Probabilistic Generative Models for Automatic Guided Drug Discovery

Yevgen Zainchkovskyy

Kongens Lyngby 2024

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

This thesis encompasses five distinct contributions presented as manuscripts, accompanied by two novel pieces of work detailed in Chapters 3 and 4. The latter has not been previously disseminated in any publications. The first chapter serves as an introduction to the field of computational drug discovery, offering an overarching perspective on the problem we aim to address and laying the groundwork for subsequent chapters.

In the second chapter, we provide a brief overview of Bayesian optimization, Gaussian Processes, and the concept of Pareto optimality. The third chapter introduces a novel concept: Distance aligning in latent spaces of Variational Autoencoders. Bringing together insights from the initial three chapters, Chapter 4 unveils the comprehensive model for automatic guided drug discovery, complemented by simulated experimental results.

Summary (Danish)

Denne afhandling omfatter fem distinkte bidrag præsenteret som manuskripter, ledsaget af to innovative værker beskrevet i Kapitel 3 og 4. Sidstnævnte er ikke tidligere blevet offentliggjort i nogen publikationer. Det første kapitel fungerer som en introduktion til feltet inden for beregningsbaseret lægemiddelopdagelse og giver en overordnet perspektiv på det problem, vi forsøger at løse, og lægger grundlaget for de følgende kapitler. I det andet kapitel giver vi en kort oversigt over bayesiansk optimering, gaussiske processer og konceptet om Pareto-optimalitet. Det tredje kapitel introducerer en ny idé - afstandstilpasning i latente rum af Variational Autoencoders. Ved at sammenfatte indsigter fra de første tre kapitler afslører Kapitel 4 den omfattende model for automatisk vejledt lægemiddelopdagelse, suppleret med simulerede eksperimentelle resultater.

Preface

This thesis was prepared at the Section for Cognitive Systems, DTU Compute, Technical University of Denmark in fulfillment of the requirements for acquiring a PhD degree at the Technical University of Denmark. The project was funded and executed under the Industrial PhD Programme in collaboration between the Technical University of Denmark and Novo Nordisk A/S. The project spanned from June 2020 to November 2023.

University supervisor	**Company supervisors**
Prof. Søren Hauberg	Carsten Stahlhut, Principal Data Scientist
DTU Compute	Jesper Ferkinghoff-Borg, Research Director
	Kilian W. Conde-Frieboes, Principal Scientist

Kongens Lyngby, July 1, 2024

Евген

Yevgen Zainchkovskyy

Acknowledgements

There is no way I could have completed this project without the world-class supervision I received. Always available, always insightful, always supportive and encouraging, I wholeheartedly thank my supervisors, Søren Hauberg, Carsten Stahlhut, Jesper Ferkinghoff-Borg, and Kilian W. Conde-Frieboes, for their guidance and support throughout the years.

I extend my sincere appreciation to my co-authors and colleagues at Novo Nordisk and DTU Compute. The collaborative and inspiring environment at both institutions has been nothing short of extraordinary, and I feel privileged to have been a part of it.

Finally, a heartfelt thanks to my family and friends, especially my parents, for their boundless love and unwavering support throughout this journey.

Contents

Chapter 1

Introduction

Advancements in computational methods have increasingly intersected with scientific research, leading to new innovative approaches in various fields, including genomics, climate modeling, astrophysics, and material science [HTT09]. One area that stands to benefit significantly from these developments is drug discovery. While effective, traditional methods are often time-consuming and resource-intensive. In response to these challenges, this thesis explores the potential of machine learning as a tool to enhance and streamline the drug discovery process. By leveraging data-driven algorithms and computational models the research aims to investigate how machine learning can contribute to more efficient and accurate predictions in drug development, ultimately leading to the discovery of novel therapeutic agents.

Drug discovery is a complex, expensive, and time-consuming process. It involves several steps, including target identification, lead discovery, lead optimization, preclinical trials, clinical trials, and regulatory approval [LC13]. The process can take up to 15 years and cost up to 2.6 billion USD per drug [AHO21]. The high cost of drug discovery is a major factor in the high cost to the patients.

Small-molecule drugs and protein-based drugs represent two major classes of therapeutic agents, each with distinct characteristics and applications. Small-molecule drugs are typically synthesized chemically and have a molecular weight of less than 900 daltons enabling the rapid diffusion across cell membranes to

reach intracellular sites of action [Veb+02]. Examples of small-molecule drugs include aspirin, ibuprofen, and statins. Protein-based drugs, the discovery of which we will focus on in this thesis, are a class of drugs that typically exert their effects extracellularly [LBG]. They are derived from living organisms or contain components of living organisms, such as proteins, antibodies, or nucleic acids. Unlike small-molecule drugs, protein-based drugs are often produced through recombinant DNA technology or other biotechnological methods. Examples of protein-based drugs include monoclonal antibodies, insulin, and vaccines.

The process of discovering protein-based drugs typically involves several key steps [LC13]:

- **Target Identification and Validation**:
  - **Target Identification:** Scientists identify biological molecules, often proteins, that are associated with a particular disease or condition. These molecules, known as targets, play a crucial role in the disease process.
  - **Target Validation:** Scientists validate the target by demonstrating that the target is involved in the disease process and that modulating the target will have a therapeutic effect.
- **Lead Discovery**:
  - Scientists search for compounds or molecules that can interact with the target and modulate its activity. This involves high-throughput screening of chemical libraries, virtual screening, and other approaches.
- **Lead Optimization**:
  - The identified leads are further optimized to enhance their efficacy, selectivity, and other pharmacological properties. This stage often involves medicinal chemistry, structural biology, and other techniques.
- **Preclinical Testing**:
  - Promising drug candidates undergo rigorous testing in laboratory and animal models to assess their safety, efficacy, and potential toxicities. This stage helps identify the most promising candidates for further development.
- **Clinical Trials**:
  - Drug candidates that pass preclinical testing enter clinical trials, a series of human trials designed to evaluate safety, efficacy, and dosing. Clinical trials have multiple phases, and they are time-consuming and resource-intensive.

The emphasis of this thesis is on the third step of the drug discovery process: **lead optimization**.

1.1 Lead Optimization in Drug Discovery

Lead optimization is a crucial phase in the drug discovery process where researchers focus on refining and improving the properties of initial drug candidates, known as leads. The primary goal during lead optimization is to enhance the efficacy, safety, and pharmacokinetic properties of the lead compounds to increase their chances of success in further development and clinical trials. It is a critical step in the drug discovery process, as it can significantly impact the success or failure of a drug candidate.

Historically, lead optimization has been an iterative and resource-intensive process, often requiring extensive medicinal chemistry efforts to modify the chemical structure of lead compounds. The goal of these modifications is to achieve a delicate balance between the desired biological activity and acceptable safety and pharmacokinetic profiles. This process involves the integration of various disciplines, including medicinal chemistry, pharmacology, toxicology, and computational modeling, to guide the design of more potent and selective compounds [War+15].

In the past, the lack of advanced screening technologies and computational tools meant that lead optimization relied heavily on empirical methods [Hug+11]. Researchers would synthesize and test numerous analogs of lead compounds, making incremental changes to their structures in the hope of improving their properties. While this approach occasionally led to successful drug candidates, it was often time-consuming and inefficient, with many promising compounds failing to meet the necessary criteria for further development.

However, the advent of high-throughput screening [PW07], combinatorial chemistry [LLL17], and computational modeling [SK23] has revolutionized the lead optimization process. These advancements have allowed researchers to explore a broader chemical space more efficiently, predict potential outcomes of structural modifications, and identify key molecular interactions that contribute to a compound's efficacy and safety. Modern lead optimization now integrates these cutting-edge techniques, enabling a more rational and systematic approach to drug design.

As the complexity of lead optimization continues to grow, the integration of advanced computational techniques has become essential for researchers work-

ing to enhance the efficiency and effectiveness of the drug discovery process itself. One such innovative approach is Bayesian optimization [Sha+16], which leverages statistical modeling to guide the exploration of chemical space more intelligently. By utilizing prior knowledge and adapting to new data, Bayesian optimization not only streamlines the identification of promising drug candidates but also facilitates the simultaneous optimization of multiple objectives—such as potency, selectivity, and safety.

This capability addresses the limitations of traditional lead optimization methods, allowing for a more systematic and data-driven approach to drug design. In the following, we briefly introduce the principles and applications of Bayesian optimization, highlighting its promise as a powerful tool for lead optimization in drug discovery. A more detailed discussion of Bayesian optimization and its application in lead optimization will be presented in the subsequent chapters.

1.2 Bayesian Optimization in lead optimization

Bayesian optimization (BO), a machine learning technique, can be a valuable tool in the lead optimization phase of drug discovery. It is a probabilistic model-based optimization technique that is used to find the maximum or minimum of an unknown, expensive, and black-box objective function. In the context of drug discovery, the objective function might represent the biological activity, binding affinity, or even a combination of properties of a lead compound.

The application of BO can be illustrated through several key use cases:

1. **Multi-Objective Optimization.** Lead optimization often requires simultaneous improvement of multiple, sometimes conflicting, objectives. For instance, enhancing potency may adversely affect selectivity or solubility. Bayesian optimization allows for the incorporation of multi-objective strategies that can identify a set of optimal trade-offs, often referred to as Pareto fronts, enabling researchers to choose candidates that best align with their development goals.

2. **Efficient Exploration of Chemical Space.** The high-dimensional nature of chemical space presents a significant challenge in lead optimization. Traditional methods can be inefficient, requiring extensive sampling to identify promising compounds. Bayesian optimization's ability to model uncertainty allows for targeted exploration, focusing

efforts on regions of the chemical space that are most likely to yield improvements, thus reducing the number of experiments needed.

3. **Integration with Experimental Data.** Bayesian optimization naturally incorporates experimental data into the optimization process. As new compounds are synthesized and tested, the surrogate model is updated, refining its predictions and improving future selections. This adaptability allows researchers to learn from past results, making the optimization process more robust and data-driven.

The incorporation of Bayesian optimization into lead optimization provides several advantages over traditional empirical methods. First, BO requires fewer evaluations to identify optimal candidates, significantly reducing the time and resources needed for lead optimization. Second, the probabilistic nature of BO allows for the quantification of uncertainty, providing researchers with valuable insights into the reliability of their predictions. This is specifically important as the biological activity of a compound is often noisy and subject to experimental error. Finally, BO improves the decision-making process by providing a systematic and data-driven approach to lead optimization. This final point is particularly relevant in the context of drug discovery in large pharmaceutical companies, where the decision-making process can be complex and involve multiple stakeholders and fragmented data sources.

1.2.1 State of the knowledge

Looking at recent related work, Bayesian optimization has proven effective in optimizing molecular properties, as seen in the work of Yang, Milas, and White [YMW22], who used BO in combination with deep ensembling of pre-trained sequence models to design peptide inhibitors with AlphaFold [Jum+21]. Cheng et al. [Che+24] suggested a streamlined, experimental design-focused closed-loop optimization framework for protein-directed evolution focusing on fitness maximization. This framework utilizes a novel low-dimensional protein encoding strategy and integrates Bayesian optimization with search space prescreening through outlier detection techniques. Stanton et al. [Sta+22] proposed a method to enhance the sequence of proteins, aiming to produce fluorescent proteins with improved brightness and stability. The study demonstrated that mutations generated by a corruption process and a denoising autoencoder, coupled with simulated stability and solvent-accessible surface area as optimization objectives, surpassing the efficacy of uniformly random mutations.

BO requires computing a gradient of the surrogate function wrt. the input sequence. This is problematic since the input is not continuous. Previously mentioned studies work around this limitation in various ways. Yang, Milas, and White [YMW22] expresses the sequence by categorical distribution parameterized with continuous logits. To effectively backpropagate a gradient through sampling from this distribution they use Gumbel-Softmax Trick [MTM15] with an addition of a trainable layer normalization inspired by Linder and Seelig [LS21]. Cheng et al. [Che+24] employs a combinatorial-based strategy (combinatorial mutagenesis) where they extensively enumerate combinations of the 20 amino acids conditioned on their fitness value. This, however, introduces an exponential increase in the number of samples which becomes problematic as Gaussian processes are hard to scale as the computational complexity of Gaussian Processes grows cubically with the number of data points [RW06]. To tackle *this* problem, an additional "Search Space Prescreening" step is added to prune out low-fitness candidates and effectively shrink the search space. In contrast, Stanton et al. [Sta+22] utilizes an encoder-decoder architecture and performs a mini-batched search in a continuous latent space. Their procedure is initiated by slightly corrupting starting sequences and encoding them to get the starting latent-space positions that are close to the observed samples. The search progresses by following the gradient of the acquisition function, while an entropy term constrains the process, preventing the search from becoming random.

1.3 Research Statement

All the works mentioned above are great examples of principled use of Bayesian Optimization in solving the task of novel protein design, however, a set of improvements are still desirable.

First, the categorical distribution parameterization used by Yang, Milas, and White [YMW22] is not ideal as it introduces a high variance in the gradient estimates while being a difficult high-dimensional optimization problem as the logits matrix is $p \times 20$ where p is the length of the sequence. Moreover, in their work, they use an LSTM-based Pre-trained Sequence Model, Unirep [All+19] while the current state-of-the-art protein language models (PLMs) are BERT-based [Dev+19] Transformers (like ESM2 [Lin+22]). Second, the combinatorial-based strategy used by Cheng et al. [Che+24] is computationally expensive and does not scale well with the length of the sequences, severely limiting its applicability for predictions far from the training data. Finally, the denoising-autoencoder architecture used by Stanton et al. [Sta+22] is not ideal for the task of protein design with Bayesian Optimization as it does not learn a true latent representation (generative model) of the data, but rather a representation that is useful for the recon-

struction of the input. The latter is a crucial distinction, as the interpolation between the latent space points is not guaranteed to be meaningful, and the model is not suitable for sampling from the latent space. We believe that a generative model that learns a meaningful latent space representation of the data is crucial for the task of protein design. Such a model can be formulated without a restrictive entropy term in the loss function, as the model itself should be able to provide meaningful in-distribution samples.

Summarizing the above, we see a gap in the current state-of-the-art methods for protein design with Bayesian Optimization. The methods are either computationally expensive, lack flexibility or do not scale well with the length or number of the sequences. In this thesis, we aim to address these limitations by proposing a novel VAE-based generative model that learns a meaningful, regularized, low-dimensional latent space representation of the data specifically tailored to Bayesian optimization while utilizing the state-of-the-art Protein Language Models (PLMs) as the backbone of the model. We will demonstrate the effectiveness of our approach through a series of experiments on real-world datasets, showcasing the improvements in efficiency and performance over existing methods.

1.3.1 Research Questions

To bridge the gaps identified in current methods for protein design using Bayesian Optimization, we have developed a set of focused research questions. These questions aim to guide the creation of a more effective and scalable framework, emphasizing the integration of state-of-the-art Protein Language Models (PLMs) and a robust latent space representation. Our research seeks to address existing challenges related to optimization complexity, scalability, and the limitations of current generative models, contributing to more reliable approaches in protein design. The following research questions are at the core of this investigation:

1. **Is it possible to design an efficient and scalable Bayesian optimization loop specifically tailored to protein sequences, accommodating arbitrary constraints and objectives?**

2. **Can we design a generative model that learns a meaningful, regularized, low-dimensional latent space representation of protein sequences for Bayesian optimization?**

3. **How can state-of-the-art Protein Language Models (PLMs) be leveraged to effectively guide the optimization process, expanding the search space beyond the immediate training data?**

The implications of this research potentially extend beyond the immediate improvements in protein design. By addressing the limitations of current methods, we hope to inspire applications in other domains that require efficient and flexible approaches to generative modeling and Bayesian optimization.

1.4 Included manuscripts

1. Guiding directed protein evolution with Bayesian Optimization

Written for a PhD course on Bayesian Optimization and Active Learning at DTU.

Yevgen Zainchkovskyy, September 2020.

↑ This manuscript was produced as part of a PhD course on Bayesian Optimization and Active Learning at DTU. It explores a somewhat naive application of Bayesian optimization within the latent space of a Variational Autoencoder (VAE).

While the approach shows some degree of informed exploration, the validation method is questionable, relying on a black-box proxy that selects the closest sequence in the test set. This work is included in the thesis due to its direct relevance to Research Question 2 and as an introduction to Bayesian optimization in the context of protein engineering. Moreover, it represents the initial steps that eventually led to the development of the methods presented in subsequent manuscripts and, ultimately, this thesis.

---

2. Robust uncertainty estimates with out-of-distribution pseudo-inputs training

2021-2022, Preprint, `arXiv:2201.05890`

Pierre Segonne, Yevgen Zainchkovskyy, Søren Hauberg

↑ Bayesian Optimization (BO) most often relies on Gaussian Processes (GPs) as surrogate models to represent the underlying unknown objective function. However, GPs are not necessarily always the best choice as they scale poorly with the number of observations and often require custom kernels to be designed for each problem. This lack of scalability makes GPs less suitable for tasks involving large and complex datasets, such as protein sequence optimization, relating to "scalability" in Research Question 1. Neural networks, on the other hand, while powerful and versatile for various machine learning tasks, often struggle with accurately estimating uncertainty. This limitation is particularly relevant in applications where understanding the model's confidence or uncertainty is crucial, such as in Bayesian optimization for drug discovery.

In this work, we explicitly trained the uncertainty predictor of a neural network in regions where there is no available data, showing that it is possible to obtain better uncertainty estimates in such regions. We demonstrate the effectiveness of the proposed approach on a number of synthetic and real-world datasets.

Specific contribution: Extending the method for Bernoulli likelihoods, collecting, analyzing, and aggregating the results of the experiments, preparation, and minor editing of the manuscript.

---

3. A kernel for continuously relaxed, discrete Bayesian optimization of protein sequences

Machine Learning for Structural Biology Workshop, NeurIPS 2021.

Yevgen Zainchkovskyy, Simon Bartels, Søren Hauberg, Jes Frellsen, Wouter Boomsma

↑ This work directly addresses Research Question 1 by exploring a novel GP kernel for Bayesian optimization of protein sequences. In line with our goal to design an efficient and scalable Bayesian optimization loop for protein sequences (RQ1), we developed a kernel that acts on the likelihood (modeled as a Categorical discrete probability distribution) of the VAE decoder, rather than performing the search in the VAE latent space directly.

The approach is based on the idea of using the Jensen-Shannon divergence (symmetrized and smoothed version of KL-Divergence) between represented protein sequences to establish a covariance function over the latent representation. Our initial experiments suggest that this kernel, generally, performs better than the traditionally used RBF kernel in the VAE latent space, but further investigation is required to establish its effectiveness.

Specific contribution: Numerical optimization and implementation of the method. Running the experiments, collecting and analyzing the results, preparation, and minor editing of the manuscript for submission. Making the illustrative figures, plots, and posters for the workshop presentation.

---

4. Probabilistic thermal stability prediction through sparsity promoting transformer representation

Workshop on Robustness in Sequence Modeling, NeurIPS 2022

`arXiv:2211.05698v1`

Yevgen Zainchkovskyy, Jesper Ferkinghoff-Borg, Anja Bennett, Thomas Egebjerg, Nikolai Lorenzen, Per Jr. Greisen, Søren Hauberg, Carsten Stahlhut

↑ Emerging from an internal Novo Nordisk competition, the method introduced in this work presents a twist on the traditional way of using PLM embeddings for protein-related prediction tasks. Instead of typical averaging across the transformer's hidden representation (mean pooling), we learn a weight mask that enables focus on the most relevant parts of the sequence for a given task. While the idea is simple, it is surprisingly effective and directly applicable in the context of Bayesian optimization, addressing aspects of Research Questions 1 and 3.

This exact method was later used in the generative model presented in Chapter 4, partially addressing Research Question 2.

Specific contribution: Formulation and implementation of the method. Running the experiments, collecting and analyzing the results, preparation, and editing of the manuscript for submission to a conference. Making illustrative figures, plots, and posters for the presentation.

---

5. All Layers Marginal Likelihood Training with Fully Correlated Linearized Laplace Approximations

`arXiv:2304.08309`

Hrittik Roy, Marco Miani, Stas Syrota, Johan Ye, Yevgen Zainchkovskyy, Silas Brack, Frederik Warburg, Nicholas Krämer, Pablo Moreno-Munoz,

> Søren Hauberg

↑ This submission was an enormous team effort, and while it is not immediately applicable to drug discovery, Marginal Likelihood training and the resulting tractable posterior approximation are key ingredients for neural networks that generalize well and provide accurate uncertainty estimates among other desirable properties.

In this context, linearized-Laplace approximation (LLA) is theoretically compelling because it can be viewed as a Gaussian process posterior, where the mean function is determined by the neural network's maximum-a-posteriori predictive function, and the covariance function is generated by the empirical neural tangent kernel. Kristiadi et al. [Kri+23] demonstrated that the method is particularly effective for Bayesian optimization, showcasing its superior performance over standard Gaussian processes when used as surrogate models, which relates to our Research Question 1.

As the proposed method relies heavily on Jacobian vector products and other non-trivial transformations, the pivotal side-effect of working on this project was learning JAX [Bra+18] - a powerful library for automatic differentiation and numerical computing in Python. The utilization of this library empowered us to develop a streamlined custom Bayesian optimization loop (Chapter 2) and the rest of the generative model (Chapter 4).

Specific contribution: Efficient implementation of the method and numerical optimization of the bound. Design and execution of large scale RESNET experiments, and minor editing of the manuscript.

1.5 Outline

In the next chapter, we briefly refresh the reader's memory on the basics of Bayesian Optimization and Gaussian Processes. We will then introduce the problem of multi-objective optimization and the concept of Pareto optimality. In the end, we will present our custom BO implementation, specifically suitable for application in free-form latent spaces and empirically show its improved efficacy, directly addressing Research Question 1.

The story continues in Chapter 3, where latent spaces of variational autoencoders are subjected to a distance-aligning technique. The result is a latent space tailored to enhance the efficiency of Bayesian optimization, directly contributing

to Research Question 2. We conclude again with an empirical evaluation of the proposed method in a real-world BO scenario.

Chapter 4 serves as the culmination of our research, addressing all Research Questions in a cohesive framework. Here, we introduce an innovative end-to-end generative model for protein sequences that merges state-of-the-art Protein Language Models (PLMs) with the distance aligning technique developed in Chapter 3. We validate its effectiveness and present the results of a BO experiment on a real-world benchmark.

Finally, in Chapter 5, we summarize the contributions of this thesis and discuss potential future research directions.

> It is worth highlighting that, to the best of our knowledge, the methodologies detailed in Chapters 3 and 4 represent original research that has not been previously published or disseminated in academic literature.

Chapter 2

Bayesian Optimization

Bayesian optimization [Sha+16] is a probabilistic model-based approach for optimizing black-box functions, which are functions for which we have no explicit form or gradient information. When optimizing, let's say, the effectiveness of a drug candidate, the black-box function is nature itself. Trying to model "that function" from first principles is a daunting task as the effectiveness of a drug is determined by a set of very complex and multifaceted relations that depend on numerous molecular and cellular processes. Essentially, we can only observe a noisy effect of the drug candidate.

Let us start with the notion of a drug candidate first and try to present it in a more formal, computable form. We mentioned previously that we focus on protein-based drugs, also called biologics. Proteins are typically large molecules that are produced in living cells from a DNA template (excuse our embarrassingly simplified description of the process). The DNA template is a sequence of nucleotides, which are the building blocks of DNA. The sequence of nucleotides is translated into a sequence of amino acids, which are the building blocks of proteins. The sequence of amino acids determines the structure of the protein, which in turn determines its function and it is this sequence of amino acids that we work with in the computer.

20 standard natural amino acids that are commonly found in proteins and every position in the protein sequence can potentially be occupied by any of the 20

amino acids. This means that the number of possible protein sequences is 20^n, where n is the length of the protein sequence. For a protein of length 100, this means that there are $20^{100} \approx 1.26 * 10^{130}$ possible sequences. To put this number into perspective, there are $\sim 10^{80}$ atoms in the observable universe.

Clearly, we are not interested in all possible protein sequences, but only in those that have some desired properties. This makes the search space much smaller, but still very large. Typically, we would start with a *wildtype* protein sequence, which is the sequence of the protein found in nature. We then "mutate" the wildtype sequence by changing one or more amino acids at specific positions in the sequence. This allows us to explore the sequence space around the wildtype and find similar sequences that have, or increase a desired property. The combinatorial explosion of the search space is then controlled by the number of mutations that we allow. Bayesian Optimization will help us select the best mutant sequences to test in the lab without having to enumerate all possible mutants.

The key idea behind Bayesian optimization is to use a probabilistic surrogate model to model the unknown objective function and iteratively choose new points to evaluate based on a trade-off between exploration and exploitation. The surrogate model is typically a Gaussian process (GP) which we will briefly describe next.

2.1 Gaussian Processes

A Gaussian Process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution. In the context of Bayesian optimization, a GP is used to model an unknown objective function $f(x)$, where x is the input variable. We write

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

where $m(x)$ is the mean function, representing the expected value of $f(x)$ and $k(x, x')$ is the covariance function, also called the kernel function. The kernel function determines the covariance between the function values at two points x and x'. The covariance function is typically parametrized by a set of hyperparameters θ (although we typically don't write $k(x, x'; \theta)$ to denote this dependency). It is the fitting of the hyperparameters that we refer to as "training the GP". This is done by maximizing the marginal likelihood of the observations. The marginal likelihood is given by:

$$p(Y|X, \theta) = \int p(Y|f, X, \theta)p(f|X, \theta)df$$

and for Gaussian likelihood and prior, the marginal likelihood has a closed form solution (note we assume zero mean function and no noise for simplicity):

$$\log p(Y \mid X, \theta) = -\frac{1}{2} Y^T K^{-1} Y - \frac{1}{2} \log|K| - \frac{n}{2} \log 2\pi \tag{2.1}$$

where Y is the vector of observations, X is the vector of inputs and K is the covariance matrix of the observations.

Before observing any data, our belief about the function $f(x)$ is described by the prior distribution,

$$f(x) \sim \mathcal{N}\left(m(x), k\left(x, x'\right)\right)$$

where we often assume the mean function $m(x) = 0$. Given observed data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, we want to update our belief about $f(x)$. This updated belief is represented by the posterior distribution.

The joint distribution of the observed values y and the function values at new test points X_* is Gaussian:

$$\begin{bmatrix} Y \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} K(X,X) + \sigma^2_{\text{noise}} I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right)$$

where $K(X, X)$ is the covariance matrix of the training points, $K(X, X_*)$ is the covariance matrix between the training points and the test points, and $K(X_*, X_*)$ is the covariance matrix of the test points. The noise variance σ^2_{noise} is typically a (learned) hyperparameter of the GP and is used to model the noise in the observations.

The **posterior predictive distribution** of the function values at new test points X_* is derived by conditioning the joint distribution on the observed data:

$$f_* \mid X_*, X, Y \sim \mathcal{N}\left(\mu(X_*), \sigma^2(X_*)\right)$$

where

$$\mu(X_*) = m(X_*) + K(X_*, X)\left[K(X,X) + \sigma^2_{\text{noise}} I\right]^{-1}(Y - m(X))$$
$$\sigma^2(X_*) = K(X_*, X_*) - K(X_*, X)\left[K(X,X) + \sigma^2_{\text{noise}} I\right]^{-1} K(X, X_*)$$

The mean $\mu(X_*)$ is the prediction of the function values at the test points and the variance $\sigma^2(X_*)$ is the uncertainty of the prediction.

As the kernel determines the shape and characteristics of the functions in the GP, the choice of kernel function is crucial. Commonly used kernels include the Radial Basis Function (RBF) or Gaussian kernel, Matérn kernel, and others. The RBF kernel is given by:

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2l^2}(x - x')^2\right)$$

where σ_f^2 is the signal variance and l is the lengthscale. The signal variance determines the amplitude of the function values and the lengthscale determines the smoothness of the function. The Matérn kernel is given by:

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

where σ_f^2 is the signal variance, l is the lengthscale, ν is the smoothness parameter and K_ν is the modified Bessel function of the second kind. The Matérn kernel is more flexible than the RBF kernel as it allows for non-integer values of ν. The RBF kernel is a special case of the Matérn kernel with $\nu \to \infty$.

Key takeaway The GP is a distribution over functions, capturing our belief about what the true function might look like.

For a more detailed introduction to Gaussian Processes, we recommend the book by Rasmussen and Williams [RW06].

2.2 Bayesian Optimization Loop

Having access to a probabilistic model of the objective function, we can now start the Bayesian optimization loop. The loop consists of three steps:

1. **Modeling** - Fit a GP to the observations of the objective function.
2. **Acquisition** - Select a new point to evaluate based on the GP.
3. **Evaluation** - Evaluate the objective function at the selected point.

The first step is to fit a GP to the observations of the objective function according to equation 2.1. The second step is to select a new point to evaluate based on the GP. This is done by maximizing an acquisition function, which is a function of the GP. Acquisition functions typically balance exploration and exploitation. Exploration means exploring the space to find new promising regions, while exploitation means exploiting the current best region. Common acquisition functions include:

- Probability of Improvement (PI) [JSW98]:

$$\text{PI}(x) = P(f(x) \geq \text{best\_observed\_value})$$

- Expected Improvement (EI) [MTZ78]:

$$\text{EI}(x) = \mathbb{E}\left[\max\left(f(x) - \text{best\_observed\_value}, 0\right)\right]$$

- Upper Confidence Bound (UCB) [Sri+12]:

$$\mathrm{UCB}(x) = \mu(x) + \kappa\sigma(x)$$

where $\mu(x)$ and $\sigma(x)$ are the mean and standard deviation of the GP at x and κ is a hyperparameter that controls the exploration-exploitation trade-off. Given the acquisition function, the third step is to evaluate the objective function at the optimized point:

$$x_{\text{next}} = \arg\max_x \text{acquisition}(x)$$

The new observation is then added to the set of observations i.e.

$$\mathcal{D} = \mathcal{D} \cup \{(x_{\text{next}}, y_{\text{next}})\}$$

and the loop is repeated until some stopping criterion is met. The stopping criterion is typically a maximum number of iterations (budget) or a maximum number of consecutive iterations without improvement.

> **Key takeaway** Bayesian optimization is a model-based optimization approach that uses a probabilistic model of the objective function to guide the search for the optimal solution. The probabilistic model is typically a Gaussian Process and the search is guided by an acquisition function that balances exploration and exploitation.
>
> For a more detailed introduction to Bayesian Optimization, we recommend the article by Shahriari et al. [Sha+16].

2.3 Pareto optimal optimization

In the previous section, we described the standard Bayesian optimization loop. However, in the context of drug discovery, we are typically interested in optimizing multiple objectives simultaneously. For example, we might want to optimize the binding affinity of a drug candidate to a target, while at the same time minimizing the binding affinity to a different target. These objectives conflict with each other, in the sense that improving one objective might degrade another. The goal is to find a set of solutions that represent a trade-off between the conflicting objectives and are not dominated by any other feasible solutions.

A solution x is said to dominate another solution x' if it is better in all objectives and strictly better in at least one objective. A solution is Pareto optimal if it is not dominated by any other solution. Finally, the set of all Pareto

optimal solutions is called the Pareto front. Commonly used acquisition functions for Pareto optimal Bayesian Optimization include the Hypervolume-based Expected Improvement (EHVI, [Yan+19]) and the ParEGO [Kno06] algorithm.

In this work, we utilize the qEHVI acquisition function by Daulton, Balandat, and Bakshy [DBB20] which extends EHVI to the parallel, evaluation setting computing the joint EHVI for **q** new candidate points. This is also very important in drug discovery as we typically evaluate multiple mutants in parallel in the lab.

> We will refrain from describing the details of qEHVI algorithm here, and instead refer the reader to the original paper and an excellent blog post by the authors [Sam20].

When first learning about Pareto optimal optimization, my initial reaction was that it was a needless complication. Why not just combine the objectives into a single function? For example, given two objectives $f_1(x)$ and $f_2(x)$, where $f_1(x)$ could model an expected improvement of binding affinity and $f_2(x)$ could be thermo-stability, we could write the combined objective as:

$$f(x) = \alpha f_1(x) + (1 - \alpha) f_2(x)$$

where α is a hyperparameter that controls the trade-off between the two objectives.

This is called *weighted sum* and is a perfectly valid approach. However, it has a number of drawbacks. First, it requires the user to specify the weights α for each objective. This in itself is not trivial and requires domain knowledge and a lot of trial and error as one objective might be easier to optimize than the other. Second, it is not possible to explore the Pareto front with this approach. The Pareto front is a set of solutions that represent a trade-off between conflicting objectives. This means that there is no single solution that is better than all other solutions in all objectives. This is not possible with weighted sum optimization as it always selects the solution with the highest weighted sum (unless of course, we vary the weights across rounds). Additionally, in drug discovery, exploring the Pareto front is valuable as it allows chemists to better understand the effect of different mutations on separate objectives.

In figure 2.1 we visually demonstrate the difference in outcome between weighted sum and Pareto optimal optimization.

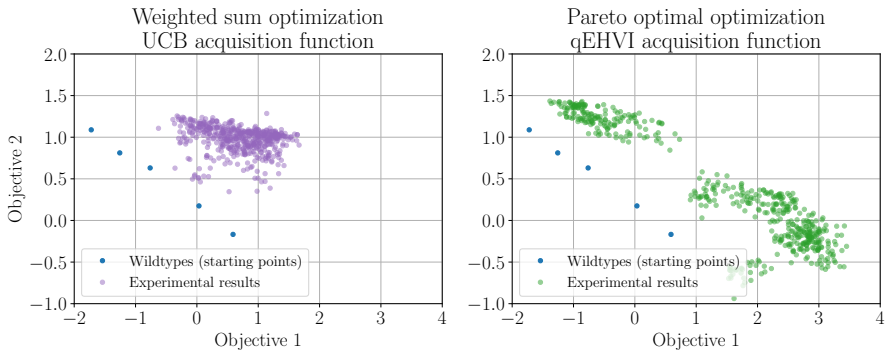

Figure 2.1: Experimental outcome of **(left)** Weighted sum optimization. **(right)** Pareto optimal optimization.

While, for both examples, the task is to maximize both objectives simultaneously (i.e. find a solution that both binds to the target and has high stability), the weighted sum approach will always select the solution with the highest weighted sum, while Pareto optimal optimization will explore the Pareto front and find a set of solutions that represent a trade-off between the two objectives.

2.4 Implementation detail

Conventional Bayesian Optimization implementation, as the one used in BoTorch [Bal+20] by default, is typically based on a multiply-restarted L-BFGS(-B) [Byr+95] approach: Each iteration of the BO loop will start from a set of uniformly distributed points in the domain of the function, and iteratively move the points towards some local maxima. The procedure is repeated `n-restart` times and in the end, the best point (the one having the highest value of the acquisition function) is selected.

L-BFGS(-B) works well for smaller problems but has a number of drawbacks. First, the number of effective points (`t-batch` size, in the lingo of BoTorch) is limited by the scalability of L-BFGS(-B). While much better than the standard BFGS algorithm (having a complexity of $O(n^2)$ in the number of parameters), it still requires storing a set of vectors representing the approximation of the inverse Hessian matrix ($n \times n$) alongside some history information about the previous updates. Second, when used with on a Monte-Carlo-based acquisition function, such as qEHVI, L-BFGS(-B) sometimes moves the points below the Pareto

front, making them "stuck" and rendered useless for further exploration. For our problem, we have also observed that a lot of points quickly converge to a local maximum and are not able to escape it as demonstrated in fig. 2.2.

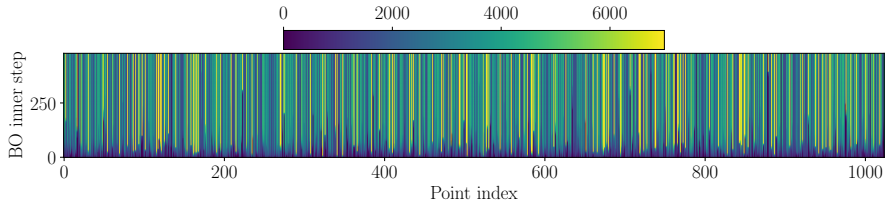

Figure 2.2: Inner view of optimization in a single BO round. We track 1024 points and see their progress over 256 steps starting from step 1 (bottom) to step 512 (top). The points are colored according to the value of the acquisition function at the respective step.

Implementing a partial reset mechanism that would allow moving "stuck" points back to the Pareto front is not trivial as libraries such as BoTorch, merely call `scipy.optimize` for L-BFGS(-B) and do not provide hooks to modify the underlying behavior of the optimizer.

> BoTorch also provides a gradient based optimizer (based on the `torch.optim` package) which has some callback functionality. However, the functionality is limited and not sufficient for our purposes as the callback function only recieves the summed loss and gradients as parameters [a]

[a] http://github.com/pytorch/botorch/blob/85ccd2d/botorch/generation/gen.py#L379

To overcome the limitations of the BoTorch library and improve the performance and scalability of the Bayesian optimization procedure, we have implemented a custom BO loop based on highly-performant JAX library [Bra+18]. This gives full control and flexibility to add custom logic and heuristics to the inner workings of the optimization. Under the hood, our implementation uses a gradient-based method (Adam optimizer), does not depend on multiple restarts and trivially scales to a large number of starting points.

We demonstrate the effect of the proposed partial reset logic in fig. 2.3. Here we reset a fraction (q-quantile) of worst-performing points and quantify the effect by looking at the mean of the final acquisition function values for 2048 points across 10 trials.

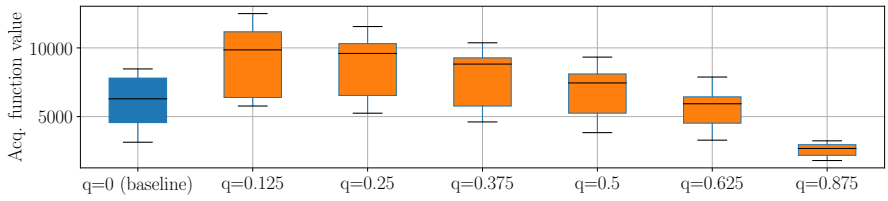

Figure 2.3: The effect of partial reset on the final acquisition function values. We reset a fraction (q-quantile) of worst-performing points and quantify the effect by looking at the mean of the final acquisition function values.

We would like to note that we do not claim any special novelty in the presented approach as the same effect is achievable with the classic multiple-restart strategy. However, we did find that our approach is more efficient, especially in the later rounds of optimization, where gradients become sparse and high acq. value areas are increasingly hard to find. This is expected, as our approach allows us to initiate some parts of the optimization process anew without waiting until the end of the round.

Our exploration of Bayesian optimization in the context of protein engineering has revealed its potential for efficiently navigating the vast sequence space to discover novel and improved protein variants. By addressing Research Question 1 - **"Is it possible to design an efficient and scalable Bayesian optimization loop specifically tailored to protein sequences, accommodating arbitrary constraints and objectives?"** - we have developed a custom BO implementation that overcomes limitations of conventional approaches. This implementation, leveraging the JAX library, offers improved scalability and flexibility, particularly in handling multiple objectives through Pareto optimization. However, the effectiveness of Bayesian optimization is intrinsically tied to the quality of the underlying latent space representation. To further enhance the efficiency of our BO process, we now turn our attention to the challenge of creating more informative latent spaces. In the following chapter, we will explore a novel approach to distance aligning in latent spaces, which promises to group similar proteins more effectively and potentially lead to more efficient exploration during the optimization process.

Chapter 3

Distance aligning in latent spaces

Ever since starting the project and seeing visualizations of the protein embeddings in latent spaces, the idea of using distances to guide the search has been in the back of my mind. Intuition is: if we can find a way to group similar proteins (in terms of, say, stability) together in the latent space, we would be able to search the space more efficiently.

Consider two 1D latent spaces in fig. 3.1 where we present two hypothetical scenarios. In the space on the left, which could be the latent space of an ordinary VAE [KW22], the samples are placed according to some similarity of the reconstructions. In the space on the right, the samples are grouped together according to their distances in the *target*-space. If we were to use gradient-based methods and optimize the proteins in the space on the left, we would arrive at the local maxima, and would not find the highest-scoring protein. In contrast, if we were to optimize the proteins in the space on the right, we would have a better chance of finding the highest-scoring protein as all of the proteins in the space are grouped together according to their *target*-space distances.

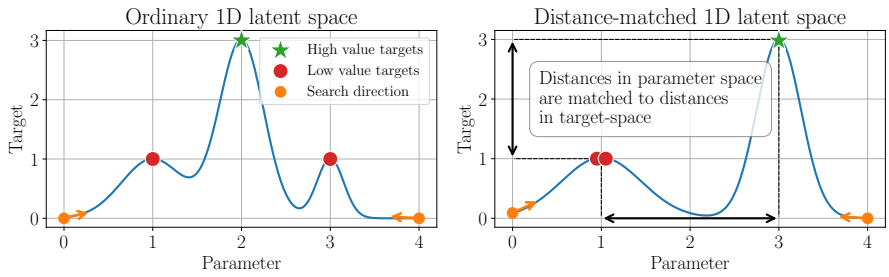

Figure 3.1: Hypothetical 1D latent spaces. **(left)** Ordinary VAE latent space where samples are encoded according to their reconstruction-similarity (and a prior). **(right)** Distance-aligned latent space where samples are additionally aligned to have matching distances in *target*-space.

The method we are going to present in the following will allow exactly that. It is naturally pluggable into VAE-based models and does not require any changes to the model architecture. It is also general enough to be used with any *target*-space distance. In the following, we will first briefly introduce the VAE model and then present the method in detail.

3.1 Variational Autoencoders (briefly)

Originally introduced by Kingma and Welling [KW22], [RMW14], Variational Autoencoders (VAE) are a class of latent variable models that are trained using variational inference. By latent variable models, we assume a model that has some hidden variables, z, that are not directly observed. In the context of VAEs, the hidden variables are called latent codes and are typically assumed to be normally distributed. A little more formally, we say that there exists a latent variable z such that the observed data x is generated by a conditional distribution $p(x \mid z)$. The goal of the VAE is to approximate the distribution $p(x)$ and the latent distribution $p(z \mid x)$. Having access to this latent distribution, we can sample from it and generate new data points.

The model consists of two parts: an encoder and a decoder. The encoder takes in the input data x and maps it to a latent space. The decoder takes in a sample from the latent space and reconstructs the input data. The encoder and decoder

are trained jointly by maximizing the Evidence Lower BOund (ELBO) defined as

$$\log p(x) \geq -D_{KL}(q(z \mid x) \| p(z)) + \mathbb{E}_{q(z|x)}[\log p(x \mid z)]. \tag{3.1}$$

ELBO consists of two terms: the KL-divergence between the latent distribution and the prior and the reconstruction loss. The first term is a regularizer that encourages the latent distribution to be close to the prior. The second term is the reconstruction loss which encourages the model to reconstruct the input data.

> For a more detailed introduction to VAEs, we refer the reader to [KW22] and the my favorite derivation based on Bayes' Theorem by Fernandez [Fer21].

3.2 Distance aligning in latent spaces

The encoder network of a VAE models latent codes z as isotropic Gaussian distributions, i.e. $z \sim \mathcal{N}(\mu, \sigma^2 I)$ and our task is to derive a way to relate pairs of them to pairs of observations in *target*-space. Looking at latent space distributions, a natural choice is to use KL-divergence (or rather some symmetric version like Jensen–Shannon divergence). Divergence, however, is measured in nats and is therefore not immediately relatable to the *target*-space values which are expressed in all kinds of arbitrary units. As an example, when optimizing protein stability, we are interested in the *change in the change* [KLB11] in Gibbs free energy, measured in Joules. It is therefore not immediately clear how to relate the two domains. Previous works have explored triplet loss [IHR23] and contrastive loss [AZ19] to impose an alignment on the latent space, but they do not, however, allow for a general solution across various kinds of continuous *targets*. Both the triplet and contrastive loss require binning to differentiate between "close" and "far" samples. For triplet loss, a triplet of samples is formed by a "positive", "negative" and an "anchor" sample. A loss term is then added to the ELBO to minimize the distance between the anchor and the positive sample and maximize the distance between the anchor and the negative sample. For contrastive loss, the dataset is split into background and target-dataset and a separate encoder is introduced to encode the background dataset. The model then learns to differentiate between the target and background samples and enhance salient latent features.

To tackle the above mentioned binning issue while allowing for a general, Bayesian solution across various kinds of continuous *targets*, we adopt the Euclidean distance as a measure of similarity between pairs of samples in the *target*-space. This same distance is then imposed on the corresponding latent codes. Little more formally, we write

$$||z_i - z_j|| \approx ||y_i - y_j||.$$

One can be tempted to naively impose the latter through the means of Z disregarding the variance σ^2 (this is what Ishfaq, Hoogi, and Rubin [IHR23] do in their work). This, however, will effectively break the probabilistic interpretation of the model. Instead, what we really want, is to maximize the likelihood of the distribution of distances between a pair of Normally distributed latent codes according to the *target*-distance of a corresponding pair in the *target*-space. The idea is illustrated in fig. 3.2.

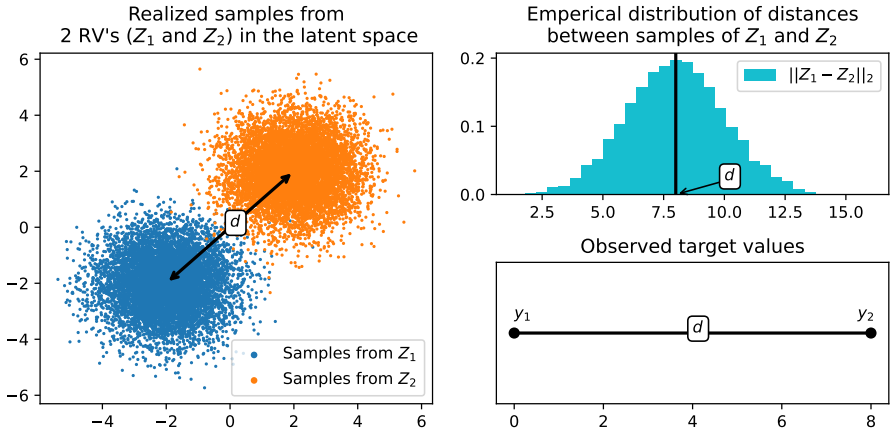

Figure 3.2: Illustration of the idea behind distance (d) aligning. **(left)** Realization of the samples from Z_1 and Z_2 in the *latent*-space. **(right,top)** Empirical distribution of distances. **(right, bottom)** Corresponding distance in the *target*-space.

The exact distribution that Euclidean distances between two normally distributed random variables (RV's) follow, is the non-central Nakagami distribution [Hau18]. Squared Euclidian distances between two normally distributed RV's, on the other hand, follow the non-central χ^2 distribution, making our objective

$$||z_i - z_j||^2 \approx ||y_i - y_j||^2.$$

We chose to use the latter as it is more widely known and has a CDF approximation allowing for a more efficient implementation. Efficiency is important, as the number of pairwise distances grows quadratically with the number of samples in a mini-batch.

3.3 Noncentral Chi-squared distribution

Let us start by relating the two RV's Z_1 and Z_2 to X:

$$\begin{aligned} Z_i &\sim \mathcal{N}(\mu_{z_i}, \sigma_{z_i}^2 I) \\ X &= Z_1 - Z_2 \\ X &\sim \mathcal{N}(\mu_{z_1} - \mu_{z_2}, \sigma_{z_1}^2 I + \sigma_{z_2}^2 I) \end{aligned}$$

as the sum of two independent normal RV's is also normal with the sum of the means and variances.

Let $X_1, X_2, ...X_D$ be D independent standard normal RV's with known means μ_i and unit variances (to be relaxed later): $X_i \sim \mathcal{N}(\mu_i, 1)$. Then

$$Y = \sum_{i=1}^{D} X_i^2$$

follows a noncentral chi-squared distribution $Y \sim \chi^2(D, \lambda)$ with D degrees of freedom and non-centrality parameter

$$\lambda = \sum_{i=1}^{D} \mu_i^2.$$

The **PDF of the noncentral chi-squared distribution** is given by

$$f_X(x; D, \lambda) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2}(\lambda/2)^i}{i!} f_{Y_{D+2i}}(x) \tag{3.2}$$

or alternatively as

$$f_X(x; D, \lambda) = \frac{1}{2} e^{-(x+\lambda)/2} \left(\frac{x}{\lambda}\right)^{D/4-1/2} I_{D/2-1}(\sqrt{\lambda x})$$

where I_ν is the modified Bessel function of the first kind of order ν given by

$$I_\nu(z) = \sum_{j=0}^{\infty} \frac{(z/2)^{\nu+2j}}{j!\Gamma(\nu + j + 1)}.$$

Another alternative, is to write the pdf as:

$$f_X(x; D, \lambda) = \mathrm{e}^{-\lambda/2} {}_0F_1(; D/2; \lambda x/4) \frac{1}{2^{D/2}\Gamma(D/2)} \mathrm{e}^{-x/2} x^{D/2-1}$$

by using the relation between Bessel functions and hypergeometric functions [Mui82].

For the **CDF of the noncentral chi-squared distribution**, we write

$$F_X(x; D, \lambda) = e^{-\lambda/2} \sum_{j=0}^{\infty} \frac{(\lambda/2)^j}{j!} Q(x; D + 2j) \tag{3.3}$$

where $Q(x; D)$ is the CDF of the ordinary χ^2 distribution with D degrees of freedom:

$$Q(x; D) = \frac{\gamma(D/2, x/2)}{\Gamma(D/2)}$$

$$\gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt$$

$\gamma(a, x)$ being the lower incomplete gamma function.

3.4 Re-scaling the variance

The derivations above assume X to have a unit variance. In practice, this is not the case as variances of Z are learned during training. To account for this, we re-scale the variance of X to be unitary. For the following derivations we assume both Z_1 and Z_2 to be isotropic, i.e. $\Sigma_{z_1} = \sigma_{z_1}^2 I$ and $\Sigma_{z_2} = \sigma_{z_2}^2 I$. This constraint is enforced by the VAE encoder network which should output a single variance value for all the dimensions of a particular Z instead of a vector. The rescaling factor is derived by normalizing the means of Z_1 and Z_2:

$$\begin{aligned}
\lambda &= \sum_{i=0}^{D} \left(\frac{\mu_{iz1}}{\sigma_{z1}} - \frac{\mu_{iz2}}{\sigma_{z2}} \right)^2 \\
&= \sum_{i=0}^{D} \left(\frac{\mu_i}{\sqrt{\sigma_{z1}^2 + \sigma_{z1}^2}} \right)^2 \\
&= \sum_{i=0}^{D} \frac{\mu_i^2}{\sigma_{z1}^2 + \sigma_{z2}^2} \\
&= \sum_{i=0}^{D} \mu_i^2 \underbrace{\frac{1}{\sigma_{z1}^2 + \sigma_{z2}^2}}_{\text{rescaling factor}}
\end{aligned}$$

The factor $(\sigma_{z1}^2 + \sigma_{z2}^2)^{-1}$ is then used to scale the variable, x, and re-normalize the PDF and CDF.

3.5 Optimized implementation

While the derivations in sec. 3.3 are correct, both the PDF and the CDF involve infinite sums or power series which could be less practical from a computational point of view. Having briefly studied the (non-differentiable) Scipy implementation, we found their approach to rely on the Bessel function computed with a numerical algorithm (Temme's method [Tem76]). Porting all this to PyTorch or JAX, neither of which currently supports non-central χ^2, was deemed too time-consuming and not guaranteed to be much faster than partially computing a vectorized infinite sum.

For a performant implementation, we will use a closed form approximation of the non-central χ^2 CDF derived by Sankaran [SAN59]:

$$F_X(x; D, \lambda) \approx \Phi(g(x))$$

where

$$\begin{aligned} g(x) &= \frac{\left(\frac{x}{D+\lambda}\right)^h - (1 + hp(h - 1 - 0.5(2 - h)mp))}{h\sqrt{2p}(1 + 0.5mp)} \\ h &= 1 - \frac{2}{3}\frac{(D+\lambda)(D+3\lambda)}{(D+2\lambda)^2} \\ p &= \frac{D+2\lambda}{(D+\lambda)^2} \\ m &= (h-1)(1-3h) \end{aligned}$$

and $\Phi(\cdot)$ is the CDF of the standard normal distribution.

We note that the PDF of a continuous random variable can be found by differentiating the CDF which we will do with the help of a Computer Algebra System:

$$f_X(x; D, \lambda) = \Phi'(g(x)) \cdot g'(x)$$

$$g'(x) = \frac{\frac{1}{\sqrt{2}}\left(\frac{x}{D+\lambda}\right)^{\frac{D^2+4D\lambda+6\lambda^2}{3(D+2\lambda)^2}}(D+\lambda)(D+2\lambda)^3}{\sqrt{\frac{D+2\lambda}{(D+\lambda)^2}}\left(8\lambda^4 + (20D+2)\lambda^3 + \left(18D^2 + \frac{2}{3}D\right)\lambda^2 + 7 \cdot D^3\lambda + D^4\right)x}$$

and $\Phi'(x)$ is the PDF of the standard normal distribution. Figure fig. 3.3 illustrates the empirical correctness of the above derivations and variance scaling.

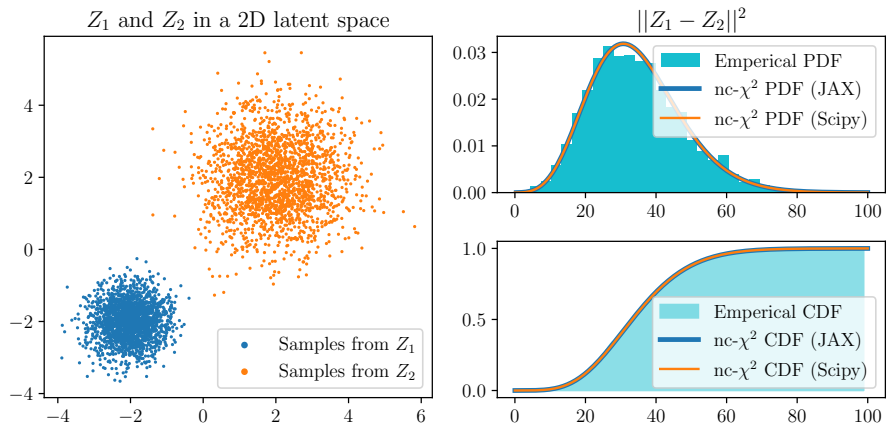

Figure 3.3: Empirical correctness of the derivations in sec. 3.4 and sec. 3.5. **(left)** Z_1 and Z_2 in the latent space. **(top right)** Empirical distribution of distances between Z_1 and Z_2 alongside the non-central χ^2 PDFs. **(bottom right)** Empirical cumulative distribution of distances between Z_1 and Z_2 alongside analytical CDFs.

3.6 Augmenting the ELBO

To impose the distance on the latent space, we augment the ELBO (3.1) with two terms: a nc-χ^2 PDF-term and a nc-χ^2 CDF-term. The PDF term is applied to the pairs of latent codes, Z_i and Z_j, which are considered to be "very close" in the *target*-space in the Euclidian sense. The thinking is that, given a differentiable manifold of a VAE, the distance between nearby samples is roughly Euclidian. The CDF term, applied to the pairs of latent codes, Z_k and Z_l, which are "far apart" captures the fact that the imposed distance is at least the Euclidean distance. The formulation is inspired by the idea of censoring in survival analysis [LW13] where the event of some observation is censored if it is not observed within a certain time frame. The censored likelihood function for iid distances between a set of N pairs of samples is given by

$$L(\{d_i\}_{i=1}^N, T) = \prod_{d_i<T} f_X(d_i) \prod_{d_i \geq T} (1 - F_X(d_i)) \tag{3.4}$$

where T is some threshold distance (a hyperparameter) and f_X and F_X are the PDF and CDF of the distribution of the distances (see equations 3.2 and 3.3).

3.7 The effect of distance aligning

To validate the effectiveness of the method for Bayesian optimization, we train a VAE on a protein dataset and use the latent space to optimize the acquisition function. As the acquisition function (in our case qEHVI) is highly non-concave, we use a multiple restarts strategy. Figure fig. 3.4 shows the final values of the acquisition function for each of the restarts:

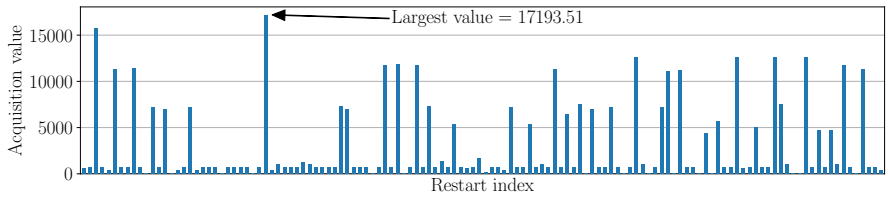

Figure 3.4: Final values of the acquisition function for each of the restarts.

We then pick the highest-scoring run and note the acquisition value as final, mimicking the behavior of a typical Bayesian optimization round where the highest protein will be synthesized in the lab.

The experiment is repeated 10 times for various thresholds T (see eq. 3.4) calculated as q-quantile, $q \in [0, 1]$ of the targets in the mini-batch. $q = 0.125$ as an example, means that the lowest 12.5% of the target distance pairs in the mini-batch are modeled by the PDF of the non-central χ^2 distribution and the rest by the CDF.

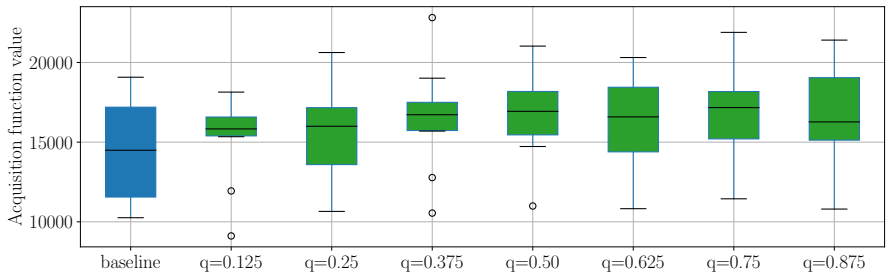

Figure 3.5: The effect of distance aligning on the final value of the acquisition function.

The results are presented in fig. 3.5. We see that the distance aligning significantly improves the performance of the model as the BO in aligned latent space is able to find higher-scoring targets compared to an ordinary VAE (Baseline).

Finally, in fig. 3.6, we present an actual view of a 2D latent space of a VAE trained on the same protein dataset. Here we amplified the distance-likelihood in the loss function for a more pronounced effect. Not surprisingly, we see that the latent space without distance aligning is noisy from a *target*-distance perspective while the latent space with distance aligning is much more structured and the samples are grouped together according to their *target*-space distances.

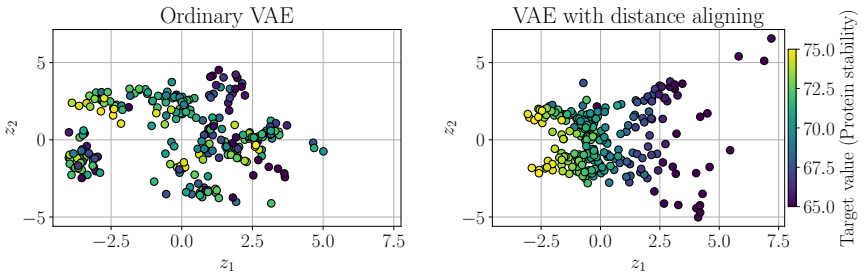

Figure 3.6: Latent space of a VAE trained on a protein dataset. **(left)** Latent space without distance aligning. **(right)** Latent space with distance aligning. Colors represent the *target* - protein stability.

3.8 Correcting the prior

A careful reader might note that the above formulation does not take into account the prior distribution of the latent codes, $z \sim \mathcal{N}(0, 1)$ (see ELBO in sec. 3.1). This results in very competing objectives during training: the prior wants to push the latent codes towards the origin while the distance aligning wants to place them according to the *target*-space distances.

We propose two solutions to this problem. The first one is to simply normalize the *target*-distances to have a unit variance. The second one is not to use a standard Gaussian prior $z \sim \mathcal{N}(\mu_p, \sigma_p^2), \mu_p = 0, \sigma_p^2 = 1$, but rather learn its variance σ_p^2 as a trainable parameter of the model. Partially based on derivations by Odaibo [Oda19], we write the solution of $-D_{KL}(q(z \mid x) \| p(z))$ for learned

prior variance σ_p^2 as

$$-D_{KL}(q(z \mid x) \| p(z)) = \frac{1}{2}\left[\log(\sigma_q^2) - \log(\sigma_p^2) + \frac{\sigma_q^2 + \mu_q^2}{\sigma_p^2} + 1\right].$$

3.9 Limitations

The method presented in this chapter is not without limitations. The most obvious one is that, rather paradoxically, given an unsupervised model, it requires a *target*-space distance for pairs of samples. This is rarely available. It is, however, possible to define augmented *targets* for some datasets. Taking MNIST [Den12] as an example, one can define the *target* to be the distance between the two digits in the image - literally calculating MSE between the pairs of images (note that MNIST is centered which makes this method even more suitable). In our early experiments, we have found that this works surprisingly well for downstream classification tasks. When using a K-Nearest Neighbour classifier, the classifier accuracy is significantly higher when using the distance-aligned latent space compared to the ordinary VAE latent space.

In the context of Bayesian optimization, the actual distance we would like to impose is the value of the acquisition function. For observed data, this value is naturally zero and therefore unusable. In future work, we will investigate the possibility of calculating the value of acquisition as a result of mini-optimization rounds and the introduction of augmented data samples. For now, we will use the distance between *targets* as a proxy.

Finally, the derivations above require a shared variance across the dimensions of the latent codes. This is not the case for a typical VAE encoder network which outputs a vector of variances - one for each dimension of Z. We have found that the method still works well in practice, but it is certainly possible to extend the derivations to account for the variance vector with the help of "generalized Chi-squared distribution". One practical implementation of which can be found in [SO77].

Our exploration of distance aligning in latent spaces has revealed a promising approach to enhance the efficiency of Bayesian optimization for protein engineering. By addressing Research Question 2 - **"Can we design a generative model that learns a meaningful, regularized, low-dimensional latent**

space representation of protein sequences for Bayesian optimization?" - we have developed a method that effectively groups similar proteins in the latent space according to their target-space distances. This alignment technique, implemented through a novel loss function based on the non-central χ^2 distribution, has demonstrated significant improvements in the performance of Bayesian optimization, as evidenced by our empirical evaluations. The ability to create a more structured and informative latent space representation lays a crucial foundation for more efficient exploration during the optimization process.

Building upon this advancement, we now turn our attention to integrating these insights into a comprehensive probabilistic generative model for protein design. In the following chapter, we will explore how this distance-aligned latent space can be combined with state-of-the-art Protein Language Models to create a powerful framework for guided drug discovery.

Chapter 4

A Probabilistic Generative Model for Automatic Guided Drug Discovery

In the pursuit of novel therapeutic compounds within the field of computational drug discovery, the need for innovative methodologies has become increasingly apparent. Conventional approaches often grapple with the complexities of exploring the vast chemical space efficiently. This chapter introduces a little methodological shift: integration of a Transformer protein language model (PLM) into the Bayesian optimization loop.

Our approach hinges on the principled utilization of Variational Autoencoders, specifically designed to learn compact latent representations of a PLM from which full protein sequences are decoded. As the PLM is trained on a vast majority of the known protein sequences, it is able to leverage the underlying patterns and structure of the protein space. This effectively expands the generative capabilities of the model beyond the small number of observed sequences in a concrete training set.

The driving force behind our approach is the Bayesian optimization framework with Gaussian Processes (GP) as the surrogate model. The GP maps the latent representations of the PLM to a posterior distribution over the protein tar-

get values. The acquisition function is then used to guide the search towards promising regions of the latent space.

Throughout this chapter, we delve into the conceptual foundations, technical intricacies, and empirical validation of our Probabilistic Generative Model. Through a series of experiments, we demonstrate the effectiveness of our approach in accelerating and refining the protein discovery process.

4.1 Transformer protein language models (briefly)

Transformers [Vas+23], which have instigated a paradigm shift in Natural Language Processing (NLP), are now being repurposed for the modeling of biological sequences. At the heart of the transformer architecture lies the attention mechanism, a crucial component enabling the capture of long-range dependencies within a sequence.

Suppose we have an input sequence $X = \{x_1, x_2, \ldots, x_p\}$, where p is the length of the sequence. The self-attention mechanism computes a set of attention weights for each position in the sequence. The attention weights are then used to compute a weighted sum of the values at each position. The attention weights are computed as follows:

$$\text{Attention } (Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

where Q, K, and V are the query, key, and value matrices, respectively. The query and key matrices are computed by multiplying the input sequence with the learned weight matrices W_Q and W_K, respectively, i.e:

$$Q = XW_Q$$
$$K = XW_K$$

The value matrix is computed by multiplying the input sequence with the learned weight matrix W_V

$$V = XW_V$$

The attention weights are computed by taking the dot product of the query and key matrices and scaling the result by $\sqrt{d_k}$, the square root of the dimensionality of the key vectors. The scaled dot product is then passed through a softmax function to obtain the attention weights. The attention weights are then used to compute a weighted sum of the values at each position. In practice, the self-attention mechanism is often used with multiple attention heads. The outputs

from each head are concatenated and linearly transformed to produce the final output. More formally, the multi-head attention mechanism is defined as follows:

$$\begin{aligned}\text{MultiHead}(Q,K,V) &= \text{Concat}\left(\text{head}_1,\ldots,\text{head}_h\right)W^O \\ \text{where}\ \ \text{head}_i &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)\end{aligned}$$

where $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. The number of attention heads is denoted by h, d_{model} is the dimensionality of the model, d_k is the dimensionality of the key vectors, and d_v is the dimensionality of the value vectors.

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model that introduced the concept of masked language modeling [Dev+19]. The masked language modeling task is a form of self-supervised learning, where the model is trained to predict a randomly masked word in a sentence. Specifically, the model is trained by minimizing the cross-entropy loss between the predicted and actual masked words given the surrounding context. After the complete general training, the model is fine-tuned on downstream tasks such as question answering and natural language inference. In the context of protein modeling, fine-tuned is not strictly necessary as the internal representations of the model can be used directly for downstream tasks.

> The key takeaway concerning BERT is that its hidden states (*embeddings*) featurize individual amino acid residues and, most importantly, their context in the sequence. In contrast, a representation such as one-hot or z-scales [San+98] encoding does not capture the context of the amino acid residues.

The PLM we use in this work is a variant of BERT, called **ESM2** [Lin+22] (specific variant `esm2_t33_650M_UR50D`) pre-trained on the UniRef50 [Suz+14] dataset which contains 45 million protein sequences. The model itself is a 33-layer transformer with 650 million parameters and inner dimensionality of $d_{\text{model}} = 1280$ effectively encoding each amino acid residue into a 1280-dimensional vector. A full embedding of a protein sequence is thus $\mathbb{R}^{p \times 1280}$, where p is the length of the sequence.

In the following, we are going to combine a PLM with a Variational Autoencoder (VAE) to learn a compact latent representation of the PLM embedding space. To reduce confusion regarding the language and notation we will refer to the PLM embeddings as **PLM embeddings** e and to the VAE latent representations as **latent embeddings** z.

4.2 Reconstructing the embeddings of a PLM with a conditional variational autoencoder

The motivation behind the need to reduce the dimensionality of the PLM embeddings is straightforward: it is computationally infeasible to perform Bayesian optimization in a $p \times 1280$-dimensional space due to the curse of dimensionality.

> Interestingly, our JAX codebase is able to handle the search in full $p \times 1280$-dimensional parameter space, but we found that this formulation is highly sensitive to initialization as the gradients are very sparse in such a high-dimensional space.

Introduced in the previous chapter, Variational Autoencoders (VAE) are a class of generative models that learn a compact latent representation of the input data. In our case, the input data in question are the PLM embeddings. Thus, the VAE is trained to reconstruct the PLM embeddings from the latent representations z. Proteins are sequences of amino acid residues, and the PLM embedding of a protein is therefore a sequence of vectors, where each vector corresponds to an amino acid residue in the sequence. One can think of the full PLM embedding for a single protein as a matrix $E \in \mathbb{R}^{1280 \times p}$, where p is the length of the protein sequence, and 1280 corresponds to the dimensionality of the PLM embeddings (recall $d_{\text{model}} = 1280$ from the previous section).

Our intial approach to model E, was to use a standard VAE architecture, where the matrix is flattened into a vector and the VAE is trained to reconstruct this full vector. The vector would then be reshaped back into a matrix and decoded by the PLM into a full protein sequence.

This approach is tried and tested not only in the context of (non-convolutional) VAEs for image reconstruction but also in the context of protein modeling by Riesselman, Ingraham, and Marks [RIM17]. The difference between our setup and that of [RIM17], however, is that they used datasets based on Multiple Sequence Alignments (MSA, [HS88]), resulting in a much larger diversity of protein sequences. In our case, we work within the context of a single protein wildtype and its immediate mutants. This means that the diversity of the protein sequences is much smaller, and the VAE is therefore not able to learn a meaningful latent representation of the PLM embeddings. What happens in practice, is that the VAE learns to reconstruct the most common amino acid residues in the sequence (the wildtype), and the latent representation is therefore not able to capture the diversity of the PLM embeddings. To address this issue, we tried several weighting schemes to balance the reconstruction loss between the shared portion of the pro-

tein (wildtype) and the individual mutants, but we found that, while technically working, the resulting model was very fragile and would easily collapse all reconstructions to the wildtype. We needed a more robust approach.

Drawing inspiration from the Conditional Autoencoder (CVAE) [SLY15] we train our model to reconstruct the individual PLM embeddings conditioned on their positions in the sequence. In a traditional VAE, the encoder network would model $q(z|x)$, in contrast, CVAE encoder models $q(z|x, y)$, where y is the conditioning variable. The decoder network is then trained to reconstruct an output y conditioned on z and x. Formally, the variational lower bound of a CVAE is written as

$$\log p_\theta(y|x) \geq -KL\left(q_\phi(z|x, y)\|p_\theta(z|x)\right) + \mathbb{E}_{q_\phi(z|x,y)}\left[\log p_\theta(y|x, z)\right].$$

For our purpose, we want to reconstruct the inputs and don't require prior of the latent variables $p_\theta(z|x)$ to be modulated by the input x as suggested in the paper. We therefore relax the constraint making the latent variables statistically independent of input variables, i.e. $p_\theta(z|x) = p_\theta(z)$ and reverting the KL-term to one of a traditional VAE:

$$\log p_\theta(y|x) \geq -KL\left(q_\phi(z|x, y)\|p_\theta(z)\right) + \mathbb{E}_{q_\phi(z|x,y)}\left[\log p_\theta(x|y, z)\right].$$

The resulting challenge is to model the conditional distribution $q_\phi(z|x, y)$ corresponding to the full protein as the encoder network outputs p vectors for both the mean and the variance of the latent representation z. To this end, we use a simple pooling strategy for means μ_E and variances σ^2_E pooling them over the sequence dimension p:

$$\mu_z = \frac{1}{p}\sum_{i=1}^{p}\mu_{E_i} \quad \text{and} \quad \sigma^2_z = \frac{1}{p}\sum_{i=1}^{p}\sigma^2_{E_i}.$$

A subtle requirement of this pooling formulation is the need to include full protein embeddings E in the input to the encoder network (training data cannot be fragmented across the positions p), so while we claim to model $q_\phi(z|x, y)$, we actually model $q_\phi(z|x, y, E)$. This is necessary to ensure that the encoder network properly maps to the latent distribution $q_\phi(z|x, y)$.

The decoder is then trained to reconstruct individual PLM embeddings $\hat{e}$ conditioned on the shared latent representation z and the position y. Here, we use a simple concatenation of the latent representation z (it is repeated to match the length of the protein sequence) and the position y as the input to the decoder network. The decoder network is then trained to reconstruct the PLM embedding e from the concatenated input. As a little twist, we also include a small Resnet block [He+15] in the decoder network to improve the reconstruction quality.

The full model architecture is shown in the fig. 4.1 below.

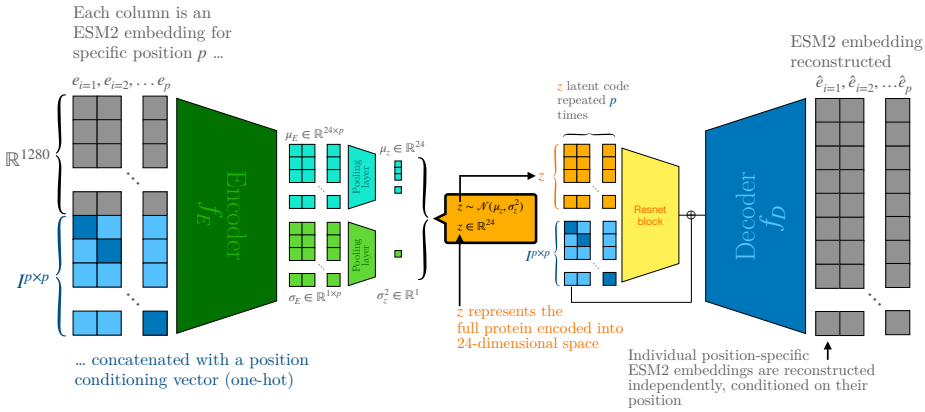

Figure 4.1: Complete VAE diagram

Key takeaway of the above formulation is that the model is trained on the full protein sequence, but the decoder is only trained to reconstruct the individual, positionally-conditioned PLM embeddings from a shared latent representation z where z represents the full protein sequence. The concatenation of the individual conditional PLM embeddings $\hat{e}$ is then used to reconstruct the full protein sequence with the PLM.

A careful reader might have noticed that the variance σ_z^2 is pooled into a single scalar value across all dimensions of z. This is a limitation of the current implementation of the distance matching loss, discussed in the previous chapter sec. 3.9.

Finally, we note that the model above supports variable-length protein sequences as the PLM embeddings can be padded to the maximum length of the longest sequence in the training set. The PLM of our choice, ESM2, natively supports variable length sequences by incorporating a special token `<pad>` into the vocabulary. That same token has special embedding which the VAE learns to model as well.

4.3 Bayesian optimization in the latent space of a Variational Autoencoder

Having designed a generative model capable of producing full PLM embeddings from a single latent code z, the rest is straightforward. On the one hand, PLM embeddings can be directly decoded into full protein sequences (by ESM2 itself), and on the other hand, the PLM embeddings $\hat{E}$ can be used as inputs to a Gaussian Process surrogate model to predict the target values with uncertainty. The acquisition function acting on the GP posterior is then used to guide the search towards promising regions of the z-latent space.

> A curious reader might ask: "Why not use the latent embeddings z directly as inputs to the GP model?". The answer is that the latent embeddings z are restricted in the sense that they reconstruct a subset of PLM embeddings E (subset corresponding to our training-set). PLM embeddings, on the other hand are of higher dimension (making them a better predictors) and are not subjected to the noisy encoding-decoding process of the VAE.

An established approach to solve protein regression tasks with PLMs is to mean-pool the embeddings across the positional dimension and use a regression model to predict the target values. This solves the problem of variable length sequences, but it also discards the positional information of the embeddings and inevitably leads to a loss of information. We tackle this problem with a masking approach, detailed in [Zai+22], included as Manuscript 4 in this thesis. The mask, being a learned vector, is applied to the PLM embeddings E before they are mean-pooled and fed into the GP model.

Having pooled the embeddings across the positional dimension, we are left with a single vector $\hat{e} \in \mathbb{R}^{1280}$, which is then used as one of the inputs to a GP regression model. The other GP input is the one-hot encoded protein sequence obtained by decoding the full PLM embedding $\hat{E}$ with ESM2. As the ESM2 decoder output is logits, we apply a "cooled" softmax function. Cooled in this context means that the softmax function is applied with a small temperature parameter $T = 10^{-8}$:

$$\text{Softmax}(\hat{l}_{i,j}) = \frac{\exp(\hat{l}_{i,j}/T)}{\sum_{k=1}^{20} \exp(\hat{l}_{i,k}/T)}.$$

where $\hat{l}_{i,j}$ is the j-th logit of the i-th amino acid residue in the sequence. This decreases the resulting entropy and effectively makes the softmax function behave like a one-hot encoding.

The vector $\hat{e}$ and the one-hot encoded representation are modeled by two sepa-

rate Matern-5/2 kernels, which are then summed together to form the full kernel, completing our surrogate model. Formally, the kernel is defined as:

$$k_{\text{full}}\ \left([\hat{e}, \mathbf{x}_{\text{one-hot}}], [\hat{e}', \mathbf{x}'_{\text{one-hot}}]\right) = k_{\text{matern}}\left(\hat{e}, \hat{e}'\right) + k_{\text{matern}}\left(\mathbf{x}_{\text{one-hot}}, \mathbf{x}'_{\text{one-hot}}\right)$$

$$k_{\text{matern}}\left(\mathbf{x}, \mathbf{x}'\right) = \alpha\left(1 + \sqrt{5}r + \frac{5}{3}r^2\right)\exp(-\sqrt{5}r)$$

$$r = \frac{\|\mathbf{x} - \mathbf{x}'\|}{\ell}$$

Here α is the amplitude and ℓ is the length scale of the kernel, both of which are learned during the training process.

Having obtained the GP posterior, the acquisition function is then used to guide the search towards promising regions of the latent space. The acquisition function we use is the qEHVI - expected hypervolume improvement [DBB20].

Additionally, we equip the VAE with the distance-matching loss (described in sec. 3.9) to ensure that the squared distance between the latent embeddings z is matched with the squared distance between the target values we are seeking to optimize for. When optimizing for multiple (Multi-Objective Bayesian Optimization, MOBO) target values, we reserve some of the dimensions of the latent space z to encode the corresponding target values distances.

An important detail to note is that the full path from z to the acquisition value is differentiable, and thus, the gradients can be backpropagated through the entire model all the way back to z.

Summarizing the above, the full model architecture is shown in the fig. 4.2:

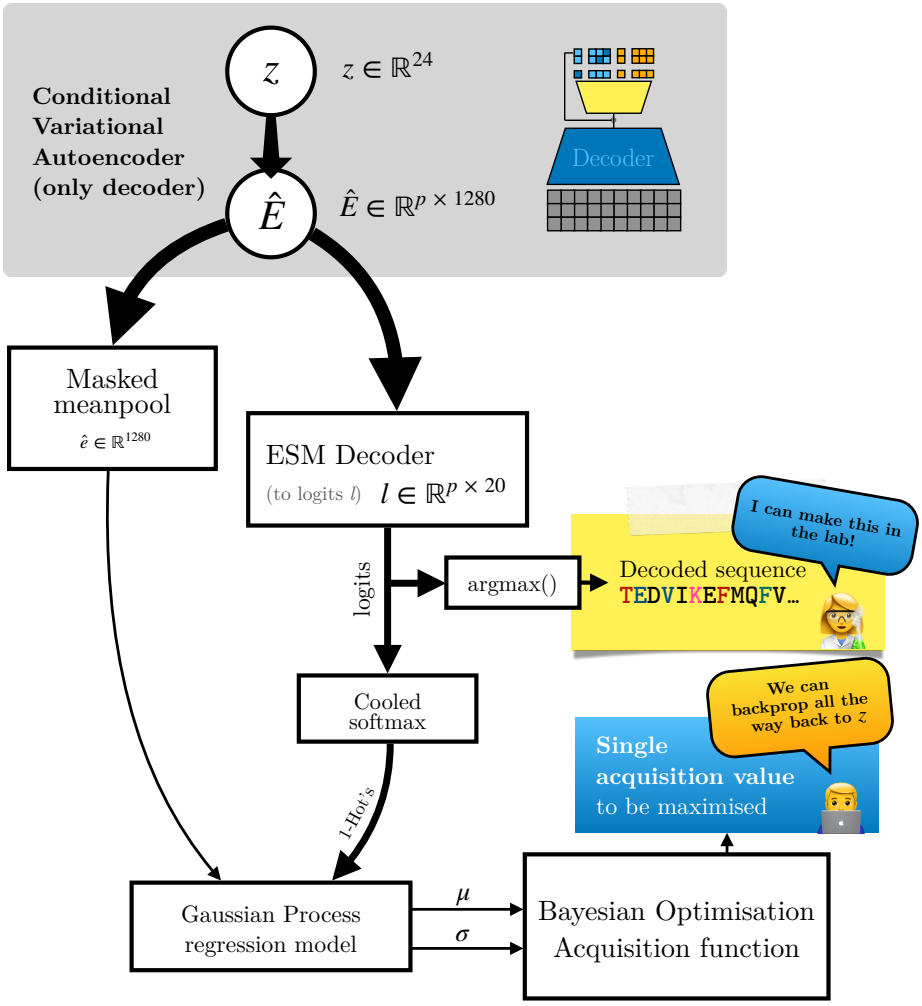

Figure 4.2: Full model diagram

The full Bayesian optimization loop with initialization and invocation of the complete model is formalized in algorithms 1 and 2.

Algorithm 1 Outer loop

Require: Dataset $\mathcal{D}_0$ where subscript 0 refers to the round of optimization, comprising of a set of n sequences S, and their corresponding target values Y_0: $\mathcal{D}_0 = \{(s_i, \mathbf{y}_i)\}_{i=1}^{n}$

Finetune PLM Decoder on $\mathcal{D}_0$ ▷ Optional

$\mathcal{E}_0 \leftarrow \text{PLMDecoder}(\mathcal{D}_0)$ ▷ Extract PLM embeddings

for $i \leftarrow 0, ..., \text{max\_bo\_rounds}$ **do**

`/* Train GP Surrogate model */`

Fit $\hat{f}_{\text{GP}}$ on $(\mathcal{D}_i, \mathcal{E}_i)$

`/* Train VAE model (`$\mathcal{D}_i$ `is used for distance-matching) */`

Fit $\hat{f}_{\text{VAE}}$ on $(\mathcal{D}_i, \mathcal{E}_i)$

`/* Latent embeddings corresponding to the Pareto frontier */`

$\mathcal{Z}_{\text{pareto}} \leftarrow \hat{f}_{\text{VAE\_Encoder}}(\mathcal{E}_i[\text{pareto\_front}(\mathcal{D}_i)])$

$z_{\text{best}} \leftarrow \text{inner\_loop}(\mathcal{Z}_{\text{pareto}}, \hat{f}_{\text{VAE}}, \hat{f}_{\text{GP}})$ ▷ Find the best latent embedding

$e_{\text{best}} \leftarrow \hat{f}_{\text{VAE}}(z_{\text{best}})$ ▷ Decode to PLM embeddings

$s_{\text{best}} \leftarrow \text{ESM2Decoder}(e_{\text{best}})$ ▷ Extract the sequence

$\mathbf{y}_{\text{best}} \leftarrow \text{Black-Box}(s_{\text{best}})$ ▷ Invoke the black-box function

$\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \cup \{(s_{\text{best}}, \mathbf{y}_{\text{best}})\}$ ▷ Update the dataset

$\mathcal{E}_{i+1} \leftarrow \mathcal{E}_i \cup \{e_{\text{best}}\}$ ▷ Update the PLM embeddings

end for

return $\text{pareto\_front}(\mathcal{D}_{\text{max\_bo\_rounds}})$

Algorithm 2 Inner loop

Require: $\mathcal{Z}_{\text{pareto}}, \hat{f}_{\text{VAE}}, \hat{f}_{\text{GP}}$ (see algorithm 1), f_a - acquisition function and η - learning rate

`/* Get stdandar deviation of the pareto embeddings */`

$\sigma_{\text{pareto}} \leftarrow \text{std}(\mathcal{Z}_{\text{pareto}})$

`/* Sample starting points from the pareto embeddings */`

$\mathcal{Z}_0 \leftarrow \mathcal{Z}_{\text{pareto}} + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2_{\text{pareto}})$

for $j \leftarrow 0, ..., \text{max\_bo\_steps}$ **do**

$\text{E} = \hat{f}_{\text{VAE}}(\mathcal{Z}_j)$ ▷ Decode to PLM embeddings

$\mathbf{s}, \mathbf{l} \leftarrow \text{ESM2Decoder}(E)$ ▷ Extract the sequences $\mathbf{s}$ and logits $\mathbf{l}$

$\mathbf{oh} \leftarrow \text{Softmax}(\mathbf{l})$ ▷ Get one-hot encodings "cooled softmax"

$E_{\text{pooled}} \leftarrow \text{m\_mean\_pool}(E)$ ▷ Masked mean-pool of the embeddings

$A \leftarrow f_a(E_{\text{pooled}}, \mathbf{oh})$ ▷ Invoke acquisition function

$\mathcal{Z}_{j+1} \leftarrow \mathcal{Z}_j + \eta \nabla_{\mathcal{Z}_j} A$ ▷ Update the latent embeddings

end for

4.4 Experiments

Assessing Bayesian Optimization techniques for protein design presents a challenging endeavor. As nature is the ultimate black-box function, we are not able to evaluate the quality of the proposed sequences directly, or at least do so promptly. A single complete round of optimization can easily take several weeks to complete, and the number of rounds required to obtain a meaningful result is in the order of hundreds. Instead, we need to depend on a surrogate function assessed through computer-based simulations.

Stanton et al. [Sta+22] introduced a framework for evaluation of the quality of the proposed sequences by using a protein folding stability predictor. The predictor is a molecular dynamics (MD) simulation-based approach that uses the FoldX software [Sch+05]. The software also produces a 3D structure of the protein, which can be used to compute other interesting target values such as Solvent Accessible Surface Area (SASA). The combination of folding stability and SASA is the multiobjective function we will optimize for (higher is better for both objectives). In the context of RFPs, Stability correlates with protein function (e.g. how long the protein can fluoresce) while SASA is a proxy for fluorescent intensity. Copying the experimental and evaluation setup from [Sta+22], we will use the same dataset (derived from FPbase [Lam19]) of red-spectrum fluorescent proteins (RFP), and the same set of seed sequences.

Fluorescent proteins are widely used in molecular and cell biology to visualize and study cellular processes. Red fluorescent proteins (RFPs) are a subset of fluorescent proteins that emit light in the red part of the electromagnetic spectrum. These proteins are valuable tools for live-cell imaging and fluorescence microscopy because their longer wavelengths can penetrate tissues more effectively than shorter wavelengths.

The most well-known and widely used red fluorescent protein is DsRed (Discosoma sp. red fluorescent protein, [Sha+05]), which was one of the first red fluorescent proteins to be cloned and characterized. Over the years, researchers have developed various derivatives and improved versions of red fluorescent proteins with enhanced brightness, photostability, and other properties.

The use of red fluorescent proteins is particularly beneficial for multicolor imaging experiments, where researchers aim to visualize multiple cellular components simultaneously. By using different colored fluorescent proteins (e.g., green, yellow, and red), researchers can label and track various cellular structures or proteins within the same sample.

> The quest to improve red fluorescent proteins (RFPs) and other fluorescent proteins has been an ongoing effort in the field of molecular and cell biology. Researchers are continually working to enhance the properties of existing fluorescent proteins or develop novel variants with improved characteristics.
>
> The improvement of RFPs often involves techniques such as protein engineering, directed evolution, and rational design. These approaches aim to introduce specific mutations into the fluorescent protein sequence to enhance desired properties while preserving its overall functionality.
>
> The continuous quest for better fluorescent proteins, including RFPs, contributes to the advancement of imaging technologies and our understanding of cellular processes. Researchers often share newly developed variants with the scientific community, fostering collaboration and accelerating progress in the field.
>
> One such place to share and discover fluorescent proteins is FPbase [Lam19], a database of fluorescent proteins and their properties. FPbase is a community driven resource that allows researchers to search for fluorescent proteins based on their characteristics and find the best variant for their experiments. The database also provides a platform for researchers to share their newly developed fluorescent proteins with the scientific community.

Not only did Stanton et al. [Sta+22] introduce a complete dataset and evaluation framework, they also did a great job of developing a method (LaMBO) for actual sequence design based on gradient-based optimization of multiobjective acquisition functions in the latent space of the autoencoder. Finally, they tested their method against other approaches based on Genetic Algorithms (GA) and established solid baselines.

It would not be an understatement to say that we are standing on the shoulders of giants and that our method is directly inspired by LaMBO. The main difference between our approach and LaMBO is that we utilize free-form Bayesian optimization in a latent space of a Variational Autoencoder while LaMBO is limited by a step-based corruption process resulting in a fixed number of mutations for each round of optimization. The next big difference is that we use a PLM to encode and decode the protein sequences, resulting in a much more expressive intermediate latent representation while LaMBO uses a simple one-hot encoding which strictly limits their search to the immediate domain of the training set. As they are corrupting the sequences one mutation at a time, the model will inevitably have a hard time capturing the effect of multiple mutations at once (non-additive effects). Finally, we shape our latent space with a distance-matching loss (sec. 3.6) further improving the quality of the latent embeddings for the purpose of Bayesian Optimization.

Multi-Objective GFlowNets (MOGFN), another great work focused on the use of Multi-Objective Bayesian optimization for protein design is done by Jain et al. [Jai+23]. Based on Generative Flow Networks by Bengio et al. [Ben+23], they utilize the same RFP dataset as Stanton et al. [Sta+22] and the same evaluation framework. This makes their work directly comparable to ours.

In the following we compare the performance of our method to LaMBO, MOGFN and the Genetic Algorithm (GA) baselines from Stanton et al. [Sta+22] (for details on the baselines, we refer to the paper). Starting with 512 examples in the initial pool, we perform 64 Bayesian Optimization rounds totaling 1024 black-box evaluations in batches of 16. All methods are evaluated by comparing the relative improvement of the hypervolume bounded by the Pareto front after x black-box function evaluations compared to the initial hypervolume. All experiments are repeated 10 times with different random seeds and the results are presented in fig. 4.3 where the midpoint, lower, and upper bounds of each curve depict the 50%, 20%, and 80% quantiles.

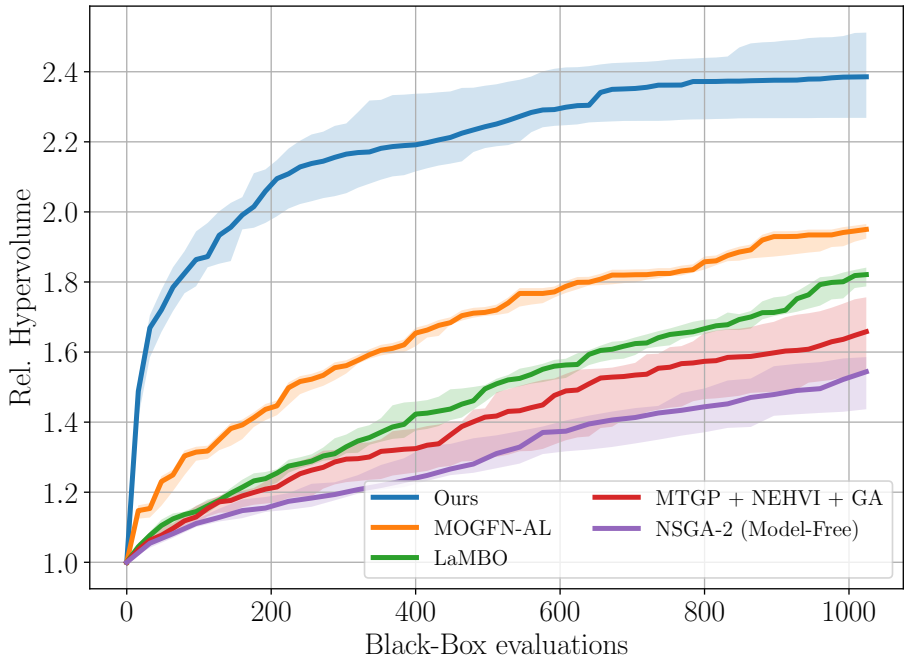

Figure 4.3: Relative hypervolume bounded by the optimized Pareto frontier for our method (blue) and other approaches. The midpoint, lower, and upper bounds of each curve depict the 50%, 20%, and 80% quantiles, estimated from 10 trials.

We notice that our approach leads to noteworthy improvements in Hypervolume compared to the initial dataset within a specified budget of black-box evaluations. Notably, our method achieves performance comparable to the next-best method, MOGFN-AL, with only about one-eighth of the black-box evaluation budget.

The main reason for the success of our method, we hypothesize, is the ability of the PLM embeddings to generalize to unseen sequences. As mentioned in the introduction, the PLM is trained on a vast majority of the known protein sequences, and it can leverage the underlying patterns and structure of the protein space. This effectively expands the generative capabilities of the model beyond the small number of observed sequences in a concrete training set.

Demonstrated performance is likely not the limit of our method. As we are doing free-form Bayesian optimization, we artificially limited the number of mutations to be at most 25 away from the wildtype. This matches the maximum number of mutations the LaMBO-method resulted in, at the end of their experiments. The reason for this limitation is twofold, first we do not want to propose sequences that are too far away from the wildtype as they might not be as stable as FoldX suggests, and second, we want to limit the time spent on each round of optimization. FoldX is a computationally expensive method, and it only gets more expensive with the number of mutations.

In fig. 4.4 we explore the distance between the wildtype and the proposed sequences at each round of optimization.

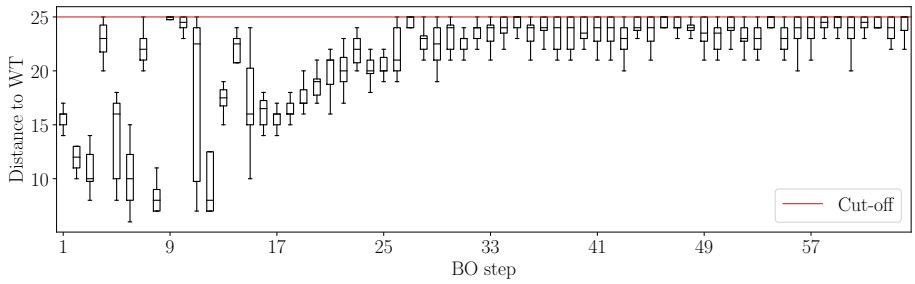

Figure 4.4: Levenshtein distance between the suggested sequences and their respective wildtypes at each round of BO optimization.

We notice that the number of mutations generally increases with each round of optimization, capping at the maximum of 25, as mentioned earlier. In other words, we are throwing away high-acquisition value sequences that are too far away from the wildtype. This might explain the saturation of the curve in

fig. 4.3.

Another unfortunate limitation of the current evaluation framework is the inability to model insertions and deletions. We have observed that our method sometimes proposes sequences with insertions and deletions, which we unfortunately, have to filter before passing them to FoldX. It is of great interest to explore the possibility of using a different stability predictor (such as PROVEAN [CC15]) and see if it improves the results further.

Finally, in fig. 4.5 we present the final Pareto frontier after combining the results of all 10 trials.

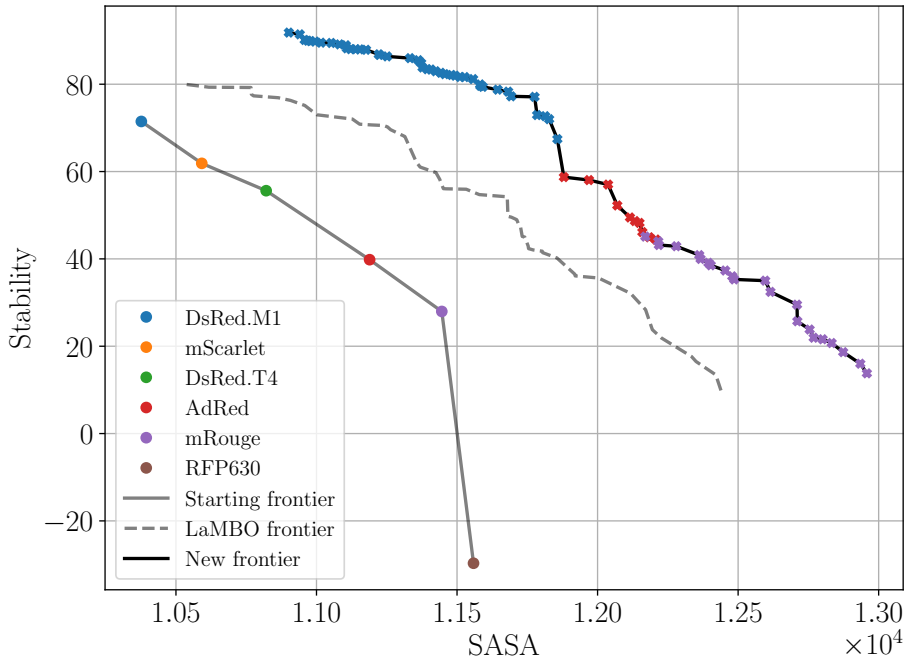

Figure 4.5: Resulting Pareto frontier after 10 independent trials. The wild-types (ancestor proteins) are shown as colored circles, with corresponding optimized offspring shown as crosses. LaMBO frontier (dashed) is included for reference.

We publish the full sequences of the new pareto frontier fig. 4.5 at the url[a] with their corresponding WT PDB identifiers and Stability/SASA values. We encourage bio chemists to explore the diversity of the proposed

sequences and use them in RFP engineering tasks.

[a] https://github.com/eugene

4.5 Running times

Our method is not computationally expensive per se, but it does require a GPU with at least 18GB of memory as we have to keep the decoder of the PLM, GP, VAE and BO-related data in memory at all times during the optimization process.

The experiments were run on a single NVIDIA RTX 4090 GPU with 24GB of memory and in fig. 4.6 we present the running time of the different components of our method at each round of optimization.

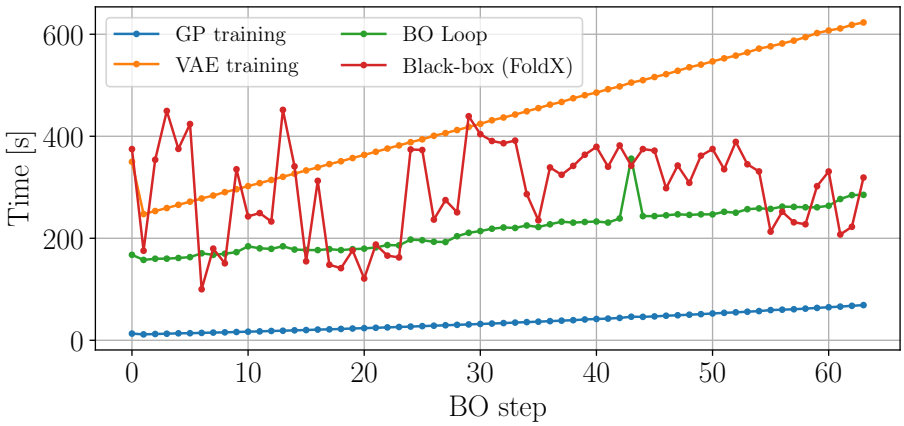

Figure 4.6: Running time of the different components of our method at each round of optimization.

Summing up the running times of the different components, we arrive at ~ 20 minutes/round for the final rounds of optimization.

4.6 Limitations

Unfortunately, the model is not without its limitations. We list the most important ones below:

The mapping that VAE incurs is not perfect in the sense that it inevitably adds noise to the PLM reconstructions. After all, we are compressing a matrix that is $p \times 1280$-dimensional into a 24-dimensional vector. When taking a test set and passing it through a VAE (encoding and decoding it back again), we found that $\sim 1/3$ of the predictive power of the PLM embeddings is lost. We hypothesize that this might be related to the variational nature of the VAE and that a different autoencoder formulation might be a better choice. Wasserstein Auto-Encoders [Tol+19] are promising candidates, but we have not yet had the time to explore this avenue.

The method can (and does) propose sequences that have already been discovered in the previous rounds of optimization. This happens because the acquisition function operates on the continuous latent space Z, and after decoding and mapping to a discrete space of the protein sequences with a PLM, the two seemingly different PLM vectors can map to the same protein sequence. The inclusion of one-hot encoded protein sequences as an input to the GP model is an attempt to mitigate this problem, but it is not a perfect solution. A carefully designed kernel function for the GP surrogate model operating on the PLM posterior (in contrast to embeddings) might be able to solve this problem. For now, we filter out the duplicates after each optimization step.

The method is most certainly biased towards a generation that follows the statistics of naturally occurring proteins. This is a limitation of the PLM itself. PLM can be swapped to any other model that can encode and decode proteins to and from a vector space. One could imagine a PLM model that is trained on a dataset of artificial proteins containing unnatural amino acids.

With the current approach, we cannot constrain to a maximum number of mutations as the Hamming distance is not differentiable. Relaxations are possible, but not trivial. This forces us to filter out the results after the optimization is done.

Our exploration of a Probabilistic Generative Model for Automatic Guided Drug Discovery represents a comprehensive approach to addressing all three Research Questions posed at the outset of this thesis.

First, we have developed an efficient and scalable framework for protein sequence optimization (RQ1). The conditional variational autoencoder design, coupled with the distance-aligning technique, has enabled us to create a meaningful, regularized, low-dimensional latent space representation of protein sequences specifically tailored for Bayesian optimization (RQ2). Furthermore, by leveraging the pre-trained ESM2 model as our PLM, we have effectively utilized state-of-the-art language models to guide the optimization process and expand the search space beyond the immediate training data (RQ3).

The empirical results on the red fluorescent protein benchmark demonstrate the effectiveness of our approach, significantly outperforming existing methods in terms of hypervolume improvement and Pareto frontier expansion.

As we move to the conclusion of this thesis, we will reflect on the broader implications of our work, its contributions to the field of computational drug discovery, and outline promising directions for future research that build upon the foundations laid here.

Chapter 5

Conclusion

This thesis addressed three specific research questions in the field of protein design using Bayesian Optimization. Our work has made progress in creating an improved Bayesian optimization framework for protein sequences, developing a generative model with a more suitable latent space representation, and incorporating state-of-the-art Protein Language Models in the optimization process.

The following summarizes our key findings and contributions.

Addressing Research Question 1, which asked about designing an efficient and scalable Bayesian optimization loop for protein sequences, we formulated and implemented an alternative BO approach based on first-order gradient methods (SGD). This method is specifically tailored for protein discovery in the latent space of a generative model. Unlike traditional LFBG-S-based approaches that typically yield a single-point estimate of the next query point, our framework optimizes multiple points in parallel. During optimization, we continuously monitor the acquisition value of these points and reset a fraction of those considered stuck in local minima, enhancing robustness and diversity. In our formulation, each BO round concludes with a set of high-value acquisition points that can be easily filtered according to user-defined constraints. This approach allows for more efficient exploration of the generative model's latent space and addresses the challenge of expressing non-differentiable (or hard-to-express) constraints, such as requiring a proposed protein to be a fixed Levenshtein distance from a reference pro-

tein.

To address Research Question 2, which focused on designing a generative model with a meaningful, regularized, low-dimensional latent space representation, we proposed a method to shape the latent space of our generative model (VAE) using a distance-matching loss. This approach manifests as an extra term in the ELBO, maximizing the likelihood of the distribution of squared distances between all pairs of data points in the mini-batch. We developed a computationally efficient approximation to this term, demonstrating its effectiveness in shaping the latent space to improve the efficiency of acquisition function optimization.

Finally, in response to Research Question 3, which asked how to leverage state-of-the-art Protein Language Models (PLMs) effectively, we proposed a method to incorporate a PLM into the acquisition function. By learning and reconstructing the PLM embeddings in a specialized VAE, we were able to use the PLM to guide the search towards novel protein sequences beyond the immediate observed dataset.

The effectiveness of these proposed methods was evaluated on a well-known protein benchmark, where our approach significantly outperformed previously published methods in terms of the multi-target maximization objective.

5.1 Future directions

Having established the feasibility of the proposed method, and relaying on the fact that we implemented everything in JAX, a clear academic direction of future work is to equip the learned representation with a Riemannian metric [Car92]

$$\mathbf{g}_\mathbf{z} = \mathbf{J}_\mathbf{z}^T \mathbf{J}_\mathbf{z}$$

where $\mathbf{J}_\mathbf{z}$ is the Jacobian of the decoder network with respect to the latent embeddings $\mathbf{z}$.

Trivially computable with JAX, this would allow us to navigate the latent space, smoothly interpolating between latent embeddings along a geodesic curve estimated on the data manifold. Moving between the latent embeddings would allow us to explore the latent space more systematically, potentially improving the quality of the proposed sequences. Additionally, biochemists can study decoded sequences alongside geodesic curves to garner insights into the impact of specific mutations and their influence on the target values.

In parallel with the above research-focused advancements, we recognize the im-

portance of translating our academic findings into practical tools for the pharmaceutical industry. To this end, our future work will also focus on refactoring the codebase to meet industry standards, improving robustness and scalability. This process will involve enhancing modularity, implementing comprehensive error handling, and optimizing performance for large-scale tasks.

We plan to develop extensive documentation, including API references, tutorials, and best practices, crucial for researchers and practitioners integrating our methods into existing drug discovery pipelines. To facilitate easy deployment and use, we aim to package our implementation as a standalone Python library with a user-friendly API, ensuring compatibility with popular scientific computing libraries.

Through these combined efforts in advancing the theoretical foundations and practical applications of our work, we aim to bridge the gap between academic research and industrial application, facilitating the adoption of our advanced Bayesian Optimization methods in real-world protein design and drug discovery processes.

Bibliography

[AHO21] D. Austin, T. Hayford, and United States. Congressional Budget Office. *Research and Development in the Pharmaceutical Industry*. Congressional Budget Office, 2021. URL: https://books.google.dk/books?id=-6huzgEACAAJ.

[All+19] Ethan C. Alley et al. "Unified rational protein engineering with sequence-only deep representation learning". In: *bioRxiv* (2019). DOI: 10.1101/589333. eprint: https://www.biorxiv.org/content/early/2019/03/26/589333.full.pdf. URL: https://www.biorxiv.org/content/early/2019/03/26/589333.

[AZ19] Abubakar Abid and James Zou. *Contrastive Variational Autoencoder Enhances Salient Features*. 2019. arXiv: 1902.04601 [cs.LG]. URL: https://arxiv.org/abs/1902.04601.

[Bal+20] Maximilian Balandat et al. "BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization". In: *Advances in Neural Information Processing Systems 33*. 2020. URL: http://arxiv.org/abs/1910.06403.

[Ben+23] Yoshua Bengio et al. *GFlowNet Foundations*. 2023. arXiv: 2111.09266 [cs.LG].

[Bra+18] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[Byr+95] R. Byrd et al. "A Limited Memory Algorithm for Bound Constrained Optimization". In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208. DOI: 10.1137/0916069. eprint:

http://epubs.siam.org/doi/pdf/10.1137/0916069. URL: http://epubs.siam.org/doi/abs/10.1137/0916069.

[Car92] M.P. do Carmo. *Riemannian Geometry*. Mathematics (Boston, Mass.) Birkhäuser, 1992. ISBN: 9783764334901. URL: https://books.google.dk/books?id=uXJQQgAACAAJ.

[CC15] Yongwook Choi and Agnes P. Chan. "PROVEAN web server: a tool to predict the functional effect of amino acid substitutions and indels". In: *Bioinformatics* 31.16 (Apr. 2015), pp. 2745–2747. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btv195. eprint: https://academic.oup.com/bioinformatics/article-pdf/31/16/2745/49035314/bioinformatics_31_16_2745.pdf. URL: https://doi.org/10.1093/bioinformatics/btv195.

[Che+24] Lixue Cheng et al. *ODBO: Bayesian Optimization with Search Space Prescreening for Directed Protein Evolution*. 2024. arXiv: 2205.09548 [q-bio.BM]. URL: https://arxiv.org/abs/2205.09548.

[DBB20] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. *Differentiable Expected Hypervolume Improvement for Parallel Multi-Objective Bayesian Optimization*. 2020. arXiv: 2006.05078 [stat.ML].

[Den12] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[Dev+19] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[Fer21] Andres Fernandez. *Deriving the Variational ELBO from Bayes Theorem*. https://aferro.dynu.net/math/elbo_derivation/. Accessed: 2023-11-13. 2021.

[Hau18] Søren Hauberg. *The non-central Nakagami distribution*. http://www2.compute.dtu.dk/~sohau/papers/nakagami2018/nakagami.pdf. Accessed: 2023-11-13. 2018.

[He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[HS88] Desmond Higgins and Paul M. Sharp. "CLUSTAL: A package for performing multiple sequence alignment on a microcomputer". English. In: *Gene* 73.1 (Dec. 1988), pp. 237–244. ISSN: 0378-1119. DOI: 10.1016/0378-1119(88)90330-7.

[HTT09] Tony Hey, Stewart Tansley, and Kristin Tolle, eds. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington: Microsoft Research, 2009. URL: http://research.microsoft.com/en-us/collaboration/fourthparadigm/.

[Hug+11] JP Hughes et al. "Principles of early drug discovery". In: *British Journal of Pharmacology* 162.6 (2011), pp. 1239–1249. DOI: https://doi.org/10.1111/j.1476-5381.2010.01127.x. eprint: https://bpspubs.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1476-5381.2010.01127.x. URL: https://bpspubs.onlinelibrary.wiley.com/doi/abs/10.1111/j.1476-5381.2010.01127.x.

[IHR23] Haque Ishfaq, Assaf Hoogi, and Daniel Rubin. *TVAE: Triplet-Based Variational Autoencoder using Metric Learning*. 2023. arXiv: 1802.04403 [stat.ML]. URL: https://arxiv.org/abs/1802.04403.

[Jai+23] Moksh Jain et al. *Multi-Objective GFlowNets*. 2023. arXiv: 2210.12765 [cs.LG].

[JSW98] Donald R. Jones, Matthias Schonlau, and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13 (1998), pp. 455–492. URL: https://api.semanticscholar.org/CorpusID:263864014.

[Jum+21] John M. Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596 (2021), pp. 583–589. URL: https://api.semanticscholar.org/CorpusID:235959867.

[KLB11] Elizabeth H. Kellogg, Andrew Leaver-Fay, and David Baker. "Role of conformational sampling in computing mutation-induced changes in protein structure and stability". In: *Proteins: Structure* 79 (2011). URL: https://api.semanticscholar.org/CorpusID:10999682.

[Kno06] J. Knowles. "ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems". In: *IEEE Transactions on Evolutionary Computation* 10.1 (2006), pp. 50–66. DOI: 10.1109/TEVC.2005.851274.

[Kri+23] Agustinus Kristiadi et al. *Promises and Pitfalls of the Linearized Laplace in Bayesian Optimization*. 2023. arXiv: 2304.08309 [cs.LG]. URL: https://arxiv.org/abs/2304.08309.

[KW22] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML].

[Lam19] Talley J. Lambert. "FPbase: a community-editable fluorescent protein database". In: *Nature Methods* 16 (2019), pp. 277–278. URL: https://api.semanticscholar.org/CorpusID:81979417.

[LBG] Benjamin Leader, Quentin Baca, and David Golan. "Leader, B., Baca, Q. J. & Golan, D. E. Protein therapeutics: a summary and pharmacological classification. Nature Rev. Drug Discov. 7, 21-39". In: *Nature reviews. Drug discovery* (). DOI: 10.1038/nrd2399.

[LC13] Jie Jack Li and Elias James Corey. *Drug discovery: practices, processes, and perspectives*. John Wiley & Sons, 2013.

[Lin+22] Zeming Lin et al. “Language models of protein sequences at the scale of evolution enable accurate structure prediction”. In: *bioRxiv* (2022).

[LLL17] Ruiwu Liu, Xiaocen Li, and Kit S Lam. “Combinatorial chemistry in drug discovery”. In: *Current Opinion in Chemical Biology* 38 (2017). Next Generation Therapeutics, pp. 117–126. ISSN: 1367-5931. DOI: https://doi.org/10.1016/j.cbpa.2017.03.017. URL: https://www.sciencedirect.com/science/article/pii/S1367593117300534.

[LS21] Johannes Linder and Georg Seelig. “Fast activation maximization for molecular sequence design”. In: *BMC Bioinformatics* 22.1 (Oct. 2021). ISSN: 1471-2105. DOI: 10.1186/s12859-021-04437-5. URL: http://dx.doi.org/10.1186/s12859-021-04437-5.

[LW13] Elisa T. Lee and John Wenyu Wang. *Statistical Methods for Survival Data Analysis*. 4th. Wiley Publishing, 2013. ISBN: 1118095022.

[MTM15] Chris J. Maddison, Daniel Tarlow, and Tom Minka. *A* Sampling*. 2015. arXiv: 1411.0030 [stat.CO]. URL: https://arxiv.org/abs/1411.0030.

[MTZ78] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. “The Application of Bayesian Methods for Seeking the Extremum”. In: *Towards Global Optimization* 2.117-129 (1978), p. 2.

[Mui82] Robb J. Muirhead. “Aspects of Multivariate Statistical Theory”. In: *Wiley Series in Probability and Statistics*. 1982. URL: https://api.semanticscholar.org/CorpusID:123513635.

[Oda19] Stephen G. Odaibo. “Tutorial: Deriving the Standard Variational Autoencoder (VAE) Loss Function”. In: *CoRR* abs/1907.08956 (2019). arXiv: 1907.08956. URL: http://arxiv.org/abs/1907.08956.

[PW07] D A Pereira and J A Williams. “Origin and evolution of high throughput screening”. In: *British Journal of Pharmacology* 152.1 (2007), pp. 53–61. DOI: https://doi.org/10.1038/sj.bjp.0707373. eprint: https://bpspubs.onlinelibrary.wiley.com/doi/pdf/10.1038/sj.bjp.0707373. URL: https://bpspubs.onlinelibrary.wiley.com/doi/abs/10.1038/sj.bjp.0707373.

[RIM17] Adam J. Riesselman, John B. Ingraham, and Debora S. Marks. *Deep generative models of genetic variation capture mutation effects*. 2017. arXiv: 1712.06527 [q-bio.QM].

[RMW14] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. 2014. arXiv: 1401.4082 [stat.ML].

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006, pp. I–XVIII, 1–248. ISBN: 026218253X.

[Sam20] Eytan Bakshy Sam Daulton Max Balandat. *Sample-efficient exploration of trade-offs with parallel expected hypervolume improvement*. https://research.facebook.com/blog/2020/12/sample-efficient-exploration-of-trade-offs-with-parallel-expected-hypervolume-improvement/. Accessed: 2023-11-29. 2020.

[San+98] Maria Sandberg et al. "New chemical descriptors relevant for the design of biologically active peptides. A multivariate characterization of 87 amino acids." In: *Journal of medicinal chemistry* 41 14 (1998), pp. 2481–91. URL: https://api.semanticscholar.org/CorpusID:13279646.

[SAN59] MUNUSWAMY SANKARAN. "On the non-central chi-square distribution". In: *Biometrika* 46.1-2 (June 1959), pp. 235–237. ISSN: 0006-3444. DOI: 10.1093/biomet/46.1-2.235. eprint: https://academic.oup.com/biomet/article-pdf/46/1-2/235/576732/46-1-2-235.pdf. URL: https://doi.org/10.1093/biomet/46.1-2.235.

[Sch+05] Joost Schymkowitz et al. "The FoldX web server: An online force field". In: *Nucleic acids research* 33 (Aug. 2005), W382–8. DOI: 10.1093/nar/gki387.

[Sha+05] Nathan Shaner et al. "Improved monomeric red, orange and yellow fluorescent proteins derived from Discosoma sp. red fluorescent protein". In: *Nature biotechnology* 22 (Jan. 2005), pp. 1567–72. DOI: 10.1038/nbt1037.

[Sha+16] Bobak Shahriari et al. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

[SK23] Anastasiia V. Sadybekov and Vsevolod Katritch. "Computational approaches streamlining drug discovery". In: *Nature* 616 (2023), pp. 673–685. URL: https://api.semanticscholar.org/CorpusID:258336875.

[SLY15] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.

[SO77] J. Sheil and I. O'Muircheartaigh. "Algorithm AS 106: The Distribution of Non-Negative Quadratic Forms in Normal Variables". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 26.1 (1977), pp. 92–98. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2346884 (visited on 11/15/2023).

[Sri+12] Niranjan Srinivas et al. "Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting". In: *IEEE Transactions on Information Theory* 58.5 (May 2012), pp. 3250–3265. ISSN: 1557-9654. DOI: 10.1109/tit.2011.2182033. URL: http://dx.doi.org/10.1109/TIT.2011.2182033.

[Sta+22] Samuel Stanton et al. *Accelerating Bayesian Optimization for Biological Sequence Design with Denoising Autoencoders*. 2022. arXiv: 2203.12742 [cs.LG].

[Suz+14] Baris E. Suzek et al. "UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches". In: *Bioinformatics* 31.6 (Nov. 2014), pp. 926–932. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu739. eprint: https://academic.oup.com/bioinformatics/article-pdf/31/6/926/49011550/bioinformatics_31_6_926.pdf. URL: https://doi.org/10.1093/bioinformatics/btu739.

[Tem76] N.M Temme. "On the numerical evaluation of the ordinary bessel function of the second kind". In: *Journal of Computational Physics* 21.3 (1976), pp. 343–350. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(76)90032-2. URL: https://www.sciencedirect.com/science/article/pii/0021999176900322.

[Tol+19] Ilya Tolstikhin et al. *Wasserstein Auto-Encoders*. 2019. arXiv: 1711.01558 [stat.ML].

[Vas+23] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[Veb+02] Daniel F Veber et al. "Molecular Properties That Influence the Oral Bioavailability of Drug Candidates". eng. In: *Journal of medicinal chemistry* 45.12 (2002), pp. 2615–2623. ISSN: 0022-2623.

[War+15] Michael Waring et al. "An analysis of the attrition of drug candidates from four major pharmaceutical companies". In: *Nature reviews. Drug discovery* 14 (June 2015). DOI: 10.1038/nrd4609.

[Yan+19] Kaifeng Yang et al. "Multi-Objective Bayesian Global Optimization using expected hypervolume improvement gradient". In: *Swarm and Evolutionary Computation* 44 (2019), pp. 945–956. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2018.10.007. URL: https://www.sciencedirect.com/science/article/pii/S2210650217307861.

[YMW22] Ziyue Yang, Katarina A. Milas, and Andrew D. White. "Now What Sequence? Pre-trained Ensembles for Bayesian Optimization of Protein Sequences". In: *bioRxiv* (2022). DOI: 10.1101/2022.08.05.502972. eprint: https://www.biorxiv.org/content/early/2022/08/06/2022.08.05.502972.full.pdf. URL: https://www.biorxiv.org/content/early/2022/08/06/2022.08.05.502972.

[Zai+22] Yevgen Zainchkovskyy et al. *Probabilistic thermal stability prediction through sparsity promoting transformer representation*. 2022. arXiv: 2211.05698 [stat.ML].

Guiding directed protein evolution with Bayesian Optimization

Guiding directed protein evolution with Bayesian Optimization

Yevgen Zainchkovskyy [1 2]

Abstract

We look into a possibility of improving the current direct evolution workflow by reducing the experimental effort associated with directed protein evolution. By incorporating Variational Autoencoders, a novel unsupervised deep generative model and Bayesian Optimization we seek to reduce the number of expensive experiments needed to find the target protein with desired properties. Starting with a discrete sequence of amino acids of our protein of choice (the wild-type) and it's multiple-sequence-aligned neighboring sequences (being proteins with the same function found elsewhere in nature), we use a VAE to learn a non-linear mapping from discrete sequence of aminoacids into a latent continuous space. We then use Bayesian Optimization in the latent space to propose promising changes to the wildtype protein (generating mutants). Finally we map the mutants back to the original sequence space for experiments in the lab. The proposed approach is validated on a large published emperical fitness landscape for all 4997 single mutations in TEM-1 β-lactamase selecting for the wild-type function (Stiffler et al., 2015). We learned that our VAE+BO approach significantly outperforms random mutant selection as it easily finds at least two of the best 9 mutants in less than 200 steps.

1. Background

The field of protein engineering is solving the problem of developing useful and valuable proteins. To do that, two common strategies are employed: rational protein design and directed evolution. Those strategies are not mutually exclusive, however for the sake of brevity, allow us to be very general in our description where we tried to summarize the main differences of both methods.

[1]Section for Cognitive Systems, Department of Applied Mathematics and Computer Science, Technical University of Denmark [2]Novo Nordisk A/S. Correspondence to: Yevgen <yeza@dtu.dk>.

Within **rational protein design** (RPD), the detailed knowledge of the structure and function of a protein is used to design new variants. This has the advantage of being both inexpensive and technically feasible, however the downside of this method is the limited availability of detailed structural information (3D structure). Additionally, this approach suffers from the fact that it is very difficult to predict the effects of various mutations. This is due to structural information only providing a static representation of a protein structure.

Mimicking natural evolution, **directed evolution** (DE) is utilizing random mutations applied to the protein of interest (called *wildtype*). A selection regime is then used to select variants having desired properties. In contrast to rational protein design, directed evolution does not require a prior structural knowledge of a protein and allows for wider exploration, but is expensive due to high cost of high-throughput screening. Not all desired activities can be screened for easily. Also, this technique is not applicable to all proteins.

2. Introduction

Motivated by the idea of reducing the number of costly high-throughput candidate screenings, we setup an experiment where we wish to test the applicability of Bayesian Optimization in the context of candidate selection. Recall that there are an enormous number of ways a protein can be mutated: at every position in the chain, a single aminoacid can be swapped into 19 other possible variants (and that number is even larger if we include non-naturally occurring aminoacids). A quick calculation reveals that for 250-amino-acid protein, there are $19 * 250 = 4,750$ possible substitutions, and for double substitutions, the number is $4750 * (4750 - 19) = 22,472,250$. Further aggravating the problem, we must also note that the vast majority of mutations lead to unfolded, useless proteins. Jumping a little forward, consider Figure 1, where we depict a histogram of the experimentally measured protein fitness for all possible single-site mutations to the TEM-1 β-lactamase enzyme. Here it is clearly seen that almost all mutations lead to a very substantial reduction in protein fitness.

Zooming in on the far right of the histogram in Figure 1 (depicted in the bottom of the figure), we find the top 9 mutations which lead to an increased protein fitness. Now,

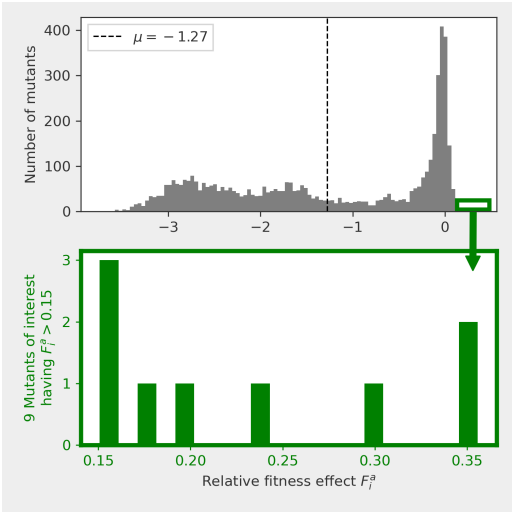

Figure 1. (top) A histogram of the experimentally measured protein fitness for all possible single-site mutations to the TEM-1 β-lactamase enzyme. (bottom) A zoomed in view on the specific region of the histogram where we find the rare mutations which resulted in the increase of the protein fitness.

for TEM-1 β-lactamase enzyme, this specifically means that those mutants provide an increased antibiotic resistance in E-Coli bacteria.

Fantasizing about work in the field of super-bacteria research, we imagine that it can be beneficial to artificially create E-Coli bacteria with those mutations for the purpose of staying ahead in novel antibiotic research. Those specific, 9 best mutations and their experimentally measured fitness values are presented in Table 1. Our tasks here is to see if it is possible to find one or more of those without traversing all possible single point mutations. Before we move on to the topic of efficiently searching for one of those 9 mutants, let us note that the TEM-1 β-lactamase has the length of 263 aminoacids, and the experimental data includes the fitness measurements[1] for all the $263 * 19 = 4997$ single mutations. For complete description of the β-lactamase data and experiments, we refer to the original source (Stiffler et al., 2015).

Before employing Bayesian Optimization, however, we are going to need a more optimal lower dimensional space to search in. As we stated before, the full combinatorial space of all the different mutations is huge, and by naively trying to optimize in the original space, we would not gain much, as the procedure will be completely uninformed, blindly suggesting candidates in an endless void of unfoldable invalid proteins. Additionally, we note the discrete nature of the original space and the problem of applying traditionally formulated BO to the functions of discrete variables. The problem here is that traditional Gaussian Processes based BO employs gradient based methods and those do not work for discrete variables[2].

3. Latent variable model

Proteins have evolved as a result of a long-term evolutionary processes that select for functional molecules. It therefore appears beneficial to try to probabilistically model the underlying process, as this will open up the ability to inference plausibility of other mutations. We model the evolutionary process as a type of a canonical "protein generator" able to generate a protein x with probability $p(x|\theta)$ where parameters θ are fit to mimic the statistics of evolutionary data. Motivated by the need to reduce the dimensionality of the original space and the fact that it has to be continuous, we will model $p(x|\theta)$ with a nonlinear latent-variable model known as a Variational Autoencoder (VAE). Intuitively, we think of the model as being a generator which samples a hidden variable z from a prior $p(z) \sim \mathcal{N}(0, 1)$ and afterwards generates the actual protein x on the basis of conditional distribution $p(x|\theta, z)$. As z is never observed directly, we will use a marginal likelihood formulation

$$p(x|\theta) = \int p(x|z, \theta)p(z)\, dz$$

which considers all possible instantiations of the hidden variables z and integrates them out. Unfortunately, the integral above is intractable, but using Variational Inference (Bishop, 2006), we can form a lower bound on the log probability of $p(x|\theta)$. This lower bound, $\mathcal{L}$, is known as the evidence lower bound (ELBO):

$$\log p(x|\theta) \geq \mathcal{L}(\phi, x) = \mathbf{E}_q[\log p(x|z, \theta)] - D_{\mathrm{KL}}(q(z|x, \phi)||p(z))$$

Where $q(z|x, \phi)$ is introduced as a variational proxy for the true posterior distribution $p(z|x, \theta)$ of hidden variables given the observations. Looking more closely at the ELBO, we see that it consists of two parts, the reconstruction part $\mathbf{E}_q[\log p(x|z, \theta)]$ which encourages the decoder to learn to reconstruct the data, and the D_{KL} part which acts as a regularizer forcing the latent variables z to be distributed close to a unit Gaussian. Practically speaking, VAE will strive to be general according to it's reconstructions and arrange similar proteins closely in the latent space as it will be penalized otherwise.

[1] Protein fitness is assessed as the logarithm of the allele counts in the selected population

[2] at least without tricks

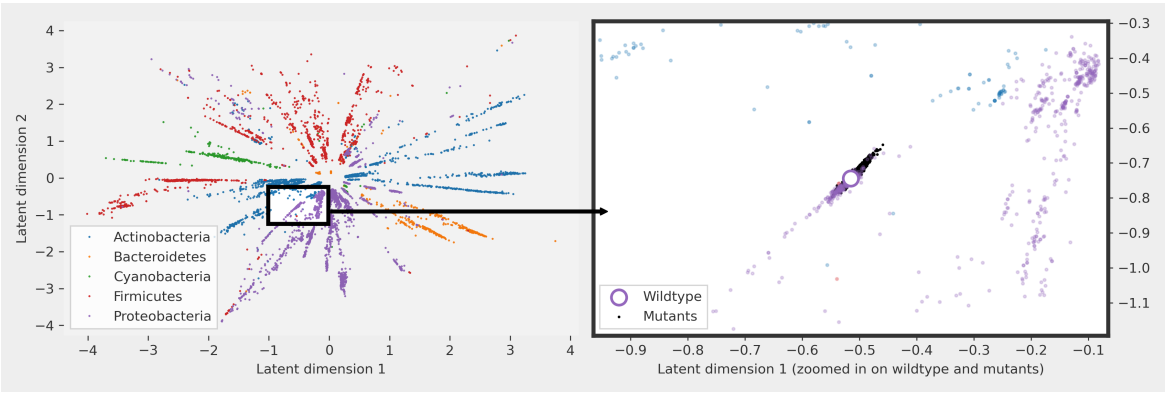

Figure 2. (left) Mapping of the Multiple Sequence Aligned β-lactamase proteins into a two dimensional latent space. (right) Zoomed in portion of the latent space with the wildtype and it's single-site mutations.

To train our VAE, we used a publicly available dataset consisting of 8403 entries of similar proteins found via the Multiple Sequence Aligned procedure (Wikipedia contributors, 2020). Those are the proteins which have the same neutralizing antibiotics-function, however, they originate from different species in nature. It turns out (and this is already well known), VAEs are able to exploit the underlying similarities between similar proteins. For the purpose of demonstration, we train a separate version of VAE with only 2 dimensions in the latent space. The result of the training is shown in Figure 2 where it is clearly seen that proteins which originate from the same species, are clustered together. Note that the training was done in an unsupervised fashion. In other words, the VAE model is trained exclusively on the 1-hot-encoded sequences of aminoacids originating from similar proteins in nature.

To further examine the latent space, we also encoded the mutants from (Stiffler et al., 2015) and were thrilled to discover that they correctly aligned to the wildtype (see the zoomed in portion of the latent space in Figure 2).

At this point, we have obtained a promising Latent Variable Model allowing us to encode and decode a protein between a regularized lower-dimensional continuous latent space and the original discrete space. But we are not limited to merely using this bi-directional mapping. By benefitting from probabilistic nature of the VAE model, we will also extract an additional feature to be used in Bayesian Optimization: 'relative favorability' (RF). As described in (Riesselman et al., 2018), the probabilities that the model assigns to any given sequence, can be used as proxy for for the relative plausibility of a molecule satisfying functional constraints. The following ratio

$$\log \frac{p(x^{\text{mutant}}|\theta)}{p(x^{\text{wildtype}}|\theta)}$$

is considered a heuristic metric for the relative favorability of a mutated sequence x^{mutant} compared to the sequence of the wildtype x^{wildtype}. In the following, we denote this metric as MODEL. We note that this metric correlates with the experimentally measured fitness value of the mutant proteins, F_i^a (Spearman $\rho = 0.74$) and is also included in Table 1 for completeness.

Table 1. **9 mutants of interest being the ones with the hightest experimentally measured fitness value (VALUE column).** MUTATION column describes the specific mutation to the wildtype, eg. `E210M` means that a change is made in position 210 where E (glutamic acid) is substituted with Y (tyrosine). Values in the MODEL column is the heuristic metric for the effect of the mutation (it is inferred in the VAE model in an unsupervised way). Columns LAT_DIM_0 to LAT_DIM_30 are the coordinates of the mutants in the latent space. We include those to emphasize their closeness and the fact the latent variable model were able to capture the underlying similarities. Being well aware of the handwavy nature of the last postulate given the non-linearity of the underlying latent space, we deliberately omit to provide a formal definition of similarity.

MUTATION	VALUE	MODEL	LAT_DIM_0	...	LAT_DIM_30
E210Y	0.3557	6.7868	-0.0180	...	0.0719
E210I	0.3485	2.2616	-0.0168	...	0.0718
A211M	0.2997	3.4414	-0.0173	...	0.0722
E210K	0.2328	3.3433	-0.0179	...	0.0721
E210M	0.1941	3.5376	-0.0166	...	0.0715
K190Q	0.1770	1.3670	-0.0176	...	0.0719
K190E	0.1549	3.2577	-0.0176	...	0.0722
E210H	0.1536	6.4048	-0.0186	...	0.0727
E210C	0.1507	6.6831	-0.0179	...	0.0719

4. Bayesian Optimization

As mentioned earlier, protein screening is an expensive and time consuming procedure. Furthermore, we also have the problem of combinatorial explosion in the number of possible mutations to the protein. One of the possible solutions, is to employ an informed searching procedure. Bayesian Optimization is one such technique allowing us to make informed decisions while being constrained by the high cost of the evaluation of the cost function. Recall that *evaluation* in this context constitutes to synthesizing a protein and testing (measuring) it in a lab.

Put a little more formally, Bayesian Optimization is a sequential procedure for global optimization of black-box functions which does not assume any functional forms. The way it works, is as follows.

1. First, we *choose a surrogate model* for modeling the true function f which in our case is the experimentally observed fitness value of a protein F_i^a. This surrogate model will be *a prior* over all possible f's.

2. Next we *observe some proteins* by evaluating the function. This will constitute to making an actual lab experiment where we measure the fitness value F_i^a of given proteins. After we obtain those measurements, the prior is updated (we have constrained the set of all possible f's to a set which now aligns with the observed measurements) resulting in *a posterior*.

3. With a help of an acquisition function $\alpha(x)$, we will make an informed decision on the location of the next promising protein in the latent space x_{t+1}. α itself is constructed by using the mean, μ, and uncertainty, σ, of the surrogate f's. Some acquisition function are parameterized, giving the user an ability to specify a tradeoff between exploitation and exploration.

4. Knowing our next sampling point x_{t+1}, we go to step 2 and continue the procedure from there.

The termination criteria of the above algorithms is up to the user, but there will typically be a budget associated with the experiments. Alternatively, one can terminate if an improvement is not achieved after a number of steps.

5. Experiment setup

The first ingredient in our experiment will naturally be the data. We denote our dataset $\mathcal{D}$ consisting of 4807 training pairs $(\mathbf{x}, t)$ where $\mathbf{x} = [z \quad \text{RF}]^\top$ is the input feature vector and y is the target variable (scalar) we seek to predict. The training procedure is then carried as follows: we initialize the BO loop with a sample of a 100 randomly selected observations[3] from the dataset. During the training, on every iteration, the procedure will suggest a new location x_{t+1} to be sampled. In a large-scale experiment, this suggestion would be mapped back to the original space through a decoder of the VAE and tested in the lab. As we do not have this luxury, a limitation will need to be imposed. Instead of obtaining a true value for x_{t+1} through a real experiment, we are going to find the closest, observed x in $\mathcal{D}$ and use it's corresponding target value $\hat{t}$ as a substitute for the real t. For the similarity metric between x_{t+1} and true, x's, we will use Euclidean distance. Obviously, this substitution will artificially constrain the optimization procedure to only see the effects of single-point mutations, but nevertheless, we hope to see some indication of an informed search. To quantify the success of the experiment, we will look at the mean of the target values of suggested mutants $\hat{\mu}_t$. It is clear that the procedure is working if $\hat{\mu}_t$ is high or at least higher than the mean of the targets across the full $\mathcal{D}$.

6. Results

We report selected findings obtained after completing a total of 54 experiments. Across the experiments we varied the dimensionality of the latent space, a choice of the GP kernel, initialization scheme and most importantly, the acquisition function and it's exploitation-exploration hyperparameter.

For the optimal dimensionality of the latent space, we learned that 8 dimensions worked best. This roughly aligned with the explained variance obtained by a PCA analysis on the latent space of 30 dimensions. In alignment with our intuition, running the optimization procedure in 30 dimensions resulted in low μ and large variance $\hat{\sigma}_t$ in the target values of the suggested mutants (See Figure 4). One possible explanation being the poor coverage of the latent space by the GP surrogate function due to the curse of dimensionality (Bishop, 2006).

For the initialization of BO loop, we hypothesized that it could be beneficial to favor samples of high relative favorability (RF, Section 3) as this metric correlates with the target value. In practice, however, it didn't yield better results. When initialized with samples of high RF, BO procedure would perform worse, although it generally recovers in the the training process (Figure 5). We think it might have to do with the need of the surrogate function to unlearn the imposed RF bias.

[3] This aligns with 10×10 sample plate used for screening in the lab.

Figure 3. Optimization progress across 5 experiments with varying β (exploitation-exploration tradeoff hyperparameter). In all experiments, the Bayesian Optimization procedure manages to suggest highly relevant mutations as seen by the high value of μ - the mean value of the fitness of the suggested targets. Arrows point to the first occurrence of the best mutant in the experiment. Comparing with Table 1, we note that the procedure successfully manages to find 2 out of 3 top mutants.

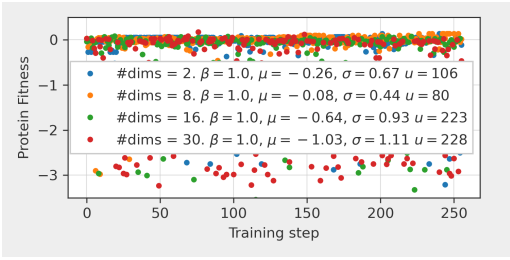

Figure 4. Finding the optimal size of the latent space. As seen on the high mean μ and low variance σ of the suggested targets, the optimal size of the latent space is 8.

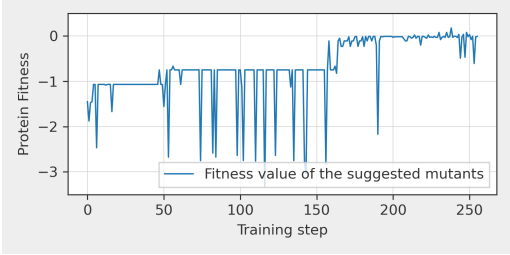

Figure 5. Optimization progress when initialized with a bad heuristic.

Trying different GP kernels for the surrogate function (RBF, Matern$^{\frac{1}{2}}$, Matern$^{\frac{3}{2}}$ and Matern$^{\frac{5}{2}}$), we learned that for our problem, Matern$^{\frac{5}{2}}$ performed best. This is not entirely surprising, as this kernel, being a generalization of the RBF) allows to capture more structure due to an extra parameter (ν). For all the kernels above, both the general and the ARD (automatic relevance determination) formulations were tried. Again, not surprisingly, ARD led to better results, due to it's ability to expresses each dimension as being independent from the other though a separate set of hyperparameters.

For the acquisition function and the exploitation-exploration hyperparameter, a grid search was performed. Here, we went through the most popular acquisition functions: Expected Improvement (EI), Probability of Improvement (POI) and finally Upper Confidence Bound (UCB). We learned that UCB performed best, however we cannot clearly pinpoint an optimal value for it's β hyperparameter as it depends on the actual goal of an experiment. By observing the mean of the mutants for the different values of β, we confirmed that the search is getting more exploratory for higher values of β. This can be seen in the general decrease in the mean target of the mutants μ an increase in the variance σ and a number of uniquely queried samples (u).

In Figure 3 we present the result of 5 experiments with varying β. In all experiments, the Bayesian Optimization procedure manages to suggest highly relevant mutations as seen by the high value of μ - the mean of the suggested targets (recall that the mean of targets in the full dataset is $\mu_t = -1.27$). While we did not manage to "find" the top `E210Y` mutation in any of our experiments, we did successfully locate the second and third. Interestingly, `A211M`, was found in two separate experiments.

7. Conclusions and future directions

Although we carried a rather hypothetical and constrained experiment, the potential of combining Variational Autoencoders and Bayesian Optimization is not to be dismissed. In this work, we have successfully shown that there are clear benefits of guiding the directed protein evolution workflow with novel machine learning models thereby potentially reducing costly experimental effort. Our central finding being the fact that the procedure, for the most part, allowed the exploration phase to focus on potentially functional mutations thereby possibly reducing the wasteful exploration of non-functional (unfoldable, invalid) proteins. Compared to strictly random single point mutations, the procedure was able to successfully propose new mutants in the area of the space where functional proteins reside. Generally, our suggested mutatants had an experimentally measured Fitness value $\mu \approx 0$ while the fitness level of underlying population was $\mu_t = -1.27$. We additionally note that the underlying true distribution is very heavily skewed towards low-fitness variants (see Figure 1).

Given the limitations imposed by the restricted scope of experimental data (only single point mutations), it is of our greatest interest to conduct a similar experiment on a larger scale. We strongly believe that a similar model will be able to propose good candidates even further away, in other words, we hypothesize that it is possible to suggest beneficial mutations in several positions in the chain at a single optimization step!

Related to the above, we also predict that improvements can be made given we augment input data with additional information. Similar to the 'relative favorability' feature (Section 3 (Riesselman et al., 2018)), potential other features could be employed through eg. molecular dynamics simulations (Gelpí et al., 2015) or protein vectors (Bepler & Berger, 2019).

Finally, we note an immediate improvement to the model itself. As we have described in Sections 3 and 5, the latent space is implicitly assumed to be Euclidean. In recent work it has been shown that this assumption decreases the capacity of the model, leading to lower performance. Therefore, by modifying the VAE to support Riemannian structure over the latent space, a potential improvement can be achieved (Kalatzis et al., 2020). Given the new latent structure, however, the GP-based surrogate model in Bayesian Optimization, will most likely also require an update.

8. Data and code

The β-lactamase multiple sequence alignment dataset is taken from (Riesselman et al., 2018) and the single-point mutations are from (Stiffler et al., 2015). After training a VAE and mapping all the single-point mutations, an aggregated dataset was created (`data/df-*.csv`, where $\star$ refers to the dimensionality of the latent space: 2, 5, 8, 16, 30). The resulting experiment data resides in `results/` and includes a total 54 experiments. Across experiments we vary: latent space dimensionality, exploitation-exploration trade-off and initialization scheme (random or model-informed). The code is written in Python using the excellent Botorch library (an intermediate version based on `fmfn/BayesianOptimization` is also included). Using a GPU, expect a training time of around 5 minutes for a single experiment. See `http://github.com/eugene/prot-bo-0` for details.

References

Bepler, T. and Berger, B. Learning protein sequence embeddings using information from structure, 2019.

Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Gelpí, J., Hospital, A., Goñi, R., and Orozco, M. Molecular dynamics simulations: Advances and applications. *Advances and Applications in Bioinformatics and Chemistry*, 10:37, 11 2015. doi: 10.2147/AABC.S70333.

Kalatzis, D., Eklund, D., Arvanitidis, G., and Hauberg, S. Variational autoencoders with riemannian brownian motion priors, 2020.

Riesselman, A. J., Ingraham, J. B., and Marks, D. S. Deep generative models of genetic variation capture the effects of mutations. *Nature Methods*, 15(10): 816–822, 2018. ISSN 15487105. doi: 10.1038/s41592-018-0138-4. URL `https://doi.org/10.1038/s41592-018-0138-4`.

Stiffler, M., Hekstra, D., and Ranganathan, R. Evolvability as a function of purifying selection in tem-1 beta-lactamase. *Cell*, 160:882–892, 02 2015. doi: 10.1016/j.cell.2015.01.035.

Wikipedia contributors. Multiple sequence alignment, 2020. URL `https://en.wikipedia.org/wiki/Multiple_sequence_alignment`. [Online; accessed 12-Sep-2020].

Robust uncertainty estimates with out-of-distribution pseudo-inputs training

Robust uncertainty estimates with out-of-distribution pseudo-inputs training

Pierre Segonne
pierre.segonne@electricitymap.org

Yevgen Zainchkovskyy
yeza@dtu.dk

Søren Hauberg
sohau@dtu.dk

Abstract

Probabilistic models often use neural networks to control their predictive uncertainty. However, when making *out-of-distribution (OOD)* predictions, the often-uncontrollable extrapolation properties of neural networks yield poor uncertainty predictions. Such models then don't *know what they don't know*, which directly limits their robustness w.r.t unexpected inputs. To counter this, we propose to explicitly train the uncertainty predictor where we are not given data to make it reliable. As one cannot train without data, we provide mechanisms for generating *pseudo-inputs* in informative low-density regions of the input space, and show how to leverage these in a practical Bayesian framework that casts a prior distribution over the model uncertainty. With a holistic evaluation, we demonstrate that this yields robust and interpretable predictions of uncertainty while retaining state-of-the-art performance on diverse tasks such as regression and generative modelling.

1 Introduction

Neural networks generally extrapolate arbitrarily (Xu et al., 2020), and high quality predictions are limited to regions of the input space where the networks have been trained. This is to be expected and is only problematic if the associated predictions are not accompanied with a well-calibrated measure of uncertainty. If a neural network is used for estimating such a measure of uncertainty, we, however, quickly run into trouble, as the reported uncertainty then exhibits arbitrary behaviour in regions with no training data. Alarmingly, these are exactly the regions where evaluating the uncertainty is most important to the safe deployment of machine learning models in real world applications (Amodei et al., 2016). One potential solution is to avoid using directly the output of neural networks for predicting uncertainty, and let it emerge from another mechanism, e.g. an *ensemble* (Hansen and Salamon, 1990; Lakshminarayanan et al., 2017) or some notion of *Monte Carlo* (MacKay, 1992; Gal and Ghahramani, 2016). Here we explore the alternative view that the networks should simply be trained where there is no data. But can we train without data? The Bayesian formalism often does so implicitly: most *conjugate priors* can be seen as additional training data (Bishop, 2006), e.g. in Gaussian models, a mean prior $\mathcal{N}(\mu_0, \sigma_0^2)$ can be realised by additional training data of μ_0 with σ_0^2 setting the amount of observations. Placing a prior over the output of a neural network can, thus, be interpreted as additional training data. Unfortunately, this view is not practical as it implies additional data *for all* possible inputs to a neural network, resulting in infinite data. Our approach is simple: we locate regions of low data density in *input space* and implicitly place observations here in *output space* by minimising an appropriate KL divergence towards a prior (see Fig. 1). The result is a simple algorithm that significantly improves uncertainty estimates in both regression and generative modeling.

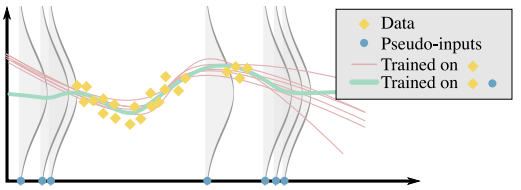

Figure 1: Pseudo-inputs are generated out of distribution, and there we train towards a prior (grey density).

1.1 Background and related work

The predictive performance of machine learning models has drastically increased in the past decade, but the quality of the accompanying uncertainties have not

followed. Uncertainties are reported as being miscalibrated (Guo et al., 2017) and overconfident (Lakshminarayanan et al., 2017; Hendrycks and Gimpel, 2016). Some models even see higher likelihoods of out-of-distribution than in-distribution data (Nalisnick et al., 2019; Nguyen et al., 2015; Louizos and Welling, 2017).

Neural networks commonly output distributions which gives a notion of predictive uncertainty. Classifiers trained with *soft-max* is an ever-present example of such. These predictions are generally observed to be *overconfident* (Lakshminarayanan et al., 2017; Hendrycks and Gimpel, 2016) and to carry little meaning outside the support of the training data (Skafte et al., 2019; Lee et al., 2017). The latter is an artifact of the hard-to-control extrapolation that comes with neural networks (Xu et al., 2021). In general, since extrapolation is difficult to control, uncertainties predicted by neural networks will exhibit seemingly arbitrary behavior outside the support of the data, yielding untrustworthy results.

Mean-variance networks for regression (Nix and Weigend, 1994) model the conditional target density as a Gaussian $p(y|x) = \mathcal{N}\left(y|\mu(x), \sigma^2(x)\right)$ with mean and variance predicted by neural networks. The predictive uncertainty is generally accurate in regions near training data, but otherwise unreliable (Hauberg, 2019). To counter this, Arvanitidis et al. (2017) and Skafte et al. (2019) proposed variance network architectures to enforce a specified extrapolation value, but these heuristics tend to be difficult to tune, and lack principle. Mean-variance networks have seen a recent uptake within generative modeling, where they are used as an *encoder* distribution in *variational autoencoders (VAEs)* (Kingma and Welling, 2013; Rezende et al., 2014).

Which uncertainty? A commonly called-upon dichotomy (Der Kiureghian and Ditlevsen, 2009) is that the uncertainty of a model's *prediction* can be decomposed into the uncertainty of the *model (epistemic)* and of the *data (aleatoric)*. The epistemic uncertainty can be lowered by increasing the amount of data, simplifying the model or otherwise reducing the complexity of the learning problem. The aleatoric uncertainty, on the other hand, is a property of the world, and cannot be changed; no prediction should ever be more certain than the uncertainty displayed by the associated data.

Bayesian methods are often used to quantify uncertainty due to their explicit formulation of uncertainty. *Gaussian processes (GPs)* (Rasmussen and Williams, 2005) provide an elegant framework that provide state-of-the-art uncertainty estimates, but, alas, the corresponding mean predictions are often not up to the standards of neural networks. GPs are tightly linked to *Bayesian neural networks (BNNs)* (MacKay, 1992) that place a prior over the network weights and seek the corresponding posterior. Despite advances in *variational approximations* (Graves, 2011; Kingma and Welling, 2013; Blundell et al., 2015), *expectation propagation* (Hernández-Lobato and Adams, 2015; Hasenclever et al., 2017), or *Monte Carlo* methods (Welling and Teh, 2011; Springenberg et al., 2016), training BNNs remains difficult. Furthermore, the predictive uncertainty seems dependent on the degree of approximation and is thus controlled by the available compute power.

Ensemble methods have long been used to produce aggregated predictions with uncertainty estimates (Hansen and Salamon, 1990; Breiman, 1996). *Deep ensembles* (Lakshminarayanan et al., 2017), a collection of differently initialized networks trained on the same data, are generally reported as state-of-the-art for uncertainty quantification in deep models (Thagaard et al., 2020; Ovadia et al., 2019). As the models in the ensemble are trained on overlapping data, they are correlated, which influence the ensemble uncertainty in ways that remains unclear (Breiman, 2001). *Monte-Carlo dropout* (Gal and Ghahramani, 2016) casts dropout training (Srivastava et al., 2014) as an ensemble model. It is computationally cheap, but experiments (Ovadia et al., 2019; Skafte et al., 2019) show that the increased correlation of ensemble elements causes overconfidence.

Robustness to distribution shift is part of a well-behaved uncertainty predictor (Ovadia et al., 2019) and must be evaluated accordingly. For out-of-distribution detection, Liang et al. (2017) proposes a pre-processing perturbation step inspired by adversarial attacks (Goodfellow et al., 2014a) that helps distinguish in-distribution and out-of-distribution inputs. Hendrycks et al. (2018) used a *Generative Adversarial Network (GAN)* (Goodfellow et al., 2014b) to generate out-of-distribution pseudo-inputs that appear in a regularizing term in the loss function, called *outlier exposure*, to enhance the predictor's ability to discriminate out-of-distribution inputs (Lee et al., 2017; Dai et al., 2017).

Out-of-distribution pseudo-inputs can improve uncertainty estimates. Deep ensembles trained to maximise diversity on such pseudo-inputs can display high uncertainty outside of their training support (Jain et al., 2020), under the strong assumption that uniformly distributed pseudo inputs can accurately capture the out-of-distribution support. High entropy priors on predictions, $p_{\text{prior}}(y|x)$, conditioned on OOD pseudo-inputs sampled from $p_{\text{prior}}(x)$, have also successfully regularised the uncertainty predictions of Bayesian models (Hafner et al., 2018; Malinin and Gales, 2018). But these negatively affect the *mean* predictions compared to contemporary regularisation techniques (Srivastava et al., 2014; Ioffe and Szegedy, 2015).

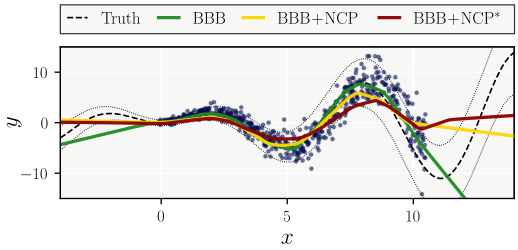

Figure 2: Noise contrastive priors improve uncertainty estimates, but overregularize the main prediction.

1.2 Robust uncertainty estimates

Notation. Let the observed variable $\mathrm{x} \in \mathcal{X}$ follow the data generating distribution $p_{\text{data}}(\mathrm{x})$, only known through the training dataset of N i.i.d samples $\mathcal{D}_{\text{train}} = \{\mathrm{x}_n\}_{n=1}^N$. In the case of supervised learning, the observed variables $\mathrm{x} = (x, y)$, with $x \in \mathbb{R}^d$ being the input and $y \in \mathbb{R}^{d'}$ the target for the model, follow the joint decomposition $p_{\text{data}}(x, y) = p_{\text{data}}(y|x)p_{\text{data}}(x)$. The proposed probabilistic model $p_\theta(\mathrm{x})$, whose weights are indicated by θ, aims to accurately emulate $p_{\text{data}}(\mathrm{x})$.

Practical problems in variance estimation. Gaussian likelihoods in the form of $p_\theta(\mathrm{x}) = \mathcal{N}\left(\mathrm{x}|\mu_\theta(\mathrm{x}), \sigma^2_\theta(\mathrm{x})\right)$ are widely adopted to model continuous covariates. Real world data cannot be expected to be *homoscedastic*, i.e constant throughout input space, and thus the predictive uncertainty, $\sigma_\theta(\mathrm{x})$, most often uses neural networks to map continuously the observed x onto the parameter space. Beyond the well-known unreliable extrapolation properties of neural networks, this parametrisation of predictive uncertainty is hamstrung by serious defects. Firstly, the predictive variance scales the learning rates of the mean and variance updates by $1/2\sigma^2_\theta(\mathrm{x})$, resulting in a bias for data regions with low uncertainty (Nix and Weigend, 1994). Secondly, the maximisation of the modelled likelihood is particularly sensitive to scarce data, as local gradient updates for the variance point towards the then undefined *maximum likelihood estimate (MLE)* (Skafte et al., 2019). Lastly, such model's likelihood is ill-defined (Mattei and Frellsen, 2018a), as it can without bound increase when the variance estimates collapse towards a detrimental 0. Overall, the naive maximisation of model likelihood is insufficient to generate robust and well-behaved uncertainty estimates, and practical implementations rely on an arbitrary lower threshold of the predictive variance.

Ensembles. The arbitrariness of the extrapolation of single mean-variance networks precludes any guarantees of robust uncertainty estimates. Previous contributions thus adopt a mixture of Gaussians as their predictive density. Whether discrete (Lakshminarayanan et al., 2017; Gal and Ghahramani, 2016; Jain et al., 2020) or continuous (Hafner et al., 2018), they commonly result in variance predictions expressed as a function of the mean. As shown in Fig. 2, this dependency creates a trade-off that limits the ability to improve the robustness of the uncertainty predictions without sacrificing some of the model's mean predictive power.

Student-t likelihood. Skafte et al. (2019) notably adopts a Gamma distributed precision, $\lambda \sim \Gamma(\alpha, \beta)$, as the conjugate of an unknown precision for a Gaussian, to yield a non-standard Student-t distributed marginal likelihood[1]. This infinite mixture of Gaussians is known to offer a more robust likelihood, especially in the scarce data regime (Gelman et al., 2013),

$$p_\theta(\mathrm{x}) = T\left(\mathrm{x}|\nu = 2\alpha, \hat{\mu} = \mu, \hat{\sigma} = \sqrt{\beta/\alpha}\right) . \quad (1)$$

Its variance $\mathrm{Var}[\mathrm{x}] = (\beta/\alpha)\cdot(\alpha/(\alpha-1))$ is explicitly decomposed into an aleatoric β/α and an epistemic term[1] $\alpha/(\alpha-1)$ (Jørgensen, 2020, p16), and offers a direct verification of whether a model knows what it knows.

Variational variance *(VV).* Stirn and Knowles (2020) assumes a latent model precision λ. It is generated by a prior $p(\lambda)$ and its posterior is approximated variationally by the family of Gamma distributions, conditioned on the inputs to reflect heteroscedasticity. Through *amortized variational inference (AVI)* (Kingma and Welling, 2013) neural networks f_ϕ map to the posterior parameters from data, $q(\mathrm{z}|f_\phi(\mathrm{x}))$. As such, variational variance preserves the modelling capacity and robustness of the Student-t marginal likelihood, without modifying its parameter architecture, while the definition of a prior over the precision induces a more robust training objective. Assuming the precision is the unique latent code, the *evidence lower bound (ELBO)*,

$$\mathcal{L}(q; \mathrm{x}) = \mathbb{E}_{q(\lambda)}\left[\log p(\mathrm{x}|\lambda)\right] - D_{\mathrm{KL}}\left(q(\lambda|\mathrm{x}) \,||\, p(\lambda)\right) \quad (2)$$

$$= \frac{1}{2}\left(\psi(\alpha) - \log\beta - \log(2\pi) - \frac{\alpha}{\beta}(\mathrm{x}-\mu)^2\right) - D_{\mathrm{KL}}\left(q(\lambda|\mathrm{x}) \,||\, p(\lambda)\right) , \quad (3)$$

takes the form of a regularised log-likelihood. It penalises predicted variances that would unrealistically get arbitrarily close to either the detrimental limits of 0 or ∞, reducing the concerns regarding the ill-definition of the objective. Additionally, the scaling effect of the learning rates of the likelihood parameters is reduced. Naturally, the effect of the regularisation will be highly dependent on the prior selected. Here, because we are mostly interested in enforcing a constant desired uncertainty extrapolation, we adopt an homoscedastic Gamma distributed prior, $p(\lambda) = \Gamma(\lambda|a, b)$, that matches the level of uncertainty observed in data.

[1] See Sec. I. of the supplementary materials.

2 Out-of-distribution pseudo-inputs

2.1 Dissipative loss

In variational variance, due to AVI, the uncertainty is controlled by α and β, the independent parameter maps of the posterior distribution, $\text{Var}[\text{x}] = \beta(\text{x})/(\alpha(\text{x}) - 1)$. The unreliable extrapolation properties of NNs therefore directly challenge the robustness of the method's uncertainty estimates outside of its training support.

Inspired by outlier exposure (Hendrycks et al., 2018) and noise contrastive priors (Hafner et al., 2018), we include deliberately generated out-of-distribution pseudo-inputs, $\{\hat{\text{x}}_k\}_{k=1}^K$ where $\hat{\text{x}}_k \sim p_{\text{out}}(\text{x})$, in the training of our variational objective to constrain the extrapolation of the posterior parametrisation. The optimal variational objective q^* is chosen such that it minimises our proposed *dissipative loss* over the combined dataset $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{out}}$, where $\mathcal{D}_{\text{out}} = \{\hat{\text{x}}_k\}_{k=1}^K$,

$$\text{L}(q; \mathcal{D}) = -\Big[\mathcal{L}_{\text{in}}(q; \mathcal{D}_{\text{train}}) + \mathcal{L}_{\text{out}}(q; \mathcal{D}_{\text{out}})\Big]. \quad (4)$$

The in-distribution component of the loss function $\mathcal{L}_{\text{in}}(q; \mathcal{D})$ naturally arises as the standard ELBO over the training set. The out-of-distribution component $\mathcal{L}_{\text{out}}(q; \mathcal{D})$ operates on a fundamentally different source of data. As the only information available for the pseudo-inputs is that they are OOD, we assert for them a constant, non-informative likelihood $p(\hat{\text{x}}|\lambda) = c$, that has thus no influence on optimisation. This is similar to *censoring* (Lee and Wang, 2003) where different likelihoods are used for observations with different properties. This simplifies the pseudo-inputs generation process, which no longer depends on modelling a likelihood, as is done by e.g Hafner et al. (2018), where the prior on targets is chosen as Gaussian data augmentation. As a result, the dissipative loss becomes,

$$\begin{aligned}\text{L}(q; \mathcal{D}) = &- \sum_{\text{x} \in \mathcal{D}_{\text{train}}} \mathbb{E}_{q(\lambda|\text{x})}\left[p_\theta(\text{x}|\lambda)\right] - D_{\text{KL}}(q(\lambda|\text{x}) \,||\, p(\lambda)) \\ &+ \sum_{\hat{\text{x}} \in \mathcal{D}_{\text{out}}} D_{\text{KL}}(q(\lambda|\hat{\text{x}}) \,||\, p(\lambda)). \end{aligned} \quad (5)$$

It shares the same motivating intuition as the *confidence loss* of Lee et al. (2017), which pushes a soft-max classifier towards the uniform distribution on OOD pseudo-inputs, and completes variational variance with a principled mechanism to learn robust variance estimates with the desired extrapolation properties. The predictor is indeed forced to match our high-entropy uncertainty prior expectations on out-of-distribution samples while learning the low-entropy covariate dependent distribution, hence the name of dissipative. The reliance of the model's predictive uncertainty on its mean predictions implies that it is primordial here to safeguard its generative performance. Previous contributions adopting a similar dual loss function (Lee et al., 2017; Jain et al., 2020; Hafner et al., 2018) contaminate the uncertainty regularisation with mean predictions, forcing joint learning of both loss terms and jeopardising the model mean predictive power. Conversely, the dissipative loss guarantees its conservation with the implementation of a split training procedure (Skafte et al., 2019); its modularity allows the application of the out-of-distribution regularisation only after the model's mean has been trained, which we view as a key conceptual advantage of our proposal.

2.2 Pseudo-input generators (PIGs)

Minimising the posterior KL divergence OOD requires an efficient sampling procedure of pseudo-inputs. As exposed in Fig. 3, their generation should leverage a-priori knowledge about $p_{\text{data}}(\text{x})$ to resolve the undefined nature of $p_{\text{out}}(\text{x})$. In this simple regression case, we show the predictive uncertainty of variational variance models trained on artificial heteroscedastic data. We use a prior uncertainty level that matches the maximum of the data uncertainty. As anticipated, without pseudo-inputs, the model extrapolates uncertainty to a constant, arbitrary level, and only the introduction of pseudo-inputs near the training data results in the desired uncertainty extrapolation. Reassuringly, this suggests that we do not need to regularise our model's extrapolation in the entire out-of-distribution space, as suggested by Jain et al. (2020). Instead, we can focus on the simpler task of generating pseudo-inputs in low-density regions of the input space that neighbours training data, as they can enforce correct extrapolation in the rest of the out-of-distribution space. Lee et al. (2017) gives supporting arguments for classification, and empirical results shows that this intuition generalises to higher dimension experiments[2].

Recent contributions have relied on GANs for generating a useful representation of $p_{\text{out}}(\text{x})$ (Lee et al., 2017; Dai et al., 2017). Although conceptually intuitive, GANs incur a heavy computational burden and induce serious practical challenges as a result of the instability of their training (Shrivastava et al., 2017). Furthermore, as one need to understand what is in-distribution to model what it is not, we instead propose to directly leverage the information at hand about the data.

Algorithm 1 generates pseudo-inputs with simple steps using the data density. Pseudo-inputs are initially drawn from $p_{\text{data}}(\text{x})$, and their positions iteratively updated with gradient descent to minimise their likelihood under $p_{\text{data}}(\text{x})$, similarly to reversed adversarial steps (Goodfellow et al., 2014a). In practice, we see

[2]Further experiments are in appendix II..1.

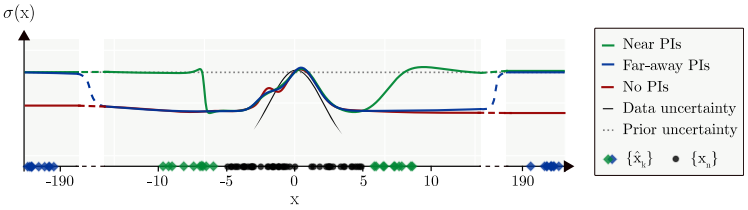

Figure 3: Predictive uncertainty of VV for different pseudo-inputs distributions. Training data is generated uniformly with $\text{Var}[\text{x}] \propto \exp\left(-0.5(||\text{x}||/s)^2\right)$.

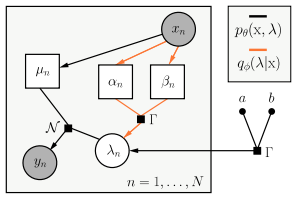

Figure 4: Graphical model for regression.

Algorithm 1: Pseudo-Input Generator (PIG)

$\forall k \in [1, K]$, $\hat{\text{x}}_k \sim p_{\text{data}}(\text{x})$. *iterations* $= 0$. $\epsilon = \infty$;
while *(iterations < max_ iterations) & ($\epsilon >$ tolerance)* **do**
  compute $\forall k \in [1, K], \nabla_{\text{x}} p(\text{x})(\hat{\text{x}}_k)$;
  $\epsilon = \max_{k\in[1,K]}(||\delta\, \nabla_{\text{x}} p(\text{x})(\hat{\text{x}}_k)||_2)$;
  $\forall k \in [1, K]$, $\hat{\text{x}}_k = \hat{\text{x}}_k - \delta\, \nabla_{\text{x}} p(\text{x})(\hat{\text{x}}_k)$;
  iterations $=$ *iterations* $+ 1$;
end

little sensitivity of the model uncertainty on the chosen hyperparameters and on the number of pseudo-inputs[3].

Remarkably, this modular procedure can run prior to training, in parallel for all $\hat{\text{x}}_k$ with automatic differentiation, and thus results in limited additional complexity for the optimisation[4]. It relies on the availability of a differentiable density estimate of the data, which is, depending on the use case, either directly available (see Sec. 3.2), or can be approximated through a variety of methods such as *Bayesian Gaussian mixture models* (Bishop, 2006) (see Sec. 3.1). A caveat here is that depending on the PIG's parameters, and on the quality of the density estimate available, pseudo-inputs might be generated in undesired regions of the input space, e.g uninformative density minima. In practice, we adopted conservative density estimates and did not observe any significant degradation of the predictive uncertainty.

3 Experiments

Holistic evaluation of uncertainty estimates. The ground truth for uncertainty is usually unknown, making its evaluation non-trivial. As in Stirn and Knowles (2020), we propose to assess it using multiple metrics. Calibration, which evaluates probabilistic predictions w.r.t the long-run frequencies that actually occur (Dawid, 1982) can be measured by *proper scoring rules* (Lakshminarayanan et al., 2017) such as the model log-likelihood $\log p_\theta(\text{x}|\lambda)$. Additionally, the *root mean squared error (RMSE)* between the predictive and empirical variance, $\text{Var}[\text{x}] - (\mu_\theta(\text{x}) - \text{x})^2$, quantifies the model's awareness of its own uncertainty. It nevertheless requires an understanding of the model's mean predictive performance, as commonly measured by the RMSE of the mean residuals, $\mu_\theta(\text{x}) - \text{x}$. We further evaluate the cooperation of mean and uncertainty estimates for generating credible samples, which constitutes a consistency check for the learned precision distribution (Gelman et al., 2013), by measuring the RMSE of sample residuals $\text{x}^* - \text{x}$, with $\text{x}^* \sim p_\theta(\text{x})$. Finally, The ELBO, despite the absence of theoretical grounding for it (Blei et al., 2017), is commonly reported as an approximation of the marginal likelihood, and thus of the model's predictive performance.

A complete assessment of a model's uncertainty further requires its evaluation under distributional shift (Ovadia et al., 2019), which we introduce voluntarily through deliberate splitting of the training set (Sec. 3.1), or by using test data from a different dataset (Sec. 3.2).

3.1 Regression

In a regression setting where the proposed model must capture the conditioning $y \,|\, x$, the precision λ of a Gaussian likelihood is the only assumed latent code.

Faithfully to variational variance (Stirn and Knowles, 2020) we adopt a Gamma heteroscedastic variational posterior $q_\phi(\lambda|x) = \Gamma\left(\lambda|\alpha_\phi(x), \beta_\phi(x)\right)$ parametrised by the independent α_ϕ and β_ϕ networks, with weights ϕ, uniquely conditioned on the inputs (see Fig. 4). This approximate posterior, independent of the targets, gives up on the dependency of the true posterior on both covariates to guarantee heteroscedasticity[5].

For more than 2 degrees of freedom, or, $\alpha_\phi(x) > 1$, the marginal likelihood $p_{\theta,\phi}(y|x) = T\left(y \,|\, 2\alpha_\phi(x), \mu_\theta(x), \sqrt{\beta_\phi(x)/\alpha_\phi(x)}\right)$, has its first two moments defined, $\mathbb{E}[y|x] = \mu_\theta(x)$ and $\text{Var}[y|x] = \beta_\phi(x)/(\alpha_\phi(x) - 1)$, providing explicit mean and uncer-

[3] Sec. II..4 of the supplements.
[4] Running times are reported in Sec. V.
[5] The true posterior is in Sec. III. of the supplements.

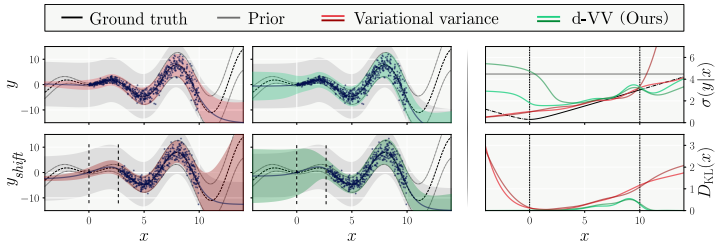

Figure 5: Toy regression results. On the left, are shown mean ± 2 std, with the training data of the bottom row presenting a shift. On the right are displayed the predictive uncertainty fit and the prior KL divergence.

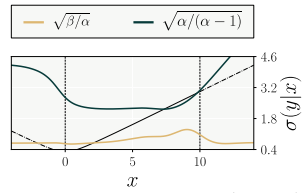

Figure 6: Aleatoric (yellow) and epistemic (dark) uncertainties.

tainty estimates with a single forward pass in the α_ϕ, β_ϕ and μ_θ networks. To ensure definition of both the posterior and marginal distributions' variance, the parameter maps use a soft-plus on their last layer to ensure positivity, and the α_ϕ network is further shifted by 1.

For pseudo-inputs generation, we estimate the input density prior to training with a Bayesian Gaussian mixture model (Bishop, 2006). We refer to it henceforth as *dissipative variational variance (d-VV)*. The implementation details are listed in Sec. III. of the appendix.

3.1.1 Toy regression

The desiderata for our method are clear: capture of the data heteroscedasticity, extrapolation to a higher uncertainty level, no underestimation of the predictive uncertainty, and posterior extrapolation to the prior out-of-distribution. Skafte et al. (2019) first showed on the toy regression task, $y = x\,\sin(x) + 0.3\,\epsilon_1 + 0.3\,x\,\epsilon_2$, where $\epsilon_1, \epsilon_2 \sim \mathcal{N}(0, 1)$, that amongst a collection of methods, only their proposed variance network architecture could realise our first three expectations. Fig. 5 demonstrates that our more principled approach also fulfills all of our requirements, without the need for arbitrarily enforcing the desired extrapolation in our architecture. The importance of out-of-distribution training is also revealed as the standard variational variance approach fails to produce uncertainty estimates that extrapolate correctly and are robust to distributional shift (bottom row of Fig. 5).

Decomposing model and data uncertainty. Fig. 6 shows that the aleatoric component captures the heteroscedastic increase of uncertainty in the training data while the epistemic uncertainty, constant in distribution, extrapolates to higher values.

3.1.2 UCI Benchmarks

Real world regression datasets from the UCI repository are used to evaluate our model against curated baselines, as in Hernández-Lobato and Adams (2015) and Skafte et al. (2019)[6]. We further copy each dataset with a distributional shift to assess the robustness of the methods. As in Foong et al. (2019), a shift is introduced, for each input feature, as a hole in the training data by assigning the middle third of observations to the test set, when sorted w.r.t that feature.

Tab. 1 aggregates best performances over the 24 datasets for two classes of methods[7]. To the left are methods that directly parametrise model uncertainty with neural networks, and to the right are methods that use ensembling for uncertainty estimation.

We note that *Bayes by backprop* offer a strong baseline, outperforming other ensemble methods. The instability of its performance on some datasets[7], as well as Hafner et al. (2018)'s demonstration of its overfitting in an active learning setting, challenge the reliability of its uncertainty estimates. As expected, the regularisation by NCP degrades significantly the mean predictions.

Our method performs best in its class. The ELBO reveals that it clearly strengthens the prior regularisation as introduced in VV, without significant degradation of the predictive power, as shown by other metrics. A caveat here is that the calibration of d-VV, as measured by the log-likelihood, suffers from the increased uncertainty of the method, which we deem acceptable if it leads to safer uncertainty estimates.

3.2 Generative models

We extend the evaluation of our proposal to the case of generative models through the lens of VAEs (Kingma and Welling, 2013; Rezende et al., 2014). VAEs infer a low dimensional latent encoding of the data $z \in \mathbb{R}^D$, on which is conditioned the generative process $p_\theta(\mathrm{x}|z)$. Its predictive uncertainty, which evaluates the confidence of the model in its ability to adequately reconstruct inputs is known to be untrustworthy.

[6] See Sec. III. for benchmark specification.

[7] Full results are included in Sec. III. of the supplements.

Table 1: Each cell counts datasets for which each method demonstrated the best average, over 5 trials. Grey shows statistical draws and "n/a" metrics impossible to evaluate for a method. Best per metric is highlighted in bold.

UCI benchmarks (shifts included)	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
$\mathcal{L}$	**21** / 18	3 / 3	0 / 0	n / a	n / a	n / a	n / a	n / a	n / a
$\log p(y\|x)$	2 / 7	5 / 9	0 / 0	0 / 0	4 / 9	0 / 0	0 / 0	2 / 8	**11** / 8
RMSE$[y, \mu(x)]$	2 / 2	1 / 4	2 / 5	1 / 5	2 / 3	5 / 12	3 / 16	0 / 0	**8** / 0
RMSE[Var]	2 / 4	5 / 9	2 / 4	0 / 0	n / a	2 / 5	5 / 17	0 / 0	**8** / 7
RMSE$[y, \tilde{y}]$	3 / 4	5 / 7	4 / 2	n / a	n / a	n / a	n / a	1 / 2	**11** / 7
$\mathbb{E}$[KL]	**23** / 14	1 / 0	0 / 0	n / a	n / a	n / a	n / a	n / a	n / a

In the case of continuous or seemingly continuous inputs, the adoption of a Gaussian decoder $p_\theta(\mathrm{x}|z) = \mathcal{N}\left(\mathrm{x}|\mu_\theta(z), \sigma^2_\theta(z)\right)$ results in an ill-defined model likelihood (Mattei and Frellsen, 2018a) that encourages decoder variance collapse, making the training of the model notoriously harder (Skafte et al., 2019). Most implementations therefore choose to fix the variance to a set level e.g $\sigma_\theta(z) = 0.1$, or elude the challenge by adopting a Bernoulli likelihood.

Motivated by our previous results, we now aim to demonstrate that VAEs, whose decoder is fitted with our method, are able to provide robust uncertainty estimates. Assuming a latent generative precision, the latent variables of the model are decomposed into $\mathrm{z} = \{z, \lambda\}$, with z the latent input representations. The marginalisation of the Gamma distributed latent variance results in a Student-T decoder, as detailed in Eq. 1. The *variational variance variational auto encoder (V3AE)* yields, with the addition of our OOD pseudo-inputs training, the dissipative loss function[8],

$$\begin{aligned} \mathrm{L}(q_\phi, \theta; \mathcal{D}_{\text{train}}) = &- \sum_{\mathrm{x} \in \mathcal{D}_{\text{train}}} \mathcal{L}(q_\phi, \theta; \mathrm{x}) \\ &+ \mathbb{E}_{q_{\text{out}}(z)} \left[D_{\mathrm{KL}} \left(q_\phi(\lambda|z) \,||\, p(\lambda) \right) \right]. \end{aligned} \tag{6}$$

Because only the decoder is regularised, the pseudo-inputs lie in the space of latent representations, $\mathcal{D}_{\text{out}} = \{\hat{z}_k\}_{k=1}^K \in \mathbb{R}^D$. The distribution of training inputs is therefore readily accessible as the aggregate posterior $q_\phi(z|\mathcal{D}_{\text{train}}) = q_\phi(z|\mathrm{x}_1) \cdots q_\phi(z|\mathrm{x}_N)$. Again, we rely on a split training procedure to leverage this perk; the encoder parameter maps μ_θ and σ_θ, as well as the decoder mean μ_ϕ are first trained until convergence, allowing the generation of the OOD pseudo-inputs and subsequently, the training of the decoder variance.

Image data. We evaluate the performance of our proposed *dissipative V3AE (d-V3AE)* against a fully Gaussian VAE on image data, coming from Fashion-MNIST, SVHN and CIFAR10. For both models, all parameter maps share the same underlying architecture,

[8]The derivation is provided in Sec. IV.

Table 2: Evaluation of the generative modelling. For each dataset, we report mean ± std over 5 trials.

		FashionMNIST	SVHN	CIFAR
$\log p(\mathrm{x})$	VAE	2215.54±68.81	**4304.90±58.45**	**2930.64±14.82**
	d-V3AE	**2349.71±11.80**	4133.41±64.28	2668.85±13.23
RMSE$(\mathrm{x}, \tilde{\mathrm{x}})$	VAE	0.171±0.003	0.097±7e-4	0.154±5e-4
	d-V3AE	**0.158±0.003**	**0.087±0.002**	**0.129±7e-4**

with the addition of either a softplus and/or a shifting last layer to ensure definition of both the variational and the generative distribution's moments[9].

Tab. 2 compares model performance on two metrics, the log-likelihood and the RMSE between the original inputs x and reconstructed samples $\tilde{\mathrm{x}}$, where $\tilde{\mathrm{x}} \sim p_\theta(\mathrm{x}|\lambda, z)\,, (\lambda, z) \sim q_\phi(\lambda, z|\mathrm{x})$. Unlike most previous implementations, we focus on actual samples, and not the mean, of the generative distributions. This comparison emphasize the cooperation between the decoder's mean and variance, allowing evaluation of the models' uncertainty estimates. Our method both qualitatively (Fig. 9), and quantitatively improves on a Gaussian VAE's sampling ability. The prior smoothens the uncertainty estimates, resulting in more realistic and less crisp samples. The log-likelihoods, evaluated at test time using truncation, i.e. $p_{\text{trunc}}(\mathrm{x}) = p_\theta(\mathrm{x})/(F_{\mathrm{x}}(1) - F_{\mathrm{x}}(0))$, to account for the finite support of data, reveal that our model can achieve a better fit, if the prior is selected correctly. In SVHN and CIFAR10, the presence of color channels complicates the selection process and challenges our choice of a single homoscedastic prior for all pixels and channels. We note that the dissipative loss also applies to classic VAEs with Bernoulli-only decoders; see Sec. IV. of the supplements for details.

Applications of robust generative uncertainty. In Figs. 7 & 8, the colouring of the 2D latent space represent the aggregated decoder variance $\sum_{i=1}^d (\sigma^2_\theta(z))_i$. It is clear that our method displays more regular uncertainty estimates, and provides the extrapolation guarantees we strove for. Beyond increased robustness

[9]Full details are included in Sec. IV. of the supplements.

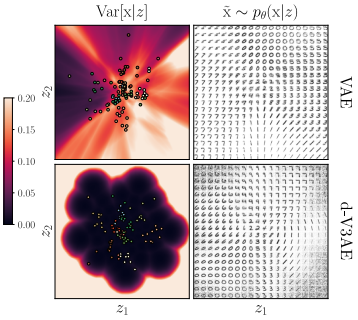

Figure 7: Decoder's aggregated variance (left) and generated samples (right) from the latent space. Coloured points correspond to latent representations of test data, with per-class colours.

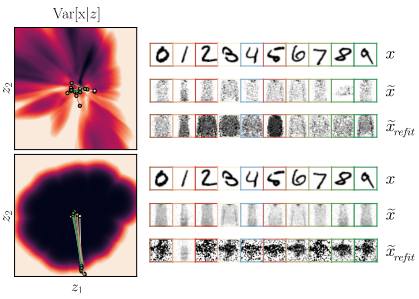

Figure 8: Effect of encoder refitting on the latent representations (left) and resulting samples (right). OOD inputs (top rows, x) initially result in in-distribution samples (second rows, $\tilde{x}$). The refitted encoder displaces the encodings (coloured trajectories), modifying the generated samples (third rows, $\tilde{x}_{\text{refit}}$).

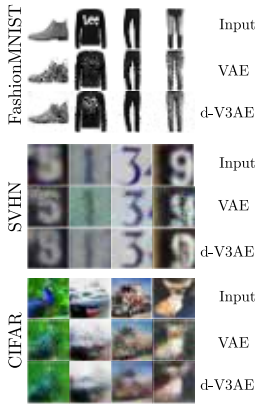

Figure 9: Generated samples.

and better generative power, this unlocks meaningful out-of-distribution detection, beating previous state-of-the-art (Havtorn et al., 2021)[10]. For Figs. 8 & 10, as argued in Mattei and Frellsen (2018b), we refit at test time the encoder of models trained on Fashion-MNIST on MNIST. The regularity and structure of the decoder variance rewards the encoder for learning to place representations of OOD data outside of the region of in-distribution latent encodings, resulting in a model that is aware of its own inability to reconstruct plausible data, as displayed by the row $\tilde{x}_{\text{refit}}$ of d-V3AE.

4 Conclusion

We have introduced a novel loss, the dissipative loss, that leverages artificial out-of-distribution pseudo-inputs for learning robust uncertainty estimates. We demonstrate through a Bayesian approach that casts a prior distribution over the model's variance a principled mechanism for controlling the extrapolation properties of neural networks governing the predictive uncertainty. Our results reflect the benefits of our principled and scalable approach, displaying better calibrated and more robust uncertainty estimates, while matching the predictive power of known baselines. Finally, and most interestingly, our approach can instill into probabilistic models a notion of their own ignorance, increasing their ability to *know what they don't know.*

The main limitation of our approach is that it depends on an input density estimate. In our experience, even coarse-grained densities are sufficient to significantly improve upon current approaches. However, as one rarely have guaranteed good estimates of the input density, our method cannot be approached as a black-box. One exception seems to be the application to VAEs, where the aggregated posterior, in our experience, always provide a suitable density estimate.

[10] Additional experiments are provided in Sec. IV..6

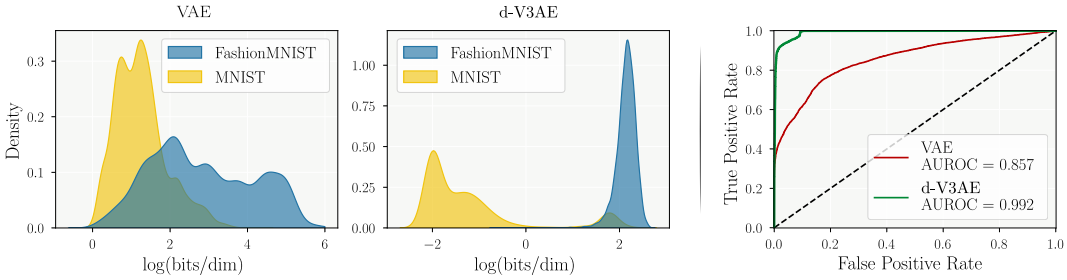

Figure 10: Empirical densities of likelihoods for FashionMNIST (ID) and MNIST (OOD). The clear separation of distributions offered by our method is reflected in the high AUROC shown on the right.

References

Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*, 2020.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4 (3):448–472, 1992.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, August 2006. ISBN 0387310738.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599*, 2017.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using typicality. *arXiv preprint arXiv:1906.02994*, 2019.

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

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

Nicki Skafte, Martin Jørgensen, and Søren Hauberg. Reliable training and estimation of variance networks. In *Advances in Neural Information Processing Systems*, pages 6326–6336, 2019.

Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325*, 2017.

Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks, 2021.

David A Nix and Andreas S Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, volume 1, pages 55–60. IEEE, 1994.

Søren Hauberg. Only bayes should learn a manifold (on the estimation of differential geometric structure from data), 2019.

Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. Latent space oddity: on the curvature of deep generative models. *arXiv preprint arXiv:1710.11379*, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural safety*, 31 (2):105–112, 2009.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24:2348–2356, 2011.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

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

Leonard Hasenclever, Stefan Webb, Thibaut Lienart, Sebastian Vollmer, Balaji Lakshminarayanan, Charles Blundell, and Yee Whye Teh. Distributed bayesian learning with stochastic natural gradient expectation propagation and the posterior server. *The Journal of Machine Learning Research*, 18(1): 3744–3780, 2017.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29:4134–4142, 2016.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Jeppe Thagaard, Søren Hauberg, Bert van der Vegt, Thomas Ebstrup, Johan D. Hansen, and Anders B. Dahl. Can you trust predictive uncertainty under real dataset shifts in digital pathology? In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Lima, Peru, October 2020.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, pages 13991–14002, 2019.

Leo Breiman. Random forests. *Machine learning*, 45 (1):5–32, 2001.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15 (1):1929–1958, 2014.

Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014a.

Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

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*, 27:2672–2680, 2014b.

Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Russ R Salakhutdinov. Good semi-supervised learning that requires a bad gan. In *Advances in neural information processing systems*, pages 6510–6520, 2017.

Siddhartha Jain, Ge Liu, Jonas Mueller, and David Gifford. Maximizing overall diversity for improved uncertainty estimates in deep ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4264–4271, 2020.

Danijar Hafner, Dustin Tran, Timothy Lillicrap, Alex Irpan, and James Davidson. Reliable uncertainty estimates in deep neural networks using noise contrastive priors. 2018.

Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. *arXiv preprint arXiv:1802.10501*, 2018.

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

Pierre-Alexandre Mattei and Jes Frellsen. Leveraging the exact likelihood of deep latent variable models. In *Advances in Neural Information Processing Systems*, pages 3855–3866, 2018a.

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. CRC press, 2013.

Martin Jørgensen. *Stochastic Representations with Gaussian Processes and Geometry*. PhD thesis, Technical University of Denmark, 2020.

Andrew Stirn and David A Knowles. Variational variance: Simple and reliable predictive variance parameterization. *arXiv preprint arXiv:2006.04910*, 2020.

Elisa T Lee and John Wang. *Statistical methods for survival data analysis*, volume 476. John Wiley & Sons, 2003.

Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017.

A Philip Dawid. The well-calibrated bayesian. *Journal of the American Statistical Association*, 77(379):605–610, 1982.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112 (518):859–877, 2017.

Andrew YK Foong, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. 'in-between'uncertainty in bayesian neural networks. *arXiv preprint arXiv:1906.11537*, 2019.

Jakob D. Havtorn, Jes Frellsen, Søren Hauberg, and Lars Maaløe. Hierarchical vaes know what they don't know, 2021.

P.-A Mattei and J Frellsen. Refit your encoder when new data comes by. In *3rd NeurIPS workshop on Bayesian Deep Learning*, 2018b.

Norman L Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous univariate distributions, volume 1, 2nd Edition*. John wiley & sons, 1994.

Christian Bauckhage. Computing the kullback-leibler divergence between two generalized gamma distributions. *arXiv preprint arXiv:1401.6853*, 2014.

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

I. Student-t likelihood

I..1 Marginal distribution of a Gaussian likelihood with a Gamma precision

In the case of a Gaussian likelihood with a latent Gamma distributed precision, the marginal distribution follows:

$$
\begin{aligned}
p_\theta(\mathrm{x}) &= \int \mathcal{N}(\mathrm{x}|\mu,\lambda)\Gamma(\lambda|\alpha,\beta)d\lambda \\
&= \int \frac{\lambda^{1/2}}{\sqrt{2\pi}} e^{-\frac{1}{2}\lambda(\mathrm{x}-\mu)^2} \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda} d\lambda \\
&= \frac{1}{\Gamma(\alpha)\sqrt{2\pi}} \frac{\beta^\alpha}{\left(\beta + \frac{(\mathrm{x}-\mu)^2}{2}\right)^{\alpha-\frac{1}{2}}} \int \left[\left(\beta + \frac{1}{2}(\mathrm{x}-\mu)^2\right)\lambda\right]^{(\alpha+\frac{1}{2})-1} e^{-\left(\beta+\frac{(\mathrm{x}-\mu)^2}{2}\right)\lambda} d\lambda \\
&= \frac{\Gamma\left(\alpha+\frac{1}{2}\right)}{\Gamma(\alpha)\sqrt{2\pi}} \frac{\beta^\alpha}{\left(\beta + \frac{(\mathrm{x}-\mu)^2}{2}\right)^{\alpha-\frac{1}{2}}} \\
&= \frac{\Gamma\left(\frac{2\alpha+1}{2}\right)}{\Gamma(\alpha)\sqrt{\pi 2\alpha}\left(\frac{\beta}{\alpha}\right)^{1/2}} \left(1 + \frac{1}{2\alpha}\left(\frac{\mathrm{x}-\mu}{\left(\frac{\beta}{\alpha}\right)^{1/2}}\right)^2\right)^{-\frac{2\alpha+1}{2}} \\
&= \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\nu\pi}\hat{\sigma}} \left(1 + \frac{1}{\nu}\left(\frac{\mathrm{x}-\hat{\mu}}{\hat{\sigma}}\right)^2\right)^{-\frac{\nu+1}{2}} \\
&= T\left(\mathrm{x}|\nu=2\alpha, \hat{\mu}=\mu, \hat{\sigma}=\sqrt{\beta/\alpha}\right) .
\end{aligned}
\tag{7}
$$

The moments of the marginal distribution are, assuming $\nu > 2$,

$$
\begin{cases}
\mathbb{E}[\mathrm{x}] & = \hat{\mu} = \mu \\
\mathrm{Var}[\mathrm{x}] & = \hat{\sigma}^2 \frac{\nu}{\nu-2} = \frac{\beta}{\alpha}\frac{\alpha}{\alpha-1} .
\end{cases}
\tag{8}
$$

I..2 Decomposition of a Student-t's uncertainty

The variance of a non standard Student-t distribution with 2α degrees of freedom and scaled by $\sqrt{\beta/\alpha}$ can be decomposed as $\mathrm{Var}[\mathrm{x}] = \frac{\beta}{\alpha}\frac{\alpha}{\alpha-1}$. The number of degrees of freedom scales with the number of observations that the distribution arises from. When the number of observations grows towards ∞, i.e towards perfect information, the term $\frac{\alpha}{\alpha-1}$ converges towards 1, motivating its casting into an epistemic factor. The natural consequence is that $\frac{\beta}{\alpha}$, which accounts for the rest of the model's uncertainty, scales as the aleatoric uncertainty. For more details see Jørgensen (2020, p16).

II. Pseudo-inputs generator

II..1 On boundary samples

Table 3: Comparison of the OOD prior KL for different pseudo-input distributions. Lower is better, and best per experiment are highlighted in bold.

method	CCPP	Wine-white
No OOD (VV)	0.655	1.719
Boundary OOD (d-VV)	**0.019**	**0.072**
Far OOD (d-VV)	0.023	0.941

Tab. 3 provides supporting evidence regarding the benefits of placing pseudo-inputs close to the training distribution. It compares the mean measured OOD KL divergence over 5 trials for two different UCI regression datasets. The boundary OOD pseudo-inputs were generated using the proposed density-based generation procedure. The far OOD pseudo-inputs were generated by adding large Gaussian noise ($\sigma = 15$ for standardised inputs) to training points. Boundary OOD pseudo-inputs result in a lower OOD prior KL divergence, meaning that the regularisation is indeed enforced in most of the OOD support.

II..2 Sensitivity to hyperparameters

Table 4: Evaluation of the influence of the number of steps in Alg. 1.

N steps	ELBO	$\log p(y\|x)$
0	**1.253**	1.360
1	1.232	1.361
5	1.192	1.423
10	1.183	**1.437**

Table 5: Evaluation of the influence of the threshold ϵ in Alg. 1.

Threshold ϵ	ELBO	$\log p(y\|x)$
0.1	1.193	**1.418**
0.01	1.192	1.418
0.001	**1.252**	1.359

The pseudo-input generator hyperparameters were chosen through a coarse grid search on a sub selection of experiments from the UCI benchmarks, and applied to the rest of the experiments. In practice we observed that Alg. 1 is relatively insensitive to hyperparameters. For example the Tab. 4 below describes the evolution of the ELBO and the log-likelihood as a function of the number of steps on the *Carbon* UCI dataset. The ELBO and LLK here have opposite evolutions based on the number of steps - hence our practical choice of a "middle ground" with n = 5. Tab. 5 evaluates similarly the influence of the threshold parameter. The results are naturally varying depending on the dataset chosen, as different overall input densities will more or less allow the gradient descent iterations to modify the original distribution of noisy pseudo-inputs. We regard this as a strength of our proposed method, if the distribution of pseudo-inputs can be improved and better separated from the training input distribution, it will leverage this possibility, but if it's not the case, pseudo inputs will not vary from their original position, generated as if they were inputs perturbed by Gaussian noise.

II..3 Influence of the pseudo-input generator

We originally envisioned to generate pseudo-inputs using the same Gaussian perturbation technique as independently proposed in Hafner et al. (2018), $\hat{x} = x + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. We nevertheless quickly come to realise that this does not actually produce pseudo-inputs that are guaranteed to be out-of-distribution, and eventually result in over-regularisation. Tab. 6 compares the mean ELBO over 5 trials of our proposal with selected parameters for the gradient descent (dVV), with our proposal without any gradient descent (dVV - 0 steps) and our method with pseudo-inputs generated as Gaussian noise (dVV - Gaussian noise, as is done in Hafner et al. (2018)) on the UCI benchmark. We observe that, as expected, better pseudo-inputs placement results in our method performing better on a larger selection of experiments (8/12).

Table 6: Comparison of the mean ELBO of dissipative VV on UCI datasets, over 5 trials, depending on the chosen pseudo-input generator. Best is highlighted in bold.

method	Boston	Carbon	CCPP	Concrete	Energy	Kin8nm	Naval	Protein	Superconduct	Wine-red	Wine-white	Yacht
d-VV	**-0.614**	1.191	-0.141	**-0.443**	**0.665**	**-0.247**	**0.522**	-1.319	**-0.543**	-1.997	-1.822	-17.599
d-VV (0 steps)	-0.658	**1.229**	**-0.131**	**-0.433**	0.656	-0.255	0.496	-1.317	-0.558	-2.192	-2.03	-21.802
d-VV (Gaussian noise)	-0.819	-0.473	-0.569	-0.645	-0.515	-0.591	-0.537	**-1.236**	-0.724	**-1.747**	**-1.554**	**-0.507**

II..4 Number of pseudo-inputs

In our implementation, to stabilise training, we use the expected value over each dataset (in or out-of-distribution) by dividing each term by the number of data points used to compute them. This results in a very limited sensibility of our practical implementation in the number of pseudo-inputs used.

III. Regression experiments

III..1 Variational Variance's ELBO Closed Form

For a Gaussian likelihood and a Gamma posterior, both terms of the ELBO have a closed form solution. Firstly the expected log-likelihood verifies:

$$\begin{aligned}\mathbb{E}_{q(\lambda|x)}\left[\log p(y|x,\lambda)\right] &= \int \log \mathcal{N}(y|\mu(x),\lambda)\Gamma\left(\lambda|\alpha(x),\beta(x)\right) d\lambda \\ &= \int -\frac{1}{2}\left(\log 2\pi - \log\lambda + \lambda(y-\mu(x))^2\right)\Gamma\left(\lambda|\alpha(x),\beta(x)\right) d\lambda \\ &= -\frac{1}{2}\left(\log 2\pi - \mathbb{E}_{q(\lambda|x)}[\log\lambda] + (y-\mu(x))^2\,\mathbb{E}_{q(\lambda|x)}[\lambda]\right) .\end{aligned} \tag{9}$$

The variational posterior being Gamma distributed, its expected value is defined as $\mathbb{E}_{q(\lambda|x)}[\lambda] = \frac{\alpha(x)}{\beta(x)}$. The logarithmic expectation of a Gamma distribution can be derived to yield (Johnson et al., 1994, 337–349) $\mathbb{E}_{q(\lambda|x)}[\log\lambda] = \psi(\alpha(x)) - \log\beta(x)$ where ψ is the digamma function. The closed-form expression of the expected likelihood is therefore:

$$\mathbb{E}_{q(\lambda|x)}\left[\log p(y|x,\lambda)\right] = -\frac{1}{2}\left(\log 2\pi - \psi(\alpha(x)) + \log\beta(x) + \frac{\alpha(x)}{\beta(x)}\left(y-\mu(x)\right)^2\right) . \tag{10}$$

Secondly, the KL-divergence between the posterior $\Gamma\left(\alpha(x),\beta(x)\right)$ and the prior $\Gamma(a,b)$ can be derived from Equation (28) in Bauckhage (2014, p6). With Bauckhage's notation, setting $p_1 = p_2 = 1$, to correspond to standard Gamma distributions, shape parameters $d_1 = \alpha(x)$ and $d_2 = a$, and scale parameters $a_1 = \frac{1}{\beta(x)}$ and $a_2 = \frac{1}{b}$ the KL-divergence can be expressed as

$$\begin{aligned}D_{\mathrm{KL}}(q(\lambda|x)\,||\,p(\lambda)) = &(\alpha(x)-a)\psi(\alpha(x)) \\ &- \log\Gamma(\alpha(x)) + \log\Gamma(a) \\ &+ a(\log\beta(x) - \log b) \\ &+ \alpha(x)\frac{b-\beta(x)}{\beta(x)} .\end{aligned} \tag{11}$$

III..2 True posterior and heteroscedasticity

The true posterior for variational variance in a regression context can be written $p(\lambda|y,x)$. As first demonstrated in Sec. 8.2 of Stirn and Knowles (2020), it factorizes as:

$$p(\lambda|y,x) = \frac{p(y|x,\lambda)p(\lambda)}{\int p(y|x,\lambda)p(\lambda)d\lambda} \tag{12}$$

$$= \frac{\Pi_{n=1}^{N} p(y_n|x_n,\lambda_n)p(\lambda_n)}{\int \Pi_{n=1}^{N} p(y_n|x_n,\lambda_n)p(\lambda_n)d\lambda_n} \tag{13}$$

$$= \Pi_{n=1}^{N}\frac{p(y_n|x_n,\lambda_n)p(\lambda_n)}{\int p(y_n|x_n,\lambda_n)p(\lambda_n)d\lambda_n} \tag{14}$$

$$= \Pi_{n=1}^{N} p(\lambda_n|y_n,x_n). \tag{15}$$

As a result, the true posterior both depends on the inputs x_n and targets y_n. It means that a single input, could theoretically imply different latent precisions for different targets $y_n \neq y_k$, thus violating the x-surjectivity of the heteroscedastic definition.

III..3 Model architecture

We adopted a unified network architecture for the regression case. All neural-network parameter maps share the same underlying architecture, a single hidden layer with 50 hidden units using *exponential linear unit (ELU)* activation functions. A final *softplus* layer is applied on the last layer of the σ, α and β parameter maps. The α parameter map is further shifted by $+1$ to ensure the definition of the marginal distribution's variance. Regression models are trained with the *Adam* (Kingma and Ba, 2014) optimiser, and both the inputs and targets are standardised prior to training and testing.

III..4 Pseudo-input generator

Tab. 7 presents the parameters used by the PIG in a regression setting. We remind that these parameters are parameters of a gradient descent, with learning rate δ. For our experiments, we approximated the input

Table 7: Parameters for the regression pseudo-input generator.

K	max_iterations	tolerance	δ
N	5	0.005	4e-1

density with a Bayesian Gaussian mixture model[11] with diagonal covariance matrices, and initialised with as many components as there are inputs in a batch.

III..5 UCI experiments

Table 8: UCI benchmarks

Name	Dimensions (N, D_x, D_y)	Link (https://archive.ics.uci.edu/ml/*)
Boston	(505,13,1)	machine-learning-databases/housing/
Carbon	(10721,5,3)	datasets/Carbon+Nanotubes
Concrete	(1030,8,1)	datasets/Concrete+Compressive+Strength
Energy	(768,8,2)	datasets/Energy+efficiency
Kin8nm	(8192,8,1)	`https://www.openml.org/d/189`
Naval	(11934,16,2)	datasets/Condition+Based+Maintenance+of+Naval+Propulsion+Plants
Power plant (CCPP)	(9568,4,1)	datasets/Combined+Cycle+Power+Plant
Protein	(45630, 9, 1)	datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure
Superconductivity	(21263,81,1)	datasets/Superconductivty+Data
Wine-red	(1599,11,1)	datasets/Wine+Quality
Wine-white	(4898,11,1)	datasets/Wine+Quality
Yacht	(308,6,1)	datasets/Yacht+Hydrodynamics

The UCI experiments (`https://archive.ics.uci.edu/ml/datasets.php`) consist of the datasets presented in Tab. 8.

The results, for the different metrics, as presented in Tab. 11 to 16, were computed as the mean $\pm$ the standard deviation over 5 trials with standardised inputs and targets. Due to a technical error, we were forced to re-run the experiments for *d-VV* and *VV (no PIG)* right before the submission deadline, and reduced the number of trials to 3 for these methods.

A method is deemed to perform best for a given metric when the mean of the evaluated metric is the best across methods. For determining statistical draws we ran for each method a two-sided test for verifying whether the mean of the evaluated metric μ is significantly different to the mean of the best method μ_{best}. To do so, we test for $\mu_{\text{best}} - \mu = 0$, under a Gaussian distribution with standard deviation $\frac{\sigma^2_{\text{best}}}{N_{\text{best}}} + \frac{\sigma^2}{N}$ at a 0.05 level.

[11] `https://scikit-learn.org/stable/modules/generated/sklearn.mixture.BayesianGaussianMixture.html`

III..6 Aggregate benchmark data

Table 9: Results for experiments **excluding distributional shifts**. Each cell counts datasets for which each method demonstrated the best average, over 5 trials. Grey shows statistical draws and "n/a" metrics impossible to evaluate for a method. Best per metric is highlighted in bold.

UCI benchmarks (shifts not included)	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
$\mathcal{L}$	**11** / 8	1 / 1	0 / 0	n / a	n / a	n / a	n / a	n / a	n / a
$\log p(y\|x)$	1 / 1	3 / 6	0 / 0	0 / 0	1 / 2	0 / 0	0 / 0	0 / 0	**7** / 6
RMSE$[y, \mu(x)]$	2 / 2	1 / 4	1 / 3	0 / 0	0 / 0	**4** / 10	0 / 0	0 / 0	4 / 0
RMSE[Var]	2 / 4	3 / 4	1 / 2	0 / 0	n / a	1 / 3	1 / 2	0 / 0	**4** / 4
RMSE$[y, \tilde{y}]$	2 / 2	3 / 3	2 / 1	n / a	n / a	n / a	n / a	0 / 0	**5** / 3
$\mathbb{E}$[KL]	**12** / 7	0 / 0	0 / 0	n / a	n / a	n / a	n / a	n / a	n / a

Table 10: Results for experiments **specifically for distributional shifts**. Each cell counts datasets for which each method demonstrated the best average, over 5 trials. Grey shows statistical draws and "n/a" metrics impossible to evaluate for a method. Best per metric is highlighted in bold.

UCI benchmarks (only shifts)	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
$\mathcal{L}$	**10** / 10	2 / 2	0 / 0	n / a	n / a	n / a	n / a	n / a	n / a
$\log p(y\|x)$	1 / 6	2 / 3	0 / 0	0 / 0	3 / 7	0 / 0	0 / 0	2 / 8	**4** / 2
RMSE$[y, \mu(x)]$	0 / 0	0 / 0	1 / 2	1 / 5	2 / 3	1 / 2	3 / 16	0 / 0	**4** / 0
RMSE[Var]	0 / 0	2 / 5	1 / 2	0 / 0	n / a	1 / 2	**4** / 15	0 / 0	4 / 3
RMSE$[y, \tilde{y}]$	1 / 2	2 / 4	2 / 1	n / a	n / a	n / a	n / a	1 / 2	**6** / 4
$\mathbb{E}$[KL]	**11** / 7	1 / 0	0 / 0	n / a	n / a	n / a	n / a	n / a	n / a

III..7 Benchmark raw data

Table 11: UCI benchmarks - $\mathcal{L}$

	UCI benchmarks	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
Not shifted	uci_boston	-0.61 ± 0.33	-0.72 ± 0.38	-64.28 ± 29.34	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_carbon	1.19 ± 0.11	1.17 ± 0.12	-3913.5 ± 580.03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_ccpp	-0.14 ± 0.01	-0.16 ± 0.04	-10.9 ± 1.06	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_concrete	-0.44 ± 0.13	-0.46 ± 0.08	-44.06 ± 14.23	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_energy	0.67 ± 0.03	0.65 ± 0.03	-118.78 ± 81.29	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_kin8nm	-0.25 ± 0.02	-0.28 ± 0.04	-16.76 ± 1.53	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_naval	0.52 ± 0.16	0.12 ± 0.4	-9.57 ± 4.57	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_protein	-1.32 ± 0.01	-1.34 ± 0.01	-14.31 ± 2.59	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_superconduct	-0.54 ± 0.03	-0.56 ± 0.01	-269.08 ± 58.88	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_wine_red	-2.0 ± 0.08	-2.36 ± 0.19	-37.86 ± 31.48	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_wine_white	-1.82 ± 0.06	-2.01 ± 0.1	-869.26 ± 1718.81	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_yacht	-17.6 ± 0.43	1.04 ± 0.1	-551.12 ± 1060.39	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
Shifted	uci_boston	-1.28 ± 0.32	-1.47 ± 0.28	-110.63 ± 110.55	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_carbon	1.16 ± 0.02	1.11 ± 0.03	-3991.86 ± 679.64	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_ccpp	-0.25 ± 0.08	-0.33 ± 0.14	-8.36 ± 1.65	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_concrete	-1.29 ± 0.27	-1.51 ± 0.38	-75.74 ± 59.7	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_energy	-0.69 ± 1.1	-0.56 ± 0.9	-891.92 ± 1570.09	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_kin8nm	-0.33 ± 0.03	-0.38 ± 0.04	-23.9 ± 1.92	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_naval	-4.73 ± 4.61	-5.31 ± 6.58	-26.02 ± 48.06	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_protein	-1.56 ± 0.11	-1.64 ± 0.16	-9.34 ± 6.68	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_superconduct	-1.43 ± 0.32	-1.52 ± 0.34	-224.0 ± 111.57	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_wine_red	-2.84 ± 0.21	-3.45 ± 0.35	-47.78 ± 56.02	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_wine_white	-2.04 ± 0.11	-2.53 ± 0.29	-346.83 ± 434.27	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_yacht	-7.59 ± 2.58	0.35 ± 0.19	-194.42 ± 149.45	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan

Table 12: UCI benchmarks - $\log p(y|x)$

	UCI benchmarks	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
Not shifted	uci_boston	-0.43 ± 0.35	-0.42 ± 0.39	-3.34 ± 1.39	-0.76 ± 0.07	-0.18 ± 0.19	-0.68 ± 0.04	-0.81 ± 0.51	-1.39 ± 0.33	-248.43 ± 163.55
	uci_carbon	1.45 ± 0.13	1.45 ± 0.11	0.98 ± 3.08	-3.78 ± 0.05	1.13 ± 0.51	-3.71 ± 0.04	0.29 ± 1.08	nan ± nan	nan ± nan
	uci_ccpp	-0.07 ± 0.01	-0.03 ± 0.03	0.05 ± 0.06	-0.58 ± 0.14	-0.18 ± 0.12	-0.61 ± 0.05	-3.36 ± 0.56	0.21 ± 0.04	4.06 ± 0.69
	uci_concrete	-0.29 ± 0.15	-0.25 ± 0.1	-0.83 ± 0.5	-0.68 ± 0.09	-0.4 ± 0.15	-0.65 ± 0.04	-0.9 ± 0.33	0.38 ± 0.04	3.84 ± 0.66
	uci_energy	0.87 ± 0.04	0.89 ± 0.03	0.47 ± 0.2	-1.22 ± 0.11	0.28 ± 0.37	-1.17 ± 0.04	0.36 ± 0.26	nan ± nan	nan ± nan
	uci_kin8nm	-0.17 ± 0.02	-0.15 ± 0.04	-0.36 ± 0.07	-0.61 ± 0.06	-0.61 ± 0.12	-0.65 ± 0.03	-0.63 ± 0.05	-0.68 ± 0.08	-0.16 ± 0.03
	uci_naval	0.71 ± 0.19	0.52 ± 0.2	-0.13 ± 0.32	-2.26 ± 0.08	-2.67 ± 0.22	-2.26 ± 0.06	-0.2 ± 0.74	nan ± nan	nan ± nan
	uci_protein	-1.16 ± 0.01	-1.12 ± 0.01	-1.42 ± 0.38	-1.13 ± 0.05	-1.54 ± 0.74	-1.05 ± 0.01	-7.41 ± 0.27	-1.02 ± 0.01	-0.96 ± 0.02
	uci_superconduct	-0.38 ± 0.02	-0.35 ± 0.02	-1.73 ± 1.71	-0.66 ± 0.04	-0.96 ± 0.18	-0.68 ± 0.03	-1.72 ± 0.25	-0.2 ± 0.19	-0.04 ± 0.06
	uci_wine_red	-1.91 ± 0.07	-2.13 ± 0.21	-7.77 ± 6.39	-2560.95 ± 5395.69	-1.15 ± 0.04	-1.24 ± 0.08	-4.24 ± 0.91	0.16 ± 0.04	3.76 ± 0.39
	uci_wine_white	-1.72 ± 0.06	-1.75 ± 0.1	-305.7 ± 549.73	-27.69 ± 48.8	-1.4 ± 0.58	-1.16 ± 0.08	-5.86 ± 1.08	0.29 ± 0.06	3.76 ± 0.82
	uci_yacht	0.9 ± 0.02	1.33 ± 0.11	0.63 ± 0.59	-0.59 ± 0.11	0.4 ± 0.14	-0.58 ± 0.04	0.33 ± 0.69	0.63 ± 0.1	1.57 ± 0.6
Shifted	uci_boston	-1.09 ± 0.32	-1.16 ± 0.27	-10.33 ± 10.87	-0.84 ± 0.09	-0.16 ± 0.09	-0.79 ± 0.07	-2.52 ± 1.34	-3.83 ± 1.82	-428.29 ± 194.72
	uci_carbon	1.34 ± 0.06	1.4 ± 0.02	-0.6 ± 2.9	-3.87 ± 0.34	1.12 ± 0.25	-3.71 ± 0.12	0.55 ± 0.1	nan ± nan	nan ± nan
	uci_ccpp	-0.18 ± 0.09	-0.2 ± 0.14	-0.1 ± 0.08	-0.54 ± 0.04	-0.16 ± 0.02	-0.65 ± 0.02	-4.31 ± 0.43	0.2 ± 0.07	3.76 ± 0.75
	uci_concrete	-1.11 ± 0.27	-1.23 ± 0.36	-10.81 ± 15.27	-0.77 ± 0.1	-0.38 ± 0.06	-0.78 ± 0.04	-2.38 ± 0.52	0.29 ± 0.05	4.17 ± 0.36
	uci_energy	-0.47 ± 1.15	-0.2 ± 0.76	-96.95 ± 259.19	-1.42 ± 0.26	0.2 ± 0.25	-1.36 ± 0.23	-1.13 ± 2.52	nan ± nan	nan ± nan
	uci_kin8nm	-0.26 ± 0.04	-0.22 ± 0.04	-0.88 ± 0.14	-0.64 ± 0.09	-0.59 ± 0.04	-0.65 ± 0.05	-0.86 ± 0.21	-0.68 ± 0.09	-0.26 ± 0.1
	uci_naval	-4.55 ± 4.62	-5.01 ± 6.53	-14.78 ± 18.92	-3.59 ± 0.89	-2.76 ± 0.15	-3.62 ± 0.89	-22.29 ± 12.05	nan ± nan	nan ± nan
	uci_protein	-1.44 ± 0.12	-1.42 ± 0.15	-2.26 ± 1.3	-1.33 ± 0.09	-1.51 ± 0.46	-1.21 ± 0.07	-9.94 ± 1.27	-1.16 ± 0.06	-1.21 ± 0.09
	uci_superconduct	-1.28 ± 0.32	-1.28 ± 0.33	-8.32 ± 11.59	-0.98 ± 0.11	-1.06 ± 0.16	-0.9 ± 0.06	-5.9 ± 1.74	-0.72 ± 0.28	-2.87 ± 5.54
	uci_wine_red	-2.7 ± 0.2	-3.1 ± 0.32	-12.13 ± 13.11	-229.56 ± 749.24	-1.14 ± 0.02	-1.57 ± 0.26	-4.42 ± 0.5	0.13 ± 0.1	3.84 ± 0.44
	uci_wine_white	-1.97 ± 0.12	-2.24 ± 0.27	-183.07 ± 323.25	-1.75 ± 0.23	-1.3 ± 0.13	-1.29 ± 0.04	-5.65 ± 0.49	0.3 ± 0.04	3.61 ± 0.88
	uci_yacht	0.71 ± 0.76	0.65 ± 0.19	0.51 ± 0.55	-0.53 ± 0.05	0.43 ± 0.07	-0.58 ± 0.05	0.37 ± 0.35	0.41 ± 0.2	-11.62 ± 25.18

Table 13: UCI benchmarks - RMSE $[y, \mu(x)]$

	UCI benchmarks	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
Not shifted	uci_boston	0.33 ± 0.09	0.33 ± 0.08	0.38 ± 0.09	0.35 ± 0.06	0.3 ± 0.07	0.29 ± 0.05	0.33 ± 0.05	0.47 ± 0.06	0.5 ± 0.04
	uci_carbon	0.03 ± 0.02	0.03 ± 0.02	0.03 ± 0.02	0.75 ± 0.01	0.09 ± 0.08	0.75 ± 0.01	0.08 ± 0.0	nan ± nan	nan ± nan
	uci_ccpp	0.23 ± 0.01	0.23 ± 0.01	0.23 ± 0.01	0.23 ± 0.01	0.27 ± 0.03	0.23 ± 0.01	0.24 ± 0.01	0.07 ± 0.03	0.0 ± 0.0
	uci_concrete	0.29 ± 0.04	0.29 ± 0.04	0.33 ± 0.04	0.29 ± 0.01	0.35 ± 0.07	0.27 ± 0.01	0.28 ± 0.02	0.08 ± 0.02	0.0 ± 0.0
	uci_energy	0.08 ± 0.01	0.08 ± 0.01	0.3 ± 0.03	0.13 ± 0.01	0.22 ± 0.08	0.13 ± 0.01	0.13 ± 0.02	nan ± nan	nan ± nan
	uci_kin8nm	0.26 ± 0.01	0.26 ± 0.01	0.28 ± 0.01	0.27 ± 0.01	0.44 ± 0.07	0.26 ± 0.01	0.33 ± 0.01	0.4 ± 0.07	0.29 ± 0.0
	uci_naval	0.09 ± 0.06	0.1 ± 0.03	0.33 ± 0.07	0.72 ± 0.01	0.86 ± 0.09	0.72 ± 0.01	0.2 ± 0.07	nan ± nan	nan ± nan
	uci_protein	0.71 ± 0.01	0.71 ± 0.0	0.75 ± 0.01	0.71 ± 0.01	1.12 ± 0.73	0.69 ± 0.01	0.7 ± 0.01	0.76 ± 0.02	0.73 ± 0.01
	uci_superconduct	0.35 ± 0.01	0.35 ± 0.01	0.4 ± 0.02	0.35 ± 0.01	0.67 ± 0.22	0.32 ± 0.01	0.33 ± 0.01	0.44 ± 0.03	0.41 ± 0.01
	uci_wine_red	0.89 ± 0.01	0.9 ± 0.04	0.77 ± 0.07	1.13 ± 0.15	0.76 ± 0.02	0.84 ± 0.06	0.77 ± 0.05	0.1 ± 0.04	0.01 ± 0.0
	uci_wine_white	0.9 ± 0.03	0.88 ± 0.02	0.82 ± 0.06	0.85 ± 0.05	0.93 ± 0.39	0.77 ± 0.06	0.79 ± 0.05	0.08 ± 0.04	0.01 ± 0.0
	uci_yacht	0.05 ± 0.02	0.04 ± 0.02	0.82 ± 0.12	0.05 ± 0.01	0.09 ± 0.06	0.05 ± 0.01	0.11 ± 0.04	0.09 ± 0.02	0.16 ± 0.03
Shifted	uci_boston	0.52 ± 0.08	0.52 ± 0.08	0.43 ± 0.1	0.48 ± 0.07	0.3 ± 0.04	0.44 ± 0.08	0.41 ± 0.06	0.5 ± 0.07	0.46 ± 0.06
	uci_carbon	0.03 ± 0.0	0.03 ± 0.0	0.03 ± 0.0	0.72 ± 0.05	0.1 ± 0.03	0.72 ± 0.05	0.08 ± 0.0	nan ± nan	nan ± nan
	uci_ccpp	0.26 ± 0.02	0.26 ± 0.03	0.25 ± 0.01	0.25 ± 0.01	0.26 ± 0.01	0.25 ± 0.01	0.25 ± 0.01	0.07 ± 0.04	0.01 ± 0.0
	uci_concrete	0.54 ± 0.07	0.54 ± 0.07	0.44 ± 0.05	0.48 ± 0.05	0.35 ± 0.03	0.43 ± 0.05	0.4 ± 0.04	0.09 ± 0.03	0.0 ± 0.0
	uci_energy	0.26 ± 0.23	0.26 ± 0.22	0.39 ± 0.14	0.28 ± 0.19	0.23 ± 0.05	0.22 ± 0.11	0.22 ± 0.11	nan ± nan	nan ± nan
	uci_kin8nm	0.28 ± 0.01	0.28 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.43 ± 0.02	0.27 ± 0.01	0.36 ± 0.02	0.37 ± 0.08	0.31 ± 0.02
	uci_naval	1.37 ± 0.96	1.35 ± 0.95	1.61 ± 0.71	0.89 ± 0.11	0.9 ± 0.06	0.89 ± 0.1	1.32 ± 0.59	nan ± nan	nan ± nan
	uci_protein	0.83 ± 0.04	0.83 ± 0.05	0.84 ± 0.04	0.84 ± 0.05	1.02 ± 0.22	0.8 ± 0.05	0.8 ± 0.03	0.82 ± 0.03	0.81 ± 0.03
	uci_superconduct	0.59 ± 0.07	0.59 ± 0.06	0.52 ± 0.06	0.6 ± 0.06	0.6 ± 0.08	0.52 ± 0.05	0.51 ± 0.05	0.54 ± 0.06	0.53 ± 0.05
	uci_wine_red	1.13 ± 0.05	1.14 ± 0.05	0.79 ± 0.03	1.62 ± 0.21	0.76 ± 0.01	1.03 ± 0.11	0.84 ± 0.03	0.11 ± 0.05	0.01 ± 0.0
	uci_wine_white	0.96 ± 0.04	0.96 ± 0.04	0.87 ± 0.03	1.03 ± 0.07	0.84 ± 0.05	0.86 ± 0.03	0.86 ± 0.04	0.06 ± 0.02	0.01 ± 0.0
	uci_yacht	0.16 ± 0.08	0.1 ± 0.04	0.74 ± 0.22	0.09 ± 0.03	0.07 ± 0.02	0.07 ± 0.02	0.13 ± 0.04	0.11 ± 0.02	0.18 ± 0.06

Table 14: UCI benchmarks - RMSE $\left[\mathrm{Var}[y|x], (y - \mu(x))^2\right]$

	UCI benchmarks	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
Not shifted	uci_boston	0.25 ± 0.11	0.37 ± 0.26	1.000000e+11 ± 3.162278e+11	0.61 ± 0.14	nan ± nan	0.53 ± 0.06	0.29 ± 0.17	36.0 ± 4.18	0.77 ± 0.31
	uci_carbon	0.05 ± 0.04	0.05 ± 0.04	0.03 ± 0.03	1.68 ± 0.06	nan ± nan	1.6 ± 0.03	0.09 ± 0.01	nan ± nan	nan ± nan
	uci_ccpp	0.15 ± 0.01	0.15 ± 0.01	0.24 ± 0.3	0.5 ± 0.18	nan ± nan	0.47 ± 0.04	0.14 ± 0.04	0.13 ± 0.01	0.0 ± 0.0
	uci_concrete	0.24 ± 0.15	0.22 ± 0.13	1.32 ± 3.53	0.55 ± 0.15	nan ± nan	0.47 ± 0.05	0.18 ± 0.03	0.25 ± 0.06	0.0 ± 0.0
	uci_energy	0.03 ± 0.0	0.02 ± 0.0	0.15 ± 0.04	0.55 ± 0.06	nan ± nan	0.49 ± 0.02	0.08 ± 0.01	nan ± nan	nan ± nan
	uci_kin8nm	0.14 ± 0.02	0.13 ± 0.02	63.3 ± 146.89	0.48 ± 0.09	nan ± nan	0.47 ± 0.04	0.2 ± 0.01	0.38 ± 0.03	0.14 ± 0.01
	uci_naval	0.03 ± 0.0	3.828603e+03 ± 6.631150e+03	6.000000e+11 ± 5.163978e+11	2.0 ± 0.2	nan ± nan	1.93 ± 0.16	0.12 ± 0.05	nan ± nan	nan ± nan
	uci_protein	0.8 ± 0.04	0.8 ± 0.01	3.000002e+11 ± 4.830458e+11	0.88 ± 0.09	nan ± nan	0.77 ± 0.03	0.88 ± 0.03	9.76 ± 3.17	4.71 ± 10.99
	uci_superconduct	0.39 ± 0.04	0.41 ± 0.07	5.371845e+04 ± 8.860245e+04	0.53 ± 0.08	nan ± nan	0.53 ± 0.05	0.33 ± 0.06	0.62 ± 0.08	0.62 ± 0.05
	uci_wine_red	1.38 ± 0.08	1.62 ± 0.38	3.46 ± 4.12	2.69 ± 0.78	nan ± nan	2.97 ± 2.35	1.18 ± 0.21	0.57 ± 0.17	0.0 ± 0.0
	uci_wine_white	1.52 ± 0.01	1.52 ± 0.14	3.795848e+03 ± 9.990566e+03	1.5 ± 0.33	nan ± nan	1.11 ± 0.2	1.24 ± 0.16	0.08 ± 0.01	0.0 ± 0.0
	uci_yacht	0.03 ± 0.0	0.01 ± 0.0	2.272965e+03 ± 4.416696e+03	0.59 ± 0.13	nan ± nan	0.52 ± 0.05	0.09 ± 0.03	0.28 ± 0.05	0.03 ± 0.02
Shifted	uci_boston	0.74 ± 0.37	0.74 ± 0.35	7.692309e+10 ± 2.773501e+11	0.72 ± 0.19	nan ± nan	0.71 ± 0.16	0.46 ± 0.21	37.59 ± 17.11	0.87 ± 0.35
	uci_carbon	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01	1.31 ± 0.51	nan ± nan	1.45 ± 0.22	0.09 ± 0.0	nan ± nan	nan ± nan
	uci_ccpp	0.15 ± 0.03	0.15 ± 0.03	0.13 ± 0.03	0.4 ± 0.06	nan ± nan	0.49 ± 0.02	0.14 ± 0.02	0.12 ± 0.01	0.0 ± 0.0
	uci_concrete	0.61 ± 0.2	0.64 ± 0.21	26.51 ± 73.79	0.57 ± 0.16	nan ± nan	0.54 ± 0.09	0.31 ± 0.06	0.28 ± 0.12	0.0 ± 0.0
	uci_energy	0.28 ± 0.29	0.25 ± 0.26	119.45 ± 271.5	0.55 ± 0.09	nan ± nan	0.59 ± 0.2	0.12 ± 0.08	nan ± nan	nan ± nan
	uci_kin8nm	0.16 ± 0.01	0.15 ± 0.02	33.73 ± 71.3	0.49 ± 0.1	nan ± nan	0.46 ± 0.06	0.23 ± 0.04	0.41 ± 0.07	0.17 ± 0.03
	uci_naval	4.51 ± 4.91	4.17 ± 4.9	4.375000e+11 ± 5.123475e+11	29.35 ± 35.96	nan ± nan	30.74 ± 37.17	2.47 ± 1.51	nan ± nan	nan ± nan
	uci_protein	0.98 ± 0.1	1.03 ± 0.12	7.777778e+11 ± 4.409585e+11	1.02 ± 0.08	nan ± nan	0.88 ± 0.09	1.04 ± 0.07	7.09 ± 2.31	1.58 ± 0.89
	uci_superconduct	0.67 ± 0.15	0.66 ± 0.14	5.061728e+11 ± 5.030769e+11	0.73 ± 0.11	nan ± nan	0.7 ± 0.1	0.53 ± 0.09	0.57 ± 0.1	0.65 ± 0.32
	uci_wine_red	2.34 ± 0.43	2.37 ± 0.45	4.02 ± 5.18	4.88 ± 1.35	nan ± nan	7.66 ± 6.9	1.32 ± 0.12	0.45 ± 0.29	0.0 ± 0.0
	uci_wine_white	1.57 ± 0.15	1.63 ± 0.16	2.788480e+03 ± 8.365328e+03	1.83 ± 0.32	nan ± nan	1.35 ± 0.15	1.4 ± 0.14	0.13 ± 0.09	0.0 ± 0.0
	uci_yacht	0.14 ± 0.09	0.04 ± 0.01	7.139074e+04 ± 1.741110e+05	0.52 ± 0.07	nan ± nan	0.51 ± 0.05	0.09 ± 0.04	0.47 ± 0.5	0.05 ± 0.03

Table 15: UCI benchmarks - RMSE $[y, \tilde{y}]$

	UCI benchmarks	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
Not shifted	uci_boston	0.56 ± 0.08	0.57 ± 0.16	0.58 ± 0.28	nan ± nan	nan ± nan	nan ± nan	nan ± nan	2.23 ± 0.68	0.51 ± 0.06
	uci_carbon	0.11 ± 0.01	0.1 ± 0.0	0.04 ± 0.02	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_ccpp	0.42 ± 0.01	0.4 ± 0.01	0.33 ± 0.01	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.33 ± 0.01	0.01 ± 0.01
	uci_concrete	0.44 ± 0.06	0.43 ± 0.05	0.43 ± 0.05	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.32 ± 0.04	0.01 ± 0.0
	uci_energy	0.21 ± 0.03	0.17 ± 0.01	0.4 ± 0.08	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_kin8nm	0.46 ± 0.01	0.43 ± 0.01	0.39 ± 0.04	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.78 ± 0.06	0.4 ± 0.01
	uci_naval	0.22 ± 0.03	0.29 ± 0.03	1.3 ± 1.0	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_protein	1.05 ± 0.01	1.05 ± 0.02	1.15 ± 0.13	nan ± nan	nan ± nan	nan ± nan	nan ± nan	1.24 ± 0.15	1.17 ± 0.38
	uci_superconduct	0.54 ± 0.0	0.56 ± 0.02	0.59 ± 0.11	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.62 ± 0.04	0.6 ± 0.04
	uci_wine_red	1.15 ± 0.1	1.07 ± 0.06	1.07 ± 0.09	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.4 ± 0.05	0.01 ± 0.0
	uci_wine_white	1.19 ± 0.04	1.17 ± 0.06	1.18 ± 0.12	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.31 ± 0.01	0.01 ± 0.0
	uci_yacht	0.18 ± 0.03	0.1 ± 0.01	0.97 ± 0.22	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.33 ± 0.13	0.23 ± 0.13
Shifted	uci_boston	0.73 ± 0.07	0.67 ± 0.08	0.58 ± 0.2	nan ± nan	nan ± nan	nan ± nan	nan ± nan	3.24 ± 1.19	0.5 ± 0.07
	uci_carbon	0.12 ± 0.01	0.11 ± 0.0	0.04 ± 0.01	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_ccpp	0.44 ± 0.01	0.43 ± 0.02	0.35 ± 0.02	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.33 ± 0.03	0.01 ± 0.0
	uci_concrete	0.73 ± 0.07	0.71 ± 0.04	0.52 ± 0.06	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.37 ± 0.04	0.01 ± 0.0
	uci_energy	0.44 ± 0.16	0.45 ± 0.19	0.69 ± 0.49	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_kin8nm	0.48 ± 0.03	0.45 ± 0.01	0.4 ± 0.02	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.79 ± 0.07	0.41 ± 0.03
	uci_naval	1.73 ± 0.8	1.71 ± 0.82	2.33 ± 1.32	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_protein	1.15 ± 0.03	1.18 ± 0.06	1.25 ± 0.13	nan ± nan	nan ± nan	nan ± nan	nan ± nan	1.2 ± 0.05	1.13 ± 0.04
	uci_superconduct	0.79 ± 0.05	0.8 ± 0.06	0.93 ± 0.42	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.71 ± 0.08	0.72 ± 0.1
	uci_wine_red	1.32 ± 0.05	1.36 ± 0.08	1.03 ± 0.05	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.44 ± 0.06	0.01 ± 0.0
	uci_wine_white	1.24 ± 0.07	1.23 ± 0.03	1.21 ± 0.09	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.31 ± 0.02	0.01 ± 0.0
	uci_yacht	0.22 ± 0.09	0.21 ± 0.02	0.97 ± 0.28	nan ± nan	nan ± nan	nan ± nan	nan ± nan	0.48 ± 0.23	0.22 ± 0.08

Table 16: UCI benchmarks - $\mathbb{E}$[KL]

	UCI benchmarks	d-VV	VV	VV (no prior)	Mean variance network	Skafte et al	Deep ensembles	Monte Carlo dropout	Noise contrastive priors	Bayes by backprop
Not shifted	uci_boston	0.24 ± 0.05	0.66 ± 0.29	109.96 ± 65.06	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_carbon	0.03 ± 0.0	0.2 ± 0.01	3.590175e+03 ± 683.49	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_ccpp	0.02 ± 0.01	0.65 ± 0.15	765.25 ± 1.304725e+03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_concrete	0.06 ± 0.01	0.54 ± 0.09	151.72 ± 197.18	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_energy	0.03 ± 0.01	1.22 ± 0.06	719.58 ± 835.96	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_kin8nm	0.01 ± 0.0	0.21 ± 0.03	25.55 ± 5.99	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_naval	0.57 ± 0.13	1.09 ± 0.37	966.35 ± 1.672143e+03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_protein	0.03 ± 0.0	0.88 ± 0.3	1.489842e+09 ± 9.803508e+08	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_superconduct	0.1 ± 0.02	0.81 ± 0.01	5.670007e+03 ± 5.534678e+03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_wine_red	0.07 ± 0.01	2.22 ± 1.01	3.798346e+03 ± 6.945584e+03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_wine_white	0.07 ± 0.01	1.72 ± 0.2	1.766196e+06 ± 2.118534e+06	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_yacht	0.14 ± 0.04	0.5 ± 0.29	869.08 ± 1.882199e+03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
Shifted	uci_boston	0.13 ± 0.03	0.39 ± 0.29	390.83 ± 741.65	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_carbon	0.03 ± 0.01	0.2 ± 0.02	3.145017e+03 ± 808.29	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_ccpp	0.02 ± 0.0	0.25 ± 0.07	29.45 ± 18.62	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_concrete	0.05 ± 0.01	0.17 ± 0.05	549.89 ± 1.020544e+03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_energy	0.02 ± 0.01	0.67 ± 0.41	2.454811e+03 ± 1.613791e+03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_kin8nm	0.01 ± 0.0	0.19 ± 0.01	43.01 ± 8.46	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_naval	0.13 ± 0.02	0.3 ± 0.11	3.223887e+04 ± 1.257591e+05	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_protein	0.04 ± 0.01	0.72 ± 0.24	1.800074e+08 ± 2.840555e+08	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_superconduct	0.03 ± 0.01	0.53 ± 0.11	2.629295e+03 ± 3.482120e+03	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_wine_red	0.06 ± 0.01	1.04 ± 0.31	210.6 ± 258.73	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_wine_white	0.08 ± 0.01	2.05 ± 0.31	3.506659e+05 ± 6.077032e+05	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan
	uci_yacht	0.33 ± 0.09	0.19 ± 0.04	196.98 ± 125.57	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan	nan ± nan

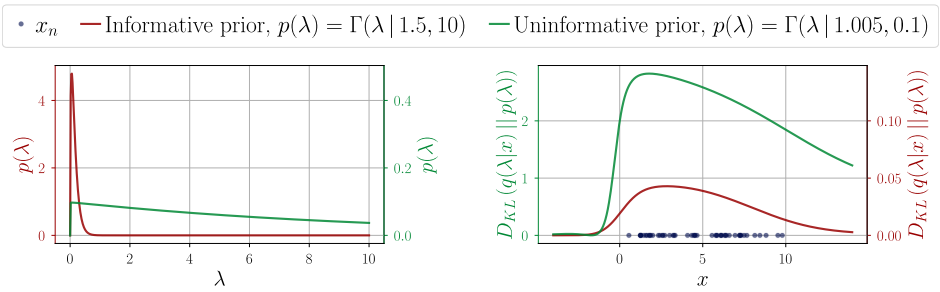

Figure 11: Effect of the informativity of the prior (as displayed on the left) on the KL divergence of the trained posterior on an artificial example (right). The scale of the respective KL divergences reveals that the heavy-tailed prior (green) allows the posterior to be significantly influenced by data, while its counterpart (red) is much more restrictive.

III..8 Prior parameters

For the toy experiments (Fig. 3, 5 and 6), an homoscedastic prior that matches the standard deviation of the targets $\bar{\sigma}$ is chosen. As shown in Fig. 11, for a Gamma prior, the rate β controls its informativity, the closer β is to 0, the more spread out the prior is, and the less penalising it is for the posterior to diverge from it. We thus deliberately choose a prior with low informativity, $\beta =$ 1e-3, and infer the shape as $\alpha = 1 + \beta/\bar{\sigma}$. For the UCI benchmarks, we aimed to adopt a prior that would match the model's empirical variance $(y - \mu(x))^2$. As such, we first ran a training run to determine the model's empirical variance on each dataset, and subsequently adopted $\alpha = 1.5$ and $\beta = (\alpha - 1)\ (y - \mu(x))^2$, with the choice for α being motivated by stability concerns, and obtained from an empirical study. All prior parameters can be found in the configuration files present in the source code.

IV. Generative models experiments

Table 17: Datasets for generative models

Name	Dimensions (N, C, D_x, D_y)	Link
MNIST	(70000, 1, 28, 28)	`http://yann.lecun.com/exdb/mnist/`
FashionMNIST	(70000, 1, 28, 28)	`https://github.com/zalandoresearch/fashion-mnist`
EMNIST	(70000, 1, 28, 28)	`https://www.westernsydney.edu.au/icns/reproducible_research/publication_support_materials/emnist`
KMNIST	(70000, 1, 28, 28)	`https://github.com/rois-codh/kmnist`
SVHN	(600000, 3, 32, 32)	`http://ufldl.stanford.edu/housenumbers/`
CIFAR	(60000, 3, 32, 32)	`https://www.cs.toronto.edu/~kriz/cifar.html`

IV..1 Datasets

Tab. 17 lists all the datasets used in the generative modelling experiments.

IV..2 Dissipative loss for generative models

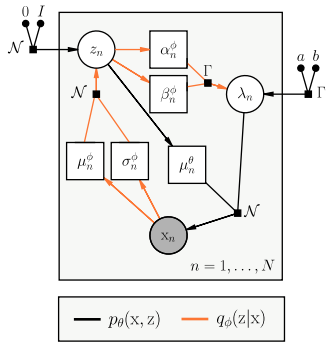

Figure 12: PGM for V3AE

The full expression of the dissipative loss for the V3AE is given as:

$$\begin{aligned}\text{Loss}(q_\phi, \theta; \mathcal{D}_{\text{train}}) = -\Big[\Big(\sum_{x \in \mathcal{D}_{\text{train}}} \mathbb{E}_{q_\phi(z|x)}\left[\mathbb{E}_{q_\phi(\lambda|z)}[\log p_\theta(x|z)] - D_{\text{KL}}\left(q_\phi(\lambda|z) \,||\, p(\lambda)\right)\right] \\ - D_{\text{KL}}(q_\phi(z|x) \,||\, p(z))\Big) + \mathbb{E}_{q_{\text{out}}(z)}\left[D_{\text{KL}}\left(q_\phi(\lambda|z) \,||\, p(\lambda)\right)\right]\Big].\end{aligned} \tag{16}$$

The expected likelihood w.r.t the posterior $q_\phi(z|x)$ is intractable as it requires the integration of the parameter maps $\alpha_\phi(z)$ and $\beta_\phi(z)$ and must be approximated through MC-integration, using multiple sampled latent codes. We observed in practice that a low number of sampled codes, typically, 2 or 3, is sufficient for ensuring convergence.

IV..3 Model architecture

In our VAEs, the encoder and decoder networks' architectures are mirrored, and all parameter maps of each stage share on the same architecture. For the MNIST and FashionMNIST datasets, we relied on fully connected encoder-decoders, with 2 hidden layers with respectively 512 and 256 neurons. Each fully connected layer is followed by *batch normalisation* (Ioffe and Szegedy, 2015). For the SVHN and CIFAR datasets, we relied on a convolutional architecture, with hidden dimensions corresponding to depths of 32, 64, 128, 256, and 512, for kernels of size 3, with a stride of 2 and a padding of 1. Again, batch normalisation is applied after each layer. In both cases, we used *leaky rectified linear units (Leaky ReLU)* for activations and here again, softplus and shifting might be applied on the last layer of the different parameter maps to ensure the proper definition of the quantities they model, and models are optimised with Adam.

IV..4 Pseudo-input generator

Table 18: Parameters for the generative model pseudo-input generator

K	max_iterations	tolerance	δ
N	10	0.007	4e-1

Tab. 18 presents the parameters used by the PIG in a generative modelling setting. We remind that these parameters are parameters of a gradient descent, with learning rate δ. Because it is too computationally expensive to use the complete aggregate posterior as the density estimate we base the PIG on, we iteratively generated pseudo-inputs using the aggregate posterior established on one batch at a time.

IV..5 Prior parameters

As for the regression experiments, an homoscedastic Gamma prior, with the same parameters for all image channels was chosen for model comparisons. The shape and rate parameters were tuned with the same base intuition as for the UCI benchmarks; the prior uncertainty should be fairly close to the empirical mean of the model. An empirical grid search was conducted to determine the best combination of prior parameters wrt to the objective to optimise. In the case of out-of-distribution detection, we adopted an heteroscedastic prior. Such prior adopts similar base parameters as the more standard homoscedastic prior, but its rate parameter, and consequently its associated uncertainty, increases linearly as a function of the distance to the closest of the C pre-determined K-means[12] cluster center, where C is the number of classes in the dataset. Again, the prior parameters used for running the experiments can be found in the configuration files provided with the source code.

IV..6 Out-of-distribution detection

The same experimental setup used to assess the OOD detection capabilities of the d-V3AE is replicated for other OOD datasets, namely *EMNIST* (Fig. 13) and *KMNIST* (Fig. 14). Here again, the benefits of the regularity of the learned decoder variance for OOD detection are clear, with our method clearly overperforming the baseline.

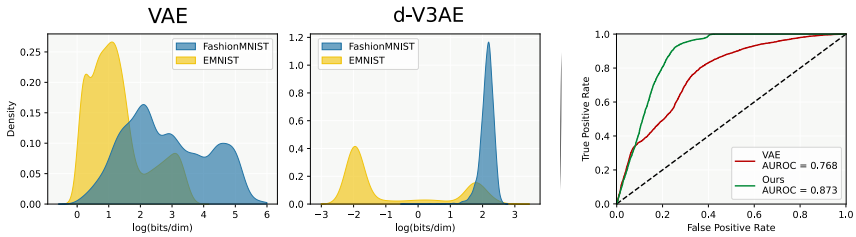

Figure 13: Empirical densities of likelihoods for FashionMNIST (ID) and EMNIST (OOD).

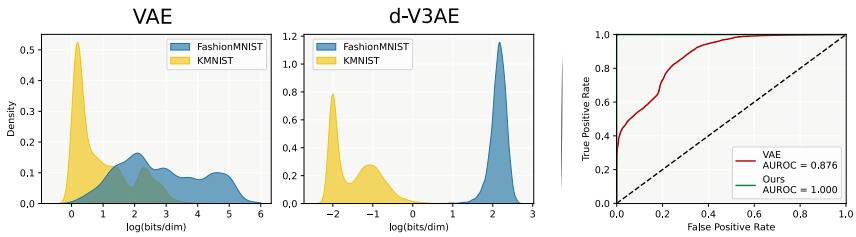

Figure 14: Empirical densities of likelihoods for FashionMNIST (ID) and KMNIST (OOD).

[12] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

IV..7 Pseudo-inputs training for VAE's with Bernoulli likelihood

Motivated by the idea of not necessarily having to adopt a non trivial $\Gamma(\lambda|a,b)$ prior, we explore the use of pseudo-input training in simpler VAE's with Bernoulli likelihood. In this setting, the combined epistemic and aleatoric uncertainty on the reconstructed $\tilde{\mathrm{x}}$ is approximated with a measure of entropy. As uncertainty is high for distributions with high entropy, we reinterpret the decoded Bernoulli distributed reconstruction $\tilde{\mathrm{x}}$ as normalized Categorical distribution and then proceed to maximize its entropy for the pseudo inputs $\hat{z}$. The resulting loss function,

$$\mathrm{Loss}(q_\phi, \theta; \mathcal{D}_{\text{train}}) = -\Big[\sum_{\mathrm{x} \in \mathcal{D}_{\text{train}}} \mathcal{L}(q_\phi, \theta; \mathrm{x}) - \sum_{\hat{z} \in \mathcal{D}_{\text{out}}} \mathrm{H}[\tilde{\mathrm{x}}|\hat{z}] \Big], \tag{17}$$

balances the overall entropy of the reconstruction by promoting entropy increase, $\mathrm{H}[\tilde{\mathrm{x}}|\hat{z}]$. Figure 15 shows the effect of this method for a VAE trained on a subset of MNIST.

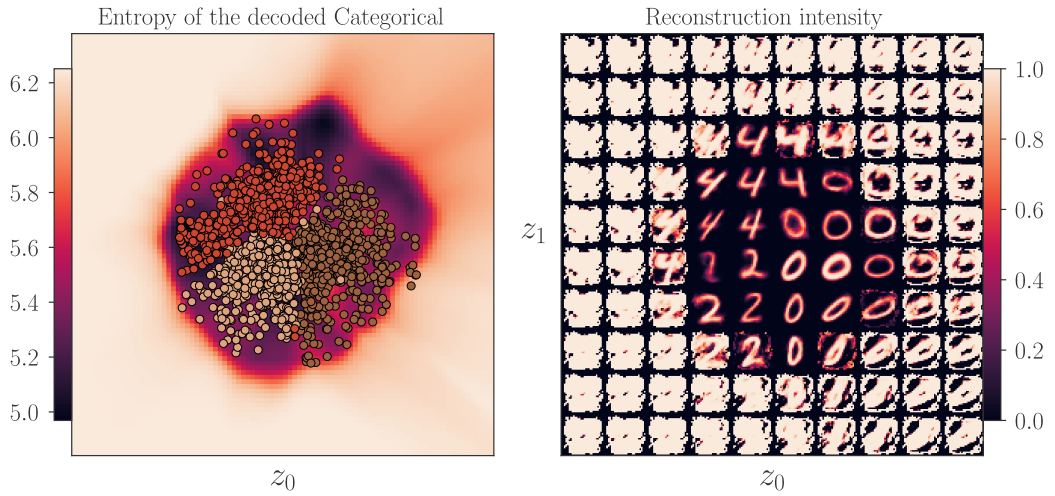

Figure 15: Pseudo inputs for Bernoulli VAE's

V. Implementation details

V..1 Source code

The source code is accessible in the GitHub repository `https://github.com/****`[13]

V..2 Hardware

Experiments were done on a 16-Core AMD Ryzen 5950X machine with a single Nvidia 3080 GPU.

V..3 Running times and carbon emissions

Table 19: Regression models

Dataset	CO_2 (kg)	Time (s)
uci_boston	0.000064	15.969
uci_carbon	0.000695	138.548
uci_ccpp	0.001464	275.824
uci_concrete	0.000102	22.480
uci_energy	0.000095	21.140
uci_kin8nm	0.000869	168.273
uci_naval	0.001427	274.547
uci_protein	0.009920	1762.071
uci_superconduct	0.002010	360.634
uci_wine_red	0.000185	36.186
uci_wine_white	0.000482	89.373
uci_yacht	0.000046	11.404

Table 20: Generative models

Dataset	CO_2 (kg)	Time (s)
fashion_mnist	0.003713	504.190
cifar	0.037554	2573.566
svhn	0.036692	2585.144

Table 21: Generative models w/o pseudo inputs

Dataset	CO_2 (kg)	Time (s)
fashion_mnist	0.002750	407.991
cifar	0.025522	2022.803
svhn	0.025130	2023.293

Tab. 20 and 21 demonstrate that the generation of artificial pseudo-inputs incurs a limited additional computational burden ($\sim +26\%$).

[13] Hidden for the review, please refer instead to the .zip folder attached.

A kernel for continuously relaxed, discrete Bayesian optimization of protein sequences

A kernel for continuously relaxed, discrete Bayesian optimization of protein sequences

Yevgen Zainchkovskyy
DTU Compute
Danish Technical University
2800 Kgs. Lyngby, Denmark
yeza@dtu.dk

Simon Bartels
Datalogisk Institut
University of Copenhagen
2200 Copenhagen, Denmark
bartels@di.ku.dk

Søren Hauberg
DTU Compute
Danish Technical University
2800 Kgs. Lyngby, Denmark
sohau@dtu.dk

Jes Frellsen
DTU Compute
Danish Technical University
2800 Kgs. Lyngby, Denmark
jefr@dtu.dk

Wouter Boomsma
Datalogisk Institut
University of Copenhagen
2200 Copenhagen, Denmark
wb@di.ku.dk

Abstract

Protein sequences follow a discrete alphabet rendering gradient-based techniques a poor choice for optimization-driven protein design. Contemporary approaches instead perform optimization in a continuous latent representation, but unfortunately the representation metric is generally a poor measure similarity between the represented proteins. This make (global) Bayesian optimization over such latent representations inefficient as commonly applied covariance functions are strongly dependent on the representation metric. Here we argue in favor of using the Jensen-Shannon divergence between the represented protein sequences to define a covariance function over the latent representation. Our exploratory experiments indicate that this kernel is worth further investigation.

1 Introduction

Proteins form a basic building block of life, and, yet at the same time, serves as core ingredients in remedies we take for granted, ranging from washing powder, to food conservation and medicine. Unfortunately, discovery of proteins with useful functions is non-trivial. Recall that proteins can be viewed as a sequence of aminoacids chosen from an alphabet of 20 variants (disregarding non-naturally occurring aminoacids). The space of proteins is therefore enormous and most sequences in this space are biologically not sensible. Even for a small protein consting of 100 amino acids, we thus have a search space of 20^{100}, of which only a very small subset are likely to be useful [Tian and Best, 2017].

These observations render both brute-force searches and direct optimization strategies for discovering useful proteins intractable. This has lead to a surge of work that use vast databases on known proteins to learn a low-dimensional continuous representation of proteins and perform the search for new and useful proteins therein. Since the evaluation of the usefulness (fitness) of a proposed protein often amount to a laboratory experiment, we do not have an explicit loss function and must rely on gradient-free search techniques. Here Bayesian optimization (BO) [Močkus, 1975, Shahriari et al., 2016] has been particular fruitful.

Unfortunately, the Euclidean distance measure in latent representations is often a poor proxy for protein similarity [Detlefsen et al., 2020], which indirectly influence Bayesian optimization techniques

Machine Learning for Structural Biology Workshop, NeurIPS 2021.

negatively. BO often rest on an a priori assumption that input points (here latent representations) covary according to a pre-specified covariance function such as the squared-exponential

$$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{x}') = \theta \exp\left(-\lambda \|\boldsymbol{x} - \boldsymbol{x}'\|^2\right). \tag{1}$$

Due to the poor performance of the Euclidean metric this covariance assumption becomes a poor modeling choice, which limit the capabilities of the associated BO. **In this paper** we propose an alternative covariance function that measure similarity between proteins directly in their one-hot representation. We base this on the Jensen-Shannon divergence, which allow us to define a positive definite covariance function. Results from experiments on a synthetic benchmark motivate a more thorough empirical evaluation.

2 Background

2.1 Problem statement

Let $L \in \mathbb{N}$ and let A be a finite set to define $\mathbb{X} := A^L$. For a *costly* black-box function $f : \mathbb{X} \to \mathbb{R}$, we wish to find the minimizer $\boldsymbol{x}_* := \arg\min_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{x})$ using as few function evaluations as possible. We can imagine $\mathbb{X}$ to contain all possible sequences of some length L for all naturally occurring amino acids. For the function f, we may think of the efficiency of a medicine or a washing powder. Hence, evaluating f for a given amino acid sequence is costly in the sense that it requires running a study or a wetlab experiment.

2.2 Bayesian optimization

The typical parameters of Bayesian optimization consist of a surrogate model m for f, and an acquisition function α. Iteratively, the model is updated given the observations of all past experiments and the acquisition function is numerically optimized on the surrogate to select the next configuration for the evaluation of f. Typically, m is a Gaussian process [Rasmussen and Williams, 2006], and popular choices for α are *Expected Improvement* and *Upper Confidence Bound* [Jones et al., 1998].

In our setting, the numerical optimization of α is not possible since the input space is discrete. An approach to this is issue is to fit a latent variable model and to perform the optimization in latent space [Lu et al., 2018]. Problematic with that approach is that euclidean distance in such latent spaces often do not reflect similarity with regard to the objective function [Detlefsen et al., 2020].

2.3 Gaussian process regression

Gaussian processes (GPs) are a typical choice for m due to their expressiveness and closed-form inference. Formally, a GP is a collection of random variables, such that every finite subset of them follows a multivariate normal distribution [Rasmussen and Williams, 2006][p. 13]. A GP is described by a mean function μ (often set to $\mu(\boldsymbol{x}) := 0$), and a positive definite covariance function (kernel) $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$. Assuming that observations of f are distorted by (not necessarily independent) Gaussian noise, the posterior over the function f conditioned on these observations is again a Gaussian process.

3 A kernel for continuously relaxed, discrete Bayesian optimization

3.1 From discrete to continuous

Recall the discrete optimization problem from Section 2.1. A way to turn this discrete optimization problem into a continuous optimization problem is to instead optimize in the space of probability distributions over $\mathbb{X}$ and to minimize the expected function value of f. The space of probability distributions $\mathbb{P}$ over $\mathbb{X}$ consists of $|\mathbb{X}|$-dimensional vectors whose entries are positive and sum to one. Formally $\mathbb{P}$ is the finite-dimensional, convex set: $\mathbb{P} = \{\boldsymbol{p} \in [0,1]^{|\mathbb{X}|} | \boldsymbol{p} \geq 0, \boldsymbol{p}^\intercal \mathbf{1} = 1\}$. Consider the function

$$\bar{f}(\boldsymbol{p}) := \mathbb{E}_{\boldsymbol{p}} f(\boldsymbol{x}) = \sum_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{x}) \boldsymbol{p}_{\boldsymbol{x}}$$

where $\boldsymbol{p}_{\boldsymbol{x}}$ denotes the probability of $\boldsymbol{x}$ under $\boldsymbol{p}$. Observe that the functions have the same optima, that is $\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \min_{\boldsymbol{p}} \bar{f}(\boldsymbol{p})$, where $\arg\min_{\boldsymbol{p}} \bar{f}(\boldsymbol{p})$ is the standard basis vector that is one in the dimension that represents the probability of $\arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$ and 0 everywhere else. Optimizing $\bar{f}$ appears more complicated, but this perspective offers the advantage that $\bar{f}$ is differentiable. In theory, we could compute the minimizer of $\bar{f}$ by solving linear program: $\max_{\boldsymbol{p}} -\boldsymbol{p}^\intercal \boldsymbol{f}(\mathbb{X})$ with the constraints $\mathbf{0} \leq \boldsymbol{p}$ and $1 = \mathbf{1}^\intercal \boldsymbol{p}$.

For a practical approach to the minimization problem we will have to restrict ourselves to a subset of $\mathbb{P}$. In this work, we explore a parameterization through the latent space of a variational autoencoder [Kingma and Welling, 2014, Rezende et al., 2014]. Further, we propose to place a Gaussian process prior over $\bar{f}$ (instead of just f). The reasons is that although a Gaussian process prior over f induces a Gaussian process belief over $\bar{f}$, the computation of the posterior over $\bar{f}$ is intractable even for $\boldsymbol{p}$ which assume independence between sequence sites. Having a prior over $\bar{f}$ avoids this problem and since elements of $\mathbb{X}$ are (as standard basis vectors) also elements of $\mathbb{P}$, the usual inference machinery of Gaussian processes can be applied. We will discuss a possible kernel function in Section 3.2.

The question remains, given a $\boldsymbol{p}$, which $\boldsymbol{x}$ to evaluate. This is an issue when optimizing in latent space have as well. In our experiments, we take the mode of $\boldsymbol{p}$.

3.2 The Jensen-Shannon-divergence kernel

In the previous section, we proposed to place a Gaussian process prior over $\bar{f}$ (as opposed to just f). To do so, we need a kernel function which acts on probability distributions. For computational tractability, such a kernel function should not involve expectations of f. In this work, we investigate the suitability of the Jensen-Shannon-divergence kernel.

Given a discrete probability distribution $\boldsymbol{p}$, its Shannon-entropy [Shannon, 1948] is defined as $H[\boldsymbol{p}] := \sum_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{p}_{\boldsymbol{x}} \log \boldsymbol{p}_{\boldsymbol{x}}$ with the convention $0 \log 0 := 0$. The Jensen-Shannon divergence [Fuglede and Topsøe, 2004] between to discrete probability distributions is defined as

$$JSD[\boldsymbol{p}, \boldsymbol{q}] := H\left[\frac{\boldsymbol{p}+\boldsymbol{q}}{2}\right] - \frac{1}{2}\left(H[\boldsymbol{p}] + H[\boldsymbol{q}]\right).$$

This function is known to be conditionally, negative definite such that

$$k(\boldsymbol{p}, \boldsymbol{q}) := \theta \exp\left(-\lambda JSD[\boldsymbol{p}, \boldsymbol{q}]\right) \tag{2}$$

is a positive definite kernel function for all $\theta, \lambda > 0$ [Feragen et al., 2015].

3.3 Numerical optimization of the acquisition function

In our experiments, we will consider specific variational autoencoders (VAEs) [Kingma and Welling, 2014, Rezende et al., 2014] to parameterize distributions $\boldsymbol{p}$ such that the numerical optimizer for our acquisition function α operates in the latent space $\mathbb{Z}$ of the VAE. Choosing a differentiable decoder, we can optimize our acquisition function $\alpha : \mathbb{P} \to \mathbb{R}$, via $\beta : \mathbb{Z} \to \mathbb{R}, \boldsymbol{z} \mapsto \alpha(\text{decode}(\boldsymbol{z}))$. We will use a VAE which decodes a given latent variable $\boldsymbol{z}$ to a categorical distribution $\boldsymbol{p}$ over $\boldsymbol{x}$, where $\boldsymbol{p}$ factorizes across the L entries of $\boldsymbol{x}$. This independence assumption simplifies the evaluation of the entropy, since entropy is additive for independent variables.

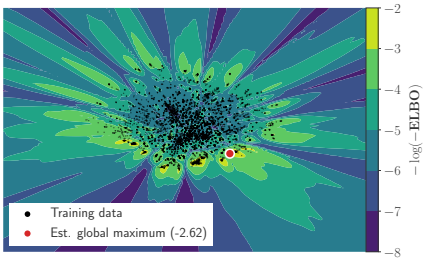

Figure 1: ELBO and the Multiple Sequence Aligned β-lactamase enzymes mapped to a two dimensional latent space.

4 Experiments

Recently, it has been shown that the ELBO of a VAE, associated with a wildtype mutant, can be used as a proxy for protein function [Riesselman et al., 2018]. We therefore use the ELBO as a synthetic,

fast-to-evaluate objective function to test our ideas from Section 3. The VAE in question is trained on a publicly available dataset consisting of 8403 entries of Multiple Sequence Aligned enzymes found in the β-Lactamase family [Riesselman et al., 2018]. For the purpose of visualisation, we limit the latent space to two dimensions and present in Figure 1 the area of the latent space of our interest with both the ELBO heatmap and the mapped training data. Not surprisingly, we observe that the ELBO is non convex and has high value in areas surrounding the data.

Motivated by the desire to uncover novel functioning protein sequences and to study the behaviour of the JSD kernel, we setup two BO experiments. The goal of both is to explore the latent space and find locations corresponding to high ELBO values of the decoded proteins associated with those latent locations. The first experiment, our baseline, is a naive application of the BO procedure with a standard RBF kernel (Eq. (1)) having latent codes as input. In the second experiment, we will use the proposed JSD kernel (Eq. (2)) having mode of categorical $\boldsymbol{p}$ in the original sequence space as its inputs. Apart from the difference in kernels, the experiments are identical in terms of the practical hyperparameters: exploration/exploitation trade-off and number of samples for initialization.

The result of a single run of the experiments is presented in Figure 2. Results of different runs of the same experiment, are included in the Appendix. The JSD kernel, while exploring less of the space and generally suggesting equally good proposals ($\mu_{\text{JSD}} = -2.73$ vs $\mu_{\text{RBF}} = -2.74$) manages to uncover location of the proteins which decode to higher ELBO's: $\max \text{ELBO}_{\text{JSD}} = -2.15$ vs $\max \text{ELBO}_{\text{RBF}} = -2.29$. Additionally, for the JSD kernel, we note a lower variance of the evaluated function values: $\sigma^2_{\text{JSD}} = 0.11$ vs $\sigma^2_{\text{RBF}} = 0.63$. This indicates that the BO routine using JSD kernel proposes less often biologically nonsensical amino acid sequences. Finally, by looking at the spikes followed by a far-away jumps in terms of Hamming distance, we note that the JSD kernel handles the jumps better. We hypothesize that by having a better metric, it must have a fuller notion of the global optimisation landscape.

5 Conclusions

5.1 Summary

This work explored an idea to transform a discrete Bayesian optimization problem into a continuous problem. The optimization is performed in the space of probability distributions over the discrete space, and the objective function is substituted for an expectation, which makes the objective differentiable. Further, we investigated a covariance function based on the Jensen-Shannon divergence that measures similarity between proteins directly in their one-hot representation. Empirically we observed that through this choice of covariance, we can improve both convergence and exploration properties of the Bayesian optimization on a synthetic benchmark.

5.2 Future work

In this initial work, we experimented with the JSD kernel on the atomic distributions, but, given a latent variable model with well-calibrated uncertainties (which VAEs typically do not posses [Arvanitidis et al., 2018]), we could associate observations of $\boldsymbol{x}$ with the original $\boldsymbol{p}$ chosen by the acquisition function. Because JSD kernel based on on $\boldsymbol{p}$ will assign high similarity to observations with high uncertainty, we could uncover a beneficial clustering effect and further improve the search procedure.

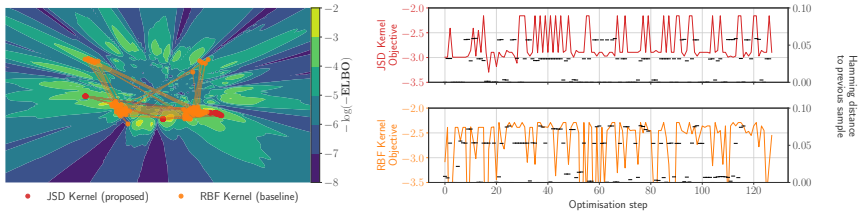

Figure 2: The progress of Bayesian optimisation search for the proposed JSD and baseline RBF kernels. (left) The visualisation of the search procedure in the latent space and (right) ELBO values of the suggested proteins at corresponding steps of optimisation with the normalized Hamming distance to the previously suggested protein on the twin-axis.

Acknowledgments and Disclosure of Funding

References

Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. Latent space oddity: on the curvature of deep generative models. In *International Conference on Learning Representations (ICLR)*, 2018.

Nicki Skafte Detlefsen, Søren Hauberg, and Wouter Boomsma. What is a meaningful representation of protein sequences?, 2020.

Aasa Feragen, Francois Bernard Lauze, and Søren Hauberg. Geodesic exponential kernels: when curvature and linearity conflict. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)*, pages 3032–3042, 2015.

Bent Fuglede and Flemming Topsøe. Jensen-shannon divergence and hilbert space embedding. In *Proceedings of the 2004 IEEE International Symposium on Information Theory, ISIT 2004, Chicago Downtown Marriott, Chicago, Illinois, USA, June 27 - July 2, 2004*, pages 31–37, 2004.

D.R. Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations (ICLR)*, 2014.

Xiaoyu Lu, Javier Gonzalez, Zhenwen Dai, and Neil Lawrence. Structured variationally auto-encoded optimization. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3267–3275, 2018.

Jonas Močkus. On Bayesian methods for seeking the extremum. In Gury I. Marchuk, editor, *Optimization Techniques IFIP Technical Conference*, volume 27 of *Lecture Notes in Computer Science*, pages 400–404, 1975.

C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. MIT, 2006.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR. URL `https://proceedings.mlr.press/v32/rezende14.html`.

Adam J. Riesselman, John B. Ingraham, and Debora S. Marks. Deep generative models of genetic variation capture the effects of mutations. *Nature Methods*, 15(10):816–822, 2018. ISSN 15487105. doi: 10.1038/s41592-018-0138-4. URL `https://doi.org/10.1038/s41592-018-0138-4`.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

C.E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, 1948.

Pengfei Tian and Robert B. Best. How many protein sequences fold to a given structure? a coevolutionary analysis. *Biophysical journal*, 113 8:1719–1730, 2017.

A Appendix

A.1 Additional experiments

We present 3 additional runs of the experiment described in Section 4. The figures show the progress of Bayesian optimisation search for the proposed JSD and baseline RBF kernels. (left) The visualisation of the search procedure in the latent space and (right) ELBO values of the suggested proteins at corresponding steps of optimisation with the normalized Hamming distance to the previously suggested protein on the twin-axis.

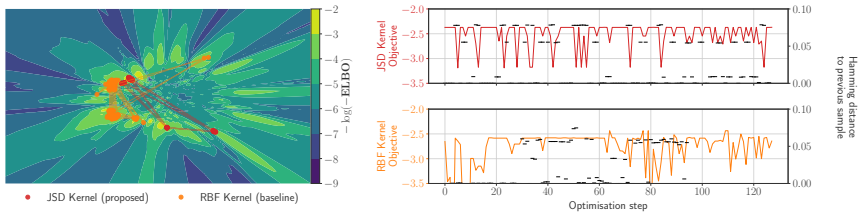

Figure 3: Experiment run 1

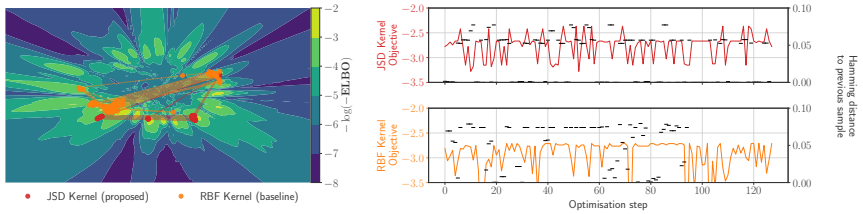

Figure 4: Experiment run 2

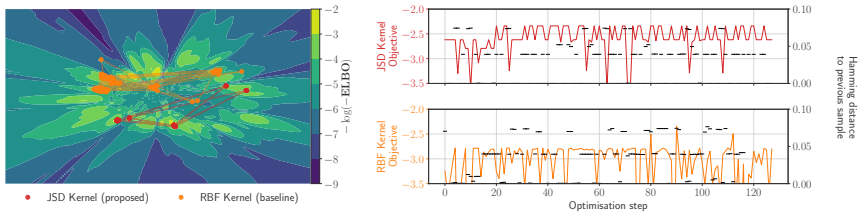

Figure 5: Experiment run 3

We note that the results described in Section 4 were comparable across all the experiments we made.

Probabilistic thermal stability prediction through sparsity promoting transformer representation

Probabilistic thermal stability prediction through sparsity promoting transformer representation

Yevgen Zainchkovskyy*
DTU Compute & Novo Nordisk A/S
yezs@novonordisk.com

Jesper Ferkinghoff-Borg
Novo Nordisk A/S
jfgb@novonordisk.com

Anja Bennett
Novo Nordisk A/S
zabx@novonordisk.com

Thomas Egebjerg
Novo Nordisk A/S
tegb@novonordisk.com

Nikolai Lorenzen
Novo Nordisk A/S
nlz@novonordisk.com

Per Jr. Greisen
Novo Nordisk A/S
pjug@novonordisk.com

Søren Hauberg
DTU Compute
sohau@dtu.dk

Carsten Stahlhut
Novo Nordisk A/S
ctqs@novonordisk.com

Abstract

Pre-trained protein language models have demonstrated significant applicability in different protein engineering task [1, 2]. A general usage of these pre-trained transformer models producing latent representation is to use a mean pool across residue positions to reduce the feature dimensions to further downstream tasks such as predicting bio-physical properties or other functional behaviours. In this paper we provide a two-fold contribution to machine learning (ML) driven drug design. Firstly, we demonstrate the power of sparsity by promoting penalization of pre-trained transformer models to secure more robust and accurate melting temperature (Tm) prediction of single-chain variable fragments with a mean absolute error of 0.23°C. Secondly, we demonstrate the power of framing our prediction problem in a probabilistic framework. Specifically, we advocate for the need of adopting probabilistic frameworks especially in the context of ML driven drug design.

1 Introduction

Peptide and protein engineering is the process of optimizing peptide and proteins towards desired and valuable features for technological or medical applications [3]. In protein engineering, we seek to optimize the function of a protein with respect to e.g. its expression level, solubility, or thermal stability. Their functional behavior is directly determined by their amino acid sequence. Thus, to develop new or optimize desired properties for e.g., biomedical applications require to invert the relationship of the function given the sequence [4], also generally known in statistics and machine learning as the inverse problem. However, existing design methods have serious problems in distinguishing the functional levels of closely related proteins [5, 6]. While both protein engineering and design is a NP-hard problem [7], a direct search in the protein space simply becomes an overwhelming and intractable approach in linear time. Directed evolution has successfully demonstrated its applicability of mapping peptide and protein sequencing to functional behavior. However, it is highly limited by the fact that even high-throughput techniques only can sample a minor fraction of sequences constructed from diversification methods [8]. Among others, Bedbrook and

*Corresponding author

Workshop on Robustness in Sequence Modeling, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

co-workers have demonstrated the direct applicability of utilizing machine learning for optimizing a property that would not have been possible to engineer through directed evolution alone, [6, 9, 10]. On the other hand, machine learning models are heavily dependent on learning from data - a crucial part in designing a machine learning driven drug design pipeline is therefore the accessibility of relevant functional data for the task at hand.

In this contribution, we demonstrate and discuss classical challenges in both designing compounds with dedicated properties from a minimal set of observations and data sets with quite scarce diversity. While we at one hand wish to provide as diverse molecules to maximize the coverage of the chemical search space and on the other hand seek to ensure optimized properties within minimum number of design rounds (experiments) - we are facing a typical active learning problem balancing explore vs exploit steps through the usage of model uncertainty estimates. In section 2, we provide a unified probabilistic framework for integrating compact latent pre-trained transformer features with Gaussian Process (GP) regression models, [11]. Through careful variant design train, development (dev), and test splits we demonstrate the applicability of uncertainty estimates to assess the models own notion of what it does not know. We examine the effect of training data with 1-5 mutations away from a wild-type sequence and the models ability to reason of its own predictive power to generalize to multiple mutations. While we in this paper limit ourselves to the quantification of predictive performance of the models, the GPs can be utilized as the surrogate model in a Bayesian Optimization framework for optimizing and searching the sequence space.

2 Background

Here, we motivate our problem and provide a brief overview of the core architecture of our probabilistic models utilizing a transformer architecture as input to our downstream regression models. Significant improvement and applicability of protein language models have been demonstrated over the last years, where among others the UniRep [12], Evolutionary Scale Modeling (ESM) [2], ProtBert [13] models can be mentioned. In [12], they utilize pretrained language model representation UniRep to generalize the representation to unseen regions of sequence space. Furthermore, Vig & Rao argues for attention in the transformer models corresponds to known biological properties like structure and binding sites that can enable contact prediction [14, 15]. In a drug design setting, we are especially interested in enabling pretrained representations in our protein engineering tasks for designing improved drugs as we are highly limited by the number of experiments we can conduct relative to the enormous sequence space at hand, e.g. 10^{130} for proteins of 100 amino-acids length. Thus, searching the space intelligently is needed even when utilizing high-throughput experimental setups.

While designing or engineering proteins, we are faced with the problem of optimizing towards specific functions of the molecules and effectively only interested in a tiny subspace of sequence space. The main challenge is naturally how can we utilize the general protein representation for a direct fine tuning to the downstream functional optimization task at hand. In this contribution, we seek to build a model for predicting the thermal stability of the antibody format single chain variable fragment (scFv). Due to the small sizes and the stranded nature of scFvs, these are commonly used as building blocks to construct recombinant multi-specific antibody formats, [16]. Unfortunately, the scFvs has been reported to be less termostable than larger antibody formats and thus more likely to lead to undesired aggregation and low Tms when utilized in a multi-specific format, [17]. To improve biophysical behavior of the scFV, our goal is to build a predictive model of the experimental measured Tm values determined by nano differential scanning fluorimetry (nanoDSF) [18]. Having an accurate model for prediction the melting temperature is needed to assess which variants to test experimentally for increased thermal stability. To quantify the accurate of our predictions, we follow a probabilistic approach where we not only obtain our mean predictions but just as importantly can provide uncertainty estimates on the predictions. Uncertainty estimates is critically needed for providing quantitative and directed search strategies balancing both exploitation and exploration.

2.1 Transformer-based models

Transformers are revolutionising NLP, have recently been repurposed to model biological sequences.In the core of a transformer architecture is the attention mechanism allowing to capture long-range dependencies between positions in a sequence. Originating as a solution to classic sequence-to-sequence (seq2seq) models, attention mechanism shows better performance and scaling characteristics

than traditional RNNs or LSTMs. Common to those architectures is a context vector comprising of a hidden state of the network being carried through subsequent propagation, resulting in degraded performance with increased sequence lengths. On the other hand, transformers utilize self-attention, which allows processing of the whole sequence while still focusing on specific parts of it.

In the context of biological sequence modelling, the hidden state of a Transformer model corresponds to individual amino acid residues and represents the given amino acid in its context as a point in a high dimensional space (embedding). Thus, similar sequences are assigned similar representations by the network and are mapped to nearby points in space.

In this contribution, we use the ESM1-b variant of a Transformer protein language model from Facebook AI Research [2] encoding each of our sequences to an embedding $\mathbf{x} \in \mathbb{R}^{250\times 1280}$.

2.2 Gaussian process regression

A Gaussian Processes (GP) is a powerful probabilistic framework enabling nonparametric, nonlinear Bayesian models [11]. A GP defines a prior distribution over the set of function $f(\mathbf{x})$ mapping the relation between our M-dimensional feature representation of our protein sequences to the target property $y = f(\mathbf{x}) + \epsilon$. Here ϵ represents additive observation noise. Using a standard zero-mean GP prior we obtain

$$p(f(\mathbf{X})) = p(\mathbf{f}_X) = \mathcal{N}(0, \mathbf{K}), \tag{1}$$

where $\mathbf{K}$ is the covariance matrix between our training input features $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]$ such that $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ defines the covariance function between input $\mathbf{x}_i$ and $\mathbf{x}_j$. We utilize one of the typically applied kernels for GP regression, Matern $\frac{5}{2}$ covariance function, with shared length-scale parameters for each input dimension σ_l, yielding

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \left(1 + \frac{\sqrt{3}r}{\sigma_l}\right) \exp\left(-\frac{\sqrt{3}r}{\sigma_l}\right) \quad \text{where} \quad r = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j)} \tag{2}$$

Assuming additive independent identically distributed Gaussian noise with variance σ_ϵ^2 our predictive distribution for our new test proteins $\mathbf{Z}$ reads, [11],

$$\begin{aligned} p(f_Z|\mathbf{X}, \mathbf{y}, \mathbf{Z}) &= \mathcal{N}(\mu_z, \mathbf{\Sigma}_z), \quad \text{where} \quad \mu_z = k(\mathbf{Z}, \mathbf{X}) \left(\mathbf{K} + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} \mathbf{y} \\ \mathbf{\Sigma}_z &= k(\mathbf{Z}, \mathbf{Z}) - k(\mathbf{Z}, \mathbf{X}) \left(\mathbf{K} + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} k(\mathbf{X}, \mathbf{Z}). \end{aligned} \tag{3}$$

3 Driving sparsity through learned masks

Having extracted an embedding for each residue in a sequence of length P, a typical approach used to represent a complete protein as a single vector $\hat{\mathbf{x}}$ is by averaging across the transformer's hidden representation $\mathbf{x}$ at each sequence position p (mean pooling):

$$\hat{\mathbf{x}} = \frac{1}{P} \sum\nolimits_{p=1}^{P} \mathbf{x}_p \tag{4}$$

While significantly reducing the overall dimensionality of the embedding and allowing to represent proteins of different lengths, this approach inevitably results in loss of information. Intuitively, averaging assigns equal weight to all residues in the sequence, while in reality, only a handful of positions might influence the target of interest.

In this work, we investigate 3 different positional-weighted approaches to the typical averaging: a positively constrained **Learned mask** $\mathbf{w_l}$, sparsity promoting **Sigmoid-transformed mask** $\mathbf{w_s}$ and a Half-Cauchy **Prior based mask** $\mathbf{w_p}$:

$$\hat{\mathbf{x}}_\mathbf{l} = \frac{\sum_{p=1}^{P} \exp(\mathbf{w_l}_p)\mathbf{x}_p}{\sum \exp(\mathbf{w_l})} \tag{5}$$

$$\hat{\mathbf{x}}_\mathbf{s} = \frac{\sum_{p=1}^{P} \mathrm{S}(\mathbf{w_s}_p)\mathbf{x}_p}{\sum \mathrm{S}(\mathbf{w_s})} \quad \text{where} \quad \mathrm{S}(x) = \frac{1}{1+\exp(-x)} \tag{6}$$

$$\hat{\mathbf{x}}_\mathbf{p} = \frac{\sum_{p=1}^{P} \mathbf{w_p}_p \mathbf{x}_p}{\sum \mathbf{w_p}} \quad \text{where} \quad \mathbf{w_p} \sim \text{Half-Cauchy}(0, \sigma) \tag{7}$$

From a practical perspective, for all three approaches, we learn the masks as a part of the standard gradient based GP Maximum Log-Likelihood maximization procedure.

4 Results

Using Mean Absolute Error (MAE) as our metric, and partitioning the data set, we ran 3 sets of experiments corresponding to 3 different splits. First, we tested on the subset of the training set where single-site mutations were used as the validation (1MUT), next we used a random sample of the training set as validation (Uniform Shuffle) and finally evaluated the proposed methods on the hold-out test set itself. The reason for this partitioning is the spread of the positions of mutations in the wild-type sequence and relative sizes of training and validation set. Naturally, having learned a specific mask on the smaller training set, performance will degrade if mask does not reflect the mutated positions in the bigger test-set. This effect is simulated for the "Uniform Shuffle" split where the size of validation set was 24 samples (vs. 10 samples for the 1-MUT split).

Table 1: Evaluation of the Baseline and the proposed methods. For each split and method we report mean $\pm$ std over 64 trials. Bold values denote statistical significance against Baseline ($p < 0.05$).

	1-MUT Shuffle	Uniform Shuffle	Test Set (Hold out)
Baseline	1.216 ± 0.306	0.819 ± 0.157	0.273 ± 0.006
Learned mask	$\mathbf{1.110 \pm 0.338}$	0.836 ± 0.231	$\mathbf{0.227 \pm 0.010}$
Learned mask (Sigmoid)	$\mathbf{1.150 \pm 0.347}$	0.813 ± 0.161	$\mathbf{0.227 \pm 0.008}$
Learned mask (Prior)	$\mathbf{1.090 \pm 0.314}$	0.841 ± 0.158	$\mathbf{0.226 \pm 0.010}$

Gross metrics are reported in Table 1. Here, we see that the **Learned Mask (prior)** proposed method outperforms the standard averaging approach referenced as *Baseline* in the cases of balanced train/validation splits. Naturally, this method also depends on the actual value of the prior: σ - the second moment of Half-Cauchy distribution. In our case, a prior of $\sigma = 0.15$ was chosen as a result of a parameter sweep on the 1-MUT Shuffle resulting in lowest MAE. That prior is then reused for all **Learned Mask (prior)** runs.

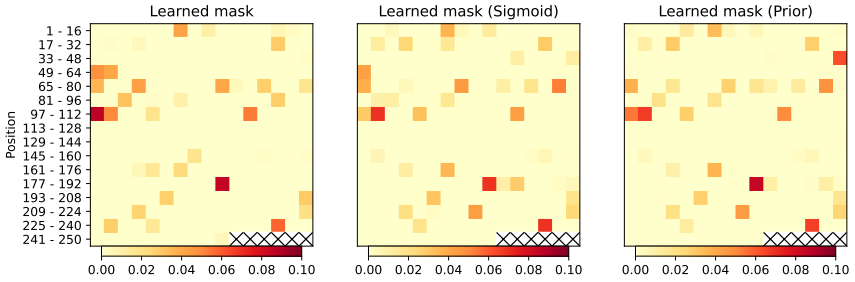

Figure 1: Masks learned on the hold out set. The number of zero entries (values below $1e^{-5}$ threshold) are: **208 for Learned mask**, **203 for Learned mask (Sigmoid)** and **9 for Learned mask (Prior)**. Unused entries are marked with a cross $\times$.

Learned masks are shown in Figure 1. In terms of the importance on melting temperature they emphasize roughly same positions in the sequence. Interestingly, the addition of Half-Cauchy prior, results in a more *dense* mask, as seen on the number of zero-entries. This appear to help generalization.

We summarize the individual test-set predictions and their corresponding uncertainties in Figure 2 (full version in Appendix, Figure 6).

Additionally, we train a model on a subset of the training-data comprising of only 80 samples all being single site mutations. While this restricted model does not perform particularly well in terms of our metric (resulting MAE $= 1.33$), it does great job in terms of uncertainties as shown in Figure 3 (full version in Appendix, Figure 7). Note that this limited model is consistently underestimating the target, as it is unable to capture additive effects of mutations.

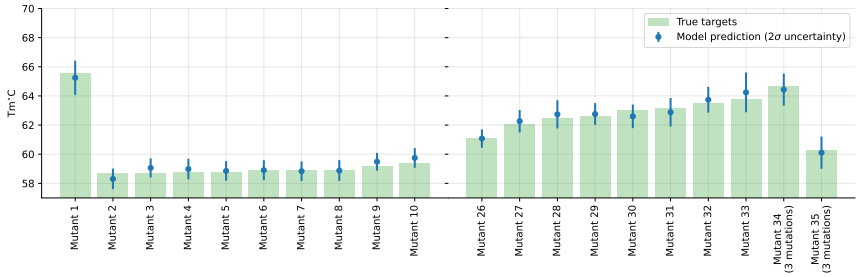

Figure 2: A subset of melting temperature prediction and corresponding uncertainties on the test-set.

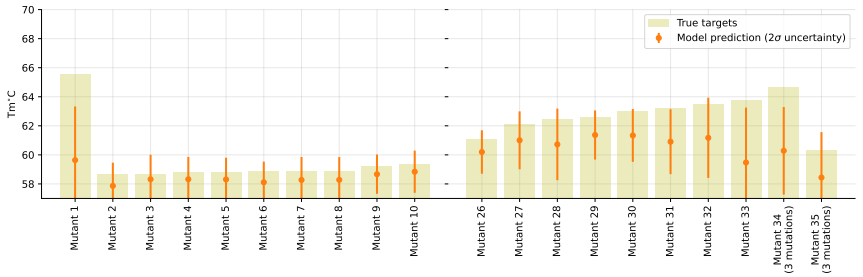

Figure 3: A subset of melting temperature prediction and corresponding uncertainties on the test-set when model is trained only on the sigle-mutations $N\text{train} = 80$

5 Conclusions

Utilization of machine learning driven techniques for navigating the drug design process towards optimized properties requires good compact representation of the molecule space of interest. We have demonstrated that sparsity promoting concentration of the larger pre-trained latent space provided by the protein language model, ESM-1b, leads to more robust estimate of a dedicated thermal stability optimization task for scFvs. We have proposed three variants of sparsity promoting effects through GP regression models integrating learned masks. In general all three models leads to improved predictive performance relative to a standard mean pooled feature representation. Even though our sparsity promoting models outperform our baseline model without mask on our final test data, our validation data indicates that the learned masks are sensitive to too aggressive sparsity when validation data is out-of-distribution of the training data. In fact, this makes sense as the sparse models exactly will seek to favor sparse representations given the training data at hand. Thus, if we seek to utilize the models solely for optimization in regions (residue positions) outside the support of previous seen data, care should be taken in utilizing the masks. From a Bayesian Optimization perspective, this type of evaluation would correspond to the explorative evaluation and thus the mean prediction evaluation is not suitable here. Instead of providing the mean prediction for evaluation we would seek opportunities to enrich the model support towards new regions e.g. through the upper-confidence-bound as acquisition function evaluation. Future work will examine the applicability sparsity promoting models in the context of steering explorative searches.

Acknowledgments and Disclosure of Funding

This project was supported in part by the Innovation Fund Denmark via the Industrial PhD Programme (grant no. 0153-00007B), Novo Nordisk R&D STAR Fellowship Programme and the Novo Nordisk Foundation (grant no. NNF20OC0062606) via the Center for Basic Machine Learning Research in Life Science (MLLS, `https://www.mlls.dk`).

References

[1] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Peter Chen, John Canny, Pieter Abbeel, and Yun Song. Evaluating protein transfer learning with tape. *Advances in neural information processing systems*, 32, 2019.

[2] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, 2019.

[3] Jesse D Bloom, Sy T Labthavikul, Christopher R Otey, and Frances H Arnold. Protein stability promotes evolvability. *Proceedings of the National Academy of Sciences*, 103(15):5869–5874, 2006.

[4] Simona Cocco, Christoph Feinauer, Matteo Figliuzzi, Rémi Monasson, and Martin Weigt. Inverse statistical physics of protein sequences: a key issues review. *Reports on Progress in Physics*, 81(3):032601, 2018.

[5] Jiayi Dou, Lindsey Doyle, Per Jr Greisen, Alberto Schena, Hahnbeom Park, Kai Johnsson, Barry L Stoddard, and David Baker. Sampling and energy evaluation challenges in ligand binding protein design. *Protein Science*, 26(12):2426–2437, 2017.

[6] Kevin K Yang, Zachary Wu, and Frances H Arnold. Machine-learning-guided directed evolution for protein engineering. *Nature methods*, 16(8):687–694, 2019.

[7] Niles A Pierce and Erik Winfree. Protein design is np-hard. *Protein engineering*, 15(10):779–782, 2002.

[8] Nobuhiko Tokuriki and Dan S Tawfik. Stability effects of mutations and protein evolvability. *Current opinion in structural biology*, 19(5):596–604, 2009.

[9] Claire N Bedbrook, Kevin K Yang, J Elliott Robinson, Elisha D Mackey, Viviana Gradinaru, and Frances H Arnold. Machine learning-guided channelrhodopsin engineering enables minimally invasive optogenetics. *Nature methods*, 16(11):1176–1184, 2019.

[10] Florian Richter, Andrew Leaver-Fay, Sagar D. Khare, Sinisa Bjelic, and David Baker. De novo enzyme design using rosetta3. *PLOS ONE*, 6(5):1–12, 05 2011.

[11] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[12] Ethan C Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature methods*, 16(12):1315–1322, 2019.

[13] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, et al. Prottrans: towards cracking the language of lifes code through self-supervised deep learning and high performance computing. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[14] Jesse Vig, Ali Madani, Lav R Varshney, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. Bertology meets biology: interpreting attention in protein language models. *arXiv preprint arXiv:2006.15222*, 2020.

[15] Roshan Rao, Joshua Meier, Tom Sercu, Sergey Ovchinnikov, and Alexander Rives. Transformer protein language models are unsupervised structure learners. *Biorxiv*, 2020.

[16] Robert E Bird, Karl D Hardman, James W Jacobson, Syd Johnson, Bennett M Kaufman, Shwu-Maan Lee, Timothy Lee, Sharon H Pope, Gary S Riordan, and Marc Whitlow. Single-chain antigen-binding proteins. *Science*, 242(4877):423–426, 1988.

[17] Anthony L Fink. Protein aggregation: folding aggregates, inclusion bodies and amyloid. *Folding and design*, 3(1):R9–R23, 1998.

[18] Wyatt Strutz. Exploring protein stability by nanodsf. *Biophysical Journal*, 110(3):393a, 2016.

A Appendix

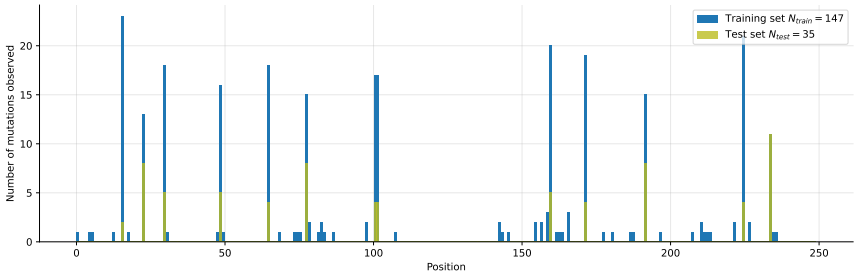

Figure 4: Histogram of the mutations occurring at the respective positions in training and test set.

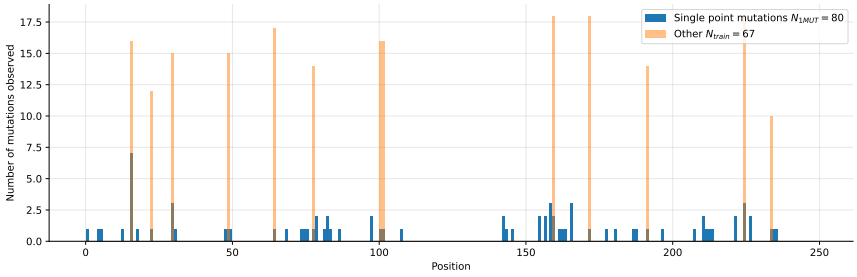

Figure 5: Histogram of the single site mutations of the training set.

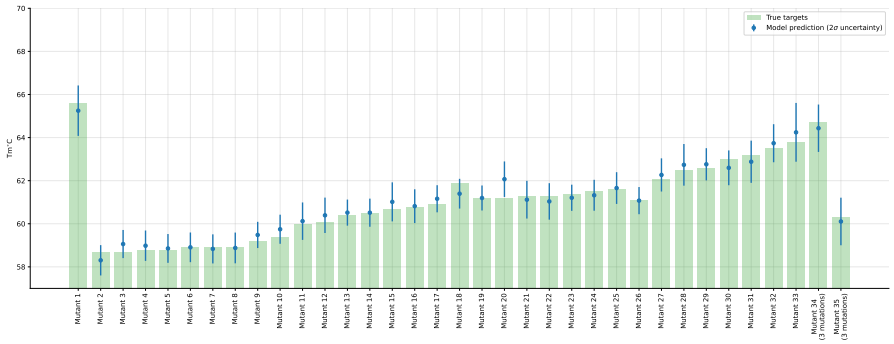

Figure 6: Complete test-set melting temperature prediction and corresponding uncertainties.

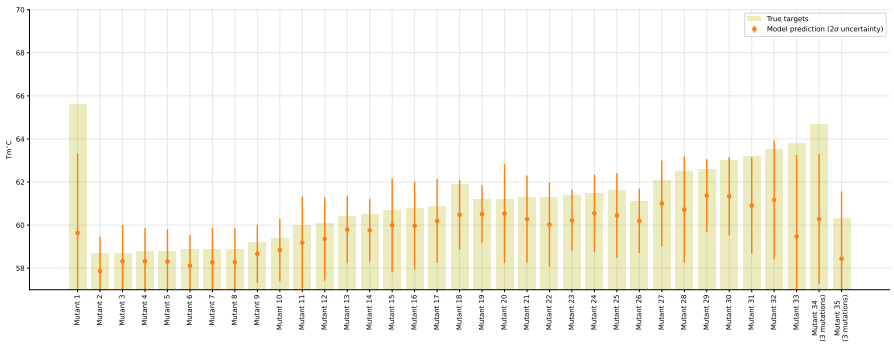

Figure 7: Complete test-set melting temperature prediction and corresponding uncertainties.

All Layers Marginal Likelihood Training with Fully Correlated Linearized Laplace Approximations

All Layers Marginal Likelihood Training with Fully Correlated Linearized Laplace Approximations

Anonymous Author
Anonymous Institution

Abstract

The linearized Laplace approximation is a simple posterior approximation for Bayesian modeling. This relies on access to the loss-Hessian with respect to the model parameters. For large models, it is even intractable to store this matrix, and crude approximations (e.g. the diagonal, KFAC, etc) are called upon. We investigate working with the full Hessian to capture posterior correlations between network parameters, relying on matrix-free methods that avoid the memory costs of storing the Hessian. We develop a scalable yet tight bound to the log-determinant of the full Hessian, which allows us to perform training on all model parameters and hyperparameters according to the marginal likelihood of the model. We further develop a scalable method which leverages the entire correlation structure of the model, while remaining tractable.

1 INTRODUCTION

Bayesian methods excel in consistently solving two *inference* tasks: *i)* for any given model, finding the free parameters that are more *plausible* given the observed data and *ii)* in the light of the data, assigning a preference or uncertainty-based ranking to alternative model hypotheses (MacKay, 2003). Unlike *orthodox* statistical approaches, Bayesian methods naturally embrace Occam's razor principle (Rasmussen and Ghahramani, 2000), which favors the simplest theory that adequately explains the data when addressing both of these tasks.

The intrinsic choice of simpler hypotheses in Bayesian models often leads to better *generalization* abilities (MacKay, 1992; Fong and Holmes, 2020), which is most naturally achieved using the marginal likelihood, also known as *evidence*. While evidence is often ignored for the first task due to its role as a normalization constant, it repeatedly plays a crucial role in the second task by consistently ranking models. This second task involves selecting the appropriate size, capacity, hyperparameters, and other model characteristics.

These ideas shaped the central message of MacKay (1995), where the commitment was that the two *uneasy bed-fellows* neural networks (NN) and Bayesian modeling could work together to suppress spurious elements while learning from the data. Today, it is well-known that Bayesian NNs lead to critical issues (Wenzel et al., 2020; Aitchison, 2021), where the intractability of the marginal likelihood stands out even for medium-sized models. Luckily, this problem was also considered by MacKay (1992), who advocated using Laplace's method (1774) to approximate the evidence.

Recently, a renewed interest in uncertainty quantification for modern NNs has put the spotlight on the linearized Laplace approximation (LLA) (Khan et al., 2019; Immer et al., 2021b; Daxberger et al., 2021). The main drawback is that LLA requires computations with Hessian matrices that are of quadratic size with the number of parameters. Despite the computational cost, the promising results in (Khan et al., 2019; Immer et al., 2021b) have motivated plenty of approximations to the Hessian, usually relying on generalized Gauss-Newton (GGN) or empirical Fisher (EF). Even in such cases, the cost is reduced via diagonal, block-diagonal approximations (Martens and Grosse, 2015) or just considering last-layer LLA (Daxberger et al., 2021; Kristiadi et al., 2020). We provide details on the main properties between LLA models in Table 1.

Reducing correlations between parameters is, however, *undesirable*. Particularly in overparametrized models, such as deep NNs, where we should expect high degrees of correlation between redundant variables.

In this paper, we investigate LLA with *full* correlation between parameters. To achieve this, we introduce an efficient lower bound for the marginal likelihood that

Preliminary work. Under review by AISTATS 2024. Do not distribute.

Table 1: Main properties of linearized Laplace approximation (LLA) methods. Acronyms REG, CLASS and CONV make reference, respectively, to regression, classification and convolutional NN. Also, LL means last-layer and ALL points out to all-layers. The column HESSIAN indicates the choice for the structure of the matrix, e.g., diagonalized or multi-output linear model (MOLM). Further details on the differences between methods are in the Appendix.

LLA METHOD	REG	CLASS	CONV	APPROX	HESSIAN	ONLINE	LML BOUND	LL	ALL
Khan et al. (2019)	✓	✓	✗	GGN	DIAG	✗	✗	NA	NA
Immer et al. (2021a)	✓	✓	✓	GGN/EF	DIAG/KFAC	✓	✗	NA	✓
Immer et al. (2021b)	✓	✓	✓	GGN	DIAG/KFAC	✗	✗	NA	✓
Daxberger et al. (2021)	✓	✓	✓	GGN/EF	DIAG/KFAC	✗	✗	✓	✓
Antorán et al. (2022)	✓	✓	✗	STOCHASTIC	MOLM	✓	✓	✓	✓
This work	✓	✓	✓	GGN	FULL	✓	✓	✗	✓

supports mini-batching. Our primary contribution lies in the elimination of Hessian approximations while preserving correlations between *all* model parameters across *all* layers. Remarkably, the bound is sufficiently tight to be considered as an *almost* exact evaluation of the marginal likelihood, such that it can be optimized via stochastic gradients.

1.1 Background

In this work, we consider N input-output pairs $\{\boldsymbol{x}_n, y_n\}$ of the dataset $\mathcal{D}$ that are modelled by a neural network function $f(\boldsymbol{x}, \boldsymbol{\theta})$, where the NN is governed by the parameters $\boldsymbol{\theta} \in \mathbb{R}^P$ and evaluates for input variables $\boldsymbol{x} \in \mathbb{R}^D$. In particular, we specify a probabilistic model $\mathcal{M}$, where given a likelihood and a prior, we obtain the posterior distribution: $p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}) \propto p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})p(\boldsymbol{\theta}|\mathcal{M})$. The marginal likelihood or *evidence* of this model $\mathcal{M}$ can be thought of as the normalizing constant and is obtained as

$$p(\mathcal{D}|\mathcal{M}) = \int p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})p(\boldsymbol{\theta}|\mathcal{M})\mathrm{d}\boldsymbol{\theta}. \tag{1}$$

Linearized Laplace approximation. Given the model above, Mackay (1992) originally proposed to use Laplace's method to approximate Eq. 1. Today, this is widely known as the *linearized* Laplace approximation (LLA), where the driving idea is that given a *mode* $\boldsymbol{\theta}_*$, one can build a local *quadratic* approximation of $\log p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})$.

To do so, the *linearization* $\ell_{\boldsymbol{\theta}_*}$ of the NN at $\boldsymbol{\theta}_*$ can be first obtained as $\ell_{\boldsymbol{\theta}_*} := f(\boldsymbol{x}, \boldsymbol{\theta}_*) + \mathbf{J}_{\boldsymbol{\theta}_*}(\boldsymbol{x})(\boldsymbol{\theta} - \boldsymbol{\theta}_*)$, where the *Jacobian* matrix is defined as $\mathbf{J}_{\boldsymbol{\theta}_*}(\boldsymbol{x}) := \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_*)|^\top_{\boldsymbol{\theta}=\boldsymbol{\theta}_*}$. Directly, this allows us to re-define the log-posterior according to $\ell_{\boldsymbol{\theta}_*}$ as

$$\begin{aligned}\log p(\boldsymbol{\theta}|\mathcal{D}, \ell_{\boldsymbol{\theta}_*}) &= \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}, \ell_{\boldsymbol{\theta}_*}) + \log p(\boldsymbol{\theta}) \\ &\quad - \log \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}, \ell_{\boldsymbol{\theta}_*})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta} \\ &:= -\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}, \ell_{\boldsymbol{\theta}_*}) - \log \mathcal{Z}(\ell_{\boldsymbol{\theta}_*}),\end{aligned} \tag{2}$$

where is worth noticing the dependencies of the last two terms in Eq. 2 w.r.t. the choice of $\ell_{\boldsymbol{\theta}_*}$ and thus $\boldsymbol{\theta}_*$. Moreover, we omitted the conditioning on the model choice $\mathcal{M}$ to keep the notation uncluttered.

Under the choice of a Gaussian prior distribution $\mathcal{N}(0, \delta^{-1}\mathbf{I})$ with precision parameter $\delta > 0$, then the $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}, \ell_{\boldsymbol{\theta}_*})p(\boldsymbol{\theta}) = \exp\left(-\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y}, \ell_{\boldsymbol{\theta}_*})\right)$, which amounts to an *unnormalized* posterior distribution. Introducing the Laplace approximation (Laplace, 1774; MacKay, 2003), we can use the second-order Taylor expansion around $\boldsymbol{\theta}_*$ to express the *loss* function as

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y}, \ell_{\boldsymbol{\theta}_*}) &= \mathcal{L}(\boldsymbol{\theta}_*|\boldsymbol{x}, \boldsymbol{y}, \ell_{\boldsymbol{\theta}_*}) + \mathbf{G}_{\boldsymbol{\theta}_*}(\boldsymbol{\theta} - \boldsymbol{\theta}_*) \\ &\quad - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_*)^\top \mathbf{H}_{\boldsymbol{\theta}_*}(\boldsymbol{\theta} - \boldsymbol{\theta}_*),\end{aligned} \tag{3}$$

with *gradient* $\mathbf{G}_{\boldsymbol{\theta}}$ and *Hessian* matrix $\mathbf{H}_{\boldsymbol{\theta}}$ given by

$$\mathbf{G}_{\boldsymbol{\theta}} := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}, \ell_{\boldsymbol{\theta}_*}), \qquad \mathbf{H}_{\boldsymbol{\theta}} := -\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}, \ell_{\boldsymbol{\theta}_*}).$$

In practice, exponentiating and normalizing Eq. 3 results in the Gaussian approximation of the posterior $q(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}) := \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{\mu} = \boldsymbol{\theta} - \boldsymbol{\Sigma}\mathbf{G}_{\boldsymbol{\theta}_*}$ and $\boldsymbol{\Sigma} = \mathbf{H}^{-1}_{\boldsymbol{\theta}_*}$. This also results in the approximation of the marginal likelihood given the linearization, which is shown below,

$$\begin{aligned}\log \mathcal{Z}(\ell_{\boldsymbol{\theta}_*}) \approx \mathcal{L}(\boldsymbol{\theta}_*|\mathcal{D}, \ell_{\boldsymbol{\theta}_*}) - \frac{1}{2}\log|\mathbf{H}_{\boldsymbol{\theta}_*}| \\ + \underbrace{\frac{1}{2}\mathbf{G}^\top_{\boldsymbol{\theta}_*}\mathbf{H}^{-1}_{\boldsymbol{\theta}_*}\mathbf{G}_{\boldsymbol{\theta}_*} + \frac{P}{2}\log(2\pi)}_{=:\,\mathcal{C}_{\boldsymbol{\theta}_*}},\end{aligned} \tag{4}$$

where the two last *residual* terms satisfy $\mathcal{C}_{\boldsymbol{\theta}_*} > 0$ whenever $\mathbf{H}_{\boldsymbol{\theta}_*}$ is positive semi-definite. Unfortunately, the computational cost is infeasible in both approximations, mainly due to the size $P \times P$ of the Hessian matrix. This problem is well-known, and usually treated with the Gauss-Newton (GGN) or empirical Fisher (EF), often accompanied by block-diagonal or diagonal approximations (Khan et al., 2018; Ritter et al., 2018; Foresee and Hagan, 1997) when LLA is used for model selection. For further reading in this direction, we recommend Immer et al. (2021a).

1.2 Beyond standard assumptions

Unlike traditional LLA derivations, we have not made any assumption on the local *optimality* of $\boldsymbol{\theta}_*$ in the form of $\mathbf{G}_{\boldsymbol{\theta}_*} = 0$. By definition $\ell_{\boldsymbol{\theta}_*}(\boldsymbol{x}, \boldsymbol{\theta}_*) = f(\boldsymbol{x}, \boldsymbol{\theta}_*)$, which results in

$$\mathcal{L}(\boldsymbol{\theta}_*|\mathcal{D}, \ell_{\boldsymbol{\theta}_*}) = -\log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_*, f) - \log p(\boldsymbol{\theta}_*). \quad (5)$$

Also, the local linearity implies that $\nabla_{\boldsymbol{\theta}} \ell_{\boldsymbol{\theta}_*}(\boldsymbol{x}, \boldsymbol{\theta}_*) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_*)$ and $\nabla^2_{\boldsymbol{\theta}} \ell_{\boldsymbol{\theta}_*}(\boldsymbol{x}, \boldsymbol{\theta}_*) = 0$. Consequently, the Hessian used in Eq. 4 to approximate the marginal likelihood is of the following form

$$\begin{aligned}\mathbf{H}_{\boldsymbol{\theta}_*} &= \sum_{n=1}^{N} \mathbf{J}^\top_{\boldsymbol{\theta}_*}(\boldsymbol{x}_n) \mathbf{H}^{\text{loss}}_{\boldsymbol{\theta}_*} \mathbf{J}_{\boldsymbol{\theta}_*}(\boldsymbol{x}_n) + \boldsymbol{\Lambda} \\ &= \mathbf{H}^{\text{GGN}}_{\boldsymbol{\theta}_*} + \boldsymbol{\Lambda}, \end{aligned} \quad (6)$$

where $\boldsymbol{\Lambda}$ is the prior precision matrix and $\mathbf{H}^{\text{GGN}}_{\boldsymbol{\theta}_*}$ coincides with the GGN approximation of the Hessian (Foresee and Hagan, 1997; Schraudolph, 2002). Additionally, the matrix $\mathbf{H}^{\text{loss}}_{\boldsymbol{\theta}_*} = -\nabla^2_o \log p(y_n|o)$ is the Hessian w.r.t. the outputs $o = f(\boldsymbol{x}_n, \boldsymbol{\theta}_*)$ of the NN.

For most common loss functions, $\mathbf{H}^{\text{loss}}_{\boldsymbol{\theta}_*}$ in Eq. 6 is positive semi-definite and only depends on $\boldsymbol{\theta}_*$ and $\boldsymbol{x}_n$, but not on y_n (Kunstner et al., 2019). As a consequence, it is satisfied that $\mathbf{H}_{\boldsymbol{\theta}_*}$ is also positive definite and $\mathcal{C}_{\boldsymbol{\theta}_*} > 0$.

The main outcome from the previous expressions is that the dependency on the linearization $\ell_{\boldsymbol{\theta}_*}$ disappears. Thus, the marginal likelihood of the linearized model can be expressed as a function of the original (non-linearized) NN, such that

$$\begin{aligned}\log \mathcal{Z}(\boldsymbol{\theta}_*) = &\sum_{n=1}^{N} \log p(y_n|f(\boldsymbol{x}_n, \boldsymbol{\theta}_*)) + \log p(\boldsymbol{\theta}_*) \\ &- \frac{1}{2} \log |\mathbf{H}_{\boldsymbol{\theta}_*}| + \mathcal{C}_{\boldsymbol{\theta}_*}. \end{aligned} \quad (7)$$

Then, the first two terms above are the usual loss, while the *extra* determinant term can be seen as a regularizer that favors points of expansion where the *volume* of the Hessian is as small as possible. The focus of the next section is on the role of $\mathcal{C}_{\boldsymbol{\theta}_*}$.

Lower bound maximization. Current approaches usually drop the $\mathcal{C}_{\boldsymbol{\theta}_*}$ term based on the argument that the gradient $\mathbf{G}_{\boldsymbol{\theta}_*} = 0$ at the point of convergence (Daxberger et al., 2021). However, we highlight that there is no guarantee that the *unnormalised* log-posterior has zero gradient at a local maximum of the marginal likelihood. In this direction, it is worth noticing that disregading $\mathcal{C}_{\boldsymbol{\theta}_*}$ from Eq. 7 results in a lower bound to the evidence (ELBO), akin to variational approximations (Blei et al., 2017).

In this work, we call this bound LLA-ELBO, which is not generally a tight bound. The following proposition gives conditions under which the LLA-ELBO is tight at its optima:

Proposition 1 — *If the gradient of both the unnormalized posterior and the log-determinant term is equal to zero, then the marginal likelihood is locally maximized, such that*

$$\nabla_{\boldsymbol{\theta}_*} \left(\sum_{n=1}^{N} \log p(y_n|f(\boldsymbol{x}_n, \boldsymbol{\theta}_*)) + \log p(\boldsymbol{\theta}_*) \right) = 0$$
$$\nabla_{\boldsymbol{\theta}_*} \log |\mathbf{H}_{\boldsymbol{\theta}_*}| = 0$$

which leads to

$$\nabla_{\boldsymbol{\theta}_*} \mathcal{C}_{\boldsymbol{\theta}_*} = 0 \quad \textit{and} \quad \nabla_{\boldsymbol{\theta}_*} \mathcal{Z}(\boldsymbol{\theta}_*) = 0.$$

Proof: In the supplementary material.

2 ALL LAYERS LOG-DETERMINANT BOUND

We seek to optimize the LLA-ELBO introduced by Eq. 7 whenever $\mathcal{C}_{\boldsymbol{\theta}_*}$ is disregarded. In practice, this amounts to evaluating the usual loss $\mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$ as well as the log-determinant of the Hessian with the latter being the challenging bit.

For large models, one critical issue is that we cannot store $\mathbf{H}_{\boldsymbol{\theta}}$ in memory due to the quadratic size $P \times P$ w.r.t. the number of parameters in the NN. Nonetheless, using automatic differentiation (Baydin et al., 2018) we can efficiently evaluate vector products without instantiating big matrices. Using Eq. 6 as reference, and some vector $\boldsymbol{v} \in \mathbb{R}^P$, we can express Hessian-vector products as

$$\begin{aligned}\mathbf{H}_{\boldsymbol{\theta}_*} \boldsymbol{v} &= \mathbf{H}^{\text{GGN}}_{\boldsymbol{\theta}_*} \boldsymbol{v} + \boldsymbol{\Lambda} \boldsymbol{v} \\ &= \sum_{n=1}^{N} \texttt{vjp}_n \left(\mathbf{H}^{\text{loss}}_{\boldsymbol{\theta}_*}(\boldsymbol{x}_n) \cdot \texttt{jvp}_n(\boldsymbol{v}) \right) + \boldsymbol{\Lambda} \boldsymbol{v}, \end{aligned} \quad (8)$$

where $\texttt{jvp}_n(\boldsymbol{v}) = \mathbf{J}_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\boldsymbol{v}$ and $\texttt{vjp}_n(\boldsymbol{v}) = \mathbf{J}_{\boldsymbol{\theta}}(\boldsymbol{x}_n)^\top \boldsymbol{v}$ denote Jacobian-vector and vector-Jacobian products, respectively. We emphasize that these products can be performed efficiently when f is a deep NN. Two important details to notice are that: *i)* the Hessian $\mathbf{H}^{\text{loss}}_{\boldsymbol{\theta}_*}$ has size $o \times o$ given the number of outputs in the NN and *ii)* we restrict the prior precision matrix $\boldsymbol{\Lambda}$ to be diagonal due to computational efficiency reasons.

Implicit log-determinants. Estimation of log-determinants could potentially be performed via matrix-vector products with the stochastic Lanczos quadrature

(Ubaru et al., 2017). This sort of quadrature combines Lanczos *bi-* or *tridiagonalisation* with stochastic trace estimation (see Sec. 2.2). While the method only requires matrix-vector products (Golub and Van Loan, 2013) for the approximation of log-determinants, efficient differentiation on stochastic Lanczos quadratures is still an open problem, and currently, no implementation efficiently supports such.

2.1 A log-determinant bound

We are particularly interested in practical alternatives to evaluate the log-determinant *iteratively*. Here we replace $\log|\mathbf{H}_{\boldsymbol{\theta}_*}|$ with an upper bound on the log-determinant $\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*}) \geq \log|\mathbf{H}_{\boldsymbol{\theta}_*}|$, such that LLA-ELBO considered w.r.t. Eq. 7 is further loosened.

The $\mathcal{U}(\cdot)$ bound considered is an *old* result by Bai and Golub (1996) that we found to be very accurate with the Hessian matrices in LLA. In the following, we *sketch* its derivation and the main computational components. We also provide further details in the Appendix, as well as we remark that the large body of numerical analysis background is included in the original paper.

First, we note that the log-determinant of $\mathbf{H}_{\boldsymbol{\theta}_*}$ can be re-written as $\log|\mathbf{H}_{\boldsymbol{\theta}_*}| = \operatorname{tr}(\log(\mathbf{H}_{\boldsymbol{\theta}_*})) = \sum_{i=1}^{P}\log(\lambda_i)$, where $\lambda_{1:P}$ are the *eigenvalues* of $\mathbf{H}_{\boldsymbol{\theta}_*}$. This sum can in turn be expressed as a Riemann-Stieltjes (RS) integral w.r.t. some step function γ as the *measure*,

$$\sum_{i=1}^{P}\log(\lambda_i) = \int_{\beta_{\text{L}}}^{\beta_{\text{U}}} \log\lambda \mathrm{d}\gamma(\lambda), \tag{9}$$

$$\text{where}\quad \gamma(\lambda) = \sum_{j=1}^{P}\mathcal{I}(\lambda-\lambda_j),$$

$$\text{and}\quad \mathcal{I}(\Delta) = \begin{cases} 1 & \Delta \geq 0 \\ 0 & \Delta < 0 \end{cases}.$$

Ideally, the integration domain $[\beta_{\text{L}}, \beta_{\text{U}}]$ should contain the *eigenspectrum* of $\mathbf{H}_{\boldsymbol{\theta}_*}$, e.g., $\beta_{\text{L}} \leq \lambda_i \leq \beta_{\text{U}}\ \ \forall i$. The inner formulation of the integral allows us to approximate the log-determinant numerically using Gaussian quadratures (Kovvali, 2011), which implicitly relies on interpolation with *orthogonal polynomials* to calculate the rules. To derive quadrature's rules, Bai and Golub assume some previous knowledge of the sums of the eigenvalues $\lambda_{1:P}$ raised to different powers. These can be calculated according to $\mu_\tau = \sum_{i=1}^{P}\lambda_i^\tau = \operatorname{tr}\left(\mathbf{H}_{\boldsymbol{\theta}_*}^\tau\right)$ for $\tau = 0, 1, 2$.

The main result provided in Bai and Golub (1996) is a convenient numerical approximation to the determinant, where the *sign* of the approximation error is available and allows the result to be interpreted as the bound

$$\log|\mathbf{H}_{\boldsymbol{\theta}_*}| \leq \mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*}) \tag{10}$$

$$\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*}) = \begin{bmatrix} \log\beta_{\text{U}} & \log\bar{t} \end{bmatrix} \begin{bmatrix} \beta_{\text{U}} & \bar{t} \\ \beta_{\text{U}}^2 & \bar{t}^2 \end{bmatrix}^{-1} \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix},$$

where $\bar{t} = (\beta_{\text{U}}\mu_1 - \mu_2)/(\beta_{\text{U}}\mu_0 - \mu_1)$. Additionally, we will also denote the *parametric* bound $\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*})$ as $\mathcal{U}(\mu_0, \mu_1, \mu_2, \beta_{\text{U}})$ with $\mathbf{H}_{\boldsymbol{\theta}_*}$ being omitted for keeping uncluttered notation. In the following, both the parametric version and $\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*})$ can be used indistinguishably upon convenience

Adapting the $\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*})$ bound to LLA. The bound proposed in Eq. 10 by Bai and Golub (1996) computes a quadrature rule to approximate the RS integral of $\log(\lambda_i)$. This approximation is based on the *easier* integral of a second-degree polynomial, which in turn depends on the *smoothness* of the eigenspectrum of the Hessian matrix $\mathbf{H}_{\boldsymbol{\theta}_*}$. At first sight, we know that the LLA is a sum between the GGN matrix and the prior precision (see Eq. 6). Based on the fact that often $\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{GGN}}$ is rank deficient, there is a *jump* in the spectrum, and significantly loosens the bound.

To overcome such a problem, we first consider the simple case of an *isotropic* prior, where $\boldsymbol{\Lambda} = \delta\mathbf{I}$. Thus, letting $\{\lambda_1, \ldots, \lambda_R\}$ denote the *non-zero* eigenvalues of $\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{GGN}}$, $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_R\}$ the corresponding normalised eigenvectors, we can write a new expression based on the eigenvectors. Importantly, the remaining $\{\boldsymbol{v}_{R+1}, \ldots, \boldsymbol{v}_P\}$ are an orthonormal base of $\operatorname{Ker}(\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{GGN}})$ such that $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_P\}$ is an orthonormal base of $\mathbb{R}^P$. Then,

$$\begin{aligned} \mathbf{H}_{\boldsymbol{\theta}_*} &= \mathbf{H}_{\boldsymbol{\theta}_*}^{\text{GGN}} + \delta\mathbf{I} \\ &= \underbrace{\sum_{i=1}^{R}\lambda_i\boldsymbol{v}_i\boldsymbol{v}_i^\top}_{\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{GGN}}} + \underbrace{\sum_{i=1}^{D}\delta\boldsymbol{v}_i\boldsymbol{v}_i^\top}_{\delta\mathbf{I}} \\ &= \sum_{i=1}^{R}(\lambda_i+\delta)\boldsymbol{v}_i\boldsymbol{v}_i^\top + \sum_{i=R+1}^{D}\delta\boldsymbol{v}_i\boldsymbol{v}_i^\top, \end{aligned} \tag{11}$$

where we also define the matrix $\overline{\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{GGN}}}$ for the first term on the r.h.s. as the δ-shifted version of the non-zero eigenvalues. By Spectral theorem we can write $\mathbf{A}^\top\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{DIAG}}\mathbf{A} = \overline{\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{GGN}}}$, with $\mathbf{A}$ being orthogonal and $\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{DIAG}}$ a $R\times R$ diagonal matrix, we can say instead

$$\log|\mathbf{H}_{\boldsymbol{\theta}_*}| = \log\left|\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{DIAG}}\right| + (P-R)\log\delta, \tag{12}$$

where we can apply the bound to $\mathbf{H}_{\boldsymbol{\theta}_*}^{\text{DIAG}}$ since its eigenspectrum is smooth.

Bound parameters and traces. The remaining details to compute the bound $\mathcal{U}(\mu_0, \mu_1, \mu_2, \beta_{\text{U}})$ are the

selection of its parameters and the computation of traces. For μ_0, μ_1, μ_2, we can say that the rank $\mu_0 = R$ and the trace is $\mu_1 = \text{tr}(\mathbf{H}^{\text{DIAG}}_{\boldsymbol{\theta}_*}) = \text{tr}(\mathbf{H}^{\text{GGN}}_{\boldsymbol{\theta}_*}) + R\delta$. For the trace *squared*, we use $\mu_2 = \text{tr}((\mathbf{H}^{\text{DIAG}}_{\boldsymbol{\theta}_*})^2) = \text{tr}((\mathbf{H}^{\text{GGN}}_{\boldsymbol{\theta}_*})^2) + 2\delta \cdot \text{tr}(\mathbf{H}^{\text{GGN}}_{\boldsymbol{\theta}_*}) + R\delta^2$. For β_{U}, we use $\beta_{\text{U}} := \text{tr}(\mathbf{H}_{\boldsymbol{\theta}_*}) + \delta$ which is indeed an upper bound of its eigenvalues, such that $\beta_{\text{U}} \geq \lambda_i + \delta$. These parameters allow us to use the bound in the r.h.s. term of Eq. 12, such that

$$\log \left|\mathbf{H}^{\text{DIAG}}_{\boldsymbol{\theta}_*}\right| \leq \mathcal{U}(\mu_0, \mu_1, \mu_2, \beta_{\text{U}}). \tag{13}$$

Then, putting all together as in Eq. 12, we arrive to the final log-determinant bound

$$\log |\mathbf{H}_{\boldsymbol{\theta}_*}| \geq \mathcal{U}(\mu_0, \mu_1, \mu_2, \beta_{\text{U}}) + (P - R) \log \delta.$$

When the prior precision $\boldsymbol{\Lambda}$ is instead diagonal, we can apply the same procedures by noting that $\log |\mathbf{H}_{\boldsymbol{\theta}_*} + \boldsymbol{\Lambda}| = \log \left|\boldsymbol{\Lambda}^{-1/2} \mathbf{H}_{\boldsymbol{\theta}_*} \boldsymbol{\Lambda}^{-1/2} + \mathbf{I}\right| + \log |\boldsymbol{\Lambda}|$, which is also computationally *cheap* because $\boldsymbol{\Lambda}^{-1/2}$ is easily accessible.

For evaluating the log-determinant bound above, we only require access to the two traces: $\text{tr}(\mathbf{H}_{\boldsymbol{\theta}_*})$ and $\text{tr}((\mathbf{H}_{\boldsymbol{\theta}_*})^2)$. This evaluation can be performed efficiently using Hutchinson (1989) estimator, and specifically letting $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ as in Nowozin (2022). Then,

$$\text{tr}(\mathbf{H}_{\boldsymbol{\theta}_*}) = \mathbb{E}[\epsilon^\top \mathbf{H}_{\boldsymbol{\theta}_*} \epsilon], \quad \text{tr}((\mathbf{H}_{\boldsymbol{\theta}_*})^2) = \frac{1}{2} \text{Var}[\epsilon^\top \mathbf{H}_{\boldsymbol{\theta}_*} \epsilon],$$

which can be combined with Monte-Carlo (MC) and S samples. This would only require a *single* vector-Jacobian product per sample (see Algorithm 1).

Algorithm 1 All-layers LLA training with $\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*})$.

Input: Observed data $\mathcal{D}$
Parameters: Initialize $\boldsymbol{\theta}$, δ
Set parameters $\boldsymbol{\theta}_* = \boldsymbol{\theta}$ for LLA
for e **in** epochs **do**
  for b **in** batches **do**
    Sample $\mathcal{D}_{\text{batch}} \sim \mathcal{D}$
    Sample $\epsilon_1, \cdots, \epsilon_S \sim \mathcal{N}(0, \mathbf{I})$
    Estimate $\text{tr}(\mathbf{H}_{\boldsymbol{\theta}_*})$, $\text{tr}((\mathbf{H}_{\boldsymbol{\theta}_*})^2)$
    Compute $\mu_0, \mu_1, \mu_2, \beta_{\text{U}}$
    Evaluate the bound $\mathcal{U}(\mu_0, \mu_1, \mu_2, \beta_{\text{U}})$
    Compute $-\log \mathcal{Z}(\boldsymbol{\theta}_*) + \mathcal{C}_{\boldsymbol{\theta}_*}$
    do Adam$(\boldsymbol{\theta}, \delta)$ optimizer step
    Set parameters $\boldsymbol{\theta}_*$ for LLA
  end for
end for

Remarks. The derivations above focus on the upper bound $\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*})$ for the log-determinant $\mathbf{H}_{\boldsymbol{\theta}_*}$. Based on Eq. 7, we highlight that an upper bound on the third term (with a negative sign) translates into a lower bound for the log-marginal likelihood (LML). For the empirical results in Sec. 4, we will indistinctly indicate our goal as the LML maximisation, which in our case is equivalent to minimise the bound $\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*})$.

2.2 Empirical tightness

To evaluate the tightness of the proposed bound, we consider a model selection problem with up to 4K parameters. Here we can compare the developed bound with both stochastic Lanczos quadrature (SLQUAD) (Ubaru et al., 2017) and the exact log-determinant. In Fig. 1, we show that the bound is *generally* closer to the true log-determinant than SLQUAD while requiring only approximately half the computational cost. Further, the differentiation through our bound is unproblematic, unlike SLQUAD. It is also worth noting that the bound, at times, underestimates the log-determinant, which is surprising for an upper bound.

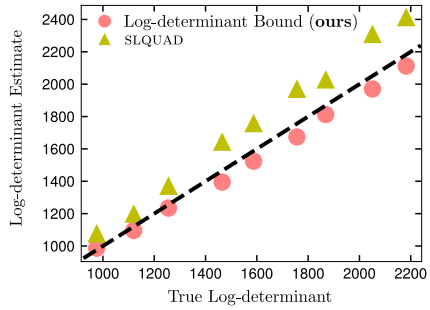

Figure 1: The log-determinant $\log |\mathbf{H}_{\boldsymbol{\theta}_*} + \boldsymbol{\Lambda}|$, estimated with SLQUAD and the $\mathcal{U}(\cdot)$ bound. Models have up to 4K parameters. The dashed line is the ground truth.

3 RELATED WORK

The broad interest in approximate inference methods for NNs has been significantly large since the seminal works on Bayesian NNs (MacKay, 1995; Lampinen and Vehtari, 2001). However, these ones commonly rely on the Laplace's method (MacKay, 1992; Daxberger et al., 2021), variational inference (Graves, 2011; Blundell et al., 2015; Khan et al., 2018), dropout (Gal and Ghahramani, 2016), stochastic weight averaging (Izmailov et al., 2018; Maddox et al., 2019) or MC methods (Neal, 1995).

Our main focus is on the LLA (Khan et al., 2019; Immer et al., 2021b) where the quadratic cost of the Hessian is often avoided with KFAC or EF together with

block-diagonal or diagonal approximations (Martens and Grosse, 2015) or just considering last-layer LLA simplifications (Daxberger et al., 2021; Kristiadi et al., 2020; Sharma et al., 2023).

Additionally, a revitalized interest has grown in using the Laplace approximation in a *post hoc* manner, i.e. performing the Taylor expansion directly around *maximum a posteriori* of parameters (Daxberger et al., 2021). While simple, this heuristic method does not reap the generalization benefits of *online* training according to the marginal likelihood (Immer et al., 2021a).

Alternative bounds in LLA. In another direction, Antorán et al. (2022) develops a scalable stochastic EM algorithm for general linear models (GLM). Like our work, it relies on the implicit representation of the model's covariance and a bound (in the M-step). While their elegant method usually requires significant deviations from standard gradient-based learning, our method can be seen as just including an additional regularization term to the loss. Importantly, their method introduces a lower bound under the log-probability density of the observed data using the Kullback-Leibler (KL) divergence. Instead, we use an upper bound on the log-determinant of the Hessian, also inspired in Bai and Golub (1996). In the end, we also induce a significantly *tight* lower bound on the log-marginal likelihood given the LLA.

4 EXPERIMENTS

In this section, we evaluate the performance of the proposed bound on different image datasets, e.g., MNIST, FMNIST or CIFAR-10, with different NN models. Particularly, we use both MLP and ResNets with parameter sizes up to the order of millions. The driving ideas behind the empirical results are to: *i)* prove the feasibility of the proposed method for optimizing according to the LLA marginal likelihood with *full* correlation structure between parameters and *ii)* show the benefits of log-marginal likelihood training with NN, particularly, in the directions regarding generalization, model selection, and hyperparameter tuning (e.g., prior precision δ). For obtaining all the empirical results, we used JAX (Bradbury et al., 2018). The code for running experiments is provided in the public repository `https://github.com/xxx/XXX/`. Additional details on the specific NN architectures and training details are included in the supplementary material.

4.1 Small data regime

In this experiment, we are interested in testing the proposed method in the *small* data regime. Often, Bayesian methods lose their advantage when the number of data points increases massively. The desired goal is to observe a performance similar to non-Bayesian models, where deterministic losses can perform poorly due to *overfitting*, particularly with low amounts of data. In line with this general principle, we test the effect of training via LML.

In Fig. 2, we show the performance on accuracy and negative log-likelihood for different dataset sizes in both training and test stages. It is particularly interesting to see that in the second and third columns (trained via $\log \mathcal{Z}(\boldsymbol{\theta}_*)$) the performance of the test set significantly increases with the data size while the accuracy of the training curves decays in a smoother way that the first column. Basically, these empirical results show that the log-marginal likelihood is successfully regularising the training process to allow for better generalization.

4.2 Model size selection

For performing model selection on the NN architecture (i.e., number of parameters), we used an MLP model with one hidden layer and different widths. For several values of hidden layers' width, we train the models on standard MAP training until convergence (the same experiments with other trainings are available in the appendix). In Fig. 3, we show the results with image MNIST, FMNIST and CIFAR-10 data. Particularly, we show the values obtained for every model in *train* negative log-marginal likelihood (NLML), accuracy and *test* negative log-likelihood (NLL). Test accuracy and test NLL are here used as ground truth for generalization performance. Train accuracy, paired with other values, serves as a measure of overfitting. Importantly, the NLML is computed on the training set and is not aware of any test point.

The shining part in these empirical results is that the NLML describes a minimum (the lower ↓, the better) for the different datasets, in correspondence of the model size that is considered *optimal* by the marginal likelihood. In this direction, it is important to realize that such a minimum is very close to the optimal value according to the other generalization and overfitting metrics in second and third row plots, which are, differently from the NLML, computed using a test set.

Accuracy and convergence. Additionally, in Fig. 4 we included the curves of training and test accuracy for ResNet18 (11.4M parameters). A similar behavior was also observed for ResNet34 (21.5M) and ResNet50 (23.9M). All curves were obtained via LML training, using the proposed bound $\mathcal{U}(\cdot)$. We can observe that training accuracy reaches ∼99.6% while test accuracy obtains scores around ∼83.4%.

4.3 Hyperparameter tuning.

Additionally to the model selection experiments, we are also interested in hyperparameter tuning. Prior adap-

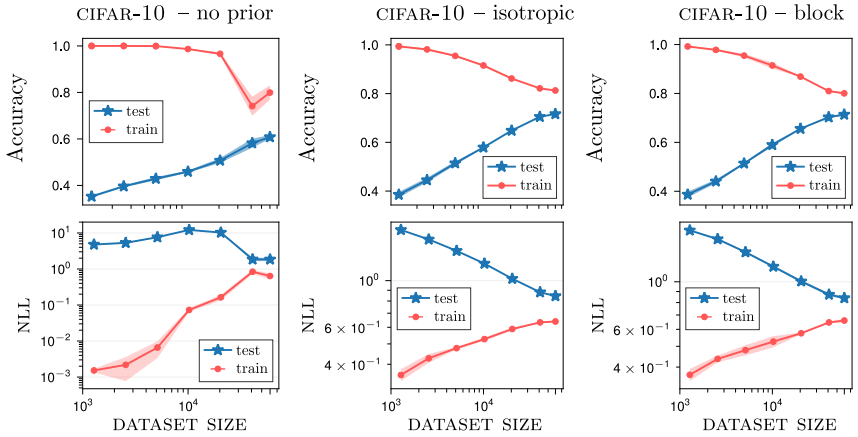

Figure 2: Training of MLP neural networks with different dataset sizes for CIFAR-10. From left to right, *no prior* makes references to maximum likelihood training, *isotropic* uses $\mathbf{\Lambda} = \delta \mathbf{I}$ and *block-prior* uses several $\delta_1, \ldots, \delta_L$ (one per layer) in the diagonal precision matrix $\mathbf{\Lambda}$. For the last two, we train using the proposed bound. **Upper row.** (↑ is better) Accuracy values for training and test data with different numbers of datapoints. **Lower row.** (↑ is better). Negative log-likelihood (NLL)function evaluation for training and test datapoints. The main observation to make is that while training data accuracy decreases as more data is observed, the test data augments. The opposite behavior is shown for NLL. The low test accuracy is because we are using MLP nets with image CIFAR-10 data.

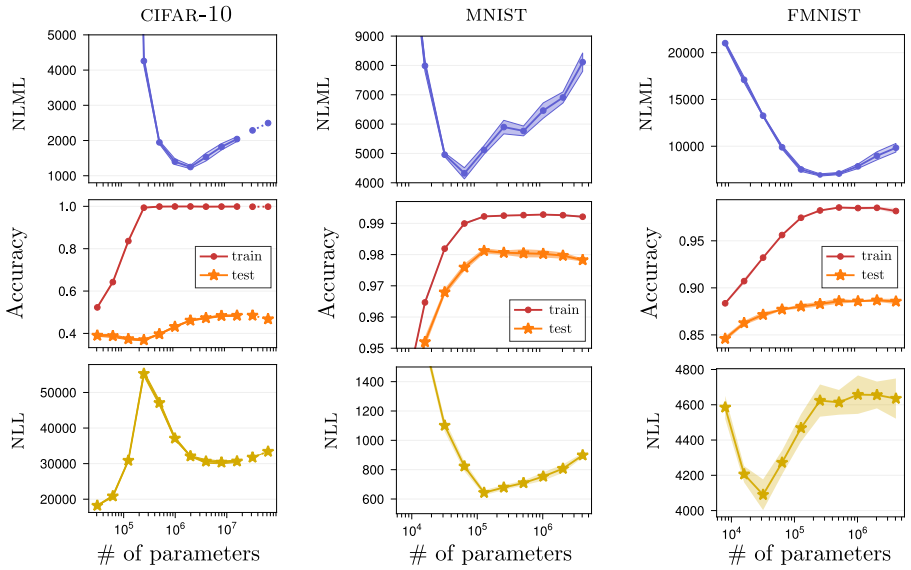

Figure 3: Model size selection results with MLP neural network for three image datasets: MNIST, FMNIST and CIFAR-10. We are interested in finding the size that generalizes better (i.e., higher test accuracy). **Upper row.** (↓ is better) Train negative log-marginal likelihood (NLML) for different model sizes. **Middle row.** (↑ is better) Training and test data accuracy. **Lower row.** (↓ is better) Test negative log-likelihood (NLL) function evaluation on test data. We used MAP training for obtaining $\boldsymbol{\theta}_*$ and $\mathcal{U}(\mathbf{H}_{\boldsymbol{\theta}_*})$ for evaluating NLML. The precision was fixed to $\delta = 1$.

tation has been, so far, a central topic in Bayesian NNs. In our results shown in Fig. 5, we plot the optimization traces of the precision δ fitted with the proposed LML bound in this work. It is remarkable that the convergence of the hyperparameters is *somehow* smooth to a similar order around 2.0. We also highlight the use of $\log \mathcal{Z}(\boldsymbol{\theta}_*)$ without any Hessian reduction and keeping its full correlation even under millions of parameters.

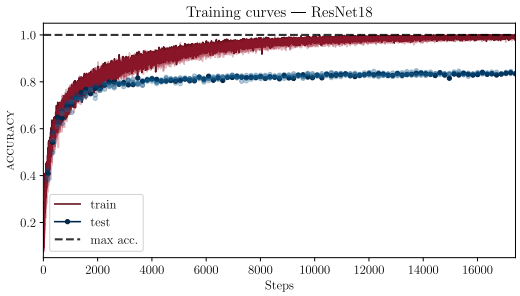

Figure 4: Training and test accuracy for ResNet18 (11.4M parameters) for CIFAR-10 data. Five seeds are included. All curves used the **full correlation** log-marginal likelihood $\log \mathcal{Z}(\boldsymbol{\theta}_*)$ as training objective.

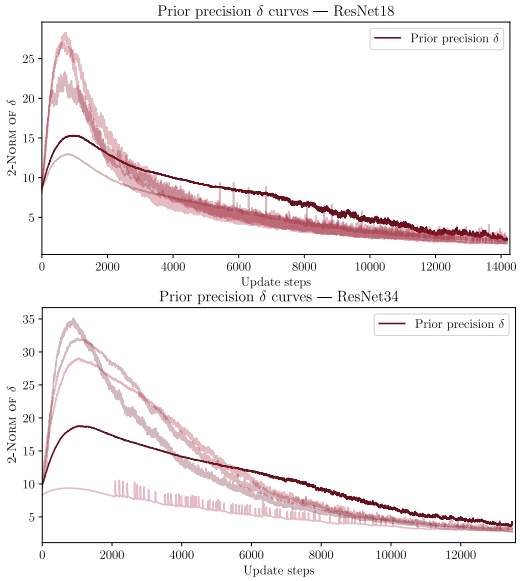

Figure 5: Optimization traces for the prior precision δ hyperparameter for 5 different seeds using $\log \mathcal{Z}(\boldsymbol{\theta}_*)$ as training objective. We show the results for two ResNet models with 11.4M and 21.5M parameters. The y-axis indicates the $||\cdot||_2$ as δ is a vector of size 62 and 110 in the two cases. We set a block-δ prior, that is, one δ_b hyperparameter per block in the ResNet.

Batch scaling. Adapting the proposed bound and the calculus of $\log \mathcal{Z}(\boldsymbol{\theta}_*)$ to batch sampling (of data) is not a straightforward task. We implemented two different routines for that: one leads to tighter bound and better results but involves a matrix with potentially problematic conditioning number, and the other one involves a looser bound (based on the results from (Immer et al., 2023)) and is more stable. Details for batch scaling are provided in the Appendix.

5 CONCLUSION

We have shown how to train a LLA according to the marginal likelihood of the fully correlated model. This contrasts existing approaches that often make significant reductions in the correlation structure in the name of feasibility. The starting point of our work is a new derivation of the LLA, which shows that previously believed expressions for the marginal likelihood are actually lower bounds thereof. Later on, we propose to further loosen this lower bound, using a bound on the log-determinant of the Hessian matrix. Empirically, we have shown that this bound is remarkably tight, such that it can practically be assumed to be exact. The benefit of using a bound instead of a direct evaluation of the log-determinant is basically the speed of evaluation, but more importantly that it is fast to differentiate through the bound, which makes training practical.

Alternative perspective. Readers *not intrigued* by the Bayesian promise can take a more pragmatic view of our work: we propose a new parameter-free regularization term, that allows for training both data likelihood and a diagonal L_2 regularization term. This avoids the *tedious* and expensive grid search for regularization parameters and allows for regularization at the individual parameter level. Irrespective of the perspective, we have empirically shown that the proposed optimization of the marginal likelihood improves generalization performance, especially in the low-data regime. We have also seen that the marginal likelihood is useful for model selection. Parameters of the Gaussian prior can easily be optimized alongside the likelihood, and the marginal likelihood can be useful when choosing neural architectures. The proposed method scales to both large models and large data sets, since the log-determinant bound scales linearly and can easily be mini-batched.

Limitations. We have further noticed that for large models the cost of memory storage increases according to the required samples for trace estimation. Also, we identify a *trade-off* between the batch size for stochastic optimization and the required number of samples. However, this does not limit the scalability of our method.

References

L. Aitchison. A statistical theory of cold posteriors in deep neural networks. *International Conference on Learning Representations (ICLR)*, 2021.

J. Antorán, S. Padhy, R. Barbano, E. Nalisnick, D. Janz, and J. M. Hernández-Lobato. Sampling-based inference for large linear models, with application to linearised Laplace. *arXiv preprint arXiv:2210.04994*, 2022.

Z. Bai and G. H. Golub. Bounds for the trace of the inverse and the determinant of symmetric positive definite matrices. *Annals of Numerical Mathematics*, 4:29–38, 1996.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622, 2015.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:20089–20103, 2021.

E. Fong and C. C. Holmes. On the marginal likelihood and cross-validation. *Biometrika*, 107(2):489–496, 2020.

F. D. Foresee and M. T. Hagan. Gauss-Newton approximation to Bayesian learning. In *Proceedings of International Conference on Neural Networks (ICNN'97)*, volume 3, pages 1930–1935. IEEE, 1997.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, pages 1050–1059, 2016.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 2013.

A. Graves. Practical variational inference for neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 24, 2011.

M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

A. Immer, M. Bauer, V. Fortuin, G. Rätsch, and K. M. Emtiyaz. Scalable marginal likelihood estimation for model selection in deep learning. In *International Conference on Machine Learning (ICML)*, pages 4563–4573, 2021a.

A. Immer, M. Korzepa, and M. Bauer. Improving predictions of Bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 703–711, 2021b.

A. Immer, T. F. A. Van Der Ouderaa, M. Van Der Wilk, G. Ratsch, and B. Schölkopf. Stochastic marginal likelihood gradients using neural tangent kernels. In *International Conference on Machine Learning (ICML)*, pages 14333–14352, 2023.

P. Izmailov, D. Podoprikhin, T. Garipov, D. P. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. In A. Globerson and R. Silva, editors, *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI*, pages 876–885, 2018.

M. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in ADAM. In *International Conference on Machine Learning (ICML)*, pages 2611–2620, 2018.

M. E. E. Khan, A. Immer, E. Abedi, and M. Korzepa. Approximate inference turns deep networks into Gaussian processes. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

N. Kovvali. Theory and applications of gaussian quadrature methods. *Synthesis lectures on algorithms and software in engineering*, 3(2):1–65, 2011.

A. Kristiadi, M. Hein, and P. Hennig. Being Bayesian, even just a bit, fixes overconfidence in relu networks. In *International Conference on Machine Learning (ICML)*, pages 5436–5446. PMLR, 2020.

F. Kunstner, P. Hennig, and L. Balles. Limitations of the empirical fisher approximation for natural gradient descent. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

J. Lampinen and A. Vehtari. Bayesian approach for neural networks: Review and case studies. *Neural Networks*, 14(3):257–274, 2001.

P. S. Laplace. Mémoire sur la probabilité des causes par les événements. *Mémoire de l'Académie Royale des Sciences*, 1774.

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

D. J. MacKay. Probable networks and plausible predictions – A review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469, 1995.

D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

D. J. C. Mackay. *Bayesian methods for adaptive models*. California Institute of Technology, 1992.

D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4 (3):448–472, 1992.

W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. A simple baseline for Bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, pages 2408–2417, 2015.

R. M. Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

S. Nowozin. Thoughts on trace estimation in deep learning. *Blog Post – 09 August*, 2022.

C. Rasmussen and Z. Ghahramani. Occam's razor. *Advances in Neural Information Processing Systems (NIPS)*, 13, 2000.

H. Ritter, A. Botev, and D. Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

M. Sharma, S. Farquhar, E. Nalisnick, and T. Rainforth. Do bayesian neural networks need to be fully stochastic? In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 7694–7722, 2023.

S. Ubaru, J. Chen, and Y. Saad. Fast estimation of tr(f(a)) via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4): 1075–1099, 2017.

F. Wenzel, K. Roth, B. Veeling, J. Swiatkowski, L. Tran, S. Mandt, J. Snoek, T. Salimans, R. Jenatton, and S. Nowozin. How good is the Bayes posterior in deep neural networks really? In *International Conference on Machine Learning (ICML)*, pages 10248–10259. PMLR, 2020.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]