seeger et al., 2003; Snelson and Ghahramani, 2006; Titsias, 2009; Hensman et al., 2013; Cheng and Boots, 2017; Shi et al., 2020). In general, these methods require inversion of a smaller matrix and reduce the computational complexity to $\mathcal{O}(NM^2)$ (M is the number of subsampled data or "inducing" points), at the cost of a larger and often more challenging optimization problem alongside the potential loss of important information from the dataset. For the special case of the exponentiated quadratic kernel, Burt et al. (2019) showed that only $\mathcal{O}(N\log^{2D}(N))$ computational complexity is needed to achieve an arbitrarily good approximation with high probability for input with either compact support or Gaussian distribution of D dimensions.

1.1.2 Bayesian optimization

I give a brief overview of Bayesian optimization (BO) in this section. For a more comprehensive treatment, Shahriari et al. (2015a) provide a thorough review of BO.

The goal of Bayesian optimization is to maximize a black-box function $f : \mathbb{R}^D \to \mathbb{R}$ over a compact set $\mathcal{B} \subset \mathbb{R}^D$. Often for simplicity, the domain is scaled and taken as $[0, 1]^D$. We assume that f is continuous and bounded on this domain. At each iteration of optimization, f is modeled with a probabilistic surrogate model given the function evaluations observed so far, producing the normally distributed posterior $f(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x}))$ for all possible $\mathbf{x}$. The location of the next function evaluation is selected by maximizing an acquisition function $\alpha(\mathbf{x}) := \mathbb{E}_f[u(\mathbf{x})]$ to balance the trade-off of exploration and exploitation, where u is a utility function that defines the acquisition function. This iterative procedure is carried out until a stopping criterion is met, e.g., the optimal solution is successfully identified or the maximum number of iterations is reached.

Gaussian process is the most popular surrogate model used in BO. Bayesian neural networks only start to be explored recently as an alternative surrogate model (Li et al., 2023), but their high computational cost limits the adoption in practice. There is an equivalence between the Gaussian process and Bayesian neural networks discovered by Neal (1994). The rise of the popularity of the Gaussian process in the machine learning community is largely due to this connection and the more affordable computation used by GP (Williams and Rasmussen, 1995) when compared to Bayesian neural networks.

Typical acquisition functions include expected improvement (EI, Jones et al., 1998) and upper confidence bound (UCB, Srinivas et al., 2010). EI is given by

$$\alpha_{\mathrm{EI}}(\mathbf{x}) = \mathbb{E}_f \left[(f(\mathbf{x}) - f(\mathbf{x}^*))_+ \right], \tag{1.4}$$

where $\mathbf{x}^*$ is the best point observed so far, and the acquisition function has a well-known analytic form when f is a GP. UCB is similarly computed directly from the marginal posterior,

$$\alpha_{\mathrm{UCB}}(\mathbf{x}) = \mu(\mathbf{x}) + \sqrt{\beta}\sigma(\mathbf{x}), \tag{1.5}$$

where β is a hyperparameter that controls the exploration-exploitation trade-off. More recently, information-theoretic acquisition functions have been developed (Hernández-Lobato et al., 2014; Wang and Jegelka, 2017).

1.2 Probabilistic Generative Models

The generative model is a powerful tool for modeling complex data distributions. In this section, I first give a brief overview of the existing literature on generative models. Then I discuss the two prevalent generative modeling settings, maximum likelihood estimation, and distribution matching. Finally, I introduce autoregressive models and how they can be trained for any-order modeling in maximum likelihood estimation.

1.2.1 Overview of generative models

Generative models aim to learn a distribution $p_\theta(\mathbf{x})$—usually θ is parameterized by neural networks—that can generate samples $\mathbf{x}$ that resemble the data distribution $p_{\text{data}}(\mathbf{x})$. There are different things that the learned distribution can be useful for, such as generating new samples, evaluating the likelihood of observed data, and performing inference on marginal or conditional distributions.

Broadly speaking, generative models can be categorized into two classes: *explicit density models* and *implicit density models*. In explicit density models, the probability density function $p_\theta(\mathbf{x})$ is explicitly defined by a parametric function, e.g., a neural network. Depending on what is explicitly available of the distribution, we can evaluate the likelihood of observed data, or even perform inference on marginal or conditional distributions. Successful examples of explicit density models include normalizing flows (Rippel and Adams, 2013; Tabak and Turner, 2013; Dinh et al., 2014; 2016), variational autoencoders (VAEs) (Kingma and Welling, 2013; Rezende and Mohamed, 2015; Kingma et al., 2016), and autoregressive models (ARMs) (Bengio and Bengio, 2000; Larochelle and Murray, 2011).

In implicit density models, the distribution $p_\theta(\mathbf{x})$ is implicitly defined by a sampling procedure, i.e. we are not able to evaluate the likelihood of data but only able to generate samples from the distribution. Generative adversarial networks (GANs) (Goodfellow et al., 2014; Radford et al., 2015; Arjovsky et al., 2017; Karras et al., 2019), and diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song and Ermon, 2019) are the most commonly used examples of implicit density models. Recently, the probability flow ODE formulation of the diffusion model (Song et al., 2020) makes it a special case of continuous normalizing flows (neural ODE), thus enabling exact log-likelihood evaluation. VAEs are also sometimes considered to be implicit density models when closed-form log-likelihood evaluation is not available with non-invertible decoders, such as in the one originally developed in Kingma and Welling (2013).

Energy-based models (EBMs) (Hinton, 2002; LeCun et al., 2006; Tieleman, 2008) are another

class of generative models that lie in between implicit and explicit density models. Instead of specifying a probability density, they explicitly model the negative log probability, i.e. the energy function. This allows flexibility in modeling the energy function with powerful neural networks and has found applications in image generation (Ngiam et al., 2011; Du and Mordatch, 2019) and language processing (Mikolov et al., 2013). But as a result, likelihood evaluation and exact sampling are intractable with the unnormalized probability.

In this dissertation, we focus on explicit density generative models for the discrete domain, due to their interpretability and wide applicability for scientific discovery applications. We first define two prevalent generative modeling settings. Then we give a brief overview of autoregressive models and how they can be trained for any-order modeling in maximum likelihood estimation.

1.2.2 Generative modeling settings

Maximum likelihood (MLE)

Given a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ drawn from a data distribution p_{data}, we aim to learn the distribution $p_\theta(\mathbf{x})$ that maximizes the probability of the data under our model. Mathematically, we aim to learn the parameters $\theta^\star$ that maximize the log-likelihood:

$$\theta^\star = \arg\max_\theta \ \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} [\log p_\theta(\mathbf{x})] \approx \arg\max_\theta \ \frac{1}{N}\sum_{i=1}^N \log p_\theta(\mathbf{x}^{(i)}) \tag{1.6}$$

which is also equivalent to minimizing the Kullback-Leibler divergence under the empirical distribution, i.e., minimizing $D_{\text{KL}}(p_{\text{data}}(\mathbf{x})||p_\theta(\mathbf{x}))$. This is the setting when we cannot evaluate $p_{\text{data}}(\mathbf{x})$ but can draw data samples from the empirical distribution, which is most commonly used in generation of images (e.g., diffusion models (Ho et al., 2020; Song and Ermon, 2019)) and language (e.g., pretrained transformers (Devlin et al., 2018; Yang et al., 2019b; Radford et al., 2019; Brown et al., 2020)).

Distribution matching (DM)

In this setting, we do not have the tool to draw samples from the distribution of interest. Instead, we have access to the unnormalized (log) probability mass (or density) function f, usually in the form of reward function or energy function, that are defined by us or by physical systems to specify how likely a sample is. For example, we can define the target PMF (or PDF) to be $f(\mathbf{x}) = \exp(r(x)/\tau)$ where $r(x)$ is the reward (or the negative energy) function and $\tau > 0$ is a temperature parameter. This expresses the intuitive idea that we would like the model to assign higher probability to data with

larger reward (lower energy). For example, in conversation systems such as ChatGPT (Ouyang et al., 2022; OpenAI, 2023), $r(x)$ can express how well a response fits the user's preference. In molecular design applications, scientists can specify the reward according to how close a particular sample's measured or calculated properties are to some functional desiderata.

Mathematically, we aim to learn the parameters θ such that $p_\theta(\mathbf{x}) \approx f(\mathbf{x})/Z$, where Z is the normalization constant of f. Two common training criteria are to minimize the KL divergence (Noé et al., 2019; Wu et al., 2019; Damewood et al., 2022):

$$\min_\theta \; D_{\mathrm{KL}}\left(p_\theta\left(\mathbf{x}\right) \middle\| f(\mathbf{x})/Z\right) = \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\left[\log p_\theta\left(\mathbf{x}\right) - \log f(\mathbf{x})/Z\right] \tag{1.7}$$

or the squared distance between the two log probabilities over a distribution of interest $q(\mathbf{x})$ (Bengio et al., 2021a; Zhang et al., 2022):

$$\min_\theta \; \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[\log p_\theta\left(\mathbf{x}\right) - \log f(\mathbf{x})/Z\right]^2 . \tag{1.8}$$

When KL divergence is used as a measure of distance, it is also sometimes referred to as *energy-based KL-training* (Köhler et al., 2020). This formulation is particularly useful for modeling physical systems such as the Ising model, molecules, proteins, and materials in thermodynamic equilibrium (Noé et al., 2019; Wu et al., 2019; Damewood et al., 2022; Köhler et al., 2023). A trained generative model can be used for evaluating properties by efficient sampling from the equilibrium distribution, whereas traditional methods such as molecular dynamics or MCMC struggle to sample efficiently due to the high energy barriers between different states. It is also useful for controlled generation, where we can specify the reward function to be a function of the desired properties, such as in instructing language models (Ouyang et al., 2022), or generating molecules with desired properties (Bengio et al., 2021a).

1.2.3 Autoregressive models

Autoregressive models (ARMs) model the distribution by factorizing the complex high-dimensional distribution $p(\mathbf{x})$ into univariate conditionals using the chain rule:

$$\log p(\mathbf{x}) = \textstyle\sum_{d=1}^{D} \log p\left(x_d \mid \mathbf{x}_{<d}\right), \tag{1.9}$$

where $\mathbf{x}_{<d} = \{x_1, \ldots, x_{d-1}\}$. Recently there has been great success in applying autoregressive models to discrete data, such as natural language (Yang et al., 2019b; Brown et al., 2020), proteins (Shin et al., 2021; Lin et al., 2023; Madani et al., 2023), and molecules (Segler et al., 2018; Flam-Shepherd et al., 2022). Due to their sequential nature, evaluation of (joint/marginal) likelihood requires up to D neural network evaluations, which is costly for long sequences.

Any-order ARMs (AO-ARMs)

Under the maximum likelihood setting, Uria et al. (2014) propose to learn the conditionals of ARMs for arbitrary orderings $\sigma \in S_D$, where S_D denotes the set of all permutations of $\{1, \ldots, D\}$. The model ϕ can be trained by maximizing a lower-bound objective (Uria et al., 2014; Hoogeboom et al., 2021a) that takes an expectation under a uniform distribution on orderings.

$$\begin{aligned}
\log p_\phi(\mathbf{x}) = \log \mathbb{E}_\sigma \, p_\phi(\mathbf{x} \mid \sigma) &\overset{\text{Jensen's inequality}}{\geq} \mathbb{E}_\sigma \sum\nolimits_{d=1}^{D} \log p_\phi \left(x_{\sigma(d)} \mid \mathbf{x}_{\sigma(<d)} \right) \\
&= \mathbb{E}_{\sigma \sim \mathcal{U}(S_D)} D \mathbb{E}_{d \sim \mathcal{U}(1,\ldots,D)} \log p_\phi \left(x_{\sigma(d)} \mid \mathbf{x}_{\sigma(<d)} \right) \\
&= D \, \mathbb{E}_d \, \mathbb{E}_\sigma \, \frac{1}{D - d + 1} \sum\nolimits_{j \in \sigma(\geq d)} \log p_\phi \left(x_j \mid \mathbf{x}_{\sigma(<d)} \right). \qquad (1.10)
\end{aligned}$$

$\mathcal{U}(S)$ denotes the uniform distribution over a finite set S. $\sigma \in S_D$ is a random ordering of the variables and S_D defines the set of all permutations of $1, 2, \cdots, D$. $\sigma(d)$ denotes the d-th element in the ordering and $\sigma(< d) = \{\sigma(1), \ldots, \sigma(d-1)\}$.

This objective allows scalable training of AO-ARMs, leveraging efficient parallel evaluation of multiple one-step conditionals in one forward pass with architectures such as the U-Net (Ronneberger et al., 2015) and Transformers (Vaswani et al., 2017). However, training of AO-ARMs remains a challenge under the distribution matching setting.

1.3 Overview

The rest of the dissertation is organized as follows. In Chapter 2, I will introduce the challenge of interpretability in Bayesian optimization when using the Gaussian process as the surrogate model and show how to address it from the angle of sparsity and multi-objective optimization. In Chapter 3, I will present a novel deep-learning-enabled approach to address the scalability challenge of using Gaussian processes as surrogate models. In Chapter 4, I will introduce a new class of generative models to address the scalability challenge of autoregressive models. Finally, I will conclude the dissertation in Chapter 5 and discuss future directions.

1.3.1 Relations to published work

Chapter 2 is based on Liu et al. (2023a):

Sulin Liu*, Qing Feng*, David Eriksson*, Benjamin Letham, and Eytan Bakshy. Sparse Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, 2023. (* indicates equal contribution).

Chapter 3 is based on Liu et al. (2020):

Sulin Liu, Xingyuan Sun, Peter J. Ramadge, and Ryan P. Adams. Task-agnostic amortized inference of Gaussian process hyperparameters. *Advances in Neural Information Processing Systems*, 2020.

Chapter 4 is based on Liu et al. (2023b):

Sulin Liu, Peter J. Ramadge, and Ryan P. Adams. Generative marginalization models. *Under Submission to Advances in Neural Information Processing Systems*, 2023.

Beyond the work presented, the following publication is also related to the dissertation on using probabilistic surrogate models for controlling safety-critical systems (Kumar et al., 2021):

Athindran Ramesh Kumar*, Sulin Liu*, Jaime F. Fisac, Ryan P. Adams, and Peter J. Ramadge. ProBF: Learning probabilistic safety certificates with barrier functions. *arXiv:2112.12210*, 2021. (* indicates equal contribution, random order).

Chapter 2

Interpretable System Design via Sparse Bayesian Optimization

2.1 Introduction

Bayesian optimization (BO) is a technique for efficient global optimization that is used to optimize the design parameters across a wide range of complex systems, including robotics (Lizotte et al., 2007; Calandra et al., 2015), machine learning pipelines (Hutter et al., 2011; Snoek et al., 2012; Turner et al., 2021), internet systems (Letham et al., 2019; Feng et al., 2020), chemistry (Gómez-Bombarelli et al., 2018; Burger et al., 2020; Shields et al., 2021) and energy (Attia et al., 2020; Liu et al., 2022). In many applications, including those just mentioned, it is preferable for the optimized parameters to be sparse. In this paper, we define *sparsity* in Bayesian optimization to be the property where the majority of optimized parameters are close to the target parameters that one wishes to regularize towards. For example, the target parameters may be a zero-vector, where setting parameters to zero encourages the removal of redundant system configurations. Alternatively, the target parameters may be the default system parameters (status quo), where sparsity favors the fewest modifications for consistency and robustness. One reason to prefer sparsity is that it increases *interpretability*, a consideration that has recently attracted a great deal of attention in machine learning (Doshi-Velez and Kim, 2017; Rudin et al., 2022). Interpretability is necessary for humans to be able to understand and evaluate the outputs of complex systems—the types of systems to which BO is often applied. In policy/process optimization, sparsity of the policy/control provides a natural way for human decision-makers to gain insight into the behavior of the system, and identify potential issues (Ustun

and Rudin, 2016; Hu et al., 2019).

Besides interpretability, sparsity can also be beneficial by producing systems that are easier to deploy and maintain, reducing the "technical debt" of complex automated systems (Sculley et al., 2015). As an example, recommender systems are essential to many internet companies, including e-commerce platforms, streaming services, and social media sites (Bobadilla et al., 2013). A typical recommendation process involves two stages, the retrieval and ranking stages (Covington et al., 2016). The parameters in the retrieval stage determine the amount of content to be fetched from various sets of candidate pools (*sources*) representing different user interest taxonomies (Wilhelm et al., 2018). Setting parameters to zero means deactivating these sources. Sparse optimization can find solutions in which low-quality sources are entirely turned off, thus simplifying the system and enabling faster development. Similarly in chemistry, a sparse solution requires fewer reagents and steps to synthesize a compound, which reduces experimentation overhead and accelerates the discovery of new compounds.

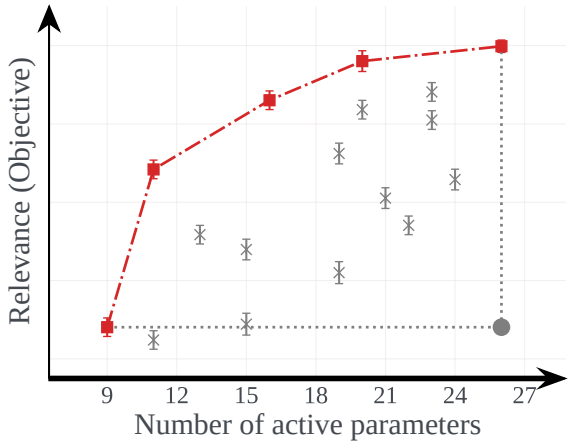


Figure 2.1: Objective and sparsity trade-offs for a real-world Internet experiment using SEBO. Points indicate recommender system configurations, where the x-axis corresponds to the number of active recommendation sources used, i.e. non-sparse parameters. Grey points indicate sub-optimal designs, while red points represent designs along the Pareto frontier found by SEBO. Decision-makers balance both system simplicity and performance when deciding which configuration to use.

Sparsity in machine learning is often achieved via regularization, such as L_1 regularization used by the lasso (Tibshirani, 1996), the group norm penalty used by the group lasso (Yuan and Lin, 2006), and L_0 regularization which directly targets setting elements to zero (Zhang, 2008). The purpose of regularization in machine learning is typically to limit overfitting and thus improve test accuracy by reducing generalization error (Evgeniou et al., 2002). In our setting, sparsity is a separate goal; interpretable sparse configurations will generally not improve the optimization objective, and in fact,

may come at some cost to other metrics. This can be seen in the sparsity-objective Pareto frontier shown in Figure 2.1 from a real-world recommender system sourcing experiment conducted at a large Internet firm. The Pareto frontier comprises all of the configurations that produce optimal trade-offs between sparsity and the optimization objective. In many real-world systems, decision-makers are willing to trade some amount of objective in order to achieve a higher level of sparsity, because of the interpretability and simplicity benefits that come with sparsity. Thus, unlike a typical BO problem, the "optimal" point per the decision maker will not necessarily be the one with the best objective, but could be some other point on the sparsity-objective Pareto frontier that has more sparsity.

A central aspect of this work is to efficiently learn these trade-offs and offer practitioners a way to balance sparsity and other metrics. Sparsity in BO is an important topic that has not yet been addressed in the literature. Past work has used regularization in acquisition function optimization or modeling, but not for the purpose of sparsity in design parameters (see Section 2.2 for a review). Our work provides a thorough and broad treatment of sparsity in BO that fills in this gap. The main contributions of this work are:

1. We study different approaches for incorporating sparse regularization into BO, and provide negative theoretical results showing that previously studied forms of regularization can fail to optimize for certain levels of sparsity, regardless of the regularization coefficient.
2. We draw connections between multi-objective BO and acquisition function regularization, and show how multi-objective BO can be used for automatic selection of the regularization coefficient. We refer to this as the SEBO ("Sparsity Exploring Bayesian Optimization") method.
3. We develop a novel relaxation strategy for optimizing directly for L_0 sparsity, and show that it significantly outperforms the typical L_1 penalty in our context.
4. We show that combining acquisition function regularization with sparse Gaussian process priors enables sparse optimization in high-dimensional spaces.
5. We provide the first results on achieving sparsity via BO, in a range of synthetic functions and on three real-world tasks (in systems configuration and AutoML), showing that SEBO is the best approach for sparse BO. We show the breadth of our method by using it to achieve different forms of sparsity such as feature-level and group sparsity.
6. We provide a new high-dimensional benchmark problem designed to emulate trade-offs found in real-world recommender systems, and show how such systems benefit from increased sparsity.

Section 2.2 describes the necessary background and related work. Section 2.3 describes two natural

approaches for incorporating sparse regularization into acquisition function optimization, both of which can fail to optimize for some levels of sparsity. Section 2.4 discusses a relationship between sparse BO and multi-objective BO, and describes how we can use methods from multi-objective BO to simultaneously optimize for all levels of sparsity. We describe how we optimize with L_0 regularization in Section 2.5. We demonstrate the usefulness of our methods by applying them to a set of synthetic and real-world benchmarks in Section 2.7. Finally, we discuss the results in Section 2.8.

2.2 Background and Related Work

2.2.1 Regularization in Bayesian optimization

Regularization has been applied to acquisition function optimization, though not for the purpose of sparsity. Shahriari et al. (2016) used regularization for unbounded BO, in which there are no bounds on the search space. They applied a form of L_2 regularization to the EI target value that penalized sampling points far from the initial center of the search space. González et al. (2016) used regularization for batch BO, where the penalty discouraged points from being chosen close to points that had already been selected for the batch. The penalty term was multiplied with the original acquisition function value.

2.2.2 Bayesian optimization with sparse models

Eriksson and Jankowiak (2021) introduced the sparse axis-aligned subspaces (SAAS) function prior in which a structured sparse prior is induced over the inverse-squared kernel lengthscales $\{\rho_i\}_{i=1}^d$ to enable BO in high dimensions. The SAAS prior has the form $\tau \sim \mathcal{HC}(\alpha), \rho_i \sim \mathcal{HC}(\tau)$ where $\mathcal{HC}$ is the half-Cauchy distribution which concentrates at zero. The goal of the SAAS prior is to turn off unimportant parameters by shrinking ρ_i to zero, which avoids overfitting in high-dimensional spaces, thus enabling sample-efficient high-dimensional BO. The global shrinkage parameter τ controls the overall sparsity: with more data, τ can be pushed to larger values, adapting the level of sparsity to the data as needed.

While sparsity in the GP model is different from the sparsity we seek here, we will show that combining the SAAS model with acquisition regularization is highly effective for sparse high-dimensional BO. By enforcing regularization in the acquisition function, the parameters identified as unimportant will be set to their baseline values, generating simpler and more interpretable policies. Other work has studied feature sparsity in GP regression but without considering sparsity

in optimization (Oh et al., 2019; Park et al., 2021).

2.2.3 Multi-objective Bayesian optimization

Multi-objective BO is used when there are several (often competing) objectives $f_1, \ldots, f_m$ and we wish to recover the Pareto frontier of non-dominated configurations. A classic method is ParEGO, which applies the standard single-objective EI acquisition function to a random scalarization of the objectives (Knowles, 2006). Many types of scalarizations have been developed for transforming multi-objective optimization (MOO) problems into single-objective problems (Ehrgott, 2005). Recent work on multi-objective BO has focused on developing acquisition functions that explicitly target increasing the hypervolume of the known Pareto frontier. Acquisition functions in this class, such as Expected Hypervolume Improvement (EHVI), are considered state-of-the-art for multi-objective BO (Yang et al., 2019a; Daulton et al., 2020; 2021).

2.3 Acquisition Function Regularization

2.3.1 External regularization

We use a regularization term $\xi(\mathbf{x})$ to model *sparsity*, which may be an L_0 quasinorm to target feature-level sparsity, $\xi(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^s\|_0$, or can be adjusted for different forms of sparsity such as group sparsity. Here $\mathbf{x}^s$ represents the target point that the decision maker wishes to drive the solution towards, e.g., a zero-vector or the current default parameters (status quo). For our analysis of regularization, we will assume that $\mathbf{x}^s$ is the unique global minimum of $\xi(\mathbf{x})$.

A straightforward approach for adding regularization is to simply add a regularization penalty directly to the acquisition function. This parallels regularized regression techniques like ridge regression and the lasso. Given a penalty term $\xi(\mathbf{x})$, we then maximize

$$\alpha_{\text{ER}}(\mathbf{x}; \lambda) = \alpha(\mathbf{x}) - \lambda\xi(\mathbf{x}) \tag{2.1}$$

to select the next point for evaluation. We refer to this approach as *external regularization* (ER). EI with external regularization is:

$$\alpha_{\text{EI-ER}}(\mathbf{x}; \lambda) = \mathbb{E}_f\left[(f(\mathbf{x}) - f(\mathbf{x}^*))_+\right] - \lambda\xi(\mathbf{x}). \tag{2.2}$$

The regularization coefficient λ must be set, just as with classic regularized regression. This

formulation separates the explore/exploit value of a point, in α, from its sparsity value, in ξ. This can perform poorly, because there is necessarily interaction between these two notions of value. We provide a negative result showing that external regularization cannot capture certain levels of sparsity.

Theorem 2.1. *Suppose $\alpha(\mathbf{x}) = 0$ for every $\mathbf{x}$ where $\xi(\mathbf{x}) \leq \theta$. Then, for any value of $\lambda > 0$, every maximizer of $\alpha_{ER}(\mathbf{x}; \lambda)$ will satisfy $\xi(\mathbf{x}) > \theta$, or will equal $\mathbf{x}^s$.*

This result is shown in Section 6.1.2, which also describes how this setting is easily encountered in practice when there is a trade-off between objective and sparsity, as in Figure 2.1. Empirically, Theorem 2.1 means that once a good non-sparse point has been found, sparse points will not be selected by the regularized acquisition function, regardless of how λ is tuned. Increasing λ will change the maximum of the regularized acquisition function from a non-sparse point directly to the trivial solution of $\mathbf{x}^s$, skipping all levels of sparsity in between. The acquisition function has no way of selecting sparse points that improve over other points with a similar level of sparsity.

2.3.2 Internal regularization

An alternative approach for adding regularization to the acquisition optimization is to add it directly to the objective function. In this approach, instead of using the posterior of f to compute the acquisition function, we compute the acquisition for the posterior of a regularized function:

$$g(\mathbf{x}; \lambda) = f(\mathbf{x}) - \lambda\xi(\mathbf{x}). \tag{2.3}$$

We refer to this as *internal regularization* (IR). The goal of the acquisition function is then to maximize g, which can be made to have a sparse maximizer by appropriately setting λ. With internal regularization, EI becomes

$$\begin{aligned} \alpha_{\text{EI-IR}}(\mathbf{x}; \lambda) &= \mathbb{E}_f \left[(g(\mathbf{x}) - g(\mathbf{x}^*))_+\right] \\ &= \mathbb{E}_f \left[(f(\mathbf{x}) - f(\mathbf{x}^*) - \lambda(\xi(\mathbf{x}) - \xi(\mathbf{x}^*)))_+\right] \end{aligned} \tag{2.4}$$

where $\mathbf{x}^*$ is now the incumbent-best of g, not of f. The difference between external and internal regularization depends on the acquisition function. It is easy to see that for the UCB acquisition of Equation (1.5), they are identical. For EI they are not, as seen by comparing Equations (2.2) and (2.4). For EI, internal regularization avoids some of the issues of external regularization by incorporating sparsity directly into the assessment of improvement. In Equation (2.4), improvement is measured both in terms of increase of objective and increase in sparsity, and it is measured

with respect to an incumbent best that has incorporated the sparsity penalty. However, internal regularization can also be incapable of recovering points at every level of sparsity, as we will show now. For this result, we are interested in the optimal objective value as a function of sparsity level:

$$h(\theta) = \max_{\mathbf{x} \in \mathcal{B}} f(\mathbf{x}) \text{ subject to } \xi(\mathbf{x}) = \theta. \tag{2.5}$$

A trade-off between sparsity and objective would result in $h(\theta)$ increasing with θ, though it need not be strictly increasing. We now give the negative result for internal regularization, see Section 6.1.2 for details.

Theorem 2.2. *For any θ in the interior of an interval where h is strictly convex, there is no maximizer of Equation (2.3) with $\xi(\mathbf{x}) = \theta$, for any $\lambda > 0$.*

This result shows that internal regularization can only hope to recover optimal points at all sparsity levels if h is concave on its entire domain. This is a strong condition, one unlikely to hold for the types of functions typically of interest in BO, even with simple regularizers. Note that this result is independent of the choice of λ and the acquisition function used. If the desired level of sparsity happens to lie within a region where h is strictly convex, internal regularization can be expected to fail to find the optimum. Figure 6.1 in Section 6.1.2 shows an illustration of this result, in a problem where h has a region of strict convexity.

We will see in the empirical results that internal regularization performs better than external regularization, though, consistent with Proposition Theorem 2.2, can fail to cover the entire objective vs. sparsity trade-off and so neither is the recommended approach for sparse BO. In this paper we focus on EI, but both forms of regularization can be applied to any acquisition function, including entropy search methods. In entropy search, the acquisition function evaluates points according to their information gain with respect to the current belief about the location or function value of the optimum. The information gain will thus depend on the level of sparsity in a similar way as with EI, and so external and internal regularization have similar considerations.

2.4 Multi-objective Optimization

There are two fundamental challenges with both of the regularization approaches developed in Section 2.3. The first is that they both have a regularization coefficient λ that must be set. In a regression setting, the regularization coefficient is usually set to maximize cross-validation accuracy through hyperparameter optimization, often using grid search or BO (Snoek et al., 2012). In sparse

BO, if there is a known desired level of sparsity, λ can be swept in each iteration of optimization to find a value that produces candidates with the desired level of sparsity. This significantly increases the overhead of BO by requiring hyperparameter optimization as part of every acquisition optimization. Furthermore, in real applications the desired level of sparsity is typically not known *a priori*.

When there is a trade-off between interpretability and system performance, the desired level of interpretability will depend on what that trade-off looks like. In practice, we thus wish to identify the best-achievable objective at any particular level of sparsity. The second challenge is that, per the results of Theorems 2.1 and 2.2, we may not be able to identify the entire objective vs. sparsity trade-off, no matter how λ is swept. Depending on the problem, it may be that the sparsity levels of interest cannot be explored via either regularization strategy. Both of these challenges can be addressed by viewing sparse BO from the lens of multi-objective BO.

2.4.1 Sparse BO as Multi-objective BO

In this section we introduce the Sparsity Exploring Bayesian Optimization method (SEBO), which takes a multi-objective approach to sparse BO. Rather than considering ξ as a penalty applied to the objective, we consider f and $-\xi$ to each be objectives that we wish to maximize.

First, we note the following connection between internal regularization and multi-objective BO.

Remark 2.1. *Internal regularization can be viewed as a linear scalarization of the two objectives f and $-\xi$, with λ the weight. Linear scalarizations are commonly used in MOO (Marler and Arora, 2010)—see Section 6.1.4 for more discussion of the connection between internal regularization and the ParEGO method for multi-objective BO.*

Casting sparse BO as MOO of the objective and sparsity has several advantages. It provides a solution for setting the regularization coefficient λ, since we can use methods from multi-objective BO to optimally balance improvements in f and ξ with the goal of exploring the Pareto frontier. We can use powerful approaches such as EHVI to select points that maximize performance for all levels of sparsity, or equivalently, maximize sparsity for all levels of performance, explicitly optimizing for the entire regularization path. The goal of multi-objective BO is to identify the optimum for every level of sparsity, which enables decision makers to make an informed trade-off between interpretability and other considerations of system performance. State-of-the-art MOO methods also avoid the issues of Theorems 2.1 and 2.2 and are able to explore the entire Pareto front.

In our experiments, we use the EHVI acquisition function. Here, the hypervolume improvement is defined with respect to a worst-case reference point $\mathbf{r} = [r_f, r_\xi]$, which can be set to estimates

for the minimum and maximum values of f and ξ respectively. Given a set of observations $X^{\text{obs}} = \{\mathbf{x}^1, \ldots, \mathbf{x}^n\}$, the Pareto hypervolume of is defined as

$$V(X^{\text{obs}}) = \lambda_M \left(\bigcup_{i=1}^{n} \left([r_f, r_\xi] \times [f(\mathbf{x}^i), \xi(\mathbf{x}^i)] \right) \right),$$

where λ_M denotes the Lebesgue measure. The expected hypervolume improvement is computed as

$$\alpha_{\text{SEBO}}(\mathbf{x}) = \mathbb{E}_f \left[V(X^{\text{obs}} \cup \{\mathbf{x}\}) - V(X^{\text{obs}}) \right]. \tag{2.6}$$

This acquisition function is hyperparameter-free, and, as we will see, is highly effective for sparse BO. In the experiments, we standardize the objectives when calculating the hypervolume. It is also possible to weight objectives differently to encourage greater exploration of sparse or high-performing solutions. We refer to the resulting method as SEBO, and explore its performance in combination with the L_0 sparse regularization, described next. The SEBO-L_0 algorithm is shown in Section 2.6.

2.5 Acquisition Functions with L_0 Sparsity

Our primary focus is L_0 sparsity, which comes with the challenge that the L_0 quasi-norm is discontinuous, making the resulting acquisition function challenging to optimize. We will follow the idea of homotopy continuation, which has been successfully applied to, for instance, solving non-linear systems of equations and numerical bifurcation analysis (Allgower and Georg, 2012).

The main idea is to define a homotopy $H(\mathbf{x}, a)$, where $H(\mathbf{x}, a_{\text{start}})$ corresponds to a problem that is easy to solve and $H(\mathbf{x}, a_{\text{end}})$ corresponds to the target problem. In particular, for $a > 0$ we define $H(\mathbf{x}, a) = \mathbb{E}_f[u([f(\mathbf{x}), \varphi_a(\mathbf{x})])]$ where $\varphi_a(\mathbf{x}) := D - \sum_{i=1}^{D} \exp\left(-0.5\,(\mathbf{x}_i/a)^2\right) \approx \|\mathbf{x}\|_0$ and $\mathbf{x} \in \mathbb{R}^D$. Under the assumption that the utility function $u(x)$ defined in Section 2.2 is continuous, we have $\lim_{a \to 0^+} H(x, a) = \mathbb{E}_f[u([f(x), \|\mathbf{x}\|_0])]$, which corresponds to the original acquisition function with the L_0 quasi-norm.

While it may be tempting to set a to a small value, e.g., $a = 10^{-3}$, and optimize the acquisition function directly, this will not work well as the gradient of the homotopy is (numerically) zero almost everywhere in the domain. On the other hand, setting a to a large value, e.g., $a = 1$ will make it much easier to optimize the acquisition function, but also result in a poor approximation of the true acquisition function that will likely not yield sparse solutions. In order to optimize the acquisition function, we will start at some value a_{start} large enough to make the acquisition function

easy to optimize and slowly decrease a towards $a_{\text{end}} = 0$. Each time we change a we re-optimize the acquisition function starting from the best solution found for the previous value of a.[1] This idea is illustrated in Figure 2.2 where we plot snapshots of $H(\mathbf{x}, a)$ for a few values of a as well as show the resulting continuous homotopy path.

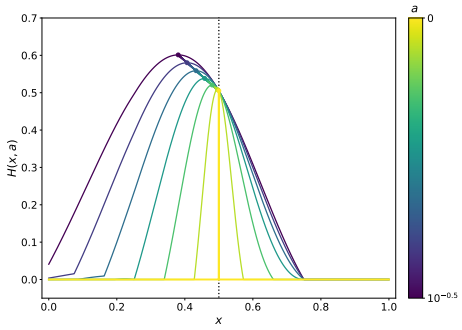


Figure 2.2: Consider the 1D problem of using SEBO to optimize $f(x) = -x^2$ with an L_0 penalty $\xi(x) = \|x - 0.5\|_0$. Assume $X^{\text{obs}} = \{0, 0.25, 0.75, 1.0\}$ have already been evaluated and we want to optimize SEBO to generate the next candidate. The global optimum of the acquisition function is given by the sparse point $x = 0.5$. We show that optimizing the acquisition function along the continuous homotopy path starting at $a_{\text{start}} = 10^{-0.5}$ allows us to eventually uncover find the true optimum of $x = 0.5$.

2.6 SEBO Algorithm

The SEBO-L_0 method is described in Algorithm 1. We start with an initial set of space-filling quasi-random experiment designs. In each iteration step, we fit a SAAS GP model and optimize the acquisition function to find the next point to evaluate, as shown at Line 1. When optimizing the acquisition function, homotopy continuation is used to handle the discontinuous L_0 norm. This part is shown on Line 11.

2.7 Experiments

We evaluate EI-IR, EI-ER and SEBO on two synthetic and three real-world problems with a focus on high-dimensional problems. Note that SEBO can be used for low-dimensional problems as well. Additional details are included in Section 6.1.6. SEBO also naturally extends to multi-objective BO problems, and our code release supports that. We focus on single-objective problems to visualize and understand 2D Pareto frontiers, which are difficult to visualize in higher dimensions. We show the

[1] This may appear similar to the idea of learning rate annealing. However, rather than decreasing a hyperparameter of the optimizer, we solve a sequence of optimization problems that approaches the true problem.

Algorithm 1 Sparsity Exploring Bayesian Optimization with L_0 norm (SEBO-L_0)

procedure SEBO-L_0 ▷ Outer loop of BO
  Place a Gaussian Process prior on f
  Observe f at n_0 quasi-random initial points and get the initial dataset $\mathcal{D}_{n_0}$
  for $n \leftarrow n_0 + 1$ to N **do**
    Update the posterior probability distribution on f using observed data $\mathcal{D}_{n-1}$
    Select the next point $\mathbf{x}_n \leftarrow$ OPTIMIZE-HOMOTOPY($\hat{f}_n$)
    Evaluate $\mathbf{x}_n$: $\mathcal{D}_n \leftarrow \{\mathcal{D}_{n-1}, (\mathbf{x}_n, f(\mathbf{x}_n))\}$
  end for
  return The best point
end procedure
procedure OPTIMIZE-HOMOTOPY($\hat{f}$) ▷ Optimize SEBO-L_0 acquisition function
  Define a homotopy $H(\mathbf{x}, a)$ using the posterior on f
  Initialize a candidate pool $\mathcal{X}_a \leftarrow \{\}$
  for $a \leftarrow a_{\text{start}}$ to a_{end} **do**
    $\mathbf{x_a} \leftarrow$ maximize $H(\mathbf{x}, a)$ based on the best points in $\mathcal{X}_a$
    $\mathcal{X}_a \leftarrow \{\mathcal{X}_a, \mathbf{x}_a\}$
  end for
  return $\mathbf{x_a}$
end procedure

results using L_0 regularization for most problems except for the last problem, where the group lasso is used to demonstrate that the methods can be applied to recover different forms of sparsity, such as group sparsity. In addition, we provide an ablation study that demonstrates the importance of using L_0 regularization by comparing it to L_1 regularization. We show in an ablation study that the homotopy continuation approach from Section 2.5 is crucial for effective L_0 regularization.

2.7.1 Experimental setup

Our experiments all have high-dimensional parameter spaces, so we use the SAAS model when optimizing with ER, IR, and SEBO. We compare performance to quasi-random search (Sobol), BO with a standard ARD Matérn-5/2 kernel and the EI acquisition function (GPEI), and SAASBO. For the SAAS model, we use the same hyperparameters as suggested by Eriksson and Jankowiak (2021) and use the No-U-Turn (NUTS) sampler for model inference. The acquisition function is computed by averaging over the MCMC samples. We always scale the domain to be the unit hypercube $[0, 1]^D$ and standardize the objective to have mean 0 and variance 1 before fitting the GP model.

For the homotopy continuation approach described in Section 2.5, we discretize the range of a to use 30 values starting from $a_{\text{start}} = 10^{-0.5}$, see Section 6.1.6 for more details. Figure 6.5 shows that SEBO is not sensitive to the choice of a_{start}. We use a deterministic model for sparsity when using it as an objective. The figures show the mean results across replications (10 replications for the adaptive bitrate simulation (ABR) problem and 20 for all other experiments), and the error bars

correspond to 2 standard errors. All experiments were run on a Tesla V100 SXM2 GPU (16GB RAM). Code for replicating the methods and benchmark experiments in this work is available at https://github.com/facebookresearch/SparseBO.

2.7.2 Evaluation plots

We evaluate optimization performance in terms of the trade-off between the objective and sparsity. To compare the trade-offs, we show the resulting Pareto frontier by treating sparsity as a separate objective, e.g., Figure 2.3 (Right) and Figure 2.4. In particular, for each level of sparsity (active dimensions), we plot the best value found using *at most* that number of non-sparse components. We also show hypervolume traces in Section 6.1.6. In cases where a method is unable to find at least one configuration for a given level of sparsity we assign replications an imputed function value corresponding to the worst label shown on the y-axis. For the synthetic problems where the true active dimensions and optima are known, we plot simple regret for a fixed level of sparsity, e.g., in Figure 2.3 (Left, Middle).

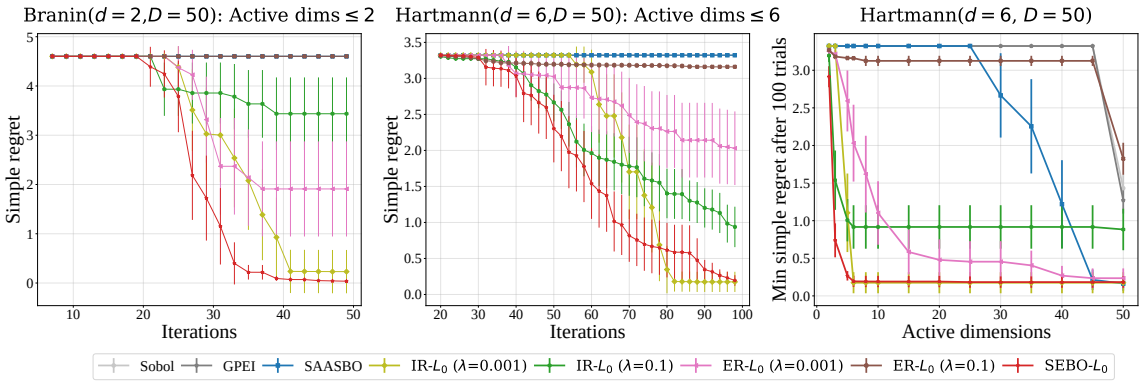


Figure 2.3: (Left) Simple regret for Branin embedded into a 50D space, considering only observations with at most 2 active (non-sparse) parameters. SEBO-L_0 performed the best followed by IR with $\lambda = 0.001$. (Middle) SEBO-L_0 and IR with $\lambda = 0.001$ performed the best for the Hartmann6 function embedded into a 50D space when considering only observations with at most 6 active parameters. (Right) The objective-sparsity trade-off after all 100 iterations on the Hartmann6 problem. Shown is the *Pareto frontier* between sparsity and simple regret after the evaluation budget has been exhausted. SEBO-L_0 is able to explore the trade-offs and is able to discover sparse configurations with fewer than 6 active parameters that are not found by the other methods.

2.7.3 Synthetic functions

We first consider two synthetic problems where the level of sparsity is known. We use the Branin and Hartmann6 functions embedded into a 50D space where 0 is considered sparse, i.e. $\mathbf{x}^s = \mathbf{0}$. We used 50 trials (evaluations) with 8 quasi-random initial points for Branin and 100 trials with 20

quasi-random initial points for Hartmann6. The results are shown in Figure 2.3. The two leftmost plots show the optimization results by evaluating the objective only on observed points whose number of active (i.e., non-zero) parameters was less than or equal to the true effective dimension (2 for Branin and 6 for Hartmann6).

We observe that SEBO-L_0 performed the best, followed by IR with $\lambda = 0.001$. This suggests IR may perform competitively if the regularization coefficient is chosen optimally. On the other hand, ER performed worse than SEBO and IR. Finally, methods with non-regularized acquisition functions (Sobol, GPEI, and SAASBO) failed to identify sparse configurations since they do not explicitly optimize for sparsity of the solutions. Figure 2.3 (Right) visualizes the trade-off between the objective and sparsity and SEBO-L_0 yielded the best sparsity trade-offs.

2.7.4 Ranking sourcing system simulation

The sourcing component of a recommendation system is responsible for retrieving a collection of items that are sent to the ranking algorithm for scoring. Items are retrieved from multiple sources, for instance that may represent different aspects of the user interest taxonomy (Wilhelm et al., 2018). Querying for more items can potentially improve the quality of the recommendation system, but comes at the cost of increasing the infrastructure load. In addition, each source may require individual maintenance; thus, deprecating poor sources could reduce technical debt and maintenance costs of an entire recommendation system (Sculley et al., 2015). Our goal is thus to identify a retrieval policy that uses a minimal number of sources while still maximizing the ranking quality score, measured by a function of content relevance and infrastructure load.

We developed a simulation of a recommender sourcing system that simulates the quality and infrastructure load of recommendations produced by a particular sourcing policy. The sourcing system is modeled as a topic model, where each source has a different distribution over topics, and topics have different levels of relevance to the user. When two sources are (topically) similar, they may obtain duplicate items, which will not improve recommendation quality.

We consider a 25D retrieval policy in which each parameter specifies the number of items retrieved from a particular source. Our desired sparsity is to set parameters to 0 ($\mathbf{x}^s = \mathbf{0}$), i.e., turning off the source. See Section 6.1.6 for more details. We used 8 initial points and ran 100 trials for all the methods. Figure 2.4 (Left) shows that SEBO-L_0 performed the best in optimizing the ranking quality score under different sparsity levels. Sobol and GPEI could not find sparse policies and obtained worse quality scores even with 25 active parameters. IR and SAASBO performed similarly, and ER

with the larger regularization parameter $\lambda = 0.01$ achieved higher quality score with less than 10 active dimensions.

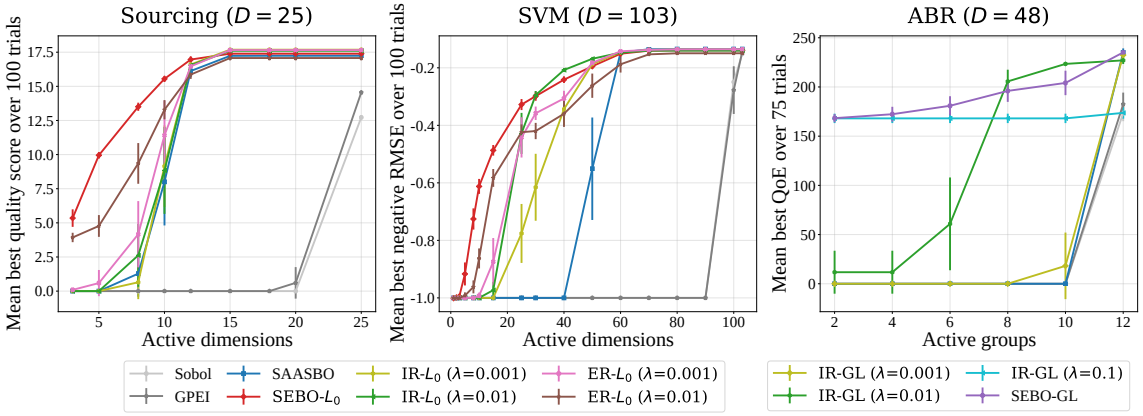


Figure 2.4: Objective-sparsity trade-offs after 100 (75 for ABR) trials for the three real-world problems. (Left) *Sourcing problem*: SEBO-L_0 regularization effectively explored all sparsity trade-offs. (Middle) *SVM problem*: ER with $\lambda = 0.01$ and IR with $\lambda = 0.01$ were able to explore parts of the Pareto frontier, however were dominated by SEBO-L_0. (Right) *ABR problem*: Similar behavior as in the SVM problem was seen here with a group lasso penalty.

2.7.5 SVM machine learning hyperparameter tuning

We consider the problem of doing joint feature selection and hyperparameter tuning for a support vector machine (SVM). We tuned the C, ϵ, and γ hyperparameters of the SVM, jointly with separate scale factors in the continuous range $[0, 1]$ for each feature. We used 100 features from the CT slice UCI dataset Asuncion and Newman (2007) and the goal was to minimize the RMSE on the test set. This produces a 103D optimization problem where we shrink each feature towards a scale factor of 0, i.e. $\mathbf{x}_i^s = 0$, as it effectively removes the feature from the dataset. We took $C \in [0.01, 1.0]$, $\epsilon \in [0.01, 1.0]$, and $\gamma \in [0.001, 0.1]$, where the center of each interval was considered sparse as this is the default value in Sklearn (i.e. $\mathbf{x}_i^s = \text{Mid}(\text{Hyperparameter Interval})$). We optimized C, ϵ, and γ on a log-scale, and initialized all methods with 20 points and ran 100 evaluations. Figure 2.4 (Middle) shows that SEBO-L_0 was best able to explore the trade-offs between sparsity and (negative) RMSE.

2.7.6 Adaptive bitrate simulation

Video streaming and real-time conferencing systems use adaptive bitrate (ABR) algorithms to balance video quality and uninterrupted playback. The goal is to maximize the quality of experience (QoE). The optimal policy for a particular ABR controller may depend on the network, for instance a stream with large fluctuations in bandwidth will benefit from different ABR parameters than a stream

with stable bandwidth. This motivates the use of a contextual policy where ABR parameters are personalized by context variables such as country or network type (Feng et al., 2020). Various other systems and infrastructure applications commonly rely on tunable parameters which can benefit from contextualization.

We suppose that the system has already been optimized with a global non-contextual policy, π_{global}, that is used for all contexts. Our goal here is to use sparse BO to find the contextualized residuals $\Delta\pi_i$ for each individual context i, i.e., $\pi_i = \pi_{\text{global}} + \Delta\pi_i$, with the target sparse point $\mathbf{x}^s$ set to be π_{global}. By regularizing the contextualized residuals $\Delta\pi_i$'s using the group lasso (GL) norm (Yuan and Lin, 2006), we hope to find policies that require minimum alteration to the global policy π_{global}, in which the minimum number of contexts have parameters that deviate from the global optimum. This adds both simplicity and interpretability to the contextual policy, since we can interpret the policy by looking at the contextual residuals $\Delta\pi_i$.

Figure 2.4 (Right) shows the results of applying our methods to the contextual ABR optimization problem from Feng et al. (2020). For this problem, we have 12 contexts and 4 parameters for each context resulting in a 48D optimization problem. We used 75 trials with 8 quasi-random initial points for all the methods. The group lasso penalty is defined by assigning parameters for each individual context to be within the same group. We observe that IR with a fixed λ was able to explore trade-offs at certain sparsity levels and that stronger regularization (larger λ) resulted in finding configurations that were more sparse. SEBO-GL, on the other hand, automatically and efficiently explored the trade-off between sparsity and reward at all sparsity levels. All other baselines (Sobol, GPEI, SAASBO) failed to find any sparse configurations that achieve non-zero reward.

2.7.7 Ablation study

We show by means of an ablation study the importance of using the homotopy continuation approach from Section 2.5 to target L_0 sparsity. We focus on SEBO as it consistently outperformed IR and ER, and refer to Figure 6.6 in Section 6.1.6 for additional results on the importance of using the SAAS model. The results from the ablation study can be seen in Figure 2.5. Using a fixed value of a for the L_0 approximation performs poorly, particularly when a is small, which is due to the acquisition function being zero almost everywhere and thus difficult to optimize. On the other hand, $a = 1$ results in a failure to discover sparse configurations and the resulting method performs similar to SAASBO (see Figure 2.3). In addition, we show that for all approaches (ER, IR, and SEBO), working directly with L_0 regularization works significantly better than the frequently used L_1 regularization.

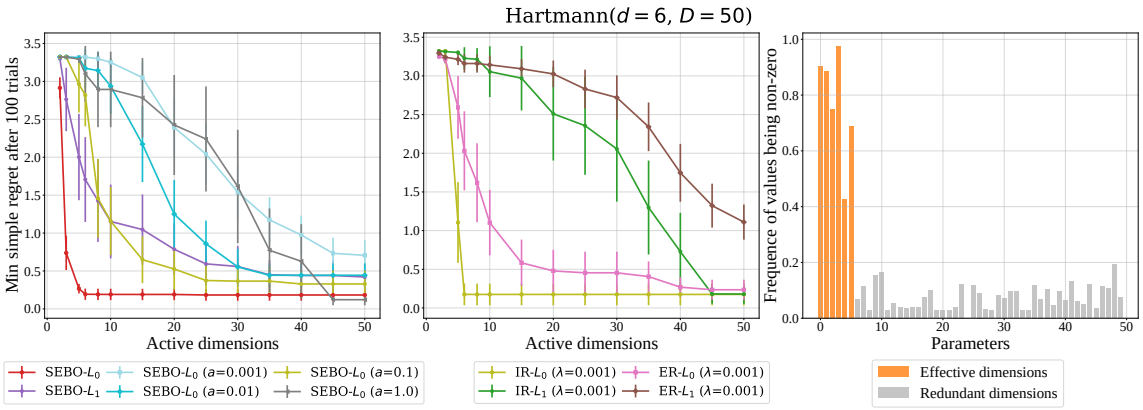


Figure 2.5: Ablation study on the Hartmann6 function embedded in a 50D space. (Left) SEBO-L_0 works much better than SEBO-L_1 as it directly targets L_0 sparsity. Using a fixed value of a performs poorly, confirming the importance of our homotopy continuation approach. (Middle) Working directly with L_0 regularization works drastically better than L_1 regularization for both IR and ER. (Right) The 6 important parameters are more frequently included in Pareto optimal configurations for the embedded Hartmann6 problem.

Finally, we show in Figure 2.5 (Right) how frequently each parameter is turned on (non-zero) in the final Pareto frontier for each replication of SEBO-L_0, which indicates the method correctly identifies the important parameters.

2.7.8 Interpretation of sparse solutions

Ranking sourcing system simulation

We examine what active dimensions are selected in the recommender sourcing system problem to understand the obtained sparse solutions. For SEBO-L_0 results across 20 replications, we obtain the optimal 25-dimensional retrieval policy and also compute the average of retrievals per source at each sparsity level. For each source, we compute a source quality scores based on the simulation setup stated in Section 6.1.6. Each source contains a mixture over a set of topics with source relevance score being q_s and the infrastructure cost per fetched item being c_s. With this, we define and compute the source quality score as $q_s - 4 \times c_s$. Note the score is computed for each source in order to interpret the obtained solutions and differ from the quality score used in the optimization.

In Figure 2.6, the left heatmap visualizes the optimal policy at different sparsity levels across 20 replications and the middle one visualizes the average retrieval policy values. Each column corresponds to one source and is sorted based on source quality score in an ascending order (from left to right); each row represents the sparsity level (number of active dimensions). The color indicates the parameter values. As it can be seen, sources with low quality scores are turned off (zero query)

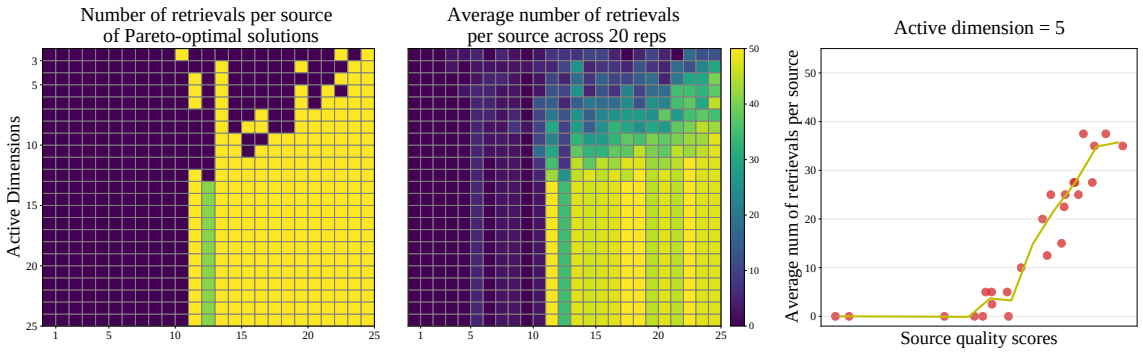


Figure 2.6: (Left). The heatmap of optimal retrieval policy at different sparsity levels. (Mid) The heatmap of average retrieval policy values at different sparsity levels. (Right) The scatter plot between average retrieval policy values with 5 active parameters and source quality score. We can see that more items are retrieved from higher quality sources, while the number of items from lower quality sources are driven to zero to achieve sparsity.

and sources with higher scores have higher number of retrievals even with smaller active dimensions. This indicates that the sparse policy obtained from SEBO identifies the most effective sources at each sparsity level. The right plot in Figure 2.6 shows the relationship between number of items retrieved from each source and source quality score with 5 active parameters. Each dot represents a source. The curve is a fitted spline to visualize the relationship. From both plots we can see that more items are retrieved from higher quality sources, while the number of items from lower quality sources are driven to zero.

Synthetic Function - Branin $(d = 2, D = 50)$

Similar to Figure 2.5 (right), we compute the frequency of each parameter is turned on (non-zero) in the final Pareto frontier for each replication of SEBO-L0. These frequencies help us to identify the important parameters and interpret the sparse policies, shown in Figure 2.7 (left). The two true effective dimensions in augmented Branin $(d = 2, D = 50)$, colored in orange bars, have the highest frequencies and are identified by SEBO-L0.

SVM Machine Learning Hyperparameter Tuning

Figure 2.7 (right) visualizes the frequency of parameter values being non-sparse in the final Pareto frontier for each replication of SEBO-L0. The sparse values are the center of each interval of the three hyperparameters γ, C and ϵ. For the augmented parameters, values being zero are considered as sparse. The three orange bars correspond to the three effective hyperparameters of the SVM, which obtains high frequencies of being non-sparse. The gray bars, corresponding to the augmented

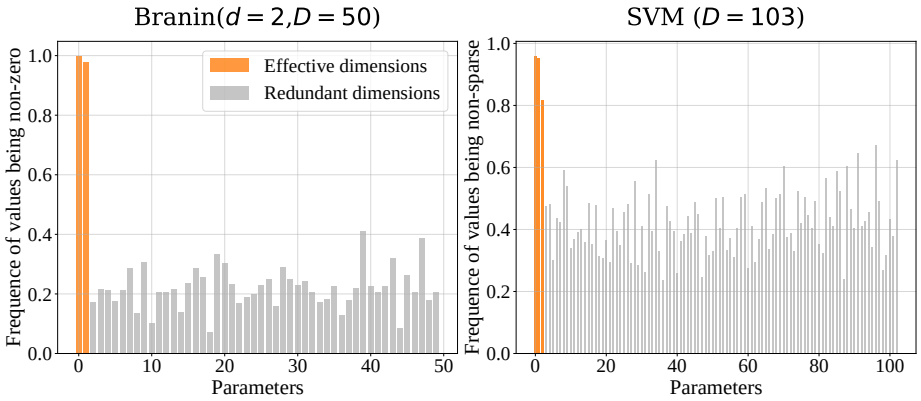


Figure 2.7: (Left) Branin ($d = 2, D = 50$). The 2 true effective parameters (colored as orange) are more frequently set to be non-zero in Pareto optimal configurations. (Right) SVM ($d = 3, D = 103$). The 3 effective hyperparameters (orange) of the SVM have higher frequencies of being non-sparse compared with the augmented dimensions (gray bars).

dimensions, have much lower frequencies.

2.8 Discussion

BO is a powerful tool for sample-efficient optimization of real-world systems. Recent developments in BO have made it possible to optimize hundreds of parameters, providing solutions to complex optimization problems in science and engineering. Yet practitioners and decision-makers often favor simplicity in the solutions, e.g., in the design space, for the sake of interpretability, managing risk, or for reducing technical debt. This poses a new challenge: how should we discover well-performing *and* parsimonious designs in a sample-efficient manner?

We show that sparsity-inducing models are not sufficient for producing sparse designs, and examine several schemes for penalizing design parameters within the acquisition function itself. We utilize theoretical insights from multi-objective optimization to identify limitations of common penalization approaches and propose SEBO, which optimizes for both sparsity and performance. In doing so, we are able to learn the entire set of optimal trade-offs between objective and sparsity, allowing decision makers to select the amount of objective they are willing to sacrifice for increased interpretability and simplicity.

Our formulation is compatible with a variety of regularizers, including L_0, L_1, and the group lasso penalties. To enable the optimization of the discontinuous L_0 penalty, we develop a novel acquisition function optimization method based on homotopy continuation that enables gradient-based optimization. We find that SEBO with L_0 penalization consistently outperforms all other methods in

identifying optimal designs, while also eliminating the need to tune regularization hyperparameters.

Our work has a few limitations that suggest areas for future work. First, SEBO can be useful for identifying the entire Pareto frontier of sparse solutions, but in some contexts decision-makers may have a desired sparsity level in mind. Further work is required to develop adaptive algorithms that can efficiently target specific sparsity levels. Second, if the goal is to reduce regret while achieving sparsity, there may be opportunities for theoretical work on selecting model and acquisition function regularization parameters simultaneously, see, e.g., Bastani and Bayati (2020).

Chapter 3

Amortized Gaussian Process Hyperparameter Inference

3.1 Introduction

Gaussian processes (GPs) are powerful tools for modeling distributions over functions. They are highly flexible Bayesian nonparametric models for which the posterior is often available in closed form. GPs are successful models for a variety of machine learning tasks, from regression and classification (Rasmussen and Williams, 2006), to Bayesian optimization (Snoek et al., 2012), to modeling of dynamics (Kuss and Rasmussen, 2004; Wang et al., 2006). The predictive performance of a Gaussian process, however, is highly dependent on the specifics of the prior on functions, as determined by the associated positive definite kernel function (Rasmussen and Williams, 2006). To find a good prior, one needs to first come up with a family of kernel functions that is capable of capturing the structure of the data.

The kernel hyperparameters must then be determined from data, usually by maximizing the log marginal likelihood (MLL), i.e., empirical Bayes (Rasmussen and Williams, 2006). There are two major issues associated with this procedure. First, evaluating the log marginal likelihood (and its gradient) for different hyperparameters is generally expensive, with $\mathcal{O}(N^3)$ computational complexity for N training data. Second, the log MLL is usually highly non-concave, making both sampling and optimization difficult. Getting stuck in bad local maxima or saddle points can result in hyperparameters with significant model mismatch and thus poor predictive performance (Rasmussen, 1997; Neal, 1999; Zhang and Leithead, 2005). Alternatively, to avoid the catastrophic failure of

bad local minima and point-estimates in MLL optimization, one may attempt a full Bayesian treatment by integrating out the kernel hyperparameters using Markov chain Monte Carlo (MCMC) techniques (Murray and Adams, 2010; Filippone and Girolami, 2014; Filippone et al., 2013). In this paper, we focus on the optimization case, although many of the ideas we present could also be applied to marginalization of hyperparameters.

Various approaches to scaling Gaussian processes have been proposed, exploring innovative ideas such as intelligently selecting a subset of the data (Smola and Bartlett, 2001; Williams and Seeger, 2001) or constructing a low-rank approximation of the covariance matrix based on virtual "inducing" points (Csató and Opper, 2002; Quiñonero-Candela and Rasmussen, 2005; Seeger et al., 2003; Snelson and Ghahramani, 2006; Titsias, 2009; Hensman et al., 2013). These scaling approaches have generally focused on how to solve the linear system more quickly without having to significantly compromise the model or the predictive performance, either by reducing the size of the linear system, solving the linear system approximately, or approximating the model with one that has computationally convenient structure. Focusing on the linear system has been a sensible approach, as effective solutions simultaneously enable both direct prediction with the Gaussian process and evaluation of hyperparameters via the log marginal likelihood.

Here however, we take an entirely different approach and focus solely on the model selection problem, without reference to linear systems at all. Instead, we *amortize* the Gaussian process model selection problem by training a neural network to consume input/output observations and emit an estimate of the hyperparameters that would otherwise arise from maximizing the log marginal likelihood. This approach is inspired by amortized variational inference approaches (Kingma and Welling, 2013; Rezende et al., 2014; Gregor et al., 2014; Ritchie et al., 2016), which similarly sidestep expensive optimization procedures in favor of directly producing estimates. In the variational inference case, the neural network produces approximate posterior distributions over the unknown latent variables; here we produce point estimates of the unknown hyperparameters.

Of course, as noted above, selection of the kernel family is just as important as determination of hyperparameters, and we view this as a crucial piece of the amortized model selection puzzle. One approach to modeling the huge space of valid kernel functions is to represent the target kernel as a composition of different base kernels (Duvenaud et al., 2013; Lloyd et al., 2014; Sun et al., 2018). In principle, various base kernels, composition rules, and associated hyperparameters could be modeled as latent random variables, and emitted by a neural network. However, since the latent variables here involve a mix of discrete variables (types and combinations of kernel functions) and continuous variables (hyperparameters associated), we have found this approach to be difficult.

Moreover, the interrelated effects of, e.g., length scale and kernel choice, make it difficult to unify the hyperparameter space across composed kernels. Thus, instead of working with compositions of commonly-used kernels, we focus on stationary kernels in the spectral domain and directly learn the spectral density of the kernel function (Wilson and Adams, 2013; Samo and Roberts, 2015; Remes et al., 2017). The space of spectral densities provides a unified and compact continuous representation of the space of stationary covariance functions. In particular, we model the spectral density of the kernel function as a mixture (Wilson and Adams, 2013) that is capable of approximating any stationary kernel arbitrarily well.

There are two particularly salient challenges associated with training a single neural network to produce effective hyperparameters for many different regression-type tasks: both the amount of data and the dimensionality of the input can vary from problem to problem. To address this, we develop a specialized hierarchical self-attention structure that consumes datasets and produces spectral densities while being invariant to permutations of the data and the dimensions. Thus, this single "meta-model" can be applied to different problems for which a Gaussian process is applicable. The parameters of the network are trained using gradients computed via reverse-mode automatic differentiation through the log marginal likelihood of the Gaussian process, for randomly-generated synthetic data from the prior. Then we directly apply the trained neural model to real-world datasets of varying size and dimension in different GP applications. Even though the model is trained with only synthetic data, experimental evidence indicates that the estimated hyperparameters are comparable in quality to those from the conventional model selection procedures while being $\sim$100 times faster to obtain.

3.2 Amortized GP Hyperparameter Inference

In this section, we frame the problem of amortized Gaussian process hyperparameter inference. The objective is to avoid the computational overhead and fragility of maximizing the log marginal likelihood, particularly within the inner loop of an iterative procedure such as Bayesian optimization or Bayesian quadrature. Our approach is to instead train a flexible function approximator, i.e., a neural network, to provide high-quality estimates of kernel hyperparameters, directly from the data. We call our method *amortized hyperparameter inference for Gaussian processes* (AHGP).

In GP regression, it is common to use stationary kernels, which include most generic covariance functions, such as the exponentiated quadratic, rational quadratic and Matérn. Consistent with this, we will study stationary kernels, but seek to make them highly flexible. One way to achieve flexibility is via composition of different commonly-used base kernels as in Duvenaud et al. (2013); Lloyd et al.

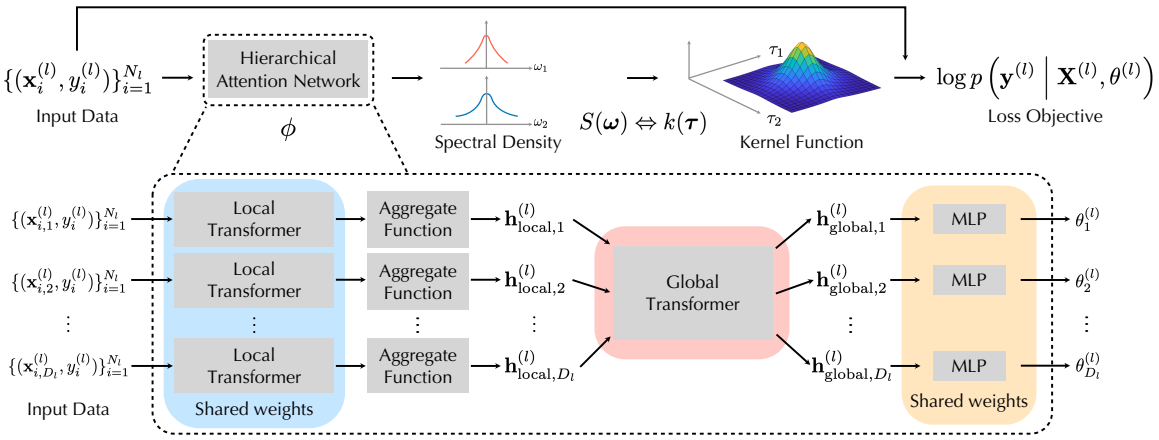


Figure 3.1: The top part of the figure gives an illustration of the computation graph in AHGP. The bottom part describes our hierarchical attention neural net architecture.

(2014); Sun et al. (2018). Model selection in this approach—choice of base kernels, composition rules, and kernel hyperparameters—could in principle be done via a neural network; however, we have found the associated optimization problem to be difficult and complicated. First, it involves optimization over both discrete (kernel types and composition rules) variables and continuous (hyperparameters) variables. Second, different kernels bring with them continuous parameters with different cardinalities and interpretations, making it challenging to identify a unified hyperparameter space.

3.2.1 Spectral modeling of stationary kernel functions

In lieu of a compositional approach to the kernel function, we build a flexible approach around the duality between stationary kernels and their spectral density, taking advantage of the well-known theorem by Bochner stated below.

Theorem 3.1. *(Bochner (1959)) A complex-valued function k on $\mathbb{R}^d$ is the covariance function of a weakly stationary (also known as covariance-stationary) mean square continuous complex-valued random process on $\mathbb{R}^d$ if and only if it can be represented as*

$$k(\tau) = \int_{\mathbb{R}^d} e^{2\pi i \omega^\top \tau} d\mu(\omega), \tag{3.1}$$

where μ is a positive finite measure, often known as the spectral measure of the random process.

When μ is absolutely continuous with respect to the Lebesgue measure, its Radon–Nikodym derivative (density) $S(\omega)$ is called the *spectral density* of the random process. The kernel function $k(\cdot)$ and the spectral density $S(\cdot)$ can be understood to be Fourier transform pairs. We take advantage of

this correspondence and use a neural network to predict the spectral density of the kernel function rather than the covariance function itself.

Following previous work (Wilson and Adams, 2013; Wilson et al., 2014), we model the spectral density via a Gaussian mixture, leading to interpretability and closed form evaluation of the kernel. Additionally, the fact that Gaussian mixtures are dense in the space of probability distribution functions (Silverman, 1986; Wilson and Adams, 2013) makes them capable of approximating the spectral density of any stationary kernel function arbitrarily well. Here we further assume that the kernel function has a product structure over different dimensions and every dimension has its own mixture of Gaussians. The product kernel structure is a common modeling choice in Gaussian process regression, and arises naturally in many generic kernels such as the exponentiated quadratic and its automatic relevance determination (ARD) version. Additionally, it is common to compose kernels via element-wise products, with each dimension's functional properties encoded in its corresponding kernel function (Rasmussen and Williams, 2006; Gönen and Alpaydin, 2011; Duvenaud et al., 2013; Wilson et al., 2014).

3.2.2 Formulation

We now formalize this problem of amortized hyperparameter inference in a Gaussian process. We are interested in fitting many different regression functions of the form $f : \mathbb{R}^D \to \mathbb{R}$, with varying values of D. For the l-th regression task $\mathcal{T}^{(l)}$, a set of input/output training data are observed and are given by $\mathcal{D}^{(l)} := \{(\mathbf{x}_i^{(l)}, y_i^{(l)})\}_{i=1}^{N_l} = \{\mathbf{X}^{(l)}, \mathbf{y}^{(l)}\}$, where $y_i^{(l)} = f_l(\mathbf{x}_i^{(l)}) + \epsilon_i^{(l)}$ with $\mathbf{x}_i^{(l)} \in \mathbb{R}^{D_l}$ and $\epsilon_i^{(l)}$ being i.i.d. zero-mean Gaussian noise. Assume we are given L tasks, with each task randomly sampled i.i.d. from a distribution over tasks, i.e., $\{\mathcal{T}^{(l)}\}_{l=1}^{L} \overset{i.i.d.}{\sim} p(\mathcal{T})$. We further assume that for each task, the function values are generated by some underlying Gaussian process with its own unique kernel hyperparameters. The spectral density of each task's GP is modeled as a mixture of M Gaussian over each dimension as discussed in Section 3.2.1, with weights, means and variances denoted as $\theta_d^{(l)} = \{\{w_{d,m}^{(l)}\}_{m=1}^{M}, \{\mu_{d,m}^{(l)}\}_{m=1}^{M}, \{\sigma^{2(l)}_{d,m}\}_{m=1}^{M}\}$ for the d-th dimension in task l. For compactness, we use $\theta^{(l)} = \{\theta_d^{(l)}\}_{d=1}^{D_l}$ to denote the collective hyperparameters of the spectral density for task l.

The neural network, parameterized by ϕ, defines a function g_ϕ from a dataset $\mathcal{D}^{(l)}$ to an estimate of its spectral density hyperparameters $\theta^{(l)}$, i.e., $\theta^{(l)} = g_\phi(\mathcal{D}^{(l)})$. Through the duality of spectral densities and stationary kernel functions, the spectral mixture product (SMP) kernel (Wilson et al.,

2014) is given by:

$$k_{\mathrm{SMP}_\theta}(\boldsymbol{\tau}) = \prod_{d=1}^{D} k_{\mathrm{SMP}_{\theta_d}}(\boldsymbol{\tau}_d)\,,$$
$$k_{\mathrm{SMP}_{\theta_d}}(\boldsymbol{\tau}_d) = \sum_{m=1}^{M} w_{d,m} \exp\left\{-2\pi^2 \boldsymbol{\tau}_d^2 \sigma_{d,m}^2\right\} \cos\left(2\pi \boldsymbol{\tau}_d \mu_{d,m}\right),$$

where $\boldsymbol{\tau}_d$ is the d-th component of $\boldsymbol{\tau} = \mathbf{x} - \mathbf{x}' \in \mathbb{R}^D$, i.e., the difference of two data points.

We can now train the neural network to produce hyperparameters using a "dataset of datasets", $\left\{\mathcal{D}^{(l)}\right\}_{l=1}^{L}$. With the closed form kernel function specified by its spectral density hyperparameters, the averaged negative log marginal likelihood evaluated from Eqn (1.3) is used as our training objective:

$$\mathcal{L}\left(\phi, \left\{\mathcal{D}^{(l)}\right\}_{l=1}^{L}\right) = -\frac{1}{L}\sum_{l=1}^{L}\frac{1}{N_l}\log p\left(\mathbf{y}^{(l)} \,\middle|\, \mathbf{X}^{(l)}, \theta^{(l)}\right), \tag{3.2}$$

where $\theta^{(l)} = g_\phi(\mathcal{D}^{(l)})$ and N_l is the number of data points in the l-th dataset. Once the neural network is trained, it can be used to estimate the GP kernel function that would be appropriate for a new set of input/output data $\mathcal{D}^{\text{test}}$ by simply doing a forward pass of the neural model.

3.3 Hierarchical Attention Network for GP Hyperparameter Learning

As described in the previous section, the neural network learns a function from a dataset $\mathcal{D}^{(l)}$ to spectral density parameters $\{\theta_d^{(l)}\}_{d=1}^{D_l}$, determining the GP kernel function for task l. As in other deep learning problems, the architecture of the neural network is critical; in particular, the structure of the network must take advantage of available symmetries. In our case, for general purpose inference of GP kernel hyperparameters, we require an architecture that is versatile enough to accommodate datasets of varying input dimension and with different number of data points. Furthermore, the model should be invariant to permutation of both the data and input dimensions. In other words, for a given dataset $\mathcal{D}^{(l)}$, neither shuffling the order of the (exchangeable) data nor shuffling the order of the dimensions should change the resulting estimate of the kernel function. Importantly, the regularization and parameter sharing induced by enforcement of such invariances should enable the neural network to learn better and faster, analogously to convolutional neural networks for images.

3.3.1 Architecture

We draw inspiration from multi-head self-attention mechanisms and propose a hierarchical Transformer (Vaswani et al., 2017) type of neural network architecture for tackling the problem of learning GP hyperparameters. A general Transformer model has multiple layers and each layer consists of a multi-head self-attention sub-layer followed by a feed forward network with residual connections and layer normalizations. It serves as an autoregressive encoder that maps a set of input data to a set of output representations. In particular, the self-attention sub-layers allow each input datum to attend to the representations of other data and produce context-aware representations. Multiple layers of self-attention enable modeling of high-order non-linear interactions between input representations. For details about multi-head self-attention mechanisms, we refer readers to Section 6.2.1.

Briefly, our network architecture mainly consists of two hierarchically nested Transformer-like blocks. A graphic illustration of the proposed architecture is presented in Figure 3.1.

Local Transformer The first transformer block, LocalTransformer, serves as an encoder of the per-dimension local information about the observed function, e.g., length scale, smoothness, periodicity, etc. It takes in a set of input/output data specific to the d-th dimension, e.g., $\mathcal{D}_d^{(l)} = \{(\mathbf{x}_{i,d}^{(l)}, y_i^{(l)})\}_{i=1}^{N_l}$, and outputs a corresponding set of representations $\{\mathbf{h}_{i,d}^{(l)}\}_{i=1}^{N_l}$. In LocalTransformer, only interactions within the d-th dimension are involved, hence each $\mathbf{h}_{i,d}^{(l)}$ is a context-aware representation of local information about the underlying function around dimension d of datum $(\mathbf{x}_i^{(l)}, y_i^{(l)})$.

Aggregate Function The outputs from LocalTransformer$\{\mathbf{h}_{i,d}^{(l)}\}_{i=1}^{N_l}$ are aggregated through an AggregateFunction that assembles a single local dimension-specific representation $\mathbf{h}_{\text{local},d}^{(l)}$ for the d-th dimension. These feature representations $\{\mathbf{h}_{\text{local},d}^{(l)}\}_{d=1}^{D_l}$ provide a summary of each dimension's local information regarding the observed function.

Global Transformer After the dimension-specific local representations $\{\mathbf{h}_{\text{local},d}^{(l)}\}_{d=1}^{D_l}$ are computed, they are fed into a second dimension-level Transformer block, GlobalTransformer, where non-linear interactions between the dimensions are modeled through multiple layers of multi-head self-attention. The final per-dimensional representations $\{\mathbf{h}_{\text{global},d}^{(l)}\}_{d=1}^{D_l}$, which serve as context-aware representations at a global (dimension) level, are further passed through a multi-layer perceptron (MLP) to produce the final spectral density hyperparameters $\{\theta_d^{(l)}\}_{d=1}^{D_l}$.

3.3.2 Versatility and permutation invariance

Self-attention enables the model to consume a set of input/output data with arbitrary data cardinality and dimensionality. The versatility of the model makes it general-purpose: one could train a single neural model to predict GP kernel hyperparameters of different tasks with varying data cardinality and dimensionality, as long as the inputs are real-valued. The proposed model also possesses the following permutation equivariance/invariance properties.

Proposition 3.1. *If* AggregateFunction *is permutation invariant, and weights of* LocalTransformer *and* MLP *are shared across dimensions, then the proposed neural network is permutation equivariant with respect to data dimensions and permutation invariant with respect to data points.*

Intuitively, these properties are inherited from the permutation equivariance of self-attention and the permutation invariance of the AggregateFunction. See Section 6.2.2 for a detailed proof. It is also noted that the versatility and inductive biases (permutation invariance/equivariance) of the proposed hierarchical attention network are generic, making our proposed neural model potentially useful for other tasks that involve learning representations over datasets, such as learning statistics of datasets in Edwards and Storkey (2016).

3.3.3 Complexity analysis.

For each multi-head self-attention layer, the computational complexity is $\mathcal{O}(h \cdot n^2)$, where h is the representation dimension and n is the size of the input set. Assuming that LocalTransformer has l_1 layers with representation dimension h_1 and GlobalTransformer has l_2 layers with representation dimension h_2, the complexity of our model is $\mathcal{O}(l_1 \cdot h_1 \cdot N^2 + l_2 \cdot h_2 \cdot D^2)$, where N is the number of data points and D is the dimensionality. In comparison, exact marginal likelihood optimization scales as $\mathcal{O}(r \cdot N^3)$, where r denotes the number of gradient evaluations during optimization. It is possible to further reduce the complexity of our model to $\mathcal{O}(l_1 \cdot h_1 \cdot m \cdot N + l_2 \cdot h_2 \cdot m \cdot D)$ if we restrict the number of attentions to m by either introducing sparse attentions (Child et al., 2019) or inducing points (Lee et al., 2019), which we leave for future work.

3.4 Related Work

3.4.1 Amortized variational inference

Amortized variational inference (Kingma and Welling, 2013; Kingma et al., 2014; Gregor et al., 2014; Ritchie et al., 2016) was a major inspiration for the present work. Kingma and Welling (2013) first proposed to use a neural network to amortize the expensive variational inference optimization through training on the evidence lower bound (ELBO) with the reparametrization trick. Our work is motivated by this idea, where we use a neural network to produce an estimate of the kernel hyperparameters conditioned on the data. Instead of producing approximate posteriors, we produce point estimates. Note that our model can be directly extended to the variational inference framework, which we leave as future work.

3.4.2 Neural processes

Another line of work, *neural processes (NPs)* (Garnelo et al., 2018b;a; Kim et al., 2019; Louizos et al., 2019), makes use of amortized variational inference to predict a distribution over regression functions from a set of observed input-output pairs. Our work targets estimation of GP hyperparameters as part of a full GP inference procedure, while neural processes serve as an end-to-end approximations to the GP inference itself. It should also be noted that our method is task-agnostic: one single trained neural model can be used across different problems with different dimensionality. In the neural process approach, different neural networks need to be trained for different types of tasks. In particular, Kim et al. (2019) proposed to extend the original NP by introducing attention mechanism between the context data and the target data for better modeling the underlying structure of the problem. In comparison, the hierarchical attention network proposed in our method models not only between data points but also between dimensions, which allows the task-agnostic nature of our method.

3.4.3 GPU-accelerated GP inference

There have been recent efforts (Gardner et al., 2018; Wang et al., 2019) to scale up GP inference with the help of GPUs by introducing smartly-designed batched conjugate gradient algorithms. In contrast, our work focuses on providing a task-agnostic method for amortizing the cost of GP model selection. Note that AHGP is also able to be deployed on GPUs.

3.4.4 Meta-learning

Under the general meta-learning framework, our model could be regarded as a meta-model for predicting hyperparameters. There have been recent explorations of meta-learning that are model-based (Santoro et al., 2016; Duan et al., 2016), metric based (Vinyals et al., 2016; Koch et al., 2015) or optimization based (Schmidhuber, 1987; Bengio et al., 1992; Finn et al., 2017). Gordon et al. (2018) extends the meta-learning framework to probabilistic inference for regression. To better warm-start BO through leveraging previous related BO runs, meta-learning has also been proposed for BO recently (Feurer et al., 2015; Perrone et al., 2018; Feurer et al., 2018). The BO meta-learning approaches focus on the few-shot learning setup: i.e. how to build a good surrogate model with very few function evaluations for a new but similar task. In comparison, our method focuses on fast inference of GP hyperparameters, and our single trained model is adaptation-free and can be directly applied to a wider range of new GP use cases.

3.5 Experimental Results

To empirically evaluate the AHGP, we studied three different GP use cases: regression, Bayesian optimization and Bayesian quadrature.

3.5.1 Baselines

We compare our method to the standard approach of maximizing the log marginal likelihood with respect to hyperparameters. We also compare with the sparse variational Gaussian processes method (SGPR) (Titsias, 2009; Hensman et al., 2013), which uses inducing points to approximate the full GP. The focus of the comparisons will be on the quality of the selected kernel hyperparameters and the run time of the hyperparameter selection procedure.

The baselines are implemented with two popular GP packages: GPy (GPy, 2012) (implemented for CPU) and GPyTorch (Gardner et al., 2018) (implemented for GPU). The spectral mixture (SM) kernel and spectral mixture product (SMP) kernel are used as the kernel functions. These give rise to eight different baselines: GPy-SM, GPy-SM-Sp, GPy-SMP, GPy-SMP-Sp, GPT-SM, GPT-SM-Sp, GPT-SMP, GPT-SMP-Sp, where we use "GPT" to denote "GPyTorch" and "Sp" to denote "SGPR". The default L-BFGS optimizer is used for GPy and the default Adam (Kingma and Ba, 2014) optimizer is used for GPyTorch. The GPyTorch baselines make use of batched conjugate gradient to invert the kernel matrix for efficient approximate inference. We additionally implement a

full GP baseline with SMP kernel that uses Cholesky decomposition in PyTorch (Paszke et al., 2019), and its MLL is optimized via reverse-mode automatic differentiation. We will refer to this baseline as PyT-AD-SMP.

3.5.2 Experimental setup

Synthetic Training Dataset Generation

In our experiments, our training dataset is constructed by sampling multiple sets of input/output data from synthetically generated GPs with stationary kernel functions. For each GP, its data dimensionality is sampled uniform randomly from 2∼15. To sample flexible kernel functions, we randomly generate mixtures of Gaussians to represent its spectral density. The weights of the Gaussian mixtures are drawn from Dirichlet distribution and the lengthscales (i.e., $1/\sqrt{2}\pi\sigma_{(d),m}$'s) are sampled from a log-uniform distribution. The number of mixtures is set to 10 and the concentration parameter of the Dirichlet distribution are set to 1. The input of the data points $\{\mathbf{x}_i^{(l)}\}_{i=1}^{N_l}$ are generated from a Poisson point process within the hypercube $[-1,1]^{D_l}$ with average density being 30. The data output values $\{y_i^{(l)}\}_{i=1}^{N_l}$ are generated from priors of the GP with its specified kernel function. The observation noise is i.i.d. randomly sampled from $\mathcal{N}(0, 0.01)$. We generate 10000 sets of input/output data to be used as the whole dataset, of which we do a split with 50% used for training and the rest for validation. It is worth noting that although our AHGP neural model is trained on datasets with relative small size (∼30), the trained neural model is able to generalize on real-world datasets of much larger size, which is presented in details in Section 3.5. This also shows the effectiveness of the inductive bias introduced in our hierarchical attention network.

Training Details

Our model is trained with PyTorch using the Adam (Kingma and Ba, 2014) optimizer with a fixed learning rate and a batch size of 64. A description of our model architecture is provided in Table 3.1, Table 3.2. We also apply 0.1 dropout to self-attention and the MLPs. The number of Gaussian mixtures in the spectral density prediction is fixed at 10. To validate the effectiveness of our neural network model, we minimize the efforts of hyperparameter tuning during training. The only hyperparameters we tuned are learning rate ($\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$) and number of layers in LocalTransformer and GlobalTransformer (2∼8). The hyperparameters are tuned based on performance on the validation set.

Since spectral mixture kernel assume the variance of the kernel function is normalized, we

Embed each $(\mathbf{x}_{i,d}^{(l)}, y_i^{(l)})$ to 256 dim
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
AggregateFunction: Average pooling

Table 3.1: LocalTransformer architecture and AggregateFunction

Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
Transformer sublayer (4 heads, representation dim 256, feedforward dim 512)
MLP for predicting weights, means and variances (each with hidden dim [256, 128])

Table 3.2: GlobalTransformer architecture and the final MLP

standardize the function values $\{y_i^{(l)}\}_{i=1}^{N_l}$. We also standardize the data input $\{\mathbf{x}_i^{(l)}\}_{i=1}^{N_l}$ as a standard procedure. During training, we find the validation performance is most sensitive to learning rate and increasing layers of the Transformers slightly helps. The final model is trained for 200 epochs with batch size 64, learning rate 10^{-5} and 8 layers in both Local and Global Transformer. The training is done on an NVIDIA GTX 1080 Ti GPU. All the evaluation experiments are run using one core of an Intel(R) Core(TM) i7-6850K CPU for CPU runtime comparisons and an NVIDIA GTX 1080 Ti GPU for GPU runtime comparisons. After training is done with the synthetic data, the *same* trained neural model is directly used to predict the kernel hyperparameters for the various *unseen* GP use-cases that are shown later.

During evaluation and training, both the data input and output are standardized (with a scaling of 0.1) and the noise variance of GP is fixed at 0.01. Note that it is possible for our method to learn the noise variance too. In our experiments, we find the predictive performance is not sensitive to the noise variance if the data are standardized and setting it to 0.01 gives competitive performance for all baselines. Meanwhile, it is noted that spectral mixture kernel is flexible enough to model the noise variance component as well: if one of the Gaussian mixture components has weight w, $\mu = 0$ and very large σ^2, the corresponding kernel matrix will be approximately wI and the weight w represents

the noise scale.

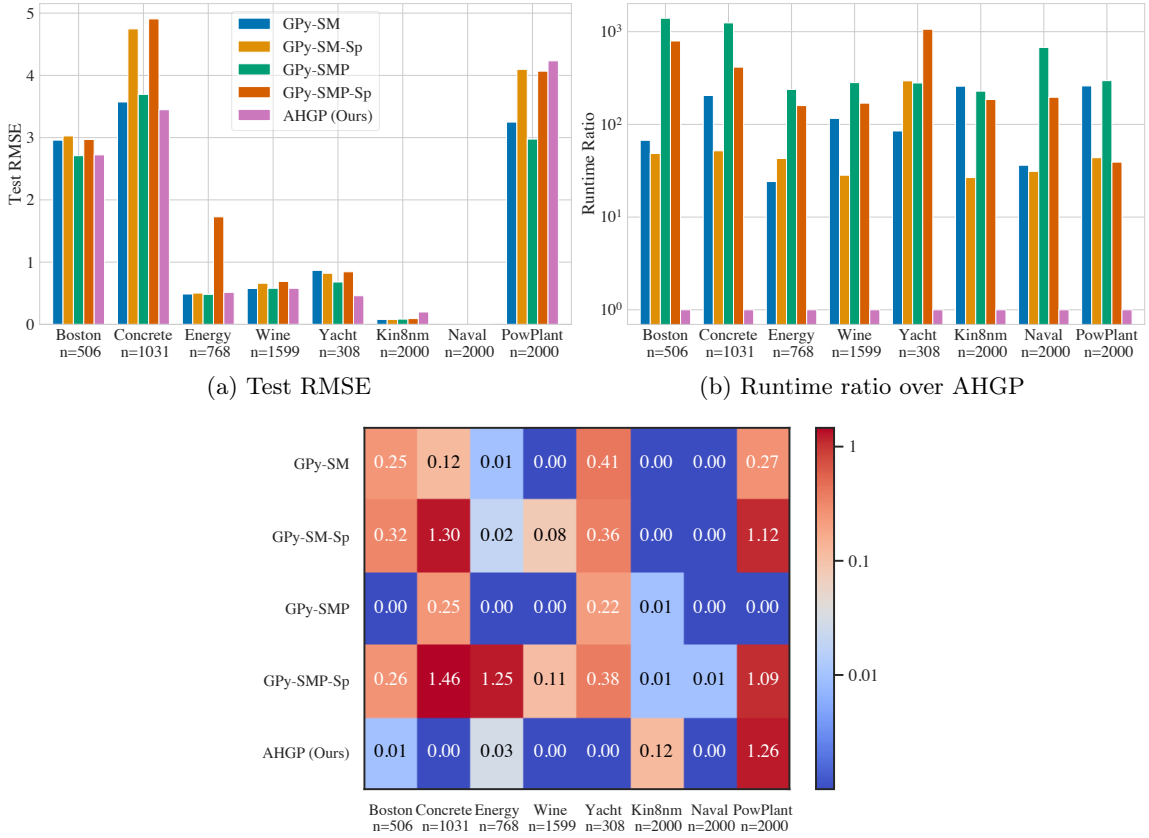
3.5.3 Regression benchmarks

We evaluate our method and the baselines on regression benchmarks from the UCI collection (Asuncion and Newman, 2007) used in Hernández-Lobato and Adams (2015) and Sun et al. (2018) following the same setup: the data are randomly split to 90% for training and 10% for testing. This splitting process is repeated 10 times and the average test performance is reported. Comparisons with CPU-based baselines on test RMSE, test log-likelihood and runtime are presented in Figure 3.2 and Table 6.3. We observe that AHGP has consistently lower run times than the baselines, averaging ∼100 times faster. Nevertheless, the predictive performance of AHGP is comparable to (and sometimes better than) the strongest baselines, which perform MLL optimization without approximation. Notably, AHGP seems to perform slightly better on datasets with fewer data points, such as Yacht. We believe this demonstrates the robustness of AHGP when there is not enough data for MLL-opt based approaches to form reasonable point estimates. The sparse variational GP methods are faster than the full GP methods in general, but with lower performance on both test RMSE and test log-likelihood. Comparisons with GPU-based methods (GPyTorch-based and PyT-AD-SMP) are in Section 6.2.3, where similar findings are obtained.

3.5.4 Bayesian optimization

Bayesian optimization (Močkus, 1975) (BO) uses a GP as a surrogate model when the objective function is expensive to evaluate. The method involves fitting the GP kernel hyperparameters and maximizing an acquisition function to select a candidate point that is highly promising to achieve the function minima (maxima) under the model. Since the method is iterative, MLL optimization needs to be conducted at every BO iteration to update the GP. An amortized approach would greatly reduce the computation involved. We pick the best performing baselines on the regression benchmarks (GPy-SM, GPy-SM-Sp, PyT-AD-SMP) and compare them with AHGP. Expected improvement (EI) is used as the acquisition function. We use standard test functions for global optimization (Derek Bingham, 2013) as the target functions for the BO experiments.

At the start of every BO iteration, the hyperparameters are randomly re-initialized for all baselines. A sample of the experimental results is shown in Figures 3.3 and 3.4. Full results on extensive benchmarks can be found in Section 6.2.3. For runtime, only average is plotted here, Tables 6.8 and 6.9 show details of mean and standard deviation. Again, AHGP is a substantial improvement in



(a) Test RMSE

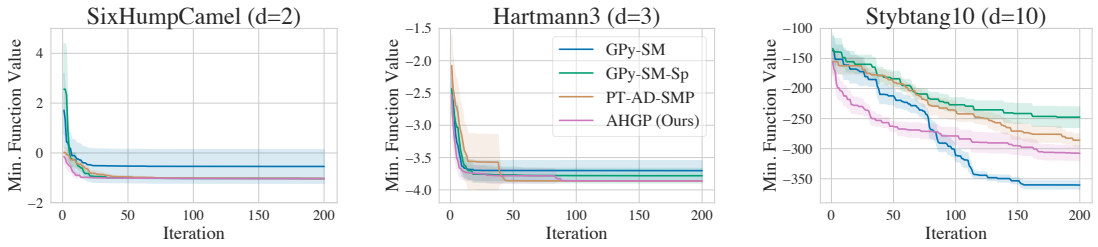
(b) Runtime ratio over AHGP

(c) Gap with best RMSE

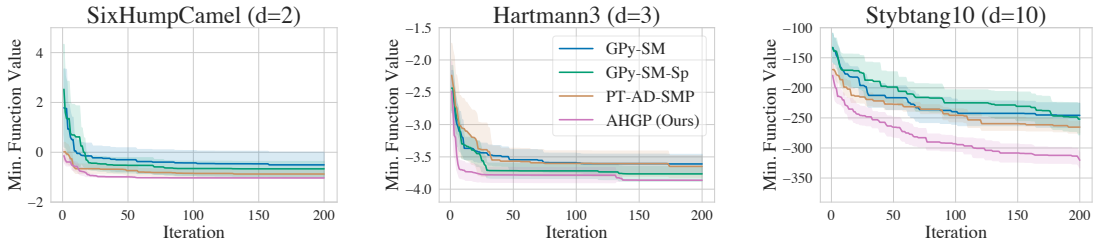
Figure 3.2: Comparison of AHGP against the CPU baselines on regression benchmarks. In (c), the numbers are the differences of the corresponding method's test RMSE with the best RMSE on the respective dataset. Note that for Naval, the RMSEs are all very close to 0. Only average test performance is shown here. Refer to Section 6.2.3 for complete results with error bars.

Dataset	GPy-SM	GPy-SM-Sp	GPy-SMP	GPy-SMP-Sp	**AHGP(Ours)**
Boston	83.56±65.67	60.16±9.98	1731.96±525.13	984.17±282.21	**1.24±0.16**
Concrete	545.08±195.05	138.01±36.99	3307.03±1484.01	1103.31±68.64	**2.66±0.05**
Energy	43.05±16.51	76.30±5.50	424.39±124.63	284.02±376.79	**1.78±0.30**
Wine	1629.73±796.54	397.35±163.41	3961.91±1617.62	2380.69±885.05	**14.04±1.59**
Yacht	18.68±8.44	64.87±1.12	61.56±26.95	233.75±157.30	**0.22±0.01**
Kin8nm	3391.64±958.03	350.49±69.52	2999.11±1036.25	2435.50±169.28	**13.13±0.84**
Naval	915.49±881.60	786.15±163.73	17070.97 ± 5960.81	4955.79 ± 2251.73	**25.25±0.57**
PowPlant	1869.34±1616.23	314.33±74.13	2131.19±490.66	281.75±207.83	**7.19±0.55**

Table 3.3: Runtime (in seconds) on regression benchmark: CPU-based methods



(a) Baseline methods use random re-initialization strategy.



(b) Baseline methods use warmstart initialization strategy.

Figure 3.3: BO performance comparisons. Minimum function values found v.s. number of BO iterations. Shaded region represents 0.5 standard deviation over 10 runs.

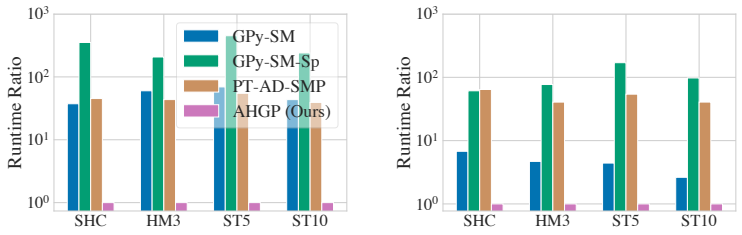


Figure 3.4: Runtime ratio over AHGP. *Left*: Baselines use random re-initialization strategy. *Right*: Baselines use warmstart initialization strategy.

run-time. In terms of minimum values found, AHGP is on par with the baselines on some functions and slightly worse on functions with higher dimensionality. Of particular note—consistent with what was seen on the regression benchmarks—AHGP has the greatest improvement in the beginning when there are few observations available.

To ensure fairness to baseline fitting procedures, we also conducted experiments where the hyperparameter selection in the BO inner loop was initialized using the best from the previous iteration. As expected, this warm-starting results in decreased run times for the baselines, although still slower than AHGP (in Figure 3.3b). This warm-starting, however, seems to compromise the hyperparameter selection—presumably due to local minima—and damage the overall outer loop optimization.

3.5.5 Bayesian quadrature

Bayesian quadrature (O'Hagan, 1991; Rasmussen and Ghahramani, 2003) performs Bayesian inference about the value of a difficult numerical integral through modeling the underlying function as a GP. The tractability of the Gaussian process makes it relatively easy to calculate a Bayesian estimate of many integrals in closed form. For example, Gunter et al. (2014) applies Bayesian quadrature to the task of probabilistic inference and achieves faster convergence than Monte Carlo methods.

We evaluate our model on four standard test functions provided in Emukit (Paleyes et al., 2019). In Figure 3.5, we plot the average distance to ground truth versus number of iterations. Table 3.4 shows the log-likelihood of the true integral evaluated at the final probabilistic estimate. AHGP compares well with the baselines across all tasks. More importantly, AHGP appears to be robust across different tasks while the baselines often suffer from large variances and occasional divergence.

Target Function	GPy-SM	GPy-SM-Sp	PyT-AD-SMP	**AHGP(Ours)**
Hennig1D	NaN	-29.66±26.12	**4.23±0.16**	3.89±0.27
Hennig2D	NaN	-39.99±36.57	-0.13±4.48	**2.41±2.30**
Sombrero2D	NaN	-0.63±3.56	**4.95±0.26**	3.23±0.43
Circular Gaussian	3.79±1.01	4.01±1.27	**5.17±0.04**	3.47±1.09

Table 3.4: Log-likelihood of the true integral evaluated at the final prediction of Bayesian quadrature.

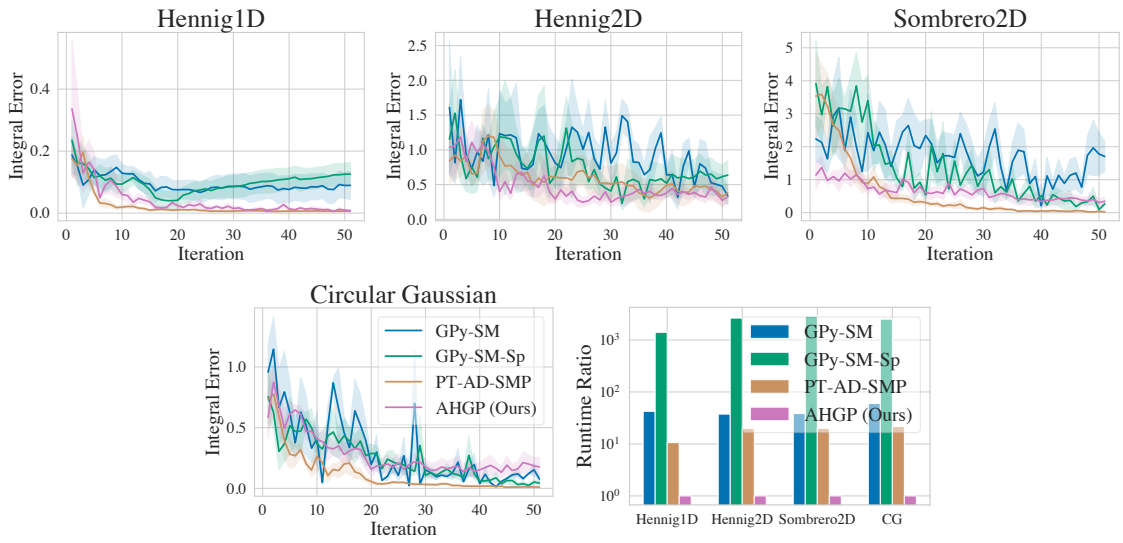


Figure 3.5: Bayesian quadrature performance comparisons. The first four plots show the integral error v.s. number of BQ iterations. Shaded region represents 0.5 standard deviation over 5 runs. The last one shows runtime ratio over AHGP.

Chapter 4

Scalable Discrete Generative Modeling with Marginalization Models

4.1 Introduction

Deep generative models have seen remarkable progress in multiple fields, encompassing image generation, audio syntheis, natural language modeling and science discovery. Most existing models do not support efficient probabilistic inference for key questions such as marginal probability $p(\mathbf{x}_s)$ and conditional probability $p(\mathbf{x}_u|\mathbf{x}_v)$, where s, u and v are disjoint subsets of the variables. The ability to answer such questions is important for many applications such as outlier detection (Ren et al., 2019; Mitchell et al., 2023), masked language modeling (Devlin et al., 2018; Yang et al., 2019b), image inpainting (Yeh et al., 2017), and constrained protein/molecule design (Wang et al., 2022; Schneuing et al., 2022). Furthermore, the capacity to conduct such inference for arbitrary subsets of variables empowers users with control and flexibility in leveraging the model according to their specific needs and preferences. For instance, in protein design domain scientists may want to manually guide the generation of a protein from a user-defined secondary (local) structure under a particular path over the relevant variables, which is only feasible if the generative model can perform arbitrary marginal inference.

Towards this end, neural autoregressive models (ARMs) (Bengio and Bengio, 2000; Larochelle and Murray, 2011) are developed to facilitate conditional/marginal inference based on the idea of modeling a high-dimensional joint distribution as a factorization of univariate conditionals using the chain rule of probability. Many efforts have been made to scale up ARMs and enable any-order

generative modeling under the setting of maximum likelihood estimation (MLE) (Larochelle and Murray, 2011; Uria et al., 2014; Hoogeboom et al., 2021a), and great progress has been made in applications such as masked language modeling (Yang et al., 2019b) and image inpainting (Hoogeboom et al., 2021a). However, marginal likelihood evaluation in most widely-used modern neural network architectures (e.g., Transformers (Vaswani et al., 2017) and U-nets (Ronneberger et al., 2015)) is limited by $\mathcal{O}(D)$ neural network passes, where D is the length of the sequence. This scaling makes it difficult to evaluate likelihoods on long sequences arising in data such as natural language and proteins. Additionally, in the setting of *distribution matching* (DM)—where a function such as a reward or energy can be evaluated pointwise and interpreted as an unnormalized (log) probability for the generative model to match—ARMs are limited to fixed-order generative modeling and lack scalability in training. The subsampling techniques developed for MLE to scale the training of conditionals are no longer applicable when matching log-probabilities (see Section 4.3.3 for details).

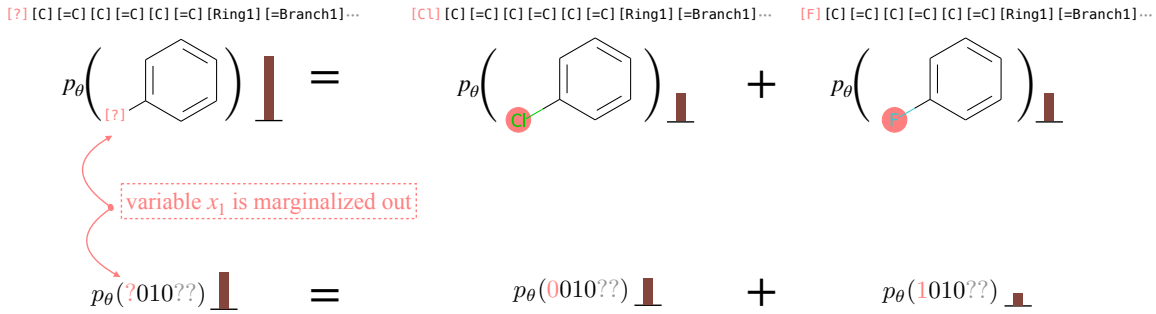


Figure 4.1: Marginalization models enable estimation of any marginal probability with a neural network θ that learns to "marginalize out" variables. The figure illustrates marginalization of a single variable on bit strings (representing molecules) with only two alternatives (versus K) for clarity. The bars represent probability masses.

To enable more scalability and flexibility in generative modeling of discrete data, we propose a new family of generative models, **marginalization models** (MaMs), that directly model the marginal distribution $p(\mathbf{x}_s)$ for any subset of variables $\mathbf{x}_s$ in $\mathbf{x}$. Direct access to marginals not only 1) scales up inference for any marginal, but also 2) enables any-order generative modeling and scalable training under both generative modeling settings.

The unique structure of the model allows it to simultaneously represent the coupled collection of all marginal distributions of a given discrete joint probability mass function. For the model to be valid, it must be consistent with the sum rule of probability, a condition we refer to as *marginalization self-consistency* (see Figure 4.1); enforcing this constraint is one of the key contributions of this work.

We show that marginalization models can be trained under both maximum likelihood and distribution matching settings. We demonstrate the effectiveness of the proposed model in both

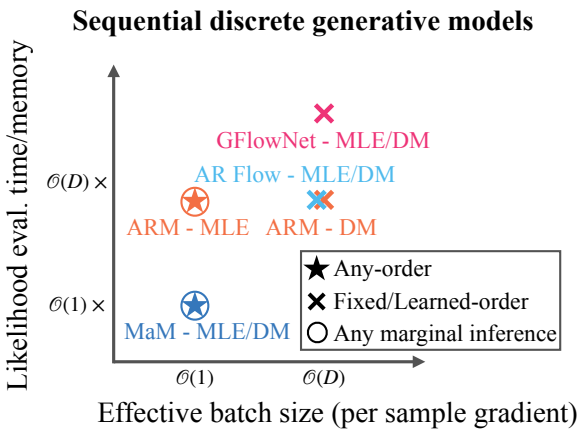


Figure 4.2: Scalability of sequential discrete generative models. The y-axis unit is # of NN forward passes.

settings on a variety of discrete data distributions, including binary images, text, physical systems, and molecules. We empirically demonstrate that marginalization models achieve orders of magnitude speedup in likelihood evaluation. For distribution matching, marginalization models enable scalable training of any-order generative models that previous methods fail to achieve.

4.2 Marginalization Models

We propose *marginalization models* (MaMs), a new type of generative model that enables both scalable any-order generative modeling and efficient marginal evaluation for both maximum likelihood and distribution matching. The flexibility and scalability of marginalization models are enabled by the explicit modeling of the marginal distribution and enforcing *marginalization self-consistency*.

In this paper, we focus on generative modeling of discrete structures using vectors of discrete variables. The vector representation encompasses various real-world problems with discrete structures, including language sequence modeling, protein design, and molecules with string-based representations (e.g., SMILES (Weininger et al., 1989) and SELFIES (Krenn et al., 2020)). Moreover, vector representations are inherently applicable to any discrete problem, since it is feasible to encode any discrete object into a vector of discrete variables.

4.2.1 Definition

We are interested in modeling the discrete probability distribution $p(\mathbf{x})$, where $\mathbf{x} = (x_1, \ldots, x_D)$ is a D-dimensional vector and each x_d takes K possible discrete values.

4.2.2 Marginalization

Given a distribution $p(\mathbf{x})$, and $\mathbf{x} = [x_1, \ldots, x_D]$, where $x_d \in \{1, \ldots, K\}$. Let $\mathbf{x}_s$ be a subset of variables of $\mathbf{x}$, $\mathbf{x}_s \subseteq \{x_1, \ldots, x_D\}$. Denote the complement set of variables as $\mathbf{x}_{s^c} = \{x_1, \ldots, x_D\} \setminus \mathbf{x}_s$. The marginal of $\mathbf{x}_s$ is obtained by summing over all values of $\mathbf{x}_{s^c}$:

$$p(\mathbf{x}_s) = \sum_{\mathbf{x}_{s^c}} p(\mathbf{x}_s, \mathbf{x}_{s^c}) \tag{4.1}$$

We refer to (4.1) as the *marginalization self-consistency* that any valid distribution should follow. The goal of a marginalization model θ is to estimate the marginals $p(\mathbf{x}_s)$ for any subset of variables $\mathbf{x}_s$ as closely as possible. To achieve this, we train a deep neural network p_θ that minimizes the distance of $p_\theta(\mathbf{x})$ and $p(\mathbf{x})$ on the full joint distribution [1] while enforcing the marginalization self-consistency.

4.2.3 Parameterization

To approximate arbitrary marginals over $\mathbf{x}_s$ with a single neural network forward pass, we additionally include the "marginalized out" variables $\mathbf{x}_{s^c}$ in the input by introducing a special symbol "?" to denote the missing values. By doing this, we create an augmented D-dimensional vector representation $\mathbf{x}_s^{\text{aug}} \in \mathcal{X}^{\text{aug}} \triangleq \{1, \ldots, K, ?\}^D$ and feed it to the NN. For example, for a binary vector of length 4, if $\mathbf{x}_s = \{x_1, x_3\}$ where $x_1 = 0$ and $x_3 = 1$, then $\mathbf{x}_s^{\text{aug}} = [0, ?, 1, ?]$ where "?" denotes x_2 and x_4 being marginalized out. From here onwards we will use $\mathbf{x}_s^{\text{aug}}$ and $\mathbf{x}_s$ interchangeably.

A marginalization model parameterized by a neural network θ takes in the augmented vector representation $\mathbf{x}_s^{\text{aug}} \in \{1, \ldots, K, ?\}^D$, and outputs the marginal log probability $f_\theta(\mathbf{x}_s) = \log p_\theta(\mathbf{x}_s)$ that must satisfy the marginalization self-consistency constraints[1]:

$$\sum_{\mathbf{x}_{s^c}} p_\theta([\mathbf{x}_s, \mathbf{x}_{s^c}]) = p_\theta(\mathbf{x}_s) \quad \forall \mathbf{x}_s \in \{1, \ldots, K, ?\}^D$$

where $[\mathbf{x}_s, \mathbf{x}_{s^c}]$ denotes the concatenation of $\mathbf{x}_s$ and $\mathbf{x}_{s^c}$. Given a random ordering of the variables $\sigma \in S_D$ where S_D defines the set of all permutations of $1, 2, \cdots, D$, the marginalization can be imposed over one variable at a time, which leads to the following one-step marginalization constraints:

$$p_\theta(\mathbf{x}_{\sigma(<d)}) = \sum_{x_{\sigma(d)}} p_\theta(\mathbf{x}_{\sigma(\leq d)}), \quad \forall \sigma \in S_D, \mathbf{x} \in \{1, \cdots, K\}^D, d \in [1 : D]\,. \tag{4.2}$$

[1] An alternative is to consider minimizing distance over some marginal distribution of interest if we only cares about a specific marginal. Note this is impractical under the distribution matching setting, when the true marginal $p(\mathbf{x}_s)$ is intractable to evaluate in general.

[1] To make sure p_θ is normalized, we can either additionally enforce $p_\theta([?? \cdots ?]) = 1$ or let $Z_\theta = p_\theta([?? \cdots ?])$ be the normalization constant.

4.2.4 Sampling

Given the learned marginalization model, one can sample from the learned distribution by picking an arbitrary order σ and sampling one variable at a time. To evaluate the conditionals at each step of the generation, we can use the product rule of probability:

$$p_\theta(x_{\sigma(d)}|\mathbf{x}_{\sigma(<d)}) = \frac{p_\theta([\mathbf{x}_{\sigma(<d)}, x_{\sigma(d)}])}{p_\theta(\mathbf{x}_{\sigma(<d)})}\,.$$

However, the above is not a valid conditional if the marginalization constraint is not strictly enforced, since it might not sum up exactly to one. Hence we use following normalized conditional:

$$p_\theta(x_{\sigma(d)}|\mathbf{x}_{\sigma(<d)}) = \frac{p_\theta([\mathbf{x}_{\sigma(<d)}, x_{\sigma(d)}])}{\sum_{x_{\sigma(d)}} p_\theta([\mathbf{x}_{\sigma(<d)}, x_{\sigma(d)}])}\,. \tag{4.3}$$

In this paper, we focus on the sampling procedure that generates one variable at a time, but marginalization models can also facilitate sampling multiple variables at a time in a similar fashion.

4.2.5 Learning marginals jointly with conditionals

In training, we impose the marginalization self-consistency by minimizing the squared error of the constraints in (4.2). When the number of discrete values each x_d can take becomes large, each marginalization constraint in (4.2) requires K NN forward passes, which makes training challenging to scale when K is large. To address this issue, we augment the marginalization models with conditionals parameterized by ϕ. The marginalization constraints in (4.2) can be broken into K parallel marginalization constraints based on conditionals and marginals that allows subsampling the constraints during training:

$$p_\theta(\mathbf{x}_{\sigma(<d)})p_\phi(\mathbf{x}_{\sigma(d)}|\mathbf{x}_{\sigma(<d)}) = p_\theta(\mathbf{x}_{\sigma(\leq d)}), \quad \forall \sigma \in S_D, \mathbf{x} \in \{1, \cdots, K\}^D, d \in [1:D]. \tag{4.4}$$

During training, we need to specify a distribution $q(\mathbf{x})$ for subsampling the marginalization constraints to optimize on. In practice, it can be set to the distribution we are interested to perform marginal inference on, such as p_{data} or the distribution of the generative model $p_{\theta,\phi}$.

4.3 Training the Marginalization Models

4.3.1 Maximum likelihood

In this setting, we train MaM with the maximum likelihood objective in Equation (1.6) while additionally enforcing the marginalization constraints in Equation (4.2):

$$\begin{aligned} \max_{\theta,\phi} \quad & \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log p_\theta(\mathbf{x}) \\ \text{s.t.} \quad & p_\theta(\mathbf{x}_{\sigma(<d)}) p_\phi(\mathbf{x}_{\sigma(d)}|\mathbf{x}_{\sigma(<d)}) = p_\theta(\mathbf{x}_{\sigma(\leq d)}),\ \forall \sigma \in S_D, \mathbf{x} \in \{1, \cdots, K\}^D, d \in [1:D]. \end{aligned} \tag{4.5}$$

Two-stage training

A typical way to solve the above optimization problem is to convert the constraints into a penalty term and optimize the penalized objective, but we empirically found the learning to be slow and unstable. Instead, we identify an alternative two-stage optimization formulation that is theoretically equivalent to Equation (4.5), but leads to more efficient training:

Claim 4.1. *Solving the optimization problem in Equation (4.5) is equivalent to the following two-stage optimization procedure, under mild assumption about the neural networks used being universal approximators:*

$$\textit{Stage 1:} \quad \max_{\phi} \ \mathbb{E}_{\mathbf{x}\sim p_{data}} \mathbb{E}_{\sigma\sim\mathcal{U}(S_D)} \sum_{d=1}^{D} \log p_\phi \left(x_{\sigma(d)} \mid \mathbf{x}_{\sigma(<d)} \right)$$

$$\textit{Stage 2:} \quad \min_{\theta} \ \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})} \mathbb{E}_{\sigma\sim\mathcal{U}(S_D)} \mathbb{E}_{d\sim\mathcal{U}(1,\cdots,D)} \left(\log[p_\theta(\mathbf{x}_{\sigma(<d)}) p_\phi(\mathbf{x}_{\sigma(d)}|\mathbf{x}_{\sigma(<d)})] - \log p_\theta(\mathbf{x}_{\sigma(\leq d)}) \right)^2,$$

The intuition is that the chain rule of probability implies that there is a one-to-one correspondence between optimal conditionals and marginals: $\log p_\theta(\mathbf{x}) = \sum_{d=1}^{D} \log p_\phi \left(x_{\sigma(d)}|\mathbf{x}_{\sigma(<d)} \right)$ for any σ and $\mathbf{x}$. By assuming neural networks are universal approximators, we can first optimize for the optimal conditionals, and then optimize for the corresponding optimal marginals. This is equivalent to first fitting an order-agnostic autoregressive model (Uria et al., 2014; Hoogeboom et al., 2021a) and then distilling that model into the marginalization network. We provide more details in Section 6.3.1.

The procedure to train MaM for maximum likelihood is presented in Algorithm 2.

4.3.2 Distribution matching

In this setting, we train MaM using the distribution matching objective in Equation (1.7) with a penalty term to enforce the marginalization constraints in Equation (4.2):

$$\min_{\theta,\phi} \quad D_{\mathrm{KL}}\left(p_\theta\left(\mathbf{x}\right)||p\left(\mathbf{x}\right)\right) \;+\; \lambda\, \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\mathbb{E}_\sigma\, \mathbb{E}_d \;\left(\log\left[p_\theta\left(\mathbf{x}_{\sigma(<d)}\right) p_\phi\left(\mathbf{x}_{\sigma(d)}|\mathbf{x}_{\sigma(<d)}\right)\right] - \log p_\theta\left(\mathbf{x}_{\sigma(\leq d)}\right)\right)^2,$$

where $\sigma \sim \mathcal{U}(S_D)$, $d \sim \mathcal{U}(1, \cdots, D)$ and $q(\mathbf{x})$ is the distribution of interest for evaluating marginals. The procedure to train MaM for distribution matching is presented in Algorithm 3. In the following, we discuss the key components that enables scalable training.

Scalable training

We use REINFORCE to estimate the gradient of the KL divergence term:

$$\begin{aligned}\nabla_\theta D_{\mathrm{KL}}(p_\theta(\mathbf{x})||p(\mathbf{x})) &= \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\left[\nabla_\theta \log p_\theta\left(\mathbf{x}\right)\left(\log p_\theta\left(\mathbf{x}\right) - \log f\left(\mathbf{x}\right)\right)\right] \\ &\approx \frac{1}{N}\sum_{i=1}^N \nabla_\theta \log p_\theta(\mathbf{x}^{(i)})\left(\log p_\theta(\mathbf{x}^{(i)}) - \log f(\mathbf{x}^{(i)})\right)\end{aligned} \tag{4.6}$$

For the penalty term, we subsample the ordering σ and step d for each data $\mathbf{x}$.

Persistent MCMC sampling

We need cheap and effective samples from p_θ in order to perform REINFORCE, so a persistent set of Markov chains are maintained by randomly picking an ordering and taking block Gibbs sampling steps using the conditional distribution $p_\phi(\mathbf{x}_{\sigma(d)}|\mathbf{x}_{\sigma(<d)})$, in similar fashion to persistent contrastive divergence used in training RBMs (Tieleman, 2008). The samples from the conditional distribution p_ϕ serve as approximate samples from p_θ when they are close to each other. Otherwise, we can additionally use importance sampling for adjustment.

4.3.3 Addressing limitations of ARMs

We discuss in more detail about how MaMs address some limitations of ARMs. The first one is general to both settings, while the latter two are specific to distribution matching.

Non-Scalable evaluation of likelihoods

Due to sequential conditional modeling, evaluation of any arbitrary marginal $p(\mathbf{x}_o)$ with ARMs requires applying the NN up to D times, which is inefficient in time and memory for high-dimensional

Algorithm 2 MaM Maximum Likelihood Estimation Training

Input: Data $\mathcal{D}_{\text{train}}$, $q(\mathbf{x})$, network θ and ϕ
Stage 1: Optimize ϕ with the lower bound objective used in AO-ARM
for i in $\{1, \cdots, N_1\}$ **do**
  minibatch $\mathbf{x} \sim \mathcal{D}_{\text{train}}$
  Sample $\sigma \sim \mathcal{U}(S_D)$, $d \sim \mathcal{U}(1, \cdots, D)$
  $\mathcal{L} \leftarrow \sum_{j \in \sigma(\geq d)} \log p_\phi \left(x_j | \mathbf{x}_{\sigma(<d)}\right)$ ▷ Parallel evaluation of one-step conditional objectives
  $\mathcal{L} \leftarrow \frac{D}{D-d+1} \mathcal{L}$
  Update ϕ with gradient of $\mathcal{L}$
end for
Stage 2: Optimize θ to distill the marginals from optimized conditionals ϕ
for i in $\{1, \cdots, N_2\}$ **do**
  minibatch $\mathbf{x} \sim q(\mathbf{x})$
  Sample $\sigma \sim \mathcal{U}(S_D)$, $d \sim \mathcal{U}(1, \cdots, D)$
  $\mathcal{L} \leftarrow \left(\log[p_\theta(\mathbf{x}_{\sigma(<d)}) p_\phi(\mathbf{x}_{\sigma(d)} | \mathbf{x}_{\sigma(<d)})] - \log p_\theta(\mathbf{x}_{\sigma(\leq d)})\right)^2$
    ▷ Squared logrithmic error of marginalization constraint
  Update θ with gradient of $\mathcal{L}$
end for

Algorithm 3 MaM Distribution Matching Training

Input: $q(\mathbf{x})$, network θ and ϕ, Gibbs sampling block size M
Joint training of ϕ and θ:
for j in $\{1, \cdots, N\}$ **do**
  Sample $\sigma \sim \mathcal{U}(S_D)$
  Sample $\mathbf{x}' \sim p_\phi(\mathbf{x}_{\sigma(\leq M)} | \mathbf{x}_{\sigma(>M)})$ ▷ Persistent block Gibbs sampling
  Sample $\tilde{\mathbf{x}} \sim q(\mathbf{x})$
  Sample $\tilde{d} \sim \mathcal{U}(1, \cdots, D)$, $\tilde{\sigma} \sim \mathcal{U}(S_D)$
  $\mathcal{L}_{\text{penalty}} \leftarrow \left(\log[p_\theta(\tilde{\mathbf{x}}_{\tilde{\sigma}(<\tilde{d})}) p_\phi(\tilde{\mathbf{x}}_{\tilde{\sigma}(\tilde{d})} | \tilde{\mathbf{x}}_{\tilde{\sigma}(<\tilde{d})})] - \log p_\theta(\tilde{\mathbf{x}}_{\tilde{\sigma}(\leq \tilde{d})})\right)^2$
    ▷ Squared logrithmic error of marginalization constraint
  $\nabla_{\theta,\phi} D_{\text{KL}} \leftarrow$ REINFORCE gradient estimate with $\mathbf{x}'$
  $\nabla_{\theta,\phi} \leftarrow \nabla_{\theta,\phi} D_{\text{KL}} + \lambda \nabla_{\theta,\phi} \mathcal{L}_{\text{penalty}}$
  Update θ and ϕ with gradient $\nabla_{\theta,\phi}$
  $\mathbf{x} \leftarrow \mathbf{x}'$
end for

data. In comparison, MaMs are able to estimate any arbitrary marginal with one NN forward pass.

Lack of support for training with any-order

In distribution matching, the objectives in Equations (1.7) and (1.8) minimize the distance between $\log p_\phi(\mathbf{x})$ and $\log p(x)$. However, unless the model is perfectly self-consistent, it will not be the case that $\log p_\phi(\mathbf{x}) = \mathbb{E}_\sigma \log p_\phi(\mathbf{x}|\sigma)$. In particular, we would not expect it to work to have an objective that takes an expectation over orderings σ and uses $\log p_\phi(\mathbf{x}|\sigma)$ to match the target $\log p(\mathbf{x})$, i.e., $\mathbb{E}_\sigma \mathbb{E}_{p_\phi(\cdot|\sigma)} d(p_\phi(\cdot|\sigma), p) \neq \mathbb{E}_{p_\phi} d(p_\phi, p)$. The MaM self-consistency constraint addresses this issue, while ARMs need to be trained with a preset order to minimize the distance between $\log p_\phi(\mathbf{x}|\sigma)$ and the target density $\log p(\mathbf{x})$. MaMs are not limited to fixed ordering because marginals are order-agnostic and we can optimize over expectation of orderings for the marginalization self-consistency constraints.

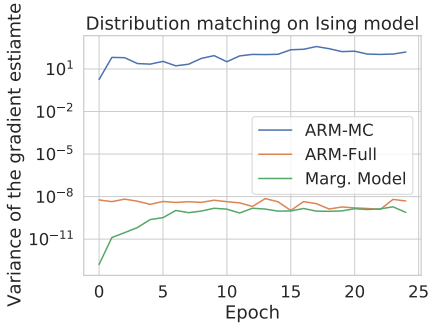


Figure 4.3: Approximating $\log p_\phi(\mathbf{x})$ with one-step conditional (ARM-MC) results in extremely high gradient variance during DM training.

Non-Scalable training

When minimizing the difference between $\log p_\phi(\mathbf{x}|\sigma)$ and the target $\log p(\mathbf{x})$, ARMs need to sum conditionals to evaluate $\log p_\phi(\mathbf{x}|\sigma)$. One might consider subsampling one-step conditionals $p_\phi(x_{\sigma(d)}|\mathbf{x}_{\sigma(<d)})$ to estimate $p_\phi(\mathbf{x})$, but this leads to high variance of the REINFORCE gradient in Equation (4.6) due to the product of the score function and distance terms, which are both high variance (We validate this in experiments, see Figure 4.3). Consequently, training ARMs for distribution matching necessitates a sequence of D conditional evaluations to compute the gradient of the objective function. This constraint leads to an effective batch size of $B \times D$, significantly limiting the scalability of ARMs to high-dimensional problems. Furthermore, obtaining Monte Carlo samples from ARMs for the REINFORCE gradient estimator is slow when the dimension is high. Due to the fixed input ordering, this process requires D sequential sampling steps, making more cost-effective sampling approaches

like persistent MCMC infeasible.

Marginalization models circumvent this challenge by directly estimating the log-likelihood with the marginal neural network. Additionally, the support for any-order training enables efficient sampling through the utilization of persistent MCMC methods.

4.4 Related Work

4.4.1 Autoregressive models

Autoregressive models (ARMs) decompose a joint probability distribution into a sequence of conditional probabilities (Bengio and Bengio, 2000; Larochelle and Murray, 2011). Recent developments in deep learning have greatly advanced the performance of ARMs across different modalities, including images, audio, and text. Any-order (Order-agnostic) ARMs were first introduced in (Uria et al., 2014) by training with the any-order lower-bound objective for the maximum likelihood setting and recently seen in ARDM (Hoogeboom et al., 2021a) with state-of-the-art performance for any-order discrete modeling of image/text/audio.

4.4.2 Discrete diffusion models

Discrete diffusion models learn to denoise from a latent base distribution into the data distribution. Sohl-Dickstein et al. (2015) first proposed diffusion for binary data and was extended in Hoogeboom et al. (2021b) for categorical data and both works adds uniform noise in the diffusion process. A wider range of transition distributions was proposed in Austin et al. (2021) and insert-and-delete diffusion processes have been explored in Johnson et al. (2021). Hoogeboom et al. (2021a) explored the connection between ARMs and diffusion model with absorbing diffusion and showed that OA-ARDMs are equivalent to absorbing diffusion models in infinite time limit, but achieves better performance with a smaller number of steps.

4.4.3 Discrete normalizing flow

Normalizing flows transform a latent base distribution into the data distribution by applying a sequence of invertible transformations (Rippel and Adams, 2013; Tabak and Turner, 2013; Dinh et al., 2014; Sohl-Dickstein et al., 2015; Rezende and Mohamed, 2015; Dinh et al., 2016; Kingma et al., 2016; Papamakarios et al., 2017). They have been extended to discrete data (Tran et al., 2019; Hoogeboom et al., 2019) with carefully designed discrete variable transformations. Their performance

is competitive on character-level text modeling, but they do not allow any-order modeling and could be limited to discrete data with small number of categories due to the use of a straight-through gradient estimators.

4.4.4 GFlowNets

GFlowNets (Bengio et al., 2021a;b) formulate the problem of generation as matching the probability flow at terminal states to the target normalized density. Zhang et al. (2022) proposes to apply GFlowNets to discrete data and additionally train an energy function from data. GFlowNets assume certain generation paths through a DAG, which limits its flexibility in sampling/generation. Compared to autoregressive models, this approach does not scale as well during training and its exact likelihood evaluation is intractable. We note that GFlowNet training also enforces a local detailed balance condition that is similar in spirit to the marginalization self-consistency presented here.

4.4.5 Probabilistic circuits

Probabilistic circuits were recently proposed as a framework for tractable probabilistic models that unify tractable probabilistic models, including Chow-Liu trees (Chow and Liu, 1968), arithmetic circuits (Darwiche, 2003), sum-product networks (Poon and Domingos, 2011), etc. This type of model uses circuits that follow smoothness and decomposibilty structural properties such that probability densities/masses are tractable. The expressiveness of the model is limited by the allowed circuit structures.

4.5 Experiments

We conduct experiments with marginalization models (MaM) on both MLE and DM settings for discrete problems including binary images, text, molecules and phyiscal systems. We consider the following baselines for comparison: Any-order ARM (AO-ARM) (Hoogeboom et al., 2021a), ARM (Larochelle and Murray, 2011), GFlowNet (Bengio et al., 2021a; Zhang et al., 2022), and Discrete Flow (Tran et al., 2019)[2]. Only MaM and (AO-)ARM supports marginal inference, so ARM will be the major focus of comparison. Discrete flow allows exact likelihood evaluation while GFlowNet needs to approximate the likelihood with sum using importance samples. For evaluating AO-ARM, we can either use it as an ensemble model (AO-ARM-E) by averaging the likelihood estimate from

[2]Results are only reported on text8 for discrete flow as there is no public code implementation.

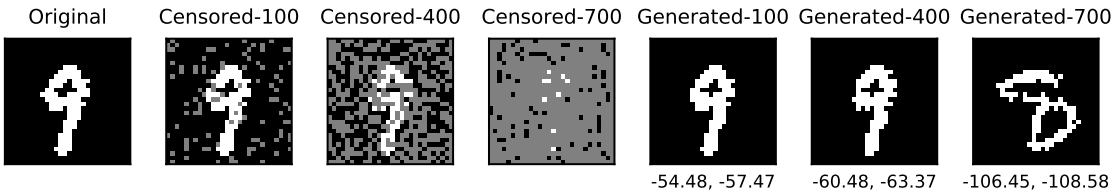


Figure 4.4: An example of the data generated (with 100/400/700 pixels masked) for comparing the quality of likelihood estimate. Numbers below the images are LL estimates from MaM's marginal network (left) and AO-ARM-E's ensemble estimate (right).

several random orderings or as a single model (AO-ARM-S) that picks a single ordering at random. Neural network architecture and training hyperparameter details can be found in Section 6.3.2.

4.5.1 Maximum likelihood estimation

Binary MNIST

We report the negative test likelihood (bits/digit), likelihood estimate quality and likelihood inference time per minibatch (of size 16) in Section 4.5.1. To keep GPU memory usage the same, we sequentially evaluate the likelihood for ARMs. Both MaM and AO-ARM use a U-Net architecture with 4 ResNet Blocks interleaved with attention layers (see Section 6.3.2). GFlowNets fail to scale to large architectures as U-Net, hence we report GFlowNet results using an MLP from Zhang et al. (2022). For MaM, we use the conditional network to evaluate the likelihood on test data, since marginals are not strictly valid but only approximations. The marginal network is used for evaluating likelihood quality and inference time.

In order to evaluate the quality of marginal likelihood estimates, we employ a controlled experiment where we randomly mask out portions of a test image and generate multiple samples with varying levels of masking (refer to Figure 4.4). This process allows us to obtain a set of distinct yet comparable samples, each associated with a different likelihood value. For each model, we evaluate the likelihood of the generated samples and compare that with AO-ARM-Ensemble's estimate since it achieves the best likelihood on test data. We report the Spearman's correlation and Pearson correlation of the likelihood estimates from the given model against that from AO-ARM-Ensemble. MaM achieves close to 4 order of magnitude speed-up while only at slightly worse quality. Additional results on generated samples, marginal likelihood on partial images are available in Section 6.3.2.

Model	NLL (bpd) ↓	Spearman's ↑	Pearson ↑	LL inference time (s) ↓
AO-ARM-E-U-Net	**0.148**	1.0	1.0	661.98 ± 0.49
AO-ARM-S-U-Net	0.149	**0.996**	**0.993**	132.40 ± 0.03
MaM-U-Net	0.149	0.992	**0.993**	**0.018 ± 0.00**
GflowNet-MLP	0.189	–	–	–

Table 4.1: Performance Comparison on Binary-MNIST

Model	NLL (bpd) ↓	Spearman's ↑	Pearson ↑	LL inf. time (s) ↓
AO-ARM-E-Transfomer	**0.652**	1.0	1.0	96.87± 0.04
AO-ARM-S-Transfomer	0.655	0.996	0.994	19.32± 0.01
MaM-Transfomer	0.655	**0.998**	**0.995**	**0.006±0.00**

Table 4.2: Performance Comparison on Molecular Sets

Molecular Sets

We test generative modeling of MaM on a benchmarking molecular dataset (Polykovskiy et al., 2020) refined from the ZINC database (Sterling and Irwin, 2015). Same metrics are reported as Binary-MNIST. Likelihood quality is measured similarly but on random groups of test molecules instead of generated ones. The generated molecules from MaM and AO-ARM are comparable to standard state-of-the-art molecular generative models, such as CharRNN (Segler et al., 2018), JTN-VAE (Jin et al., 2018), and LatentGAN (Prykhodko et al., 2019) (see Section 6.3.2), with additional controllability and flexibility in any-order generation. MaM supports much faster marginal inference, which is useful for domain scientists to reason about likelihood of substructures. Generated molecules and histogram plots of their properties are available in Section 6.3.2.

text8

Text8 (Mahoney, 2011) is a character level language modeling task with 100M characters split into chunks of 250 character. We follow the same procedure as Binary-MNIST and report the same metrics. The results of discrete flow is from Tran et al. (2019) (for which unfortunately no open-source implementations exist to measure other metrics).

4.5.2 Distribution matching training

We compare with ARM that uses sum of conditionals to evaluate $\log p_\phi$ with fixed forward ordering and ARM-MC that uses a one-step conditional to estimate $\log p_\phi$. ARM can be regarded as the golden standard of learning autoregressive conditionals, since its gradient is the most informative. MaM uses marginal network to evaluate $\log p_\theta$ and subsamples a one-step marginalization constraint for each data

Model	NLL (bpc) ↓	Spearman's ↑	Pearson ↑	LL inf. time (s) ↓
Discrete Flow (8 flows)	1.23	–	–	–
AO-ARM-E-Transfomer	1.494	1.0	1.0	207.60 ± 0.33
AO-ARM-S-Transfomer	1.529	0.982	0.987	41.40 ± 0.01
MaM-Transfomer	1.529	0.937	0.945	0.005 ± 0.000

Table 4.3: Performance Comparison on text8

point in the batch. The effective batch size for ARM and GFlowNet is $B \times \mathcal{O}(D)$ for batch of size B, and $B \times \mathcal{O}(1)$ for ARM-MC and MaM . MaM and ARM optimizes KL divergence using REINFORCE gradient estimator with baseline. GFlowNet minimizes squared distance following (Zhang et al., 2022).

Ising model

Ising models (Ising, 1925) model interacting spins and are widely studied in mathematics and physics (see MacKay (2003)). We study Ising model on a square lattice. The spins of the D sites are represented a D-dimensional binary vector and its distribution is $p^*(\mathbf{x}) \propto f^*(\mathbf{x}) = \exp\left(-\mathcal{E}_J(\mathbf{x})\right)$ where $\mathcal{E}_{\mathbf{J}}(\mathbf{x}) \triangleq -\mathbf{x}^\top \mathbf{J}\mathbf{x} - \boldsymbol{\theta}^\top \mathbf{x}$, with $\mathbf{J}$ being the binary adjacency matrix. These models, although simplistic, serve as bear analogies to the complex behavior of high-entropy alloys (Damewood et al., 2022). A quantum extentions of the Ising model, transverse-field Ising model, is extensively studied in quantum physical systems (Kogut, 1979; Sachdev, 1999; Amico et al., 2008).

We compare MaM with ARM, ARM-MC, and GFlowNet on a 10×10 ($D = 100$) and a larger 30×30 ($D = 900$) Ising model where ARMs and GFlowNets fail to scale. 2000 ground truth samples are generated following Grathwohl et al. (2021) and we measure test negative log-likelihood on those samples. We also measure $D_{\mathrm{KL}}(p_\theta(\mathbf{x}) || p^*)$ by sampling from the learned model and evaluating $\sum_{i=1}^{M} (\log p_\theta(\mathbf{x}_i) - \log f^*(\mathbf{x}_i))$. Figure 4.5 contains KDE plots of $-\mathcal{E}_{\mathbf{J}}(\mathbf{x})$ for the generated samples.

As described in Section 4.3.3, the ARM-MC gradient suffers from high variance and fails to converge. It also tends to collapse and converge to a single sample. MaM-DM has significant speedup in inference time and is the only model that supports any-order generative modeling. The performance in terms of KL divergence and likelihood are slightly worse than models with fixed/learned order, which is expected since any-order modeling is harder than fixed-order modeling, and MaM-DM is solving a more complicated task of jointly learning conditionals and marginals. On a 30×30 ($D = 900$) Ising model, MaM-DM achieves a bpd of 0.835 on ground-truth samples while ARM and GFlowNet fails to scale. Distribution of generated samples is shown in Figure 4.5.

Model	NLL (bpd) ↓	KL divergence ↓	Inference time (s) ↓
ARM-Forward-Order-MLP	0.79	**-78.63**	5.29±0.07e-01
ARM-MC-Forward-Order-MLP	24.84	-18.01	5.30±0.07e-01
MaM-Any-Order-MLP	0.80	-77.77	**3.75±0.08e-04**
GFlowNet-Learned-Order-MLP	**0.78**	-78.17	–

Table 4.4: Performance Comparison on Ising model (10×10)

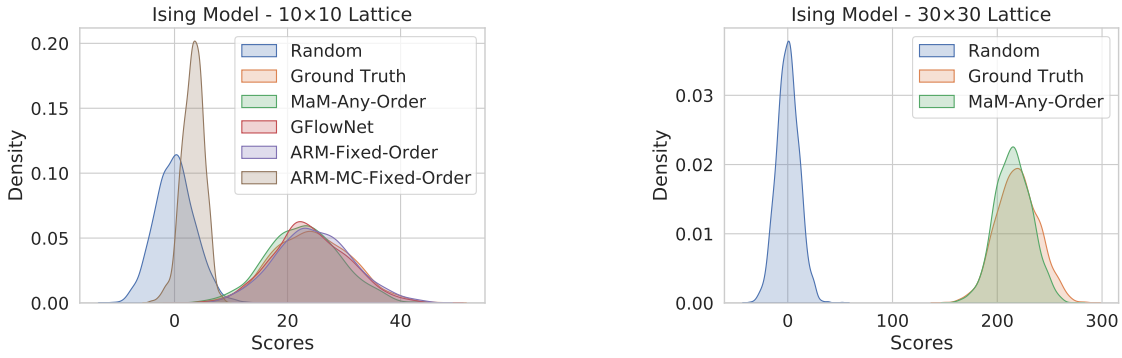


Figure 4.5: Ising model: 2000 samples are generated for each method.

Molecular generation with target property

In this task, we are interested in training generative models towards a specific target property of interest $g(x)$, such as lipophilicity (logP), synthetic accessibility (SA) etc. We define the distribution of molecules to follow $p^*(x) \propto \exp(-(g(x) - g^*)^2/\tau)$, where g^* is the target value of the property and τ is a temperature parameter.

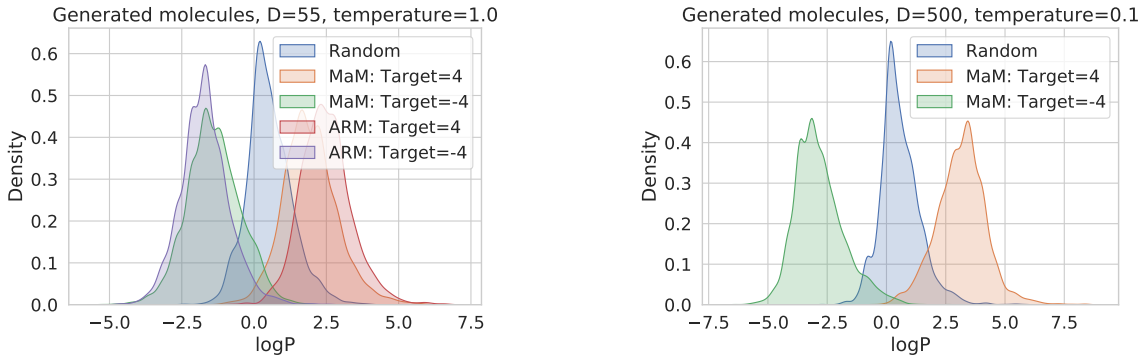


Figure 4.6: Target property matching: 2000 samples are generated for each method.

We train ARM and MaM for lipophilicity of target values 4.0 and -4.0, both with $\tau = 1.0$ and $\tau = 0.1$. Both models are trained for 4000 iterations with batch size 512. Results are shown in Figure 4.6 and Section 4.5.2.Additional Figures 6.26 and 6.27 are presented in Section 6.3.2. Findings

are consistent with the Ising model experiments. There is a small gap in the performance of MaM against ARM, but MaM supports any-order modeling and scales to problems with much larger dimension.

Model	**KL divergence** $\downarrow$			
Distribution	$\log P = 4, \tau = 1.0$	$\log P = -4, \tau = 1.0$	$\log P = 4, \tau = 0.1$	$\log P = 4, \tau = 0.1$
ARM-FO-MLP	-174.25	-168.62	-167.83	-160.2
MaM-AO-MLP	-173.07	-166.43	-165.75	-157.59

Table 4.5: Performance Comparison on Target Lipophilicity

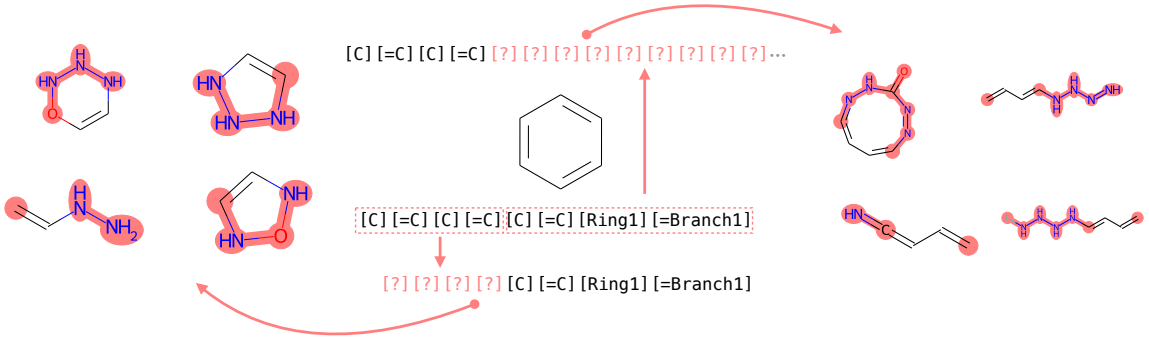


Figure 4.7: Conditionally generate towards low lipophilicity from user-defined substructures of Benzene. Left: Masking out the left 4 SELFIES characters. Right: masking the right 4-20 characters.

4.6 Conclusion

In conclusion, marginalization models are a novel family of generative models for high-dimensional discrete data that offer scalable and flexible generative modeling with tractable likelihoods. These models explicitly model all induced marginal distributions, allowing for fast evaluation of arbitrary marginal probabilities with a single forward pass of the neural network. Marginalization models also support scalable training for any-order generative modeling, which previous methods struggle to achieve under the setting of distribution matching. Potential future work includes designing new neural network architectures that automatically satisfy the marginalization self-consistency.

Chapter 5

Conclusion and Future Works

In this chapter, I briefly discuss some promising future directions.

5.1 Bayesian Optimization With Sparsity Constraints

In Chapter 2, we have shown that the proposed SEBO method can automatically identify the Pareto frontier trade-off between sparsity and system performance. However, in some contexts, decision-makers may have a desired sparsity level in mind. Further work is required to develop adaptive algorithms that can efficiently optimize system performance given a specific sparsity budget. Mathematically, this problem can be formulated as a constrained optimization problem:

$$\begin{aligned} \max_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \|\mathbf{x}\|_0 \leq s \end{aligned}$$

where f is an unknown black-box function and s is the desired sparsity level.

For the method to be successful, the major challenge will be how to take account of the sparsity constraint in the design of the acquisition function. On one hand, there is no final utility for a solution that violates the sparsity constraint. A natural way to extend SEBO for this setting would be to treat all solutions that violate the sparsity constraint to have the same sparsity objective, i.e.

$$\xi(\mathbf{x}) = \begin{cases} s+1 & \text{if } \|\mathbf{x}\|_0 > s \\ \|\mathbf{x}\|_0 & \text{otherwise} \end{cases}$$

However, this formulation only considers the utility of dense solutions in terms of achieving better system performance but neglects their "hidden utility" in providing more information that helps improve the quality of the GP surrogate model. Therefore it would be interesting to explore methods that take account of the "hidden utility" of dense solutions in the design of the acquisition function.

Another aspect of sparse system designs is to consider the varying deployment cost of designs with different sparsity levels. Prior works (Snoek et al., 2012; Lee et al., 2020) have studied how to design proper acquisition functions that take account of the deployment cost. But it remains open to design methods that optimize for sparse solutions while taking account of the deployment cost.

5.2 Equivariance in Discrete Generative Modeling

Many problems in physical sciences comprise a large amount of symmetries. For example, molecules, crystals, and Ising models are invariant with respect to global rotations or permutations of identical elements. In the context of generative modeling, we would like to have models that are equivariant to these symmetries. Recently, equivariant generative models have been developed for the continuous setting, such as equivariant normalizing flows (Köhler et al., 2020), flow matching (Klein et al., 2023) and diffusion models (Hoogeboom et al., 2022). Discrete generative models such as any-order autoregressive models and the proposed marginalization models will benefit from taking account of the symmetries to improve the sample efficiency and generalization.

Gebauer et al. (2019) has explored enforcing equivariance under translation and rotation using distance-based featurization in autoregressive models for molecule 3D point clouds generation. For discrete data, it remains an open question how to enforce equivariance for more general symmetries such as permutation and rotation in any-order autoregressive models and marginalization models.

5.3 Structured Generative Modeling

The major types of generative models—such as diffusion models and language models—do not assume any explicit latent structure but instead, generate directly in the data input space. But often latent structures exist in the data, such as the collective variables and coarse-graining in molecular dynamics. It would be interesting to explore how to incorporate these latent structures into the generative models for two reasons. First, the latent structures can improve the interpretability of the generative models for end-users such as domain scientists. For example, evaluating the likelihood of latent structures provide insights about the underlying dynamics. Also, the dependence on latent structures

allows the end-users to explicitly control the generation process.

5.4 Summary of Contributions

Probabilistic models have huge potential in accelerating scientific discovery and engineering design. This dissertation has developed the first Bayesian optimization method for interpretable system configuration optimization. I also developed amortized inference for Gaussian process, a probabilistic surrogate model that is widely used in scientific discovery. Finally, I developed a new class of generative models—marginalization models—that scales up any-order autoregressive modeling of discrete data for distribution matching.

Bibliography

Eugene L Allgower and Kurt Georg. *Numerical continuation methods: an introduction*, volume 13. Springer Science & Business Media, 2012. (page 27)

Luigi Amico, Rosario Fazio, Andreas Osterloh, and Vlatko Vedral. Entanglement in many-body systems. *Reviews of modern physics*, 80(2):517, 2008. (page 68)

Martin Arjovsky, Soumith Chintala, and Leon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017. (page 14)

Arthur Asuncion and David Newman. UCI machine learning repository, 2007. (pages 32, 50, 94, and 108)

Peter M Attia, Aditya Grover, Norman Jin, Kristen A Severson, Todor M Markov, Yang-Hung Liao, Michael H Chen, Bryan Cheong, Nicholas Perkins, Zi Yang, et al. Closed-loop optimization of fast-charging protocols for batteries with machine learning. *Nature*, 578(7795):397–402, 2020. (page 19)

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021. (page 64)

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Advances in Neural Information Processing Systems 33*, NeurIPS, pages 21524–21538, 2020. (page 94)

Hamsa Bastani and Mohsen Bayati. Online decision making with high-dimensional covariates. *Operations Research*, 68(1):276–294, 2020. (page 37)

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021a. (pages 16 and 65)

Samy Bengio and Yoshua Bengio. Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE Transactions on Neural Networks*, 11(3):550–557, 2000. (pages 14, 55, and 64)

Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Conf. Optimality in Artificial and Biological Neural Networks*, 1992. (page 47)

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. GFlowNet foundations. *arXiv preprint arXiv:2111.09266*, 2021b. (page 65)

James O. Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013. (page 12)

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022, 2003. (page 98)

Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013. (page 20)

Salomon Bochner. *Lectures on Fourier integrals*, volume 42. Princeton University Press, 1959. (page 41)

V. Joseph Bowman. On the relationship of the Tchebycheff norm and the efficient frontier of multiple-criteria objectives. In H. Thiriez and S. Zionts, editors, *Multiple Criteria Decision Making. Lecture Notes in Economics and Mathematical Systems (Operations Research), vol 130*, pages 76–86. Springer Berlin, Heidelberg, 1976. (page 96)

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. (pages 8, 15, and 17)

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015. (page 116)

Benjamin Burger, Phillip M Maffettone, Vladimir V Gusev, Catherine M Aitchison, Yang Bai, Xiaoyan Wang, Xiaobo Li, Ben M Alston, Buyi Li, Rob Clowes, et al. A mobile robotic chemist. *Nature*, 583(7815):237–241, 2020. (page 19)

David Burt, Carl Edward Rasmussen, and Mark Van Der Wilk. Rates of convergence for sparse variational gaussian process regression. In *International Conference on Machine Learning*, 2019. (page 12)

Roberto Calandra, André Seyfarth, Jan Peters, and Marc P. Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1):5–23, 2015. (page 19)

Ching-An Cheng and Byron Boots. Variational inference for Gaussian process models with linear complexity. In *Advances in Neural Information Processing Systems*, 2017. (page 12)

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. (page 45)

CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968. (page 65)

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys, pages 191–198, 2016. (page 20)

Lehel Csató and Manfred Opper. Sparse on-line Gaussian processes. *Neural computation*, 14(3): 641–668, 2002. (pages 12 and 39)

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. (page 115)

James Damewood, Daniel Schwalbe-Koda, and Rafael Gómez-Bombarelli. Sampling lattices in semi-grand canonical ensemble with autoregressive machine learning. *npj Computational Materials*, 8(1):61, 2022. (pages 16 and 68)

Adnan Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305, 2003. (page 65)

Indraneel Das and John Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. *Structural Optimization*, 14:63–69, 1997. (page 96)

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. In *Advances in Neural Information Processing Systems 33*, NeurIPS, pages 9851–9864, 2020. (page 23)

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement. In *Advances in Neural Information Processing Systems 34*, NeurIPS, pages 2187–2200, 2021. (page 23)

Derek Bingham. Optimization test functions and datasets. https://www.sfu.ca/~ssurjano/optimization.html, 2013. (pages 50 and 110)

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. (pages 15, 55, and 105)

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014. (pages 14 and 64)

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*, 2016. (pages 14 and 64)

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017. (page 19)

Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32, 2019. (page 15)

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016. (page 47)

David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014. (page 10)

David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302.4922*, 2013. (pages 9, 39, 40, and 42)

Harrison Edwards and Amos Storkey. Towards a neural statistician. *arXiv preprint arXiv:1606.02185*, 2016. (page 45)

Matthias Ehrgott. *Multicriteria Optimization*. Springer Berlin, Heidelberg, 2005. (page 23)

David Eriksson and Martin Jankowiak. High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence*, UAI, pages 493–503, 2021. (pages 22 and 29)

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. In *Advances in Neural Information Processing Systems 32*, NeurIPS, 2019. (page 102)

Theodoros Evgeniou, Tomaso Poggio, Massimiliano Pontil, and Alessandro Verri. Regularization and statistical learning theory for data analysis. *Computational Statistics & Data*, 38(4):421–432, 2002. (page 20)

Qing Feng, Benjamin Letham, Hongzi Mao, and Eytan Bakshy. High-dimensional contextual policy search with unknown context rewards using Bayesian optimization. In *Advances in Neural Information Processing Systems 33*, NeurIPS, pages 22032–22044, 2020. (pages 19 and 33)

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. (page 47)

Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for Bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2018. (page 47)

Maurizio Filippone and Mark Girolami. Pseudo-marginal Bayesian inference for Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2214–2226, 2014. (pages 12 and 39)

Maurizio Filippone, Mingjun Zhong, and Mark Girolami. A comparative evaluation of stochastic-based inference methods for Gaussian process models. *Machine Learning*, 93(1):93–114, 2013. (page 39)

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017. (page 47)

Daniel Flam-Shepherd, Kevin Zhu, and Alán Aspuru-Guzik. Language models can learn complex molecular distributions. *Nature Communications*, 13(1):3293, 2022. (page 17)

Reeves Fletcher and Colin M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964. (page 12)

Jacob Gardner, Geoff Pleiss, Kilian Q. Weinberger, David Bindel, and Andrew G. Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586, 2018. (pages 46 and 47)

Marta Garnelo, Dan Rosenbaum, Chris J. Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Eslami. Conditional neural processes. *arXiv preprint arXiv:1807.01613*, 2018a. (page 46)

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b. (page 46)

Niklas Gebauer, Michael Gastegger, and Kristof Schütt. Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. *Advances in neural information processing systems*, 32, 2019. (page 72)

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018. (page 19)

Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *Journal of machine learning research*, 12(64):2211–2268, 2011. (page 42)

Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch Bayesian optimization via local penalization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, AISTATS, pages 648–657, 2016. (page 22)

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27, 2014. (page 14)

Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E. Turner. Meta-learning probabilistic inference for prediction. *arXiv preprint arXiv:1805.09921*, 2018. (page 47)

GPy. GPy: A Gaussian process framework in python. http://github.com/SheffieldML/GPy, 2012. (page 47)

Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris Maddison. Oops I took a gradient: Scalable sampling for discrete distributions. In *International Conference on Machine Learning*, pages 3831–3841. PMLR, 2021. (pages 68 and 117)

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. In *International Conference on Machine Learning*, pages 1242–1250, 2014. (pages 39 and 46)

Tom Gunter, Michael A. Osborne, Roman Garnett, Philipp Hennig, and Stephen J. Roberts. Sampling for inference in probabilistic models with fast Bayesian quadrature. In *Advances in Neural Information Processing Systems*, pages 2789–2797, 2014. (page 53)

James Hensman, Nicolo Fusi, and Neil D. Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, page 282, 2013. (pages 12, 39, and 47)

José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015. (page 50)

José Miguel Hernández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems 27*, NIPS, pages 918–926, 2014. (page 13)

Magnus R. Hestenes et al. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952. (page 12)

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002. (page 14)

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. (pages 8, 14, and 15)

Emiel Hoogeboom, Jorn Peters, Rianne Van Den Berg, and Max Welling. Integer discrete flows and lossless compression. *Advances in Neural Information Processing Systems*, 32, 2019. (page 64)

Emiel Hoogeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*, 2021a. (pages 17, 56, 60, 64, 65, and 117)

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34:12454–12465, 2021b. (page 64)

Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pages 8867–8887. PMLR, 2022. (page 72)

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2): 251–257, 1991. (page 115)

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. (page 115)

Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. In *Advances in Neural Information Processing Systems 32*, NeurIPS, pages 7267–7275, 2019. (page 20)

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2078–2087. PMLR, 2018. (page 8)

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, LION, pages 507–523, 2011. (page 19)

Ernst Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift fur Physik*, 31(1):253–258, February 1925. doi: 10.1007/BF02980577. (page 68)

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR, 2018. (page 67)

Daniel D Johnson, Jacob Austin, Rianne van den Berg, and Daniel Tarlow. Beyond in-place corruption: Insertion and deletion in denoising probabilistic models. *arXiv preprint arXiv:2107.07675*, 2021. (page 64)

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998. (page 13)

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019. (page 14)

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019. (page 46)

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (pages 47 and 48)

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013. (pages 14, 39, and 46)

Diederik P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014. (page 46)

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, 29, 2016. (pages 14 and 64)

Leon Klein, Andreas Krämer, and Frank Noé. Equivariant flow matching. *arXiv preprint arXiv:2306.15030*, 2023. (page 72)

Joshua Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1): 50–66, 2006. (pages 23 and 97)

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, 2015. (page 47)

John B Kogut. An introduction to lattice gauge theory and spin systems. *Reviews of Modern Physics*, 51(4):659, 1979. (page 68)

Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: exact likelihood generative learning for symmetric densities. In *International conference on machine learning*, pages 5361–5370. PMLR, 2020. (pages 16 and 72)

Jonas Köhler, Michele Invernizzi, Pim De Haan, and Frank Noé. Rigid body flows for sampling molecular crystal structures. *arXiv preprint arXiv:2301.11355*, 2023. (page 16)

Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024, 2020. (pages 57 and 116)

Athindran Ramesh Kumar, Sulin Liu, Jaime F. Fisac, Ryan P. Adams, and Peter J. Ramadge. Probf: Learning probabilistic safety certificates with barrier functions. *arXiv preprint arXiv:2112.12210*, 2021. (page 18)

Malte Kuss and Carl E. Rasmussen. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 751–758, 2004. (page 38)

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37. JMLR Workshop and Conference Proceedings, 2011. (pages 14, 55, 56, 64, and 65)

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and Fujie Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006. (page 14)

Eric Hans Lee, Valerio Perrone, Cedric Archambeau, and Matthias Seeger. Cost-aware bayesian optimization. *arXiv preprint arXiv:2003.10870*, 2020. (page 72)

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753, 2019. (page 45)

Benjamin Letham, Brian Karrer, Guilherme Ottoni, and Eytan Bakshy. Constrained Bayesian optimization with noisy experiments. *Bayesian Analysis*, 14(2):495–519, 2019. (page 19)

Yucen Lily Li, Tim GJ Rudner, and Andrew Gordon Wilson. A study of bayesian neural network surrogates for bayesian optimization. *arXiv preprint arXiv:2305.20028*, 2023. (page 13)

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023. (page 17)

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. (page 12)

Sulin Liu, Xingyuan Sun, Peter J. Ramadge, and Ryan P. Adams. Task-agnostic amortized inference of gaussian process hyperparameters. *Advances in Neural Information Processing Systems*, 2020. (page 18)

Sulin Liu, Qing Feng, David Eriksson, Benjamin Letham, and Eytan Bakshy. Sparse bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, 2023a. (page 18)

Sulin Liu, Peter J. Ramadge, and Ryan P. Adams. Generative marginalization models. *Under Submission*, 2023b. (page 18)

Zhe Liu, Nicholas Rolston, Austin C Flick, Thomas W Colburn, Zekun Ren, Reinhold H Dauskardt, and Tonio Buonassisi. Machine learning with knowledge constraints for process optimization of open-air perovskite solar cell manufacturing. *Joule*, 6(4):834–849, 2022. (page 19)

Daniel J. Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. Automatic gait optimization with Gaussian process regression. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, IJCAI, pages 944–949, 2007. (page 19)

James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua Tenenbaum, and Zoubin Ghahramani. Automatic construction and natural-language description of nonparametric regression models. In *AAAI conference on artificial intelligence*, 2014. (pages 9, 39, and 40)

Christos Louizos, Xiahan Shi, Klamer Schutte, and Max Welling. The functional neural process. In *Advances in Neural Information Processing Systems*, pages 8743–8754, 2019. (page 46)

David J.C. MacKay. The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736, 1992. (page 12)

David JC MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. (page 68)

Ali Madani, Ben Krause, Eric R Greene, Subu Subramanian, Benjamin P Mohr, James M Holton, Jose Luis Olmos Jr, Caiming Xiong, Zachary Z Sun, Richard Socher, et al. Large language models

generate functional protein sequences across diverse families. *Nature Biotechnology*, pages 1–8, 2023. (page 17)

Matt Mahoney. Large text compression benchmark, 2011. (page 67)

R. Timothy Marler and Jasbir S. Arora. The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisciplinary Optimization*, 41:853–862, 2010. (page 26)

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013. (page 15)

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature. *arXiv preprint arXiv:2301.11305*, 2023. (page 55)

Jonas Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975. (page 50)

Iain Murray and Ryan P. Adams. Slice sampling covariance hyperparameters of latent Gaussian models. In *Advances in Neural Information Processing Systems*, pages 1732–1740, 2010. (pages 12 and 39)

Iain Murray and Matthew Graham. Pseudo-marginal slice sampling. In *Artificial Intelligence and Statistics*, pages 911–919, 2016. (page 12)

Radford M. Neal. Priors for infinite networks. Technical Report CRG-TR-94-1, Department of Computer Science, University of Toronto, 1994. (page 13)

Radford M. Neal. Regression and classification using Gaussian process priors. *Bayesian statistics 6*, pages 475–501, 1999. (pages 12 and 38)

Jiquan Ngiam, Zhenghao Chen, Pang W Koh, and Andrew Y Ng. Learning deep energy models. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1105–1112, 2011. (page 15)

Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457), 2019. (page 16)

Changyong Oh, Jakub M. Tomczak, Efstratios Gavves, and Max Welling. Combinatorial Bayesian optimization using the graph Cartesian product. In *Advances in Neural Information Processing Systems 32*, NeurIPS, pages 2914–2924, 2019. (page 23)

Anthony O'Hagan. Bayes–Hermite quadrature. *Journal of statistical planning and inference*, 29(3): 245–260, 1991. (page 53)

OpenAI. Chatgpt, 2023. URL https://openai.com. (page 16)

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022. (page 16)

Andrei Paleyes, Mark Pullin, Maren Mahsereci, Neil D. Lawrence, and Javier González. Emulation of physical processes with Emukit. In *Second Workshop on Machine Learning and the Physical Sciences, NeurIPS*, 2019. (page 53)

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in Neural Information Processing Systems*, 30, 2017. (page 64)

Chiwoo Park, David J. Borth, Nicholas S. Wilson, and Chad N. Hunter. Variable selection for Gaussian process regression through a sparse projection. *IISE Transactions*, 54(7):699–712, 2021. (page 23)

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019. (page 48)

Valerio Perrone, Rodolphe Jenatton, Matthias W Seeger, and Cédric Archambeau. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, pages 6845–6855, 2018. (page 47)

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in Pharmacology*, 11:565644, 2020. (pages 67 and 122)

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011. (page 65)

Oleksii Prykhodko, Simon Viet Johansson, Panagiotis-Christos Kotsias, Josep Arús-Pous, Esben Jannik Bjerrum, Ola Engkvist, and Hongming Chen. A de novo molecular generation method using latent vector based generative adversarial network. *Journal of Cheminformatics*, 11(1):1–13, 2019. (page 67)

Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005. (pages 12 and 39)

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. (page 14)

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. (page 15)

Carl Edward Rasmussen. *Evaluation of Gaussian processes and other methods for non-linear regression*. PhD thesis, University of Toronto Toronto, Canada, 1997. (page 38)

Carl Edward Rasmussen and Zoubin Ghahramani. Bayesian Monte Carlo. *Advances in Neural Information Processing Systems*, pages 505–512, 2003. (page 53)

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2006. (pages 9, 38, and 42)

Sami Remes, Markus Heinonen, and Samuel Kaski. Non-stationary spectral kernels. In *Advances in Neural Information Processing Systems*, pages 4642–4651, 2017. (page 40)

Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in neural information processing systems*, 32, 2019. (page 55)

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015. (pages 14 and 64)

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014. (page 39)

Oren Rippel and Ryan P. Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013. (pages 14 and 64)

Daniel Ritchie, Paul Horsfall, and Noah D Goodman. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016. (pages 39 and 46)

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. (pages 17 and 56)

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: fundamental principles and 10 grand challenges. *Statistics Surveys*, 16:1–85, 2022. (page 19)

Subir Sachdev. Quantum phase transitions. *Physics world*, 12(4):33, 1999. (page 68)

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, pages 872–879, 2008. (page 116)

Yves-Laurent Kom Samo and Stephen Roberts. Generalized spectral kernels. *arXiv preprint arXiv:1506.02236*, 2015. (page 40)

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016. (page 47)

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987. (page 47)

Arne Schneuing, Yuanqi Du, Charles Harris, Arian Jamasb, Ilia Igashov, Weitao Du, Tom Blundell, Pietro Lió, Carla Gomes, Max Welling, et al. Structure-based drug design with equivariant diffusion models. *arXiv preprint arXiv:2210.13695*, 2022. (pages 8 and 55)

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems 28*, NIPS, pages 2503–2511, 2015. (pages 20 and 31)

Matthias W. Seeger, Christopher K. I. Williams, and Neil D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Artificial Intelligence and Statistics*, 2003. (pages 12 and 39)

Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Central Science*, 4(1): 120–131, 2018. (pages 17 and 67)

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: a review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015a. (page 13)

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015b. (page 7)

Bobak Shahriari, Alexandre Bouchard-Côté, and Nando de Freitas. Unbounded Bayesian optimization via regularization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, AISTATS, pages 1168–1176, 2016. (page 22)

Jiaxin Shi, Michalis Titsias, and Andriy Mnih. Sparse orthogonal variational inference for Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, 2020. (page 12)

Benjamin J. Shields, Jason Stevens, Jun Li, Marvin Parasram, Farhan Damani, Jesus I. Martinez Alvarado, Jacob M. Janey, Ryan P. Adams, and Abigail G Doyle. Bayesian reaction optimization as a tool for chemical synthesis. *Nature*, 590(7844):89–96, 2021. (page 19)

Jung-Eun Shin, Adam J Riesselman, Aaron W Kollasch, Conor McMahon, Elana Simon, Chris Sander, Aashish Manglik, Andrew C Kruse, and Debora S Marks. Protein design and variant prediction using autoregressive generative models. *Nature Communications*, 12(1):2403, 2021. (page 17)

Bernard W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986. (page 42)

Alex J. Smola and Peter L. Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems*, pages 619–625, 2001. (pages 12 and 39)

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264, 2006. (pages 12 and 39)

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012. (pages 19, 25, 38, and 72)

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015. (pages 14 and 64)

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019. (pages 8, 14, and 15)

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020. (page 14)

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*, ICML, pages 1015–1022, 2010. (page 13)

Teague Sterling and John J Irwin. ZINC 15–ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015. (page 67)

Shengyang Sun, Guodong Zhang, Chaoqi Wang, Wenyuan Zeng, Jiaman Li, and Roger Grosse. Differentiable compositional kernel learning for Gaussian processes. In *International Conference on Machine Learning*, pages 4828–4837, 2018. (pages 9, 39, 41, and 50)

Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013. (pages 14 and 64)

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288, 1996. (page 20)

Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071, 2008. (pages 14 and 61)

Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009. (pages 12, 39, and 47)

Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. Discrete flows: Invertible generative models of discrete data. *Advances in Neural Information Processing Systems*, 32, 2019. (pages 64, 65, and 67)

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*, pages 3–26, 2021. (page 19)

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*, pages 467–475. PMLR, 2014. (pages 17, 56, 60, and 64)

Berk Ustun and Cynthia Rudin. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102(3):349–391, 2016. (page 19)

Jonathan Vandermause, Steven B Torrisi, Simon Batzner, Yu Xie, Lixin Sun, Alexie M Kolpak, and Boris Kozinsky. On-the-fly active learning of interpretable bayesian force fields for atomistic rare events. *npj Computational Materials*, 6(1):20, 2020. (page 7)

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017. (pages 17, 44, 56, 104, and 105)

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016. (page 47)

Jack Wang, Aaron Hertzmann, and David J Fleet. Gaussian process dynamical models. In *Advances in Neural Information Processing Systems*, pages 1441–1448, 2006. (page 38)

Jue Wang, Sidney Lisanza, David Juergens, Doug Tischer, Joseph L Watson, Karla M Castro, Robert Ragotte, Amijai Saragovi, Lukas F Milles, Minkyung Baek, et al. Scaffolding protein functional sites using deep learning. *Science*, 377(6604):387–394, 2022. (pages 8 and 55)

Ke Wang, Geoff Pleiss, Jacob Gardner, Stephen Tyree, Kilian Q. Weinberger, and Andrew G. Wilson. Exact Gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, pages 14622–14632, 2019. (page 46)

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning*, ICML, pages 3627–3635, 2017. (page 13)

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Freitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016. (page 102)

David Weininger, Arthur Weininger, and Joseph L Weininger. SMILES. 2. algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Computer Sciences*, 29(2): 97–101, 1989. (pages 57 and 116)

Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H. Chi, and Jennifer Gillenwater. Practical diversified recommendations on YouTube with determinantal point processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM, pages 2165–2173, 2018. (pages 20 and 31)

Christopher Williams and Carl Rasmussen. Gaussian processes for regression. *Advances in neural information processing systems*, 8, 1995. (page 13)

Christopher K.I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, pages 682–688, 2001. (pages 12 and 39)

Andrew G. Wilson and Ryan P. Adams. Gaussian process kernels for pattern discovery and extrapolation. In *International Conference on Machine Learning*, pages 1067–1075, 2013. (pages 9, 40, and 42)

Andrew G. Wilson, Elad Gilboa, Arye Nehorai, and John P Cunningham. Fast kernel learning for multidimensional pattern extrapolation. In *Advances in Neural Information Processing Systems*, pages 3626–3634, 2014. (page 42)

Andrew G. Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016. (page 9)

Dian Wu, Lei Wang, and Pan Zhang. Solving statistical mechanics using variational autoregressive networks. *Physical review letters*, 122(8):080602, 2019. (page 16)

Kaifeng Yang, Michael Emmerich, André Deutz, and Thomas Bäck. Multi-objective Bayesian global optimization using expected hypervolume improvement gradient. *Swarm and Evolutionary Computation*, 44:945–956, 2019a. (page 23)

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, 32, 2019b. (pages 15, 17, 55, and 56)

Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5485–5493, 2017. (page 55)

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B*, 68(1):49–67, 2006. (pages 20 and 33)

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017. (page 105)

Dinghuai Zhang, Nikolay Malkin, Zhen Liu, Alexandra Volokhova, Aaron Courville, and Yoshua Bengio. Generative flow networks for discrete probabilistic modeling. In *International Conference on Machine Learning*, pages 26412–26428. PMLR, 2022. (pages 16, 65, 66, and 68)

Tong Zhang. Multi-stage convex relaxation for learning with sparse regularization. In *Advances in Neural Information Processing Systems 21*, NIPS, 2008. (page 20)

Yunong Zhang and William E. Leithead. Exploiting Hessian matrix and trust-region algorithm in hyperparameters estimation of Gaussian process. *Applied Mathematics and Computation*, 171(2): 1264–1281, 2005. (page 38)

Chapter 6

Appendix

6.1 Appendix of Chapter 2

6.1.1 Code Implementations

The GPEI, SAASBO and EHVI used in SEBO were implemented using BoTorch (Balandat et al., 2020), a framework for BO in PyTorch and are available in Ax (https://github.com/facebook/Ax). The code is licensed under the MIT License. The SVM hyperparameter tuning experiment uses the SVM implementation in Sklearn and the CT slice dataset in the UCI machine learning repository (Asuncion and Newman, 2007). The Adaptive bitrate simulation experiment is available at https://github.com/facebookresearch/ContextualBO, licensed under the MIT License.

6.1.2 Proof of Theorem 2.1

Here we provide the proofs of Theorems 2.1 and 2.2, as well as an illustration of the result of Theorem 2.2.

Proof of Theorem 2.1. Suppose $\mathbf{x}^\dagger \in \arg\max \alpha_{\mathrm{ER}}(\mathbf{x}; \lambda)$ and $\xi(\mathbf{x}^\dagger) \leq \theta$. Then, $\alpha(\mathbf{x}^\dagger) = 0$, so $\alpha_{\mathrm{ER}}(\mathbf{x}^\dagger; \lambda) = -\lambda\xi(\mathbf{x}^\dagger)$.

By $\mathbf{x}^\dagger$ being a maximizer of α_{ER} we must have

$$-\lambda\xi(\mathbf{x}^\dagger) = \alpha_{\mathrm{ER}}(\mathbf{x}^\dagger; \lambda) \geq \alpha_{\mathrm{ER}}(\mathbf{x}^s; \lambda) = -\lambda\xi(\mathbf{x}^s).$$

Thus $\xi(\mathbf{x}^\dagger) \leq \xi(\mathbf{x}^s)$. Because $\mathbf{x}^s$ is a strict global minimum, we have then that $\mathbf{x}^\dagger = \mathbf{x}^s$. □

This setting where the acquisition value is 0 for all sparse points is easily encountered in practice when there is a trade-off between the objective function and sparsity, as in Figure 2.1, and we have sampled a point close to the (non-sparse) optimum. Consider the EI acquisition function with external regularization. Once the GP is confident that sparse points have worse objective value than non-sparse points, sparse points will have acquisition value approximately 0, as their improvement is being evaluated with respect to a non-sparse incumbent best $\mathbf{x}^*$.

6.1.3 Proof of Theorem 2.2

We assume ξ is continuous and bounded, which implies h is continuous and bounded:

Assumption 6.1. *ξ is continuous on $\mathcal{B}$, and has minimum value $\xi(\mathbf{x}^s) = s_l$ and maximum value s_u.*

Proposition 6.1. *h is continuous and bounded on the domain $[s_l, s_u]$.*

Sketch of Proof. This result falls from the continuity and boundedness of f, and by applying the intermediate value theorem to ξ. □

Proof of Theorem 2.2. Suppose h is strictly convex over the interval $[\theta_l, \theta_u]$. For the sake of contradiction, assume that there exists a $\theta_\dagger \in (\theta_l, \theta_u)$ and an $\mathbf{x}^\dagger$ such that $\mathbf{x}^\dagger \in \arg\max g(\mathbf{x}; \lambda)$ and $\xi(\mathbf{x}^\dagger) = \theta_\dagger$.

It is clear that $\mathbf{x}^\dagger \in \arg\max f(\mathbf{x})$ subject to $\xi(\mathbf{x}) = \theta_\dagger$, otherwise the point with strictly larger f and equal ξ value would have a higher value for g, and $\mathbf{x}^\dagger$ could not be optimal for g. Thus, $f(\mathbf{x}^\dagger) = h(\theta_\dagger)$.

We can express $\theta_\dagger = t\theta_l + (1-t)\theta_u$ for some $t \in (0, 1)$. By strict convexity of h on this interval, we have that

$$h(\theta_\dagger) < th(\theta_l) + (1-t)h(\theta_u). \tag{6.1}$$

Take $\mathbf{x}^u \in \arg\max f(\mathbf{x})$ subject to $\xi(\mathbf{x}) = \theta_u$, and $\mathbf{x}^l \in \arg\max f(\mathbf{x})$ subject to $\xi(\mathbf{x}) = \theta_l$. These are the points in $\mathcal{B}$ corresponding to $h(\theta_l)$ and $h(\theta_u)$. The optimality of $\mathbf{x}^\dagger$ implies that $g(\mathbf{x}^\dagger; \lambda) \geq g(\mathbf{x}^u; \lambda)$ and $g(\mathbf{x}^\dagger; \lambda) \geq g(\mathbf{x}^l; \lambda)$. Thus,

$$\begin{aligned} g(\mathbf{x}^\dagger; \lambda) &\geq tg(\mathbf{x}^l; \lambda) + (1-t)g(\mathbf{x}^u; \lambda) \\ h(\theta_\dagger) - \lambda\theta_\dagger &\geq th(\theta_l) - t\lambda\theta_l + (1-t)h(\theta_u) - (1-t)\lambda\theta_u \\ h(\theta_\dagger) &\geq th(\theta_l) + (1-t)h(\theta_u), \end{aligned} \tag{6.2}$$

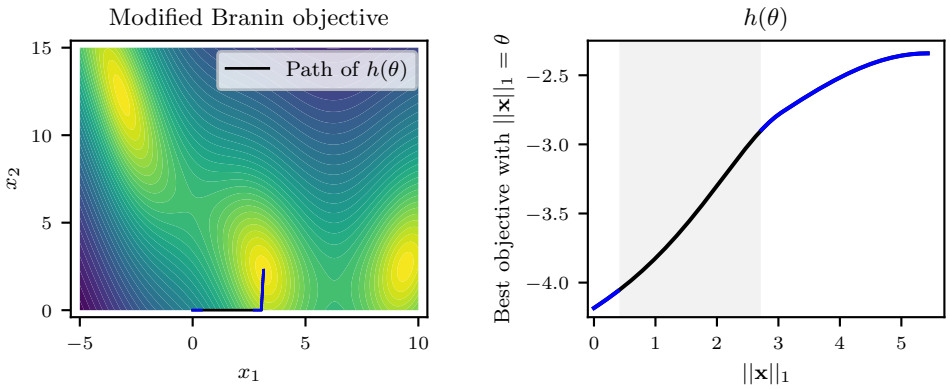


Figure 6.1: An illustration of the negative theoretical result on internal regularization. (Left) The objective f is a modified Branin function. The sparsity penalty ξ is the L_1 norm. (Right) The optimal objective vs. sparsity trade-off, $h(\theta)$, shows the best-achievable objective value for any specified value of L_1 norm. The shaded region is an interval where h is strictly convex.

using $\theta_\dagger = t\theta_l + (1-t)\theta_u$. The result in Equation (6.2) contradicts the convexity in Equation (6.1), and so $\mathbf{x}^\dagger$ cannot be optimal for g. □

Figure 6.1 shows an illustration of the result of Theorem 2.2 on a log-transformed version of the classic Branin problem, where $f(x_1, x_2) = -\log(10 + \text{Branin}(x_1, x_2))$, and we are using a traditional L_1 regularization penalty, $\xi(x_1, x_2) = |x_1| + |x_2|$. The right panel shows $h(\theta)$, from Equation (2.5), as it traces the trade-off from the minimum of ξ to the maximum of f. There is a wide interval of L_1-norm values in the middle, 0.4 to 2.7, where $h(\theta)$ is strictly convex. By Theorem 2.2, there is no value of λ under which the maximizer of Equation (2.3) has L_1 norm in that range. That range of sparsity levels thus cannot be reached by maximizing the regularized function g.

6.1.4 Relationship between ParEGO and Internal Regularization

As described in Section 2.2, ParEGO applies the EI acquisition function to a random scalarization of multiple objectives. With internal regularization, random sampling of λ for each acquisition optimization produces a ParEGO-style strategy for sparse BO, that differs only in the form of the scalarization.

The inability of linear scalarizations to capture the entire Pareto front, seen in Theorem 2.2, is a well-known failure mode for MOO. This result has inspired a large number of alternative scalarizations (Das and Dennis, 1997). ParEGO avoids the issue by replacing the linear scalarization with an augmented Chebyshev scalarization (Bowman, 1976). When applied to the context of sparse

regularization, this means maximizing

$$T(\mathbf{x};\lambda) = C(f(\mathbf{x}) - \lambda\xi(\mathbf{x})) - \max(f^* - f(\mathbf{x}), \lambda(\xi(\mathbf{x}) - \xi(\mathbf{x}^s)),$$

where f^* is an estimate for the maximum of f and C is a constant, usually set to 0.05. Unlike g in Equation (2.3), maximizers of T span the entire objective vs. sparsity trade-off (Knowles, 2006). Using EI to optimize this regularized function with randomly sampled values of λ is equivalent to applying ParEGO to the objective and the (negative) sparsity penalty.

6.1.5 Additional Details for Optimization with L_0 Sparsity

In this section we provide some additional details for the homotopy continuation described in Section 2.5. For computational reasons, we use a sequence of 30 a's starting from $a_{\text{start}} = 10^{-0.5}$ and ending at 10^{-3} that is linearly spaced on a log-scale. First, we optimize the acquisition function using L-BFGS-B from 20 different starting points to obtain 20 local optima of $H(x, a_{\text{start}})$. We then increment the value of a and use L-BFGS-B to re-optimize the homotopy starting from each of the previously found 20 local optima. This process is continued until we reach $a = 0$ which is the acquisition function corresponding to the true L_0 norm. Note that this procedure traces 20 curves $c(a) \in \arg\min_x H(x, a)$ from $a = a_{\text{start}}$ to $a = 0$ and that this curve is of finite length under the assumption that the domain is compact. These curves are potentially different as the acquisition function may be non-convex and have multiple local optima. Finally, we choose the candidate as the point that achieves the best acquisition function value.

We use $a_{\text{start}} = 10^{-0.5}$ as it strikes a balance between being large enough to find initial points with non-zero acquisition function values, and being small enough to discover points that are almost sparse. To better understand this choice note that $\max_{x,z\in[0,1]} \left|\varphi'_{10^{-0.5}}(\mathbf{x} - \mathbf{z})\right| \approx 0.067$ while, e.g., $\max_{x,z\in[0,1]} |\varphi'_{0.1}(\mathbf{x} - \mathbf{z})| \approx 2 \times 10^{-20}$ which shows that 0.1 may be too small to serve as a_{start}. We also investigate this choice in an ablation study in Section 6.1.6 and find that the performance of SEBO-L_0 is not sensitive to the choice of a_{start} as long as the value is not too small.

6.1.6 Additional Experimental Studies

Ranking sourcing system simulation

In the sourcing simulation experiment in Section 2.7, the recommender sourcing system has 25 content sources and 1000 possible pieces of content (i.e., *items*) for retrieval. We consider a 25-dimensional

retrieval policy $\mathbf{x}$ over the integer domain $[0, 50]^{25}$. We take inspiration from the Latent Dirichlet Allocation (LDA) model (Blei et al., 2003) in defining a generative probabilistic model of items recommended by each source. We assume there are 8 latent topics and that each item can be represented as a mixture over topics. Each source contains a mixture over a set of topics, and particular items will be more likely to be recommended by topically related sources. Such topical overlaps can create redundancy of recommendations across sources. Retrieving more items from additional sources comes at a cost, making sparse retrieval policies preferred.

Before describing the simulation in pseudo-code, we need the following definitions:

- T is the number of latent topics.
- K is the number of distinct items.
- S is the number of content sources.
- $\theta_s \in \Delta^T$ is the topic distribution for source s, where Δ^T denotes the T-dimensional simplex. $\{\theta_s\}_{s=1}^S$ follow a Dirichlet distribution, i.e., $\theta_s \sim \text{Dir}(\alpha)$ where $\alpha = 0.2$.
- $\phi_i \in \Delta^K$ is the item distribution for each topic i, where Δ^K denotes the K-dimensional simplex. $\{\phi_i\}_{i=1}^T$ also follow a Dirichlet distribution, i.e., $\phi_i \sim \text{Dir}(\beta)$ where $\beta = 0.5$.
- $z_{s,k}$ is the topic assignment for item k in source s and follows a multinomial distribution: $z_{s,k} \sim \text{Multi}(\theta_s)$
- $w_{s,k}$ is the indicator of item k is retrieved from source s and follows multinomial distribution: $w_{s,k} \sim \text{Multi}(\phi_{z_{s,k}})$.
- Q_i is the relevance score of each topic i and is sampled from a log-normal distribution with mean 0.25 and standard deviation 1.5.
- m_k is the relevance score of each item k, which is derived as the weighted average across topic scores based on the item distribution over 8 latent topics, i.e., $m_k = \sum_{i=1}^T \phi_{i,k} Q_i$.
- c_s is the infrastructure cost per fetched item for source s. The cost c_s is assumed to be positively correlated with source relevance score $q_s = \sum_{i=1}^T \theta_{s,i} Q_i$ and follows a Gaussian distribution with mean $\frac{q_s}{2\sum_{s=1}^S q_s}$ and standard deviation of 0.1.

To simulate the retrieval of one item from the source s, we sample a topic for an item k from the multinomial $\text{Multi}(\theta_s)$, i.e., $z_{s,k} \sim \text{Multi}(\theta_s)$, and sample an item $w_{s,k} \sim \text{Multi}(\phi_{z_{s,k}})$ where $w_{s,k}$

indicates item k being retrieved from source s. Given the sourcing policy $\mathbf{x} \in \mathbb{R}^S$, we execute the above sampling $\mathbf{x_s}$ times for each source s as described at Line 1 in Algorithm 4, and then compute the quality score given a list of retrieved items.

The overall content relevance score is the sum of the content relevance scores after de-duplicating the retrieved content. The infrastructure load is a sum of products of a number of retrievals and the cost per fetched item c_s for each source, in which c_s varies across sources and positively correlates with the source relevance score. This setup is based on the real-world observation that sources providing higher relevance content are generally more computationally expensive. The objective in the benchmark experiments is a weighted sum of overall content relevance and negative infrastructure load. In the experiment, we repeat this simulation (at Line 10) 1000 times for a given policy and compute the mean and standard error of the objective values, which we refer to as the *quality score* in the main text.

Algorithm 4 Recsys Simulation

1: **procedure** ITEM-RETRIEVAL(x_s)
2: $\vec{n}_s \leftarrow \overrightarrow{0} \in \mathbb{R}^K$ ▷ number of retrievals for K distinct items
3: **for** $n \leftarrow 1$ to x_s **do** ▷ retrieve x_s items
4: Sample a topic for an item k in source s i.e. $z_{s,k} \sim \text{Multi}(\theta_s)$
5: Sample an item $w_{s,k} \sim \text{Multi}(\phi_{z_{s,k}})$
6: $\vec{n}_s \leftarrow \vec{n}_s + \vec{w}_s$
7: **end for**
8: **return** $\vec{n}_s$
9: **end procedure**

10: **procedure** SOURCING($\mathbf{x}$)
11: $\vec{n} \leftarrow \overrightarrow{0} \in \mathbb{R}^K$ ▷ number of retrievals for K distinct items
12: **for** $s \leftarrow 1$ to S **do** ▷ retrieve items for each source s
13: $\vec{n}_s \leftarrow$ ITEM-RETRIEVAL(x_s)
14: $\vec{n} \leftarrow \{\vec{n} + \vec{n}_s\}$
15: **end for**
16: Compute relevance score $\text{RS} = \sum_{k=1}^{K} \mathbb{1}(n_k > 0) m_k$ and infrastructure cost $C = \sum_{s=1}^{S} c_s \times x_s$
17: **return** quality score $Q = \text{RS} - 0.6 \times C$
18: **end procedure**

Hypervolume Trace Plots

We evaluate optimization performance by showing the average best obtained hypervolume across 20 replicates, with 95% confidence interval over 100 trials. The results are shown for the sourcing problem (left), the SVM problem (middle) and the Hartmann6 function embedded into a 50D (right) in Figure 6.2. It can be seen that SEBO-L_0 (red traces) outperforms all the other methods and achieved the best hypervolume value over 100 iterations. The IR and ER methods with well selected

regularization parameter values can sometimes achieve competitive results and usually outperform the methods with non-regularized acquisition functions, e.g. SAASBO.

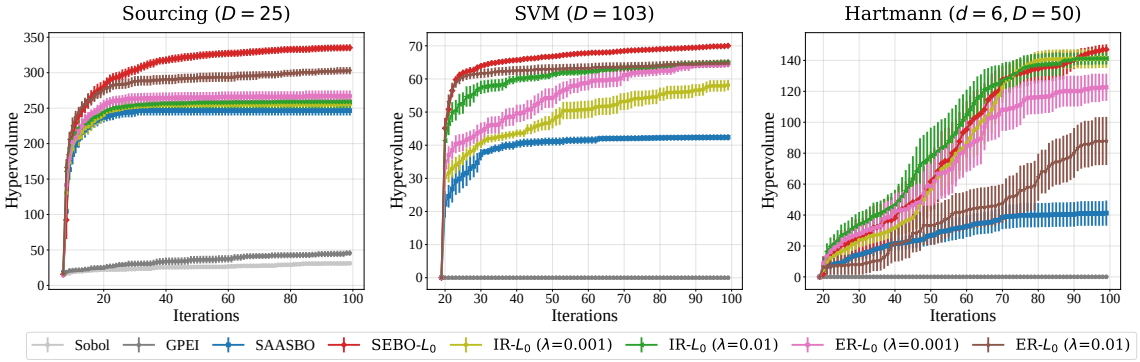


Figure 6.2: Hypervolume benchmark traces. (Left) Sourcing problem.(Middle) SVM problem. (Right) Hartmann6 function embedded into a 50D. The results are the average best hypervolume (with 95% confidence interval) obtained over 100 iterations across 20 replications. SEBO-L_0, shown in red, performs the best in all three problems.

Sensitivity Analysis of Regularization Parameter λ

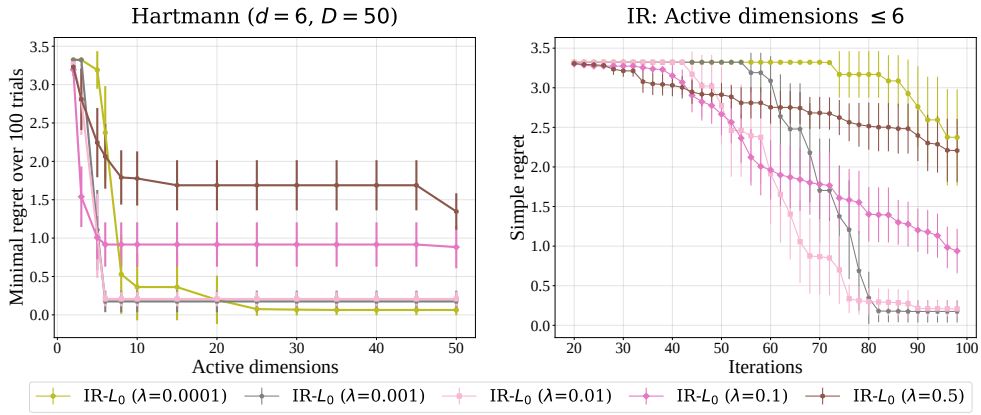


Figure 6.3: Results of IR with different λ values for Hartmann6 function embedded into a 50D space. (Left) The objective-sparsity trade-off after all 100 iterations. (Right) The simple regret considering only observations with at most 6 active (non-sparse) parameters.

We conduct a sensitivity analysis of regularization parameter λ used by IR and ER by sweeping different values of λ on the 50D Hartmann6 benchmark. The results are given in Figure 6.3 and Figure 6.4. We observe that we are able to control the sparsity level by appropriately choosing λ. In general, larger λ implies stronger regularization and results in finding configurations with a higher level of sparsity. When λ increases above a certain point, the regularization becomes too strong and fails to help find high-quality sparse points.

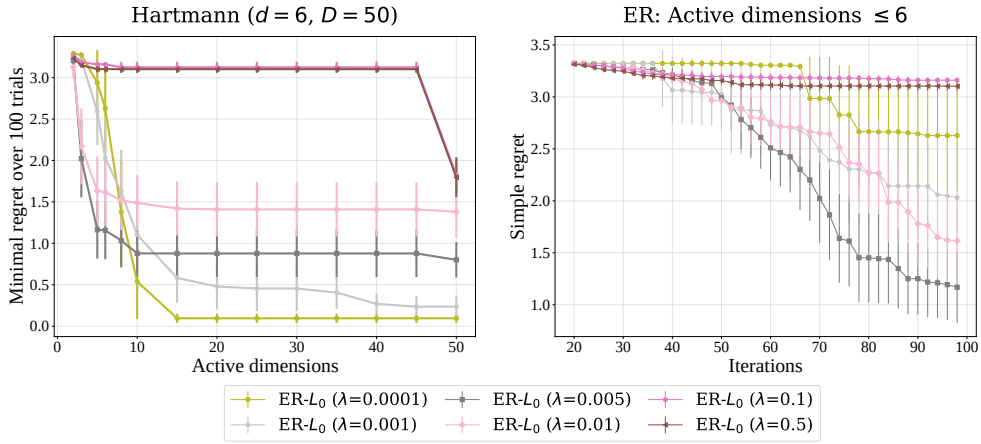


Figure 6.4: Results of ER with different λ values for Hartmann6 function embedded into a 50D space. (Left) The objective-sparsity trade-off after all 100 iterations. (Right) The simple regret considering only observations with at most 6 active (non-sparse) parameters.

By comparing results of IR and ER for different λ values, we note that IR is able to achieve effective optimization performance over a wider range of λ's while ER is more sensitive to the value of λ. This validates the discussion about ER in Section 2.3.1 that ER is not as effective as IR due to ER's inability to select a new sparse point that improves over sparse points from previous iterates if the new sparse point does not improve on the dense points that are already observed.

Sensitivity Analysis of a_{start} in SEBO-L_0 Optimization

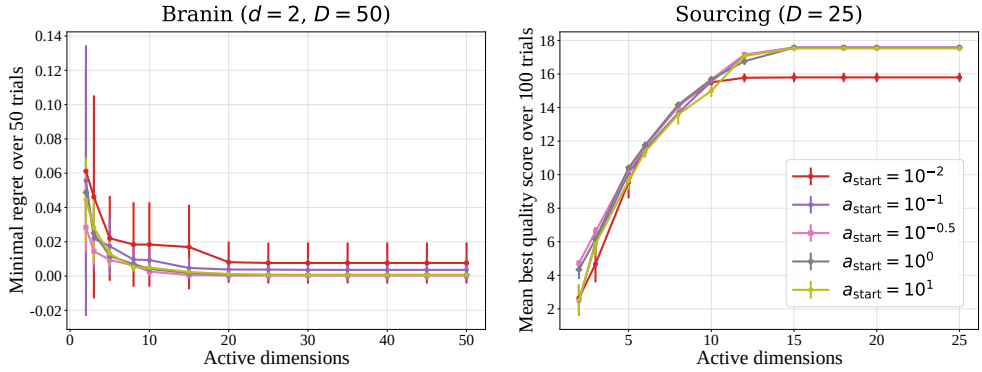


Figure 6.5: Ablation study of a_{start} in SEBO-L_0. (Left). Results of Branin ($d = 2, D = 50$). (Right). Results of Sourcing ($D = 25$). There is no statistically significant difference between using different a_{start} except for the extremely small a_{start} ($= 10^{-2}$). This shows the robustness of having a default a_{start} for optimizing SEBO-L_0 acquisition function.

The value of a_{start} is set to be $10^{-0.5}$ for all the experiments. To better understand the robustness of this choice we conducted an ablation study on the Branin($d = 2, D = 50$) and Sourcing ($D = 25$) problems considered in Section 2.7. The results in Figure 6.5 show that there is no statistically

significant difference between using 10^{-1}, $10^{-0.5}$, 10^{0} and 10^{1} as the value of a_{start}. However, using a value of 10^{-2} leads to a clear drop in performance as this starting value is too small to optimize the acquisition function.

Ablation Study on Using SAAS

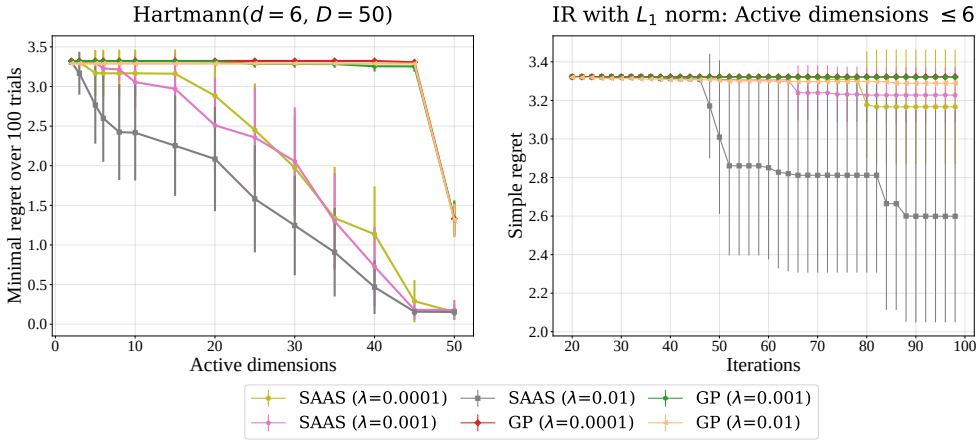


Figure 6.6: Results for the Hartmann6 function embedded in a 50D space. IR-L_1 using the SAAS model significantly outperforms IR-L_1 using a standard GP.

To illustrate the importance of using the SAAS model, we compare to using IR-L_1 with a standard GP in Figure 6.6. We observe that IR-L_1 with a standard GP fails to discover non-trivial sparse configurations for all values of λ. This confirms that sparsity in the GP model is crucial for finding sparse configurations. This can also be observed by comparing performances of SAASBO and GPEI in Figure 2.3 where there is a huge gap in terms of the best function value optimized even when looking at dense points (active dimensions = 50).

Benchmark with Additional High-dimensional BO methods

We conduct evaluations of additional high-dimensional BO methods for the Hartmann6 function embedded in a 50D space, including trust region BO (TuRBO) by (Eriksson et al., 2019) and Random Embedding BO (REMBO) by (Wang et al., 2016). The left plot in Figure 6.7 shows the trade-off between the objective and sparsity after all 100 iterations. Although SAASBO and TuRBO achieve good non-sparse solutions, they fail to obtain sparse solutions. REMBO does not obtain better sparse solution than SAASBO. In the right plot, we show the simple regret considering only observations with at most 35 active (non-sparse) parameters. SEBO-L_0 outperforms these high-dimensional BO since these methods do not encourage sparse solutions.

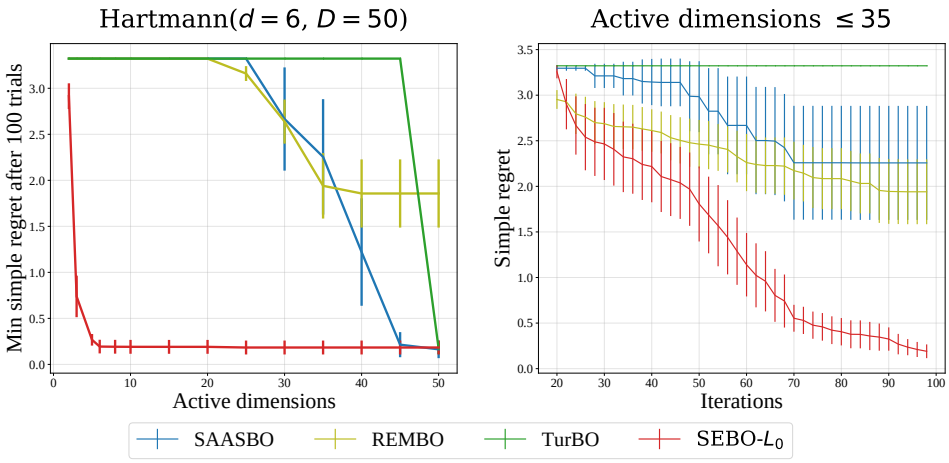


Figure 6.7: Results of additional high-dimensional BO methods for the Hartmann6 function embedded in a 50D space. (Left) The objective-sparsity trade-off after all 100 iterations. SAASBO and TuRBO, although obtaining competitive objective values with 50 active parameters, do not encourage sparse solutions. (Right) The simple regret for Hartmann6 function considering only observations with at most 35 active (non-sparse) parameters.

Low-dimensional BO Problem

SEBO can also be applied to low-dim problems using arbitrary GP models as it targets the trade-off between objective and sparsity. In the experiments section (Section 2.7), we focus on high-dimensional problems because sparsity (and interpretability) tends to be more important with more parameters. In Figure 6.8, we compare the performance on the Hartmann6 problem for Sobol, SAASBO, GPEI, and SEBO with L_0 or L_1 penalty using a standard GP as a surrogate model. This problem is known to have structures where some dimensions are more important than others for maximizing function value. SEBO-L_0 with a standard GP achieves the best trade-off in the low-dimensional (6D) problem.

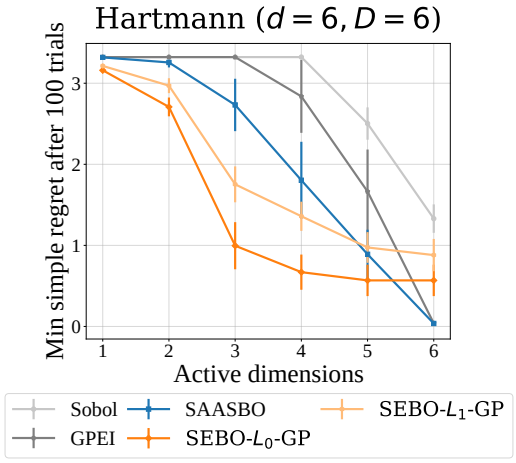


Figure 6.8: Hartmann6 function where 0 is considered sparse. Standard GP (without SAAS model) is used as the GP surrogate model for SEBO-L_1 and SEBO-L_0.

6.2 Appendix of Chapter 3

6.2.1 Attention Mechanism

In this section, we give a brief overview of the most commonly used attention mechanisms.

Dot-product attention is an attention mechanism that takes in m queries $\mathbf{Q} \in \mathbb{R}^{m \times d_k}$ and maps them to outputs using n key-value pairs $\mathbf{K} \in \mathbb{R}^{n \times d_k}$, $\mathbf{V} \in \mathbb{R}^{n \times d_k}$. The output is a weighted sum of values with each value's weight being determined by the dot product of the corresponding key with the query:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{W}\mathbf{V} \in \mathbb{R}^{m \times d_v}, \text{with } \mathbf{W} = \text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d_k}) \in \mathbb{R}^{m \times n}.$$

Intuitively, for each query, attention computes an aggregated vector based on its relevance to the key-value pairs.

Multi-head attention was first introduced in Vaswani et al. (2017), extends a single dot-product attention to multiple attentions. It first projects the original keys, values and queries onto H different keys, values and queries. And dot attention is applied for each of these H projections. The final values are calculated by taking a linear transformation of the concatenated different head-specific values:

$$\begin{aligned} \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &:= \text{concat}\left(\text{head}_1, \ldots, \text{head}_H\right) \mathbf{W}, \\ \text{head}_h &:= \text{Attention}(\mathbf{Q}\mathbf{W}_h^{\mathbf{Q}}, \mathbf{K}\mathbf{W}_h^{\mathbf{K}}, \mathbf{V}\mathbf{W}_h^{\mathbf{V}}). \end{aligned}$$

The multiple heads allow the attention mechanism to learn richer representations by combining representations from different subspaces.

Self-attention For sequential data, *self-attention* defines a key-value-query tuple for each datum in a sequence so that they can attend to each other. Combined with multi-head attention, one could apply a feed-forward network right after with residual connection to make up a multi-head self-attention sub-block. By stacking up multiple multi-head self-attention sub-blocks, one could formulate a multi-layer *Transformer* encoder (Vaswani et al., 2017) that allows extracting feature representations hierarchically through modeling interactions between input representations at different levels and maps a sequence of data to a sequence of final feature representations for downstream

tasks. It is most commonly used as the fundamental functional block of the powerful Transformer type of models that achieves state-of-the-art performance on many NLP tasks (Vaswani et al., 2017; Devlin et al., 2018).

6.2.2 Proof of Proposition 3.1

We start by defining permutation equivariant and permutation invariant functions.

Definition 6.1. *(Permutation equivariant function) Let S_n be the set of all permutations of indices $\{1, 2, \cdots, n\}$. A function $f : \mathcal{X}^n \to \mathcal{Y}^n$ is permutation equivariant if and only if for any permutation $\pi \in S_n$, $f(\pi x) = \pi f(x)$.*

Definition 6.2. *(Permutation invariant function) Let S_n be the set of all permutations of indices $\{1, 2, \cdots, n\}$. A function $f : \mathcal{X}^n \to \mathcal{Y}$ is permutation invariant if and only if for any permutation $\pi \in S_n$, $f(\pi x) = f(x)$.*

Next, we start the proof.

Proof. We assume our neural network takes in the l-th dataset $\mathcal{D}^{(l)}$ as input. By definition of the multi-head self-attention mechanism, it is obvious that a single multi-head self-attention sublock is permutation equivariant. From (Zaheer et al., 2017) we know stacks of permutation equivariant layers are still permutation equivariant. Therefore, for every dimension, the input/output data $\{(\mathbf{x}_{i,d}^{(l)}, y_i^{(l)})\}_{i=1}^{N_l}$ are always mapped to the same corresponding $\{\mathbf{h}_{i,d}^{(l)}\}_{i=1}^{N_l}$ regardless of their order. Further if AggregateFunction is permutation invariant, we have $\mathbf{h}_{\text{local},d}^{(l)}$ invariant of the order of data points for each dimension.

As the LocalTransformer is sharing the same weights for all dimensions,$\{\mathbf{h}_{\text{local},d}^{(l)}\}_{d=1}^{D_l}$ will remain equivariant with regards to permutation of dimensions. Next $\{\mathbf{h}_{\text{local},d}^{(l)}\}_{d=1}^{D_l}$ are passed through stacks of multi-head self-attention subblocks which is permutation equivariant, therefore the final $\{\mathbf{h}_{\text{global},d}^{(l)}\}_{d=1}^{D_l}$ are permutation equivariant. Again the weight sharing of the final MLP ensures that the final predicted spectral density hyperparameters $\{\theta_d^{(l)}\}_{d=1}^{D_l}$ are permutation equivariant with regards to data dimensions. They are also permutation invariant with regards to data points, as $\{\mathbf{h}_{\text{local},d}^{(l)}\}_{d=1}^{D_l}$ are invariant of the order of input/output data. □

6.2.3 Additional Experimental Details and Results

Is the Model Actually Learning?

Our method (AHGP) has demonstrated competitive performance across different applications. However, is the neural model really learning? In this section, we will construct experiments to conduct a sanity-check on this empirically. In the experiments, we augment each dataset by adding a new feature dimension to each datapoint. We test with two extreme cases: 1. an i.i.d random Gaussian noise and 2. the exact same label of the data point. The new dimension is also standardized and treated just like normal features. And then we apply our trained neural model on the augmented regression datasets.

One should expect a sensible model to predict the noise dimension with large lengthscales, as the dimension does not contain any useful information and should be discarded by assigning large lengthscales. Meanwhile, the label dimension should be identified with the smallest lengthscales because they carry the most information across all dimensions. Table 6.1 shows a summary of the weighted lengthscales predicted by our model on the regression benchmarks. Results show that our model is able to consistently identify the noise dimension with very large lengthscales and the label dimension with the smallest lengthscales, which suggests that our model is learning generalizable representations.

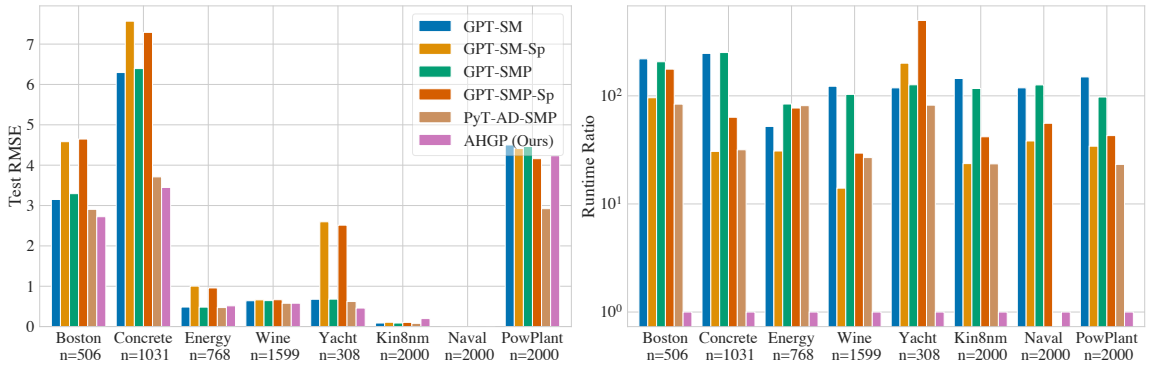
Dataset	Boston	Concrete	Energy	Wine	Yacht	Kin8nm	Naval	PowPlant
Noise dim. len.	163.14±1.61	185.74±3.88	180.71±0.91	87.31±3.84	217.94±4.67	129.77±1.81	148.88±0.59	118.65±2.46
Label dim. len.	1.77±0.00	1.71±0.00	1.51±0.00	1.40±0.01	1.15±0.01	1.46±0.00	1.37±0.00	1.63±0.00

Table 6.1: Predicted weighted lengthscales of noise and label dimensions

Regression Benchmarks

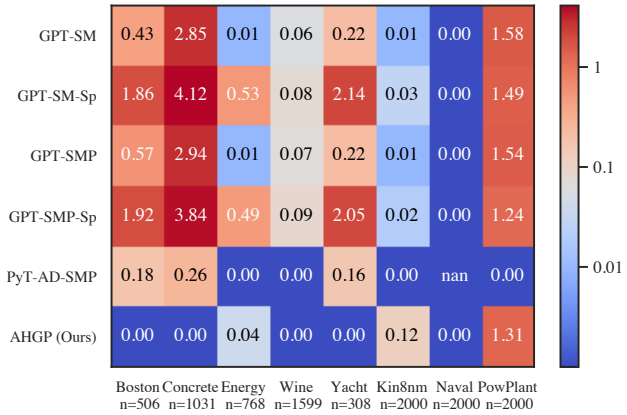
For SGPR methods, the number of inducing points is set to 10% of the number of training data. The full comparisons with the CPU-based baselines in terms of test RMSE and log marginal likelihood in Tables 3.3 and 6.3.

The comparisons of AHGP with the GPU-based baselines are presented in Figure 6.9 and Tables 6.4 to 6.6. Results are similar to comparisons with the CPU-based baselines in the main section. Note that GPyTorch baselines perform slightly worse than GPy baselines, since GPyTorch uses conjugate gradient method to approximately solve for matrix inverse instead of doing the Cholesky decomposition. In comparison, PyT-AD-SMP uses Cholesky decomposition and generally achieves better performance than GPT-SMP.



(a) Test RMSE

(b) GPU runtime ratio over AHGP



(c) Gap with best RMSE

Figure 6.9: Comparison of AHGP against the GPU-based baselines on regression benchmarks. In (c), the numbers are the differences of the corresponding method's test RMSE with the best RMSE on the respective dataset. Note that for Naval, the RMSEs are all very close to 0 except PyT-AD-SMP which runs out of GPU memory.

Dataset	GPT-SM	GPT-SM-Sp	PyT-AD-SMP	GPT-SMP	GPT-SMP-Sp	**AHGP(Ours)**
Boston	10.99±0.05	4.80±0.04	4.19±1.16	10.33±0.07	8.83±0.47	**0.05±0.00**
Concrete	32.10±3.69	3.98±0.14	4.12±0.02	32.69±10.72	8.24±0.14	**0.13±0.00**
Energy	4.17±0.12	2.47±0.13	6.49±1.76	6.72±0.13	6.17±0.56	**0.08±0.02**
Wine	46.63±0.19	5.32±0.15	10.20±0.08	39.19±0.07	11.22±0.31	**0.38±0.00**
Yacht	2.37±0.16	4.01±0.16	1.64±0.05	2.53±0.15	9.96±0.20	**0.02±0.00**
Kin8nm	66.55±21.34	10.86±0.14	10.80±0.15	53.79±1.13	19.23±0.69	**0.46±0.00**
Naval	99.58±1.98	32.12±0.70	Out of memory	106.01±0.52	46.83±0.57	**0.84±0.13**
PowPlant	49.36±0.14	11.29±0.67	7.66±0.06	32.17±0.06	14.15±0.89	**0.33±0.01**

Table 6.6: Runtime (in seconds) on regression benchmark: GPU-based methods

Dataset	GPy-SM	GPy-SM-Sp	GPy-SMP	GPy-SMP-Sp	**AHGP(Ours)**
Boston	2.960±0.506	3.027±0.506	**2.710±0.482**	2.971±0.487	2.723±0.387
Concrete	3.573±0.848	4.747±0.386	3.695±0.805	4.907±0.393	**3.448±0.448**
Energy	0.488±0.036	0.503±0.058	**0.483±0.044**	1.728±1.233	0.515±0.071
Wine	**0.577±0.027**	0.661±0.028	0.580±0.028	0.690±0.038	0.580±0.035
Yacht	0.868±0.567	0.821±0.283	0.681±0.288	0.845±0.387	**0.460±0.265**
Kin8nm	**0.080±0.003**	0.080±0.004	0.085±0.004	0.093±0.009	0.199±0.008
Naval	**0.000±0.000**	0.001±0.001	0.000±0.000	0.006±0.002	0.002±0.000
PowPlant	3.250±0.299	4.097±0.348	**2.977±0.210**	4.068±0.221	4.234±0.242

Table 6.2: Test RMSE on regression benchmarks: CPU-based methods

Dataset	GPy-SM	GPy-SM-Sp	GPy-SMP	GPy-SMP-Sp	**AHGP (Ours)**
Boston	-2.649±0.364	-3.527±1.086	-2.538±0.281	-2.939±0.642	**-2.367±0.115**
Concrete	**-2.656±0.637**	-3.435±0.238	-2.690±0.574	-3.757±0.579	-3.460±1.334
Energy	-1.059±0.019	-1.103±0.046	-1.079±0.028	-2.534±0.557	**-0.837±0.215**
Wine	-0.427±0.058	-4.331±1.328	-0.410±0.053	-1.053±0.051	**-0.321±0.075**
Yacht	-1.573±0.254	-1.526±0.108	-1.500±0.077	-1.979±0.742	**-0.997±0.092**
Kin8nm	**1.119±0.044**	0.655±0.177	1.020±0.054	0.748±0.198	0.192±0.039
Naval	6.157±0.267	5.433±1.063	**6.211±0.011**	3.643±0.164	5.393±0.102
PowPlant	-2.591±0.115	-3.831±0.330	**-2.475±0.060**	-2.936±0.047	-3.112±0.151

Table 6.3: Test log-likelihood on regression benchmarks: CPU-based methods

Dataset	GPT-SM	GPT-SM-Sp	PyT-AD-SMP	GPT-SMP	GPT-SMP-Sp	**AHGP(Ours)**
Boston	3.150±0.674	4.582±1.168	2.906±0.488	3.294±0.900	4.647±1.133	**2.723±0.387**
Concrete	6.295±1.071	7.568±0.558	3.710±0.742	6.393±0.877	7.290±0.535	**3.448±0.448**
Energy	0.485±0.041	1.004±0.166	**0.472±0.041**	0.483±0.050	0.958±0.201	0.515±0.071
Wine	0.641±0.030	0.663±0.033	**0.579±0.027**	0.644±0.030	0.666±0.034	0.580±0.035
Yacht	0.678±0.315	2.597±1.149	0.622±0.265	0.681±0.316	2.514±1.539	**0.460±0.265**
Kin8nm	0.089±0.005	0.108±0.004	**0.081±0.004**	0.089±0.005	0.105±0.003	0.199±0.008
Naval	**0.000±0.000**	0.003±0.000	Out of memory	0.001±0.000	0.003±0.000	0.002±0.000
PowPlant	4.501±0.238	4.412±0.241	**2.922±0.256**	4.462±0.230	4.161±0.241	4.234±0.242

Table 6.4: Test RMSE on regression benchmarks: GPU-based methods

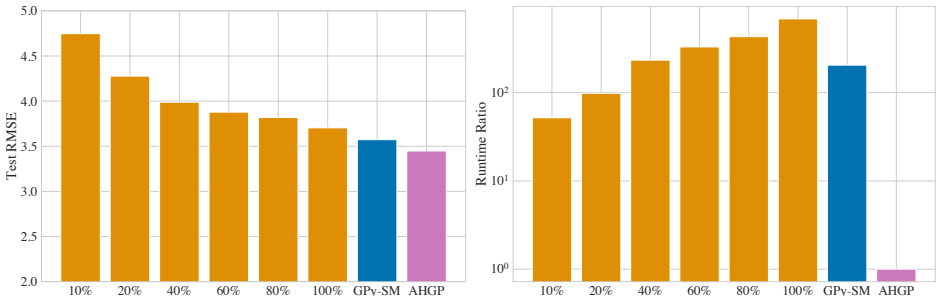
Dataset	GPT-SM	GPT-SM-Sp	PyT-AD-SMP	GPT-SMP	GPT-SMP-Sp	**AHGP (Ours)**
Boston	-2.692±0.442	-12.440±5.930	-2.757±0.416	-2.687±0.439	-11.390±5.270	**-2.367±0.115**
Concrete	**-3.030±0.640**	-3.347±0.107	-3.132±1.262	-3.095±0.720	-3.285±0.078	-3.460±1.334
Energy	-1.073±0.020	-1.269±0.064	-1.496±0.400	-1.068±0.023	-1.266±0.067	**-0.837±0.215**
Wine	-0.517±0.079	-1.067±0.129	**-0.306±0.064**	-0.527±0.076	-1.028±0.115	-0.321±0.075
Yacht	-1.492±0.105	-2.416±0.634	-1.030±0.770	-1.492±0.106	-2.477±1.071	**-0.997±0.092**
Kin8nm	0.921±0.086	0.819±0.042	**1.108±0.047**	0.917±0.090	0.854±0.045	0.192±0.039
Naval	5.892±0.028	5.253±0.215	Out of memory	**5.933±0.017**	5.372±0.113	5.393±0.102
PowPlant	-2.923±0.120	-2.903±0.135	**-2.540±0.153**	-2.920±0.150	-2.834±0.078	-3.112±0.151

Table 6.5: Test log-likelihood on regression benchmarks: GPU-based methods

Comparing with Sparse Variational GP with Different Number of Inducing Points

To fully compare the sparse variational GP (SGPR) with our method, we conducted another experiment by varying the number of inducing points used in SGPR. To illustrate the difference when different number of inducing points is used, we pick the *Concrete* in UCI datasets (Asuncion

and Newman, 2007), on which we see an obvious performance gap between GPy-SM (MLL-opt) and GPy-SM-Sp (SGPR). In the experiment, the number of inducing points in GPy-SM-Sp is set to $\{10\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$ of the total number of training data. The comparisons with GPy-SM and AHGP in terms of test RMSE, test log-likelihood and run time is shown in Figure 6.10 and Table 6.7. By increasing the number of inducing points, the gap between SGPR and MLL-opt is gradually reduced, although a small gap remains even when the number of inducing points is set to 100%. We also observe increased run times when then number of inducing points is increased. As SGPR has a more complicated optimization problem than MLL-opt, its run time is slower than MLL-opt when the number of inducing points is set to the same as the training data.



(a) Test RMSE (b) CPU runtime ratio over AHGP

Figure 6.10: Comparison of SGPR (GPy-SM-Sp) with MLL-opt (GPy-SM) and AHGP. The percentage represents how many inducing points (in terms of the number of training data) are used in SGPR.

Method	Test RMSE ↓	Test log-likelihood ↑	Runtime ↓
GPy-SM-Sp-10%	4.747±0.386	-3.435±0.238	138.01±36.99
GPy-SM-Sp-20%	4.277±0.268	-3.155±0.135	261.36±68.15
GPy-SM-Sp-40%	3.988±0.332	-2.999±0.184	620.73±118.55
GPy-SM-Sp-60%	3.878±0.464	-2.870±0.297	876.00±157.93
GPy-SM-Sp-80%	3.818±0.614	-2.699±0.175	1144.39±111.37
GPy-SM-Sp-100%	3.704±0.396	-2.690±0.144	1818.10±190.88
GPy-SM	3.573±0.848	**-2.656±0.637**	545.08±195.05
AHGP(Ours)	**3.448±0.448**	-3.460±1.334	**2.66±0.05**

Table 6.7: Comparison of test RMSE, test log-likelihood and runtime

Better Predictive Performance for Free (Sometimes)?

Surprisingly, AHGP performs even better than the MLL-opt approaches in terms of RMSE/log-likelihood on dataset with fewer data points (e.g., Yacht with $n = 308$) in terms both test RMSE and test log-likelihood. It is therefore of great interest to investigate the underlying reason behind

this phenomenon. One natural question to ask would be: "Is AHGP learning a better kernel function than the optimized one on the training data? Or is it some other reason". We examine the marginal likelihood on the training data achieved by AHGP and MLL-opt. It turns out the MLL-opt method still achieves a slightly better marginal likelihood on the training data (1.04 v.s. 0.85). This demonstrates that MLL-opt method is doing a good job in optimizing the marginal likelihood. However, its performance on test set shows that a perfectly optimized kernel hyperparameter on a relative small number of training data can still lead to overfitting. In comparison, AHGP only uses synthetic data during training and this training procedure seems to add extra regularization effect which prevents kernel hyperparamter estimate of AHGP from overfitting towards the training data and hence helps AHGP achieve a better performance on the in test time.

Bayesian Optimization

We pick the best performing baselines on the regression benchmarks (GPy-SM, GPy-SM-Sp, PyT-AD-SMP) and compare them with our method. The number of inducing points in GPy-SM-Sp is set to 20. Standard test functions for global optimization (Derek Bingham, 2013) are used as the target functions for Bayesian optimization. Five initial input points are randomly sampled and function values at the points are evaluated; those input points with their function values serve as the initial input/output data of GP. At the beginning of each BO iteration, hyperparameters are randomly reinitialized for all the baselines and then optimized through MLL optimization. The full results are shown in Figure 6.11, Figure 6.12a and Table 6.8.

For comparison, we also implement another warmstart initialization strategy which sets the initialization as the best hyperparameters from the previous BO iteration. The full results are shown in Figure 6.13, Figure 6.12b and Table 6.9.

Function	GPy-SM	GPy-SM-Sp	PyT-AD-SMP	**AHGP(Ours)**
Ackley	2275.32±343.28	5871.42±489.04	1288.16±70.09	**47.86±3.66**
Griewank	2192.96±1003.29	5863.13±626.95	1272.42±76.59	**46.36±2.84**
Hartmann3	614.48±169.35	5801.73±559.05	711.35±7.47	**16.44±1.44**
Hartmann6	1721.76±568.42	5935.85±503.19	961.31±24.82	**28.65±2.44**
Levy	3903.18±483.52	5707.24±628.55	1282.01±70.42	**45.04±5.59**
SixHumpCamel	1127.34±466.70	5898.18±608.69	625.29±4.76	**12.84±1.66**
Michalewicz10	1115.01±152.13	5891.81±461.76	1295.83±73.77	**48.48±4.24**
Michalewicz2	383.41±62.54	5762.95±486.45	629.56±4.56	**13.10±1.57**
Stybtang10	2826.95±581.05	5849.69±526.99	1276.52±77.09	**50.12±3.14**
Stybtang2	886.45±424.75	5776.96±588.76	633.21±10.43	**12.80±1.63**
Stybtang5	1119.24±186.60	6186.69±562.42	872.66±19.39	**25.74±2.87**

Table 6.8: Runtime (in seconds) on Bayesian optimization tasks: random initialization strategy

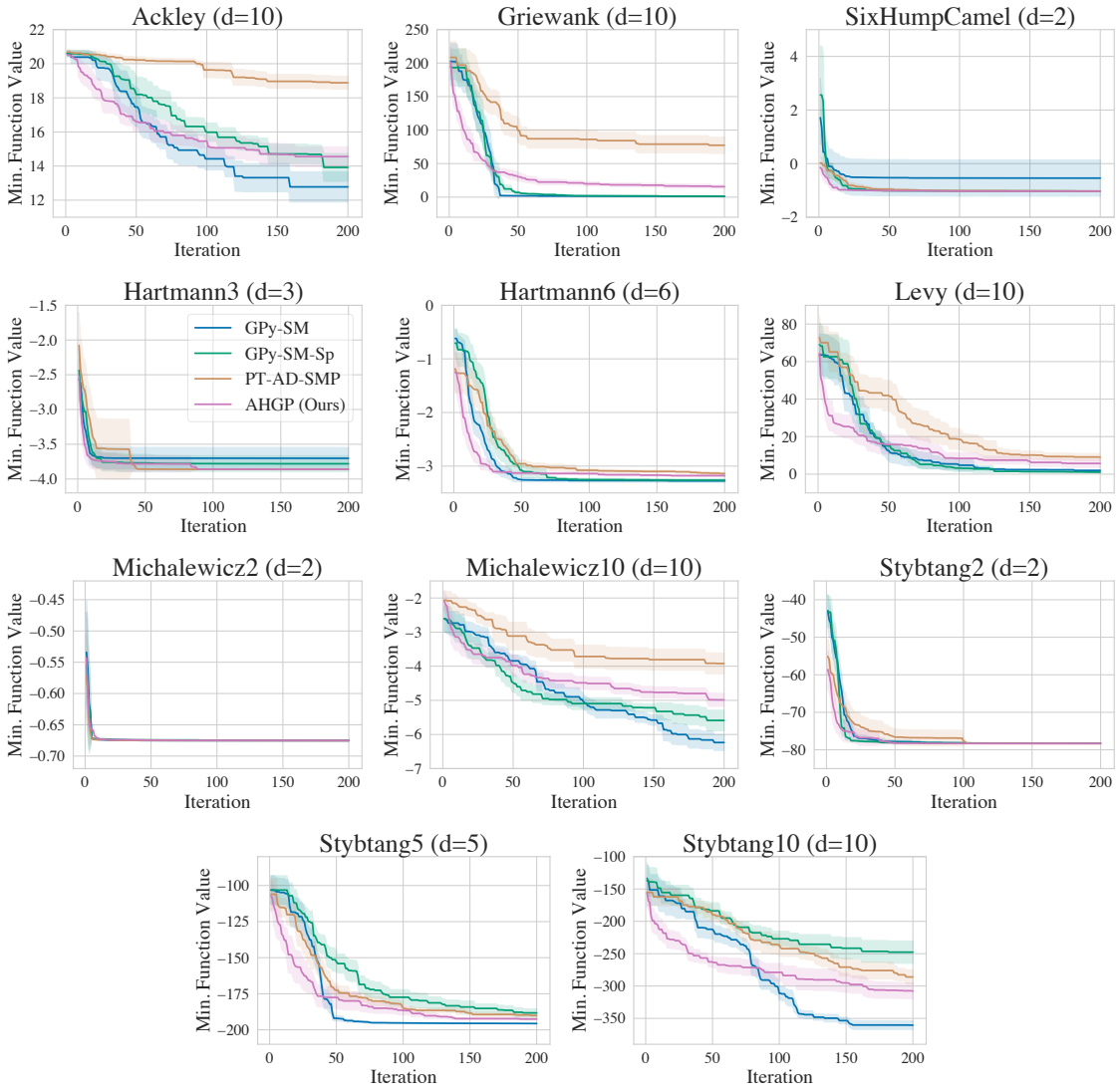
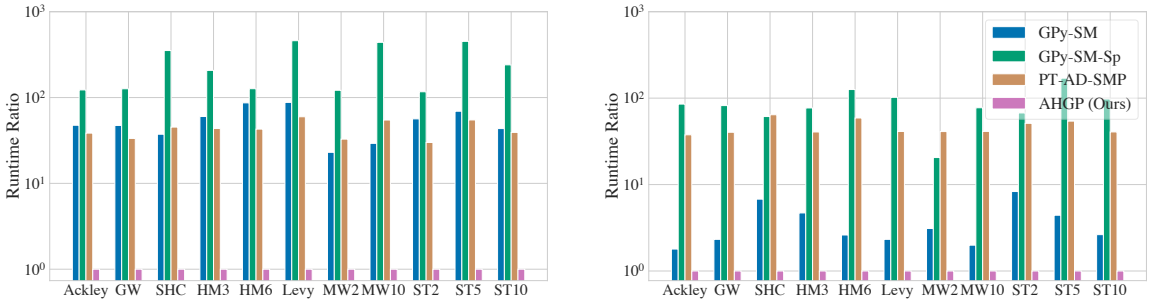


Figure 6.11: Bayesian optimization performance comparison: random initialization strategy. Shaded region indicates 0.5 standard deviations over 10 runs.

To evaluate our method on real-world Bayesian optimization problems, we additionally applied our method to tuning learning and model hyperparameters of logistic regression via BO. The goal is to find the hyperparameters that achieves the highest test accuracy when a fixed amount of time is used for training. We experiment on the task of training logistic regression on MNIST data with stochastic gradient descent. The training involves four hyperparameters: learning rate, ℓ_2 regularization parameter, ℓ_1 regularization parameter and mini-batch size. We present the results in Figure 6.14 and Table 6.10. AHGP achieves comparable test accuracy with the strongest baselines (GPy-SM, PyT-AD-SMP) while being much faster. The SGPR baseline, however, has a lower test accuracy and much longer runtime.



(a) Random initialization strategy.

(b) Warmstart initialization strategy.

Figure 6.12: Runtime on Bayesian optimization tasks.

FUNCTION	GPy-SM	GPy-SM-Sp	PyT-AD-SMP	**AHGP(Ours)**
Ackley	92.12±49.74	4381.99±2374.99	1933.38±517.41	**51.35±2.11**
Griewank	115.26±45.74	4066.03±1975.20	1998.14±864.66	**49.59±5.81**
Hartmann3	91.61±37.23	1501.48±1299.30	792.04±74.54	**19.51±0.98**
Hartmann6	82.10±35.08	3957.00±2495.70	1853.98±751.37	**31.47±0.92**
Levy	114.29±60.44	5024.81±1751.92	2022.96±461.09	**49.18±5.91**
SixHumpCamel	101.05±66.46	912.45±634.74	957.04±375.30	**14.90±0.44**
Michalewicz10	103.03±64.24	4005.42±2546.93	2133.61±1378.63	**51.78±1.84**
Michalewicz2	47.00±19.18	311.37±209.75	622.09±36.82	**15.16±0.42**
Stybtang10	126.53±109.21	4692.10±2186.68	1952.56±806.56	**48.05±5.44**
Stybtang2	119.43±105.15	969.05±945.01	734.70±219.21	**14.38±0.26**
Stybtang5	120.84±64.67	4655.30±1802.18	1482.06±504.21	**27.35±1.28**

Table 6.9: Runtime (in seconds) on Bayesian optimization tasks: warmstart initialization strategy

GPy-SM	GPy-SM-Sp	PyT-AD-SMP	**AHGP(Ours)**
195.90 ± 99.81	1067.70±41.40	23.85±0.45	**1.23±0.06**

Table 6.10: Runtime (in seconds) on Bayesian optimization for training logistic regression on MNIST

Bayesian Quadrature

For all the tasks, we start with 10 initial points and their function values. The function values are standardized. And the results are reported on the standardized function values. Random initialization strategy is used for GP hyperparameters at the start of each outer iteration. The number of inducing points in GPy-SM-Sp is set to 20. The full runtime results are reported in Table 6.11.

Model Sensitivity

To evaluate the sensitivity of our method with regards to the random initialization during training, we run 10 different trials with the same training hyperparameters. The 10 different trained models are evaluated on the regression benchmarks. For each trained model, the same 10 data-splits are

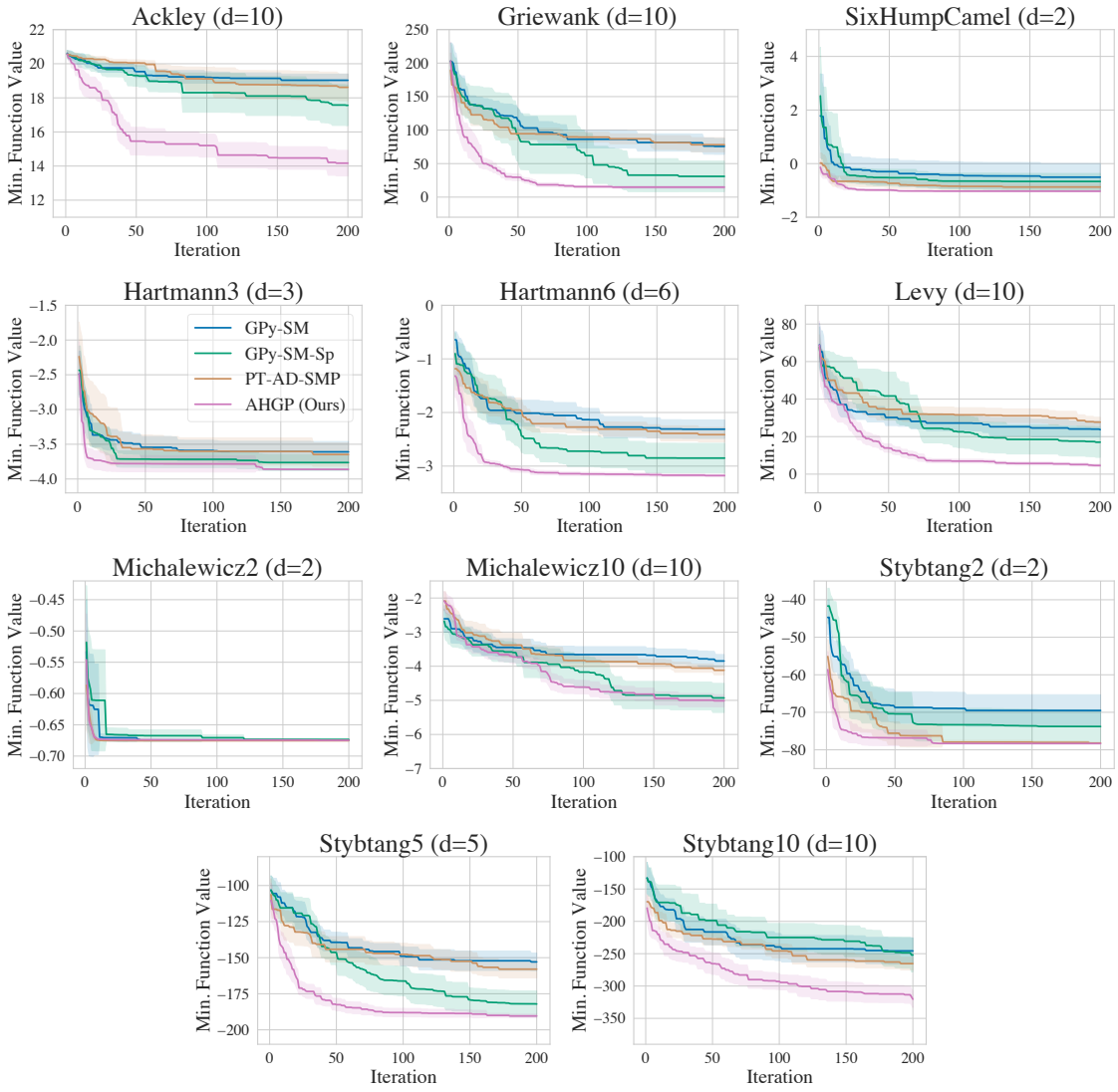


Figure 6.13: Bayesian optimization performance comparison: warmstart initialization strategy. Shaded region indicates 0.5 standard deviations over 10 runs.

used. The average test RMSEs by the different trained models are reported in Figure 6.15. For better visualization, we scaled the median of the RMSEs to 1 for each regression task. In general, AHGP is not very sensitive to the randomness involved in deep learning training. We also observe a slightly bigger variation on dataset with fewer data points.

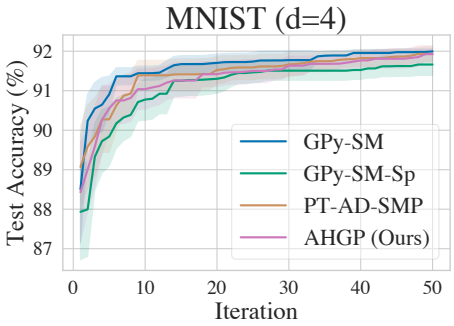


Figure 6.14: Bayesian optimization for training logistic regression on MNIST.

FUNCTION	GPy-SM	GPy-SM-Sp	PyT-AD-SMP	**AHGP(Ours)**
Circular Gaussian	1.59±0.68	66.90±23.00	0.57±0.03	**0.03±0.00**
Hennig1D	0.90±0.17	29.77±19.83	0.22±0.01	**0.02±0.00**
Hennig2D	1.03±0.19	72.47±26.82	0.54±0.05	**0.03±0.00**
Sombrero2D	1.02±0.26	75.42±20.06	0.53±0.02	**0.03±0.00**

Table 6.11: Runtime (in seconds) on Bayesian quadrature tasks

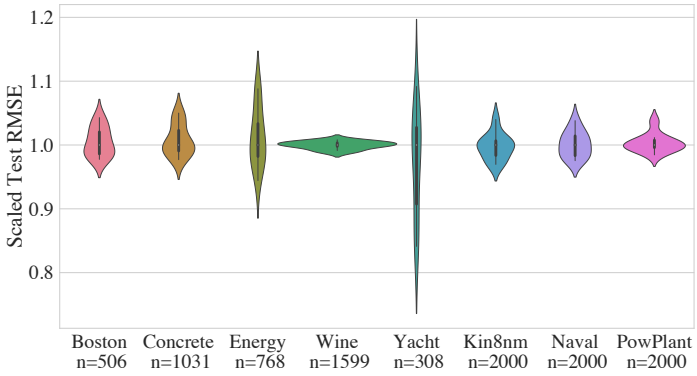


Figure 6.15: Violin plot of the test RMSEs evaluated with the 10 trained models. (The violin plot shows the probability density of the RMSEs, smoothed by a kernel density estimator.)

6.3 Appendix of Chapter 4

6.3.1 Additional Technical Details

Proof of Claim 4.1

Proof. From the single-step marginalization self-consistency in Equation (4.4), we have

$$\log p_\theta(\mathbf{x}) = \sum_{d=1}^{D} \log p_\phi \left(x_{\sigma(d)} | \mathbf{x}_{\sigma(<d)} \right), \ \forall \mathbf{x}, \sigma.$$

Therefore we can rewrite the optimization in Equation (4.5) as:

$$\begin{aligned} \max_{\phi} \quad & \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \mathbb{E}_{\sigma\sim\mathcal{U}(S_D)} \sum_{d=1}^{D} \log p_\phi \left(x_{\sigma(d)} \mid \mathbf{x}_{\sigma(<d)} \right) \\ \text{s.t.} \quad & p_\theta(\mathbf{x}_{\sigma(<d)}) p_\phi(\mathbf{x}_{\sigma(d)} | \mathbf{x}_{\sigma(<d)}) = p_\theta(\mathbf{x}_{\sigma(\leq d)}),\ \forall \sigma \in S_D, \mathbf{x} \in \{1, \cdots, K\}^D, d \in [1:D]. \end{aligned} \tag{6.3}$$

Let p^* be the optimal probability distribution that maximizes the likelihood on training data, and from the chain rule we have:

$$p^* = \arg\max_p \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log p(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \mathbb{E}_{\sigma\sim\mathcal{U}(S_D)} \sum_{d=1}^{D} \log p \left(x_{\sigma(d)} | \mathbf{x}_{\sigma(<d)} \right)$$

Then p^* is also the optimal solution to Equation (6.3) the marginalization constraints are automatically satisfied by p^* since it is a valid distribution. From the universal approximation theorem (Hornik et al., 1989; Hornik, 1991; Cybenko, 1989), we can use separate neural networks to model p_θ (marginals) and p_ϕ (conditionals), and obtain optimal solution to Equation (6.3) with θ^* and ϕ^* that approximates p^* arbitrarily well.

Specifically, if θ^* and ϕ^* satisfy the following three conditions below, they are the optimal solution to Equation (6.3):

$$p_{\phi^*} \left(x_{\sigma(d)} \mid \mathbf{x}_{\sigma(<d)} \right) = p^* \left(x_{\sigma(d)} \mid \mathbf{x}_{\sigma(<d)} \right), \quad \forall\, \mathbf{x}, \sigma \tag{6.4}$$

$$p_{\theta^*}(\mathbf{x}_s) = p^*(\mathbf{x}_s) Z_{\theta^*}, \quad \forall\, \mathbf{x}, s \subseteq \{1, \cdots, D\} \tag{6.5}$$

$$p_{\theta^*}(\mathbf{x}_{\sigma(<d)}) p_{\phi^*}(\mathbf{x}_{\sigma(d)} | \mathbf{x}_{\sigma(<d)}) = p_{\theta^*}(\mathbf{x}_{\sigma(\leq d)}),\ \forall \sigma \in S_D, \mathbf{x} \in \{1, \cdots, K\}^D, d \in [1:D] \tag{6.6}$$

where Z_{θ^*} is the normalization constant of p_{θ^*} and is equal to $p_{\theta^*}([?? \cdots ?])$. It is easy to see from the definition of conditional probabilities that satisfying any two of the optimal conditions leads to the third one.

To obtain the optimal ϕ^*, it suffices to solve the following optimization problem:

$$\textbf{Stage 1:} \quad \max_{\phi} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \mathbb{E}_{\sigma\sim\mathcal{U}(S_D)} \sum_{d=1}^{D} \log p_\phi \left(x_{\sigma(d)} \mid \mathbf{x}_{\sigma(<d)} \right)$$

because $p^* = \arg\max_p \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \mathbb{E}_{\sigma\sim\mathcal{U}(S_D)} \sum_{d=1}^{D} \log p^* \left(x_{\sigma(d)} | \mathbf{x}_{\sigma(<d)} \right)$ due to chain rule. Solving Stage 1 is equivalent to finding ϕ^* that satisfies condition in Equation (6.4). Then we can obtain the

optimal θ^* by solving for the condition in Equation (6.6) given the optimal conditionals ϕ^*:

$$\textbf{Stage 2:} \quad \min_{\theta} \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\mathbb{E}_{\sigma\sim\mathcal{U}(S_D)}\mathbb{E}_{d\sim\mathcal{U}(1,\cdots,D)}\left(\log[p_\theta(\mathbf{x}_{\sigma(<d)})p_{\phi^*}(\mathbf{x}_{\sigma(d)}|\mathbf{x}_{\sigma(<d)})] - \log p_\theta(\mathbf{x}_{\sigma(\leq d)})\right)^2$$

□

6.3.2 Additional Experimental Details and Results

Dataset details

Binary MNIST Binary MNIST is a dataset introduced in (Salakhutdinov and Murray, 2008) that stochastically set each pixel to 1 or 0 in proportion to its pixel intensity. We use the training and test split of (Salakhutdinov and Murray, 2008) provided at `https://github.com/yburda/iwae/tree/master` (Burda et al., 2015).

To evaluate the quality of the likelihood estimates, we employ a controlled experiment where we randomly mask out portions (100, 400, and 700 pixels) of a test image and generate multiple samples with varying levels of masking (refer to Figure 4.4). We repeat this for 160 (randomly subsampled) test images and created a dataset of 640 sets of comparable images. To further test the quality of the marginal likelihood estimates on partially observed images, we curate a dataset of 160 sets of partial test images ($7 \sim 9$ images in each set) by randomly subsampling from the test set and masking the upper half of the images. To make sure the partial images are comparable but different in their log-likelihood, in each set, we remove samples that have a log-likelihood close to another sample within the threshold of 5.0.

Molecular Sets The molecules in MOSES are represented either in SMILES (Weininger et al., 1989) or SELFIES (Krenn et al., 2020) strings. We construct a vocabulary (including a stop token) from all molecules and use discrete valued strings to represent molecules. It is worth noting that MaM can also be applied for modeling molecules at a coarse-grained level with predefined blocks, which we leave for future work.

The test set used for evaluating likelihood estimate quality is constructed in a similar manner to Binary MNIST, by drawing sets of random samples from the test dataset.

text8 In this dataset, we use a vocabulary of size 27 to represent the letter alphabet with an extra value to represent spaces.

The test set of datasets used for evaluating likelihood estimate quality is constructed in a similar

manner to Binary MNIST, each set is generated by randomly masking out portions of a test text sequence (by 50, 100, 150, 200 tokens) and generating samples.

Ising model The Ising model is defined on a 2D cyclic lattice. The $\mathbf{J}$ matrix is defined to be $\sigma \mathbf{A}_N$, where σ is a scalar and $\mathbf{A}_N$ is the adjacency matrix of a $N \times N$ grid. Positive σ encourages neighboring sites to have the same spins and negative σ encourages them to have opposite spins. The bias term $\boldsymbol{\theta}$ places a bias towards positive or negative spins. In our experiments, we set σ to 0.1 and $\boldsymbol{\theta}$ to $\mathbf{1}$ scaled by 0.2. Since we only have access to the unnormalized probability, we generate 2000 samples following (Grathwohl et al., 2021) using Gibbs sampling with $1,000,000$ steps for 10×10 and 30×30 lattice sizes. Those data serve as ground-truth samples from the Ising model for evaluating the test log-likelihood.

Molecular generation with target property During training, we need to optimize on the loss objective on samples generated from the neural network model. However, if the model generates SMILES strings, not all strings correspond to a valid molecule, which makes training at the start challenging when most generated SMILES strings are invalid molecules. Therefore, we use SELFIES string representation as it is a 100% robust in that every SELFIES string corresponds to a valid molecule and every molecule can be represented by SELFIES.

Training details

Binary MNIST

- 0, 1 and "?" are represented by a scalar value ("?" takes the value 0) and additionally a mask indicating if it is a "?".
- U-Net with 4 ResNet Blocks interleaved with attention layers for both AO-ARM and MaM. MaM uses two separate neural networks for learning marginals ϕ and conditionals θ. Input resolution is 28×28 with 256 channels used.
- The mask is concatenated to the input. 3/4 of the channels are used to encode input. The remaining 1/4 channels encode the mask cardinality (see Hoogeboom et al. (2021a) for details).
- MaM first learns the conditionals ϕ and then learns the marginals θ by finetuning on the downsampling blocks and an additional MLP with 2 hidden layers of dimension 4096. We observe it is necessary to finetune not only on the additional MLP but also on the downsampling

blocks to get a good estimate of the marginal probability, which shows marginal network and conditional network rely on different features to make the final prediction.

- Batch size is 128, Adam is used with learning rate 0.0001. Gradient clipping is set to 100. Both AO-ARM and MaM conditionals are trained for 600 epochs. The MaM marginals are finetuned from the trained conditionals for 100 epochs.

MOSES and text8

- Transformer with 12 layers, 768 dimensions, 12 heads, 3072 MLP hidden layer dimensions for both AO-ARM and MaM. Two separate networks are used for MaM.
- SMILES or SELFIES string representation and "?" are first converted into one-hot encodings as input to the Transformer.
- MaM first learns the conditionals ϕ and then learns the marginals θ by finetuning on the MLP of the Transformer.
- Batch size is 512 for MOSES and 256 for text8.
- AdamW is used with learning rate 0.0005, betas 0.9/0.99, weight decay 0.001. Gradient clipping is set to 0.25. Both AO-ARM and MaM conditionals are trained for 1000 epochs for text8 and 200 epochs for MOSES. The MaM marginals are finetuned from the trained conditionals for 200 epochs.

Ising model and molecule generation with target property

- Ising model input are of $\{0, 1, ?\}$ values and are one-encoded as input to the neural network. The same is done for molecule SELFIES strings.
- MLP with residual layers, 3 hidden layers, feature dimension is 2048 for Ising model. 6 hidden layers, feature dimension 4096 for molecule target generation.
- Adam is used with learning rate of 0.0001. Batch size is 512 and 4096 for molecule target generation. ARM, GFlowNet and MaM are trained with $19,800$ steps for the Ising model. ARM and MaM are trained with $3,000$ steps for molecule target generation.
- Separate networks are used for conditionals and marginals of MaM. They are trained jointly with penalty parameter λ set to 4.

Compute

- All models are trained on a single NVIDIA A100. The evaluation time is tested on an NVIDIA GTX 1080Ti.

Additional results on Binary MNIST

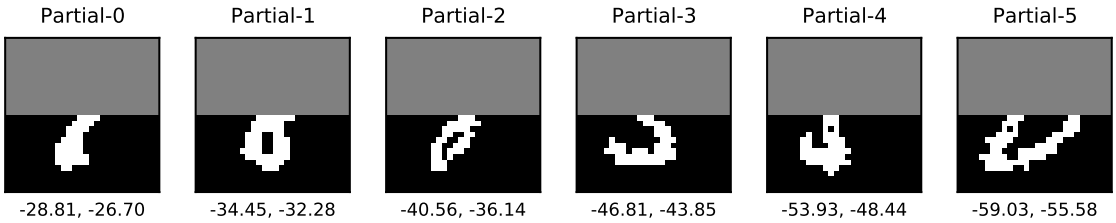


Figure 6.16: An example set of partial images for evaluating marginal likelihood estimate quality. The numbers in the captions show the log-likelihood calculated using learned marginals (left) v.s. learned conditionals (right)

Likelihood estimate on partial Binary MNIST images Figure 6.16 illustrates an example set of partial images that we evaluate and compare likelihood estimate from MaM against ARM. Table 6.12 contains the comparison of the marginal likelihood estimate quality and inference time.

Table 6.12: Performance Comparison on Binary-MNIST partial images

Model	Spearman's ↑	Pearson ↑	LL inference time (s) ↓
AO-ARM-E-U-Net	1.0	1.0	248.96 ± 0.14
AO-ARM-S-U-Net	**1.0**	**0.997**	49.75 ± 0.03
MaM-U-Net	0.998	0.995	**0.02 ± 0.00**

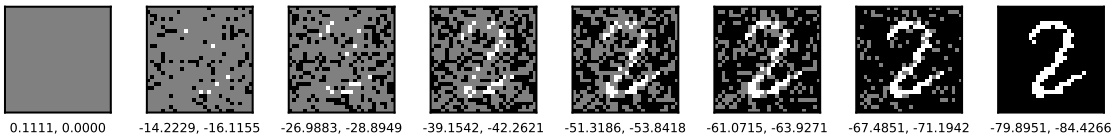


Figure 6.17: An example of the trajectory every 112 step when generating an MNIST digit following a random order. The future pixels are generated by conditioning on the existent filled-in pixels. The numbers in the captions show the log-likelihood calculated using learned marginals (left) v.s. learned conditionals (right)

Generated image samples Figure 6.17 shows how a digit is generated pixel-by-pixel following a random order. We show generated samples from MaM using the learned conditionals ϕ in Figure 6.18.

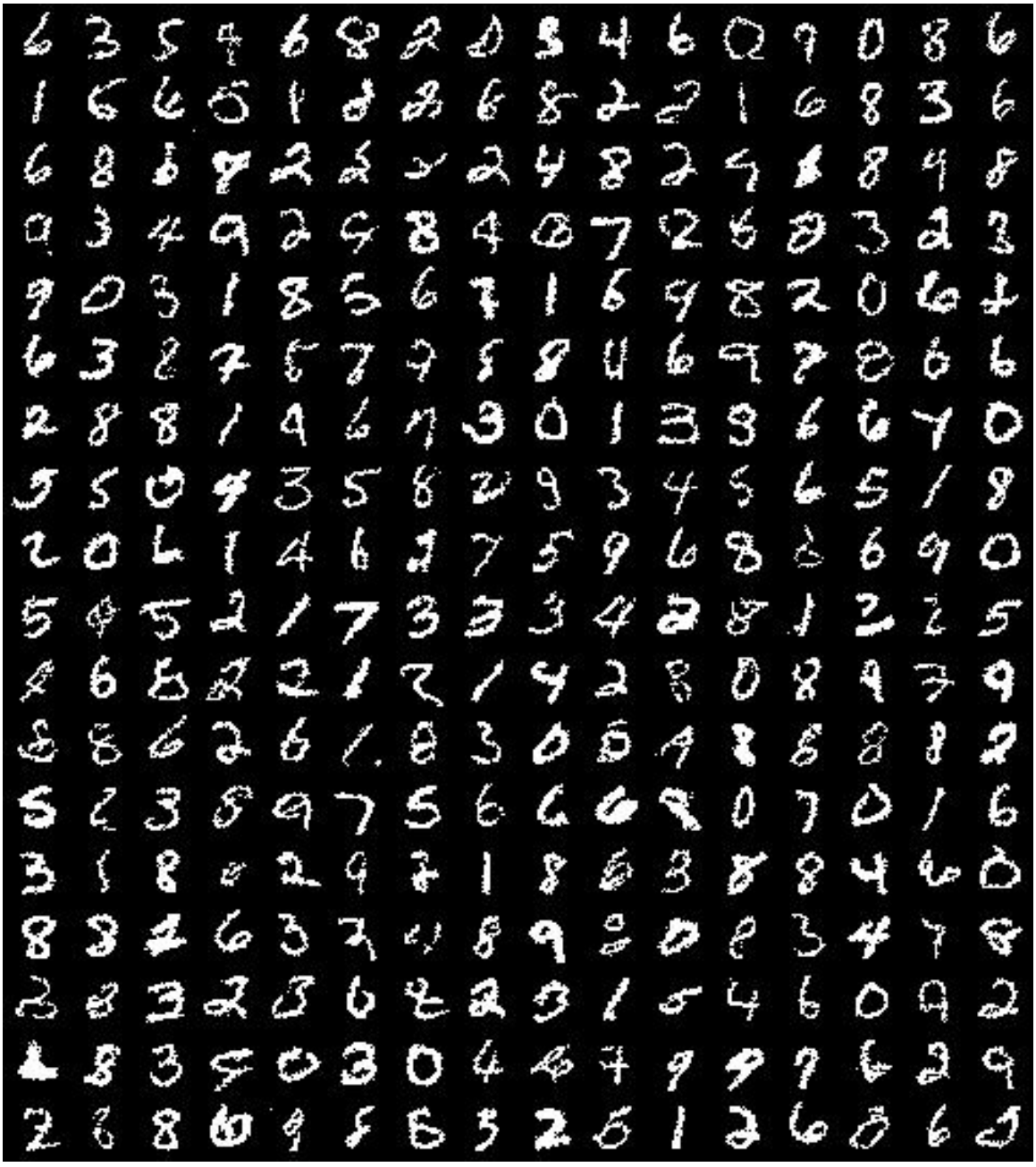


Figure 6.18: Generated samples: Binary MNIST

Table 6.13: Performance Comparison on MOSES

Model	**Valid↑**	**Unique 10k↑**	**Frag Test↓**	**Scaf TestSF↑**	**Int Div1↑**	**Int Div2↑**	**Filters↑**	**Novelty↑**
Train	1.0	1.0	1.0	0.9907	0.8567	0.8508	1.0	1.0
HMM	0.076	0.5671	0.5754	0.049	0.8466	0.8104	0.9024	**0.9994**
NGram	0.2376	0.9217	0.9846	0.0977	**0.8738**	**0.8644**	0.9582	**0.9694**
CharRNN	**0.9748**	0.9994	**0.9998**	**0.1101**	**0.8562**	**0.8503**	**0.9943**	0.8419
JTN-VAE	**1.0**	**0.9996**	0.9965	**0.1009**	0.8551	0.8493	**0.976**	0.9143
MaM-SMILES	0.7192	**0.9999**	**0.9978**	**0.1264**	0.8557	0.8499	**0.9763**	**0.9485**
MaM-SELFIES	**1.0**	**0.9999**	**0.997**	0.0943	**0.8684**	**0.8625**	0.894	0.9155

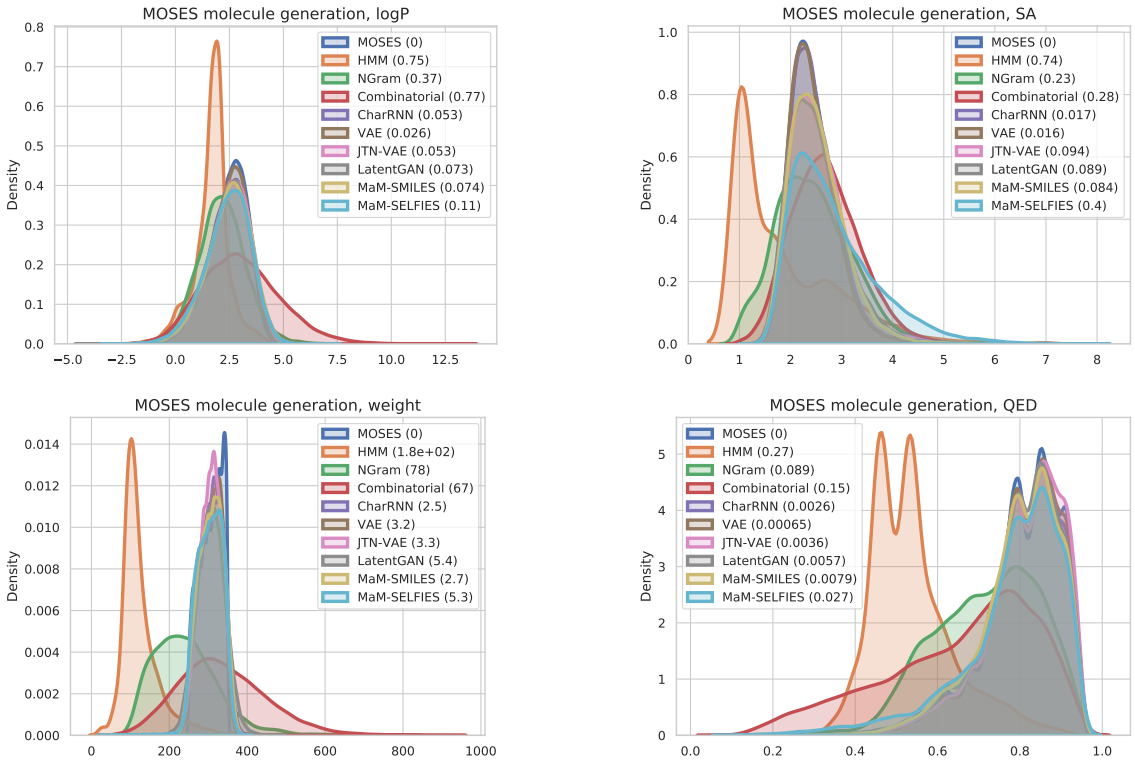


Figure 6.19: KDE plots of lipophilicity (logP), Synthetic Accessibility (SA), Quantitative Estimation of Drug-likeness (QED), and molecular weight for generated molecules. 30,000 molecules are generated for each method.

Additional results on MOSES

Comparing MaM with SOTA on MOSES molecule generation We compare the quality of molecules generated by MaM with standard baselines and state-of-the-art methods in Table 6.13 and Figure 6.19. Details of the baseline methods are provided in (Polykovskiy et al., 2020). MaM-SMILES/SELFIES represents MaM trained on SMILES/SELFIES string representations of molecules. MaM performs either better or comparable to SOTA molecule generative modeling methods. The major advantage of MaM and AO-ARM is that their order-agnostic modeling enables generation in any desired order of the SMILES/SELFIES string (or molecule sub-blocks).

Generated molecular samples Figure 6.20 and Figure 6.21 plot the generated molecules from MaM-SMILES and MaM-SELFIES.

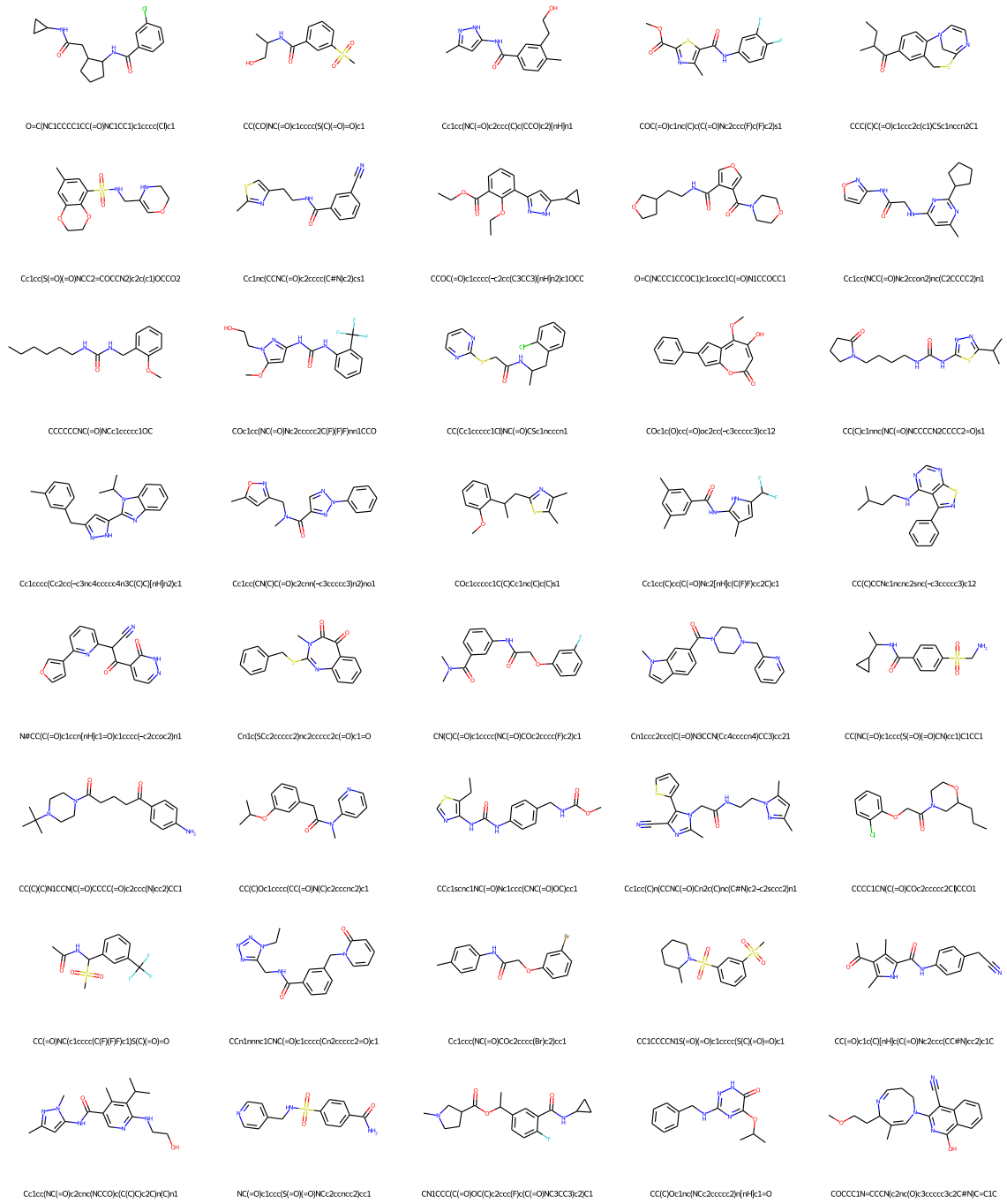


Figure 6.20: Generated samples from MaM-SMILES: MOSES

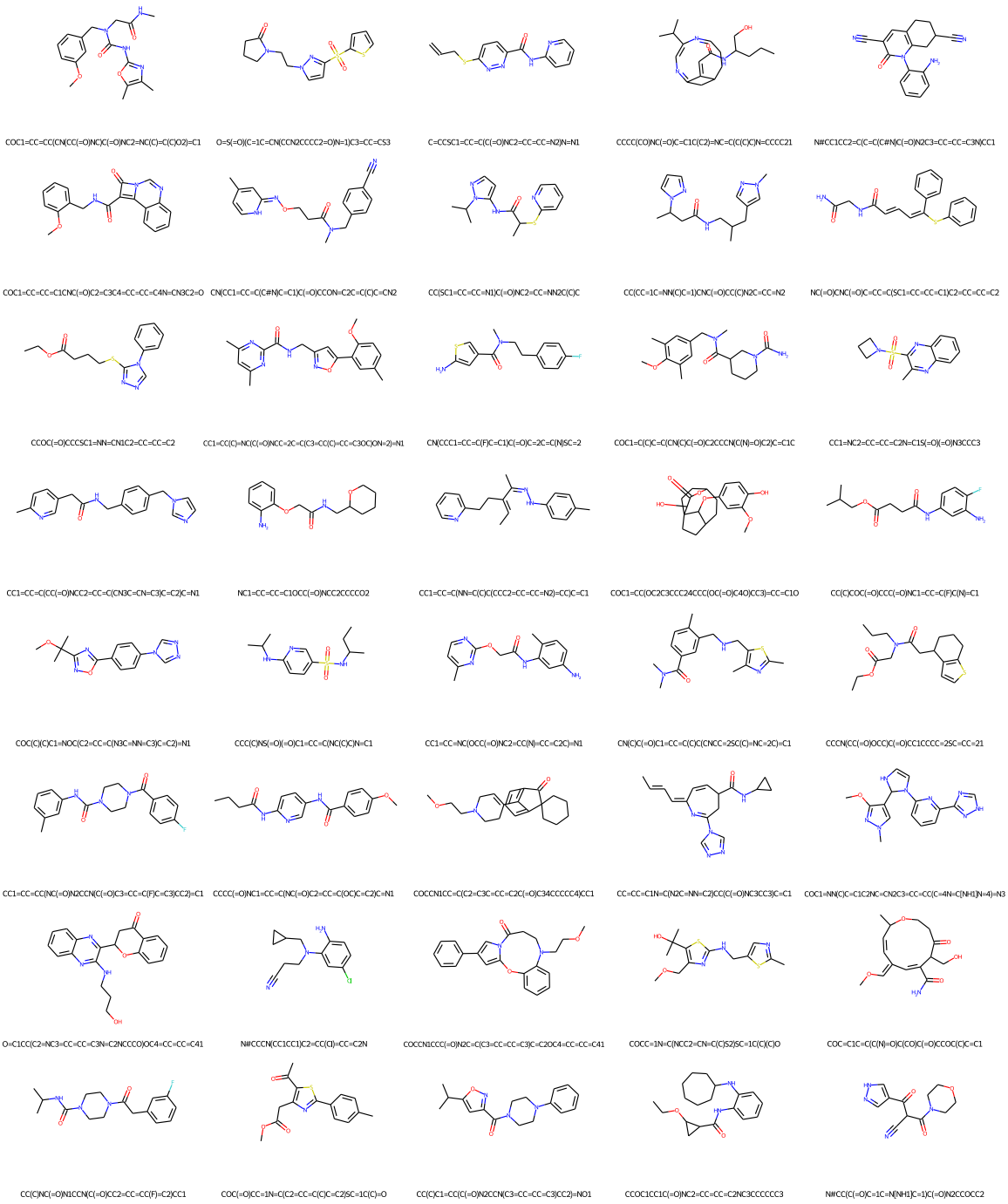


Figure 6.21: Generated samples from MaM-SELFIES: MOSES

Additional results on text8

Samples used for evaluating likelihood estimate quality We show an example of a set of generated samples from masking different portions of the same text, which is then used for evaluating and comparing the likelihood estimate quality. Their log-likelihood calculated using the conditionals with the AO-ARM are in decreasing order. We use MaM marginal network to evaluate the log-likelihood and compare its quality with that of the AO-ARM conditionals.

```
Original text:
the subject of a book by lawrence weschler in one nine nine five entitled mr
wilson s cabinet of wonder and the museum s founder david wilson received a
macarthur foundation genius award in two zero zero three the museum claims to
attract around six
```

```
Text generated from masking out 50 tokens:
 the_su_je_t of a b_ok_by_la_r_nce _es_h___ _n o___nine n_ne five entitled
mr_wilson s_cabinet of wonder and the museum s founder __vid w_l_o_ r__eive_
a macarthur fou__a__on _e___s_awa_d in two _ero z_r_ _hree _he museum c_aims
_o attr_ct ar_u_d s__
the subject of a book by lawrence heschell in one nine nine five entitled mr
wilson s cabinet of wonder and the museum s founder david wilson received a
macarthur foundation dennis award in two zero zero three the museum claims to
attract around sev
```

```
Text generated from masking out 100 tokens:
_the_su_je_t _f __b__k__y_l__r_nc_ _es_h_____n o___nine n_ne five_
enti_l_d mr_wil_o_ __c_b__et of wond_r an_ _h_ mu_eu_ s f_u_der___vid
_w_l_______eive_ a_maca_thur f_u__a___n _e_____a_a_d __ two _er_ z_r_
_h_ee____ museum c_a_ms__o __tr_ct ar_u__ ___
the subject of a book by lawrence bessheim in one nine nine five entitled mr
wilson s cabinet of wonder and the museum s founder david wilson received a
macarthur foundation leaven award in two zero zero three the museum claims to
detract around the
```

```
Text generated from masking out 150 tokens:
_the__u____t__f __________l__r_nc_ _es_h_____n o___n__e n_ne__ive_
e_ti_l__m__wil______c____et of won___ an___________ s___u_der___vid_
w_________eiv__ a__a_a_th________a___n _e_____a_______ two__e__ z_r_
___e_____ _use_m c_a_ms___ __tr_ct_a_____ ___
the tudepot of europe de laurence desthefs in one nine nine five entitled
mr wild the cabinet of wonder anne cedallica s founder david wright received
arnasa the culmination team sparked in two zero zero three the museum claims
to retract athlet c a
```

```
Text generated from masking out 200 tokens:
_t___________f __________l__r____ _______ _____o______e__n___iv__
e__i_l__ ____wil______c_____t___ w______a____________________der_____d_
w_____ ____e________a_a__________a___n_e_____a_______t___________
___e_____ _u_e___c_a__s___ ___r_c__a_____ ___
the builder of the pro walter a a e sec press one nine nine five esciele
the wild men convert of wark flax notes the world undergroand whirl spiken
america ascent and martin decree a letter to the antler s default museum
chafes in america ascent vis
```

Additional experiments on Ising model

Generated samples We compare ground truth samples and MaM samples in Figure 6.22 and Figure 6.23.

Additional experiments on molecule target generation

Target property distribution matching on lipophilicity (logP) Figure 6.24 and Figure 6.25 show the logP of generated samples of length $D = 55$ towards target values 4.0 and -4.0 under distribution temperature $\tau = 1.0$ and $\tau = 0.1$. For $\tau = 1.0$, the peak of the probability density (mass) appears around 2.0 (or -2.0) because there are more valid molecules in total with that logP than molecules with 4.0 (or -4.0), although a single molecule with 4.0 (or -4.0) has a higher probability than 2.0 (or -2.0). When the temperature is set to much lower ($\tau = 0.1$), the peaks concentrate around 4.0 (or -4.0) because the probability of logP value being away from 4.0 (or -4.0) quickly

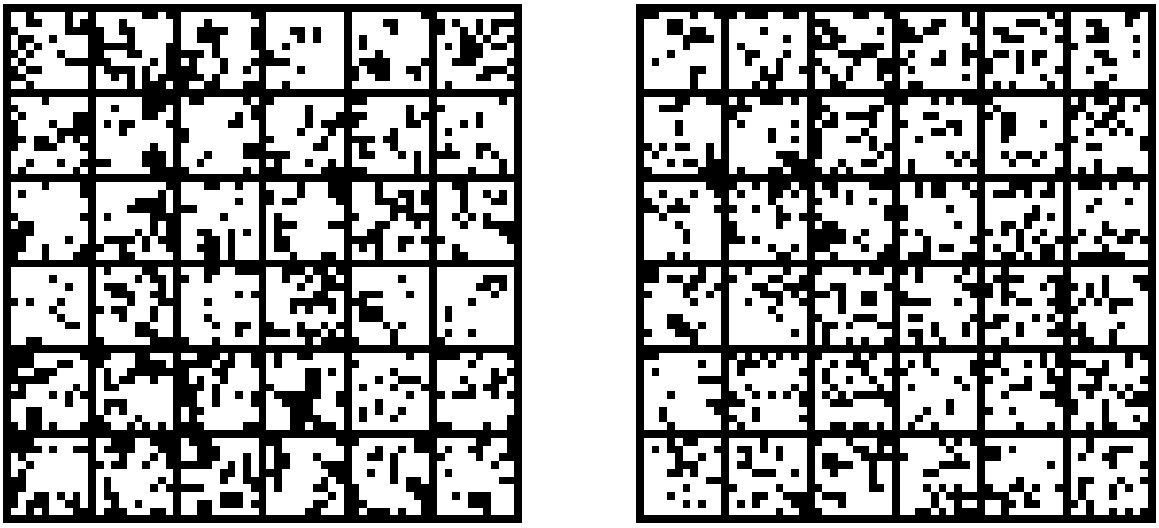


Figure 6.22: Samples: 10×10 Ising model. Ground truth (left) v.s. MaM (right).

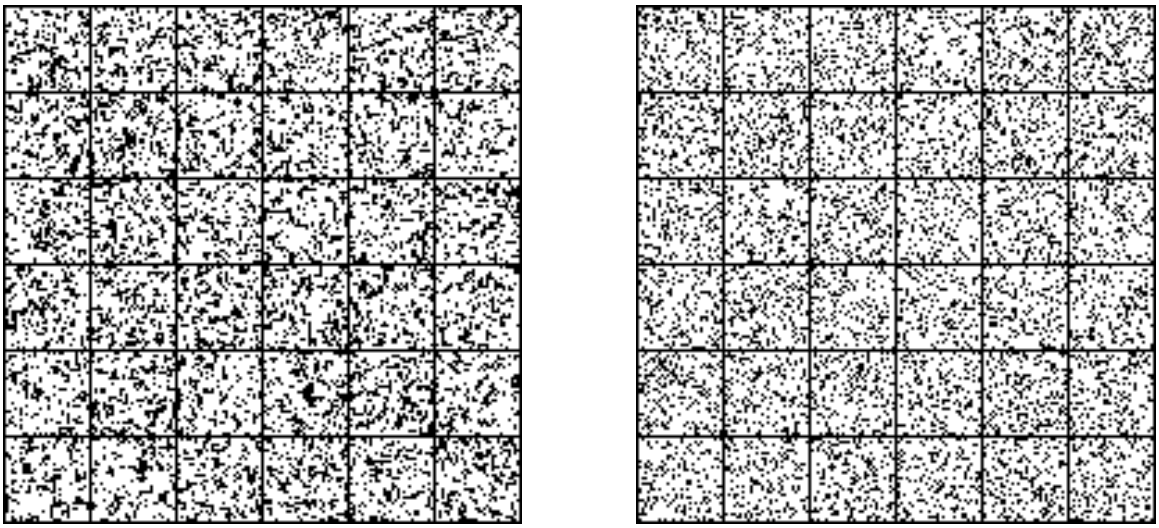
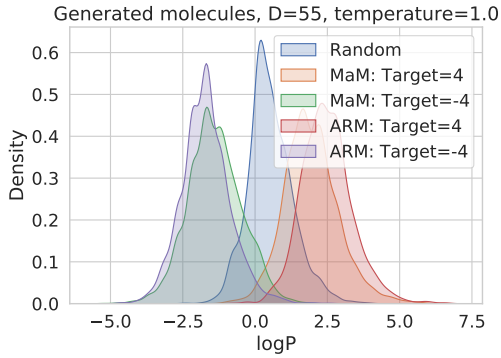


Figure 6.23: Samples: 30×30 Ising model. Ground truth (left) v.s. MaM (right).

diminishes to zero. We additionally show results on molecules of length $D = 500$. In this case, logP values are shifted towards the target but their peaks are closer to 0 than when $D = 55$, possibly due to the enlarged molecule space containing more molecules with logP around 0. Also, this is validated by the result when $\tau = 0.1$ for $D = 500$, the larger design space allows for more molecules with logP values that are close to, but not precisely, the target value.

Conditionally generated samples More samples from conditionally generating towards low lipophilicity (target $= -4.0$, $\tau = 1.0$) from user-defined substructures of Benzene. We are able to generate from any partial substructures with any-order generative modeling of MaM. Figure 6.26 shows conditional generation from masking out the left 4 SELFIES characters. Figure 6.27 shows



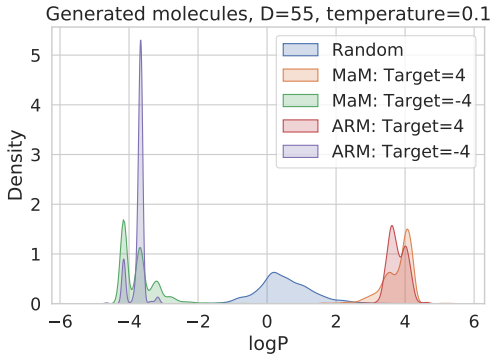
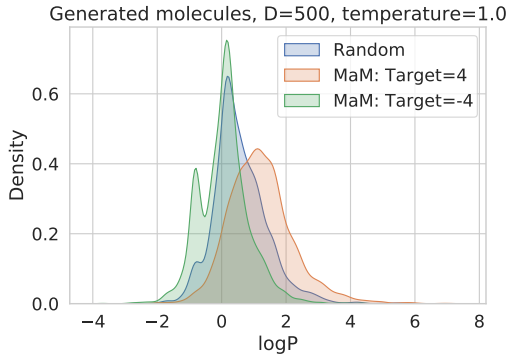


Figure 6.24: Target property matching with different temperatures. 2000 samples are generated for each method.



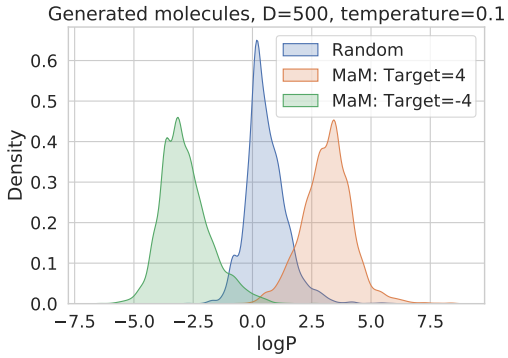


Figure 6.25: Target property matching with different temperatures. 2000 samples are generated for each method.

conditional generation from masking the right $4 \sim 20$ characters.

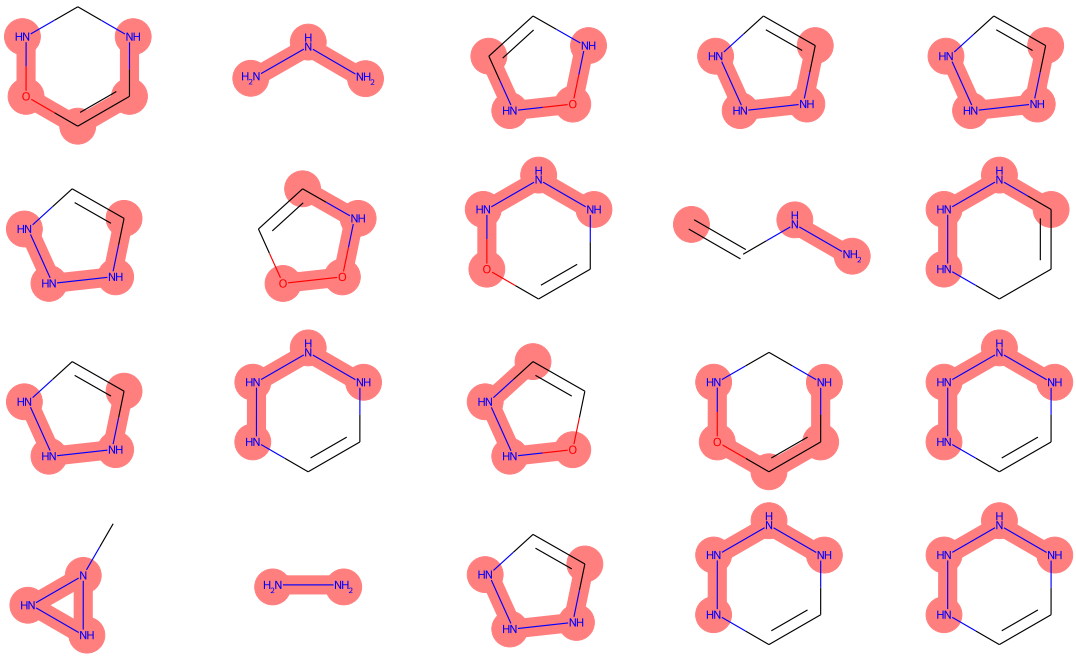


Figure 6.26: Generated samples from masking out the left 4 SELFIES characters of a Benzene.

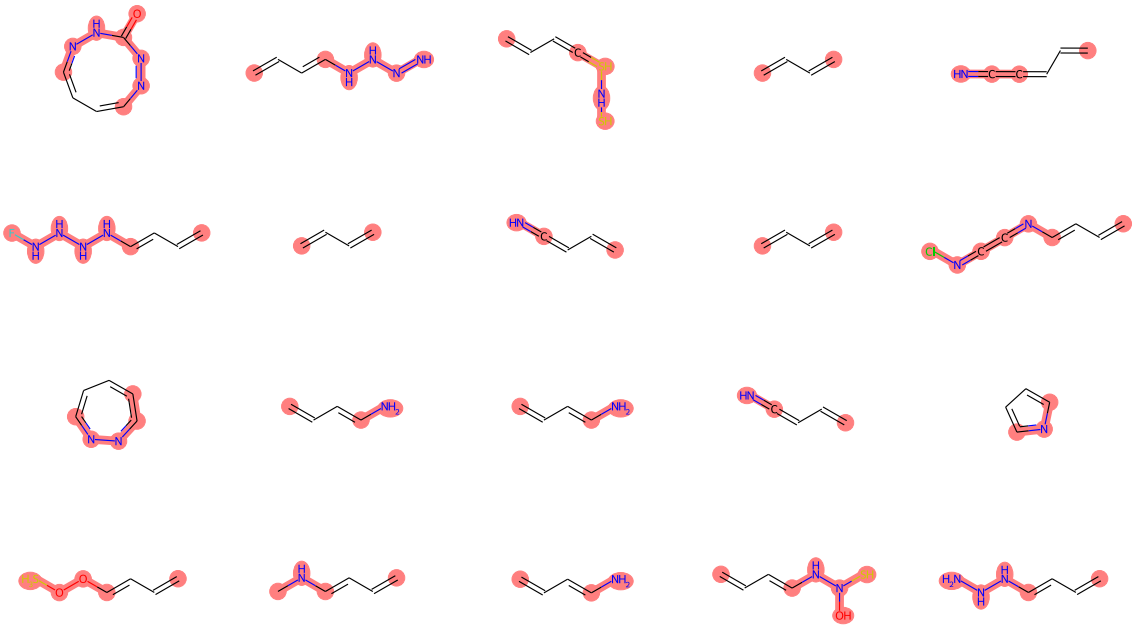


Figure 6.27: Generated samples from masking out the right 4-20 SELFIES characters of a Benzene.