# Random features for dot product kernels and beyond

Jonas Wacker

# Random Features for Dot Product Kernels and Beyond

Dissertation

*submitted to the*

Doctoral School 130: Computer Science, Telecommunications and Electronics of Paris of the Sorbonne Université

*in partial fulfillment of the requirements for the degree of Doctor of Philosophy*

*Author:*

**Jonas WACKER**

*Presented on the 12th July 2022 to the committee in charge:*

| | | |
|---|---|---|
| **Reviewer** | **Edwin V. BONILLA** | *CSIRO's Data61 and Australian National University, Australia* |
| **Reviewer/ Examiner** | **Zoltán SZABÓ** | *London School of Economics and Political Science, United Kingdom* |
| **Examiner** | **Alessandro RUDI** | *INRIA and École Normale Supérieure, Paris, France* |
| **Examiner** | **Maria A. ZULUAGA** | *EURECOM, France* |
| **Advisor** | **Maurizio FILIPPONE** | *EURECOM, France* |
| **Invited** | **Damien GARREAU** | *Université Côte d'Azur and INRIA Sophia Antipolis, France* |



# Caractéristiques aléatoires pour les noyaux de produit scalaire et les noyaux similaires

Thèse

*soumise à*

l'École Doctorale 130: Informatique, Télécommunications & Électronique (EDITE) de Paris de la Sorbonne Université

*pour l'obtention du grade de docteur*

*présentée par:*

**Jonas WACKER**

*et soutenue publiquement le 12 juillet devant un jury composé de:*

| | | |
|---|---|---|
| **Rapporteur** | **Edwin V. BONILLA** | *CSIRO's Data61 and Australian National University, Australia* |
| **Rapporteur/ Examinateur** | **Zoltán SZABÓ** | *London School of Economics and Political Science, United Kingdom* |
| **Examinateur** | **Alessandro RUDI** | *INRIA and École Normale Supérieure, Paris, France* |
| **Examinateur** | **Maria A. ZULUAGA** | *EURECOM, France* |
| **Directeur de Thèse** | **Maurizio FILIPPONE** | *EURECOM, France* |
| **Invité** | **Damien GARREAU** | *Université Côte d'Azur and INRIA Sophia Antipolis, France* |



## Abstract

Dot product kernels, such as polynomial and exponential (softmax) kernels, are among the most widely used kernels in machine learning, as they enable modeling the interactions between input features, which is crucial in applications like computer vision, natural language processing, and recommender systems. However, a fundamental drawback of kernel-based statistical models is their limited scalability to a large number of inputs, which requires resorting to approximations.

In this thesis, we study techniques to linearize kernel-based methods by means of random feature approximations and we focus on the approximation of polynomial kernels and more general dot product kernels to make these kernels more useful in large scale learning.

We begin by analyzing a class of random feature approximations for polynomial kernels from the literature that we refer to as *Product-Sketches*. In particular, we focus on a variance analysis as a main tool to study and improve the statistical efficiency of such sketches. We thus develop structured and complex generalizations of Product-Sketches, and elucidate conditions under which they can yield significant variance improvements. These generalizations allow us to construct a structured *Complex-to-Real (CtR) TensorSRHT* sketch that achieves state-of-the-art kernel approximation errors as well as regression and classification performance, while being faster than comparable sketches in the literature.

Moreover, we combine the Product-Sketches discussed in our thesis to approximate more general dot product kernels and Gaussian kernels. The aforementioned variance analysis hereby becomes the foundation to develop a data-driven optimization approach to allocate the number of random features between Product-Sketches of different degrees. We describe the improvements brought by these contributions with extensive experiments on a variety of tasks and data sets.

Lastly, we show that an optical hardware (the Optical Processing Unit - OPU), computes random feature approximations that are closely related to the ones studied in this thesis. We derive the associated limiting kernel, and we demonstrate its usefulness for a set of computer vision tasks. We show that the OPU can potentially be faster than a recent GPU while using less energy, making such devices interesting from a sustainability viewpoint.

# Résumé

Les noyaux de produit scalaire, tels que les noyaux polynomiaux et exponentiels (softmax), sont parmi les noyaux les plus utilisés en apprentissage automatique, car ils permettent de modéliser les interactions entre les composantes des vecteurs d'entrée, ce qui est crucial dans des applications telles que la vision par ordinateur, le traitement du langage naturel et les systèmes de recommandation. Cependant, un inconvénient fondamental des modèles statistiques basés sur les noyaux est leur évolutivité limitée à un grand nombre de données d'entrée, ce qui nécessite de recourir à des approximations.

Dans cette thèse, nous étudions des techniques pour linéariser les méthodes à base de noyaux au moyen d'approximations de *caractéristiques aléatoires* et nous nous concentrons sur l'approximation de noyaux polynomiaux et de noyaux plus généraux de produit scalaire pour rendre ces noyaux plus utiles dans l'apprentissage à grande échelle.

Nous commençons par analyser une classe d'approximations de caractéristiques aléatoires pour les noyaux polynomiaux de la littérature que nous appelons *croquis de produit (Product-Sketches)*. En particulier, nous nous concentrons sur une analyse de variance comme outil principal pour étudier et améliorer l'efficacité statistique de ces croquis (sketches). Nous développons donc des généralisations structurées et complexes de croquis de produit, et nous élucidons les conditions dans lesquelles elles peuvent produire des améliorations significatives de la variance. Ces généralisations nous permettent de construire un croquis de produit structuré, nommé *Complex-to-Real (CtR) TensorSRHT*, qui atteint des erreurs d'approximation de noyau faibles ainsi que des performances de régression et de classification de pointe, tout en étant plus rapide que les caractéristiques aléatoires comparables dans la littérature.

De plus, nous combinons les croquis de produit discutés dans notre thèse pour approximer des noyaux de produit scalaire plus généraux et des noyaux gaussiens. L'analyse de variance susmentionnée devient ainsi la base pour développer une approche d'optimisation basée sur les données pour allouer le nombre de caractéristiques aléatoires entre les croquis de produit de différents degrés. Nous décrivons les améliorations apportées par ces contributions à l'aide d'expériences approfondies sur une variété de tâches et d'ensembles de données.

Enfin, nous montrons qu'un matériel optique (nommé *Optical Processing Unit - OPU*), calcule des approximations de caractéristiques aléatoires qui sont étroitement liées à celles étudiées dans cette thèse. Nous dérivons le noyau limite associé, et nous démontrons son utilité pour un ensemble de tâches de vision par ordinateur. Nous montrons que l'OPU peut potentiellement être plus rapide qu'un GPU récent tout en utilisant moins d'énergie, ce qui rend ces dispositifs intéressants du point de vue de la durabilité.

# Acknowledgements

During my PhD, I have been granted a great amount of freedom to explore a scientific topic of my choice during more than three years. This experience has taught me how to become an independent researcher and to immerse myself in a less well-known, but fundamentally important research topic.

First and foremost, I would like to thank my supervisor Maurizio Filippone who provided me with a safe research environment in which I could freely develop the contents of this thesis. I am especially grateful for his trust and his optimism that I believe to be a rare gift given to a PhD student.

Exploring fundamental research ideas may not yield immediate rewards to financial stakeholders, which is why I would like to thank the French *Agence Nationale de Recherche* and thereby the French government for their trust and financial support for my PhD. Being given such a support without being bound to commercial use cases allowed me to work on a topic that may shape research in the long term.

Next, I would like to thank Motonobu Kanagawa for his incredible endurance and support in reviewing the contents of this thesis. Thanks to him, I could sharpen the main motivation and contributions of this work, and present results with the scientific rigor that is expected in this field.

I would also like to thank the academic jury that voluntarily accepted to review my thesis and attend my PhD defense. I have the privilege to count true experts in the field among the jury members and their feedback is of incredible value to this work.

Last but not least, I would like to thank my parents and all my friends who accompanied me during my PhD experience at the Côte d'Azur. It is thanks to them that this time was also an incredibly enriching social and cross-cultural experience from which I probably drew as much personal development as from the PhD itself. A special thanks hereby goes to Sagar, Marine and Julien, with whom I shared my flat, and who supported me during the good and bad times with helpful conversations and first-class culinary experiences. Another special thanks goes to my friends from the FROST group with whom I have shared some of the most joyful moments of my life.

Jonas Wacker

# Contents

# Preface

This thesis is divided into 7 main chapters. Chapter 1 is an introductory chapter and Chapter 2 is a literature review that provides a context for the contributions made in this thesis. Chapters 3-5 should be read consecutively since intermediate contributions build the basis for the contributions that follow. Chapter 6 can be read as an independent chapter, but some technical aspects are closely related to Chapters 3-5. Chapter 7 is a concluding chapter that discusses future work. We briefly summarize each chapter including the contributions made therein in the following.

**Chapter 1** is an introductory chapter that motivates the use of random features, in particular for dot product kernels. We define positive definite kernels, discuss associated scalability bottlenecks for Gaussian processes regression, and provide an intuitive introduction to modern example applications that make use of random features for dot product kernels.

**Chapter 2** is a literature review that discusses random feature approximations for shift-invariant kernels and for polynomial kernels. We further draw connections between the two. The main purpose of this chapter is to provide a context for Chapter 3. Some of the methods discussed also serve as baselines for our experiments in subsequent chapters.

In **Chapter 3**, we discuss and extend random feature approximations for polynomial kernels that we refer to as *polynomial sketches*. We make several contributions to the literature in this chapter.

Our first contribution is the derivation of closed form variance formulas for a group of polynomial sketches complementing existing theoretical results. Prior to our work, only upper bounds for such variances had been derived in the literature. Our novel variance formulas make it possible to study conditions under which a certain polynomial sketch yields lower mean squared errors than another. In particular, we show for the first time that the variance of structured polynomial sketches is upper-bounded by the variance of their unstructured counterparts when the degree of the polynomial is odd.

Our second contribution in this chapter is to construct complex-valued generalizations of the aforementioned polynomial sketches. We show that complex polynomial sketches can yield much lower variances than their real-valued coun-

terparts, especially when the degree of the polynomial is large.

In **Chapter 4**, we convert the complex polynomial sketches introduced in Chapter 3 into real-valued ones, and discuss the implications of such complex-to-real (CtR) transformations on their variances. In particular, we prove that CtR Gaussian and Rademacher polynomial sketches yield lower variances than their real-valued analogs under the same conditions as it is the case for the complex polynomial sketches introduced before. We demonstrate empirically that our novel structured *CtR-TensorSRHT* sketch achieves state-of-the-art performance in terms of kernel approximation errors and feature construction time.

In **Chapter 5**, we show how any dot product kernel can be represented as a positively weighted sum of polynomial kernels using a Maclaurin series. This observation allows us to construct random features for dot product kernels and even for the Gaussian kernel using the polynomial sketches introduced in the chapters before. Using our derived variance formulas from Chapter 3, we develop a data-driven optimization approach to determine the number of random features assigned to each polynomial sketch used in this construction. In an extensive empirical evaluation, we show that our optimized sketches achieve state-of-the-art performance for the tasks of polynomial and Gaussian kernel approximation. Lastly, we identify a pathology of our approach when approximating a Gaussian kernel with short length scales and propose a cure for this pathology.

**Chapter 6** deals with the *Optical Processing Unit (OPU)*, an optical hardware used to compute random projections. We show that such random projections naturally give rise to a limiting kernel that is closely related to a degree-2 polynomial kernel. In addition, we numerically modify these random feature maps to obtain higher-degree polynomials and derive associated variances in closed form. Finally, we benchmark projection times and energy consumption of the OPU against a recent GPU.

In **Chapter 7**, we conclude our work and discuss improvements that could be tackled in future work. In particular, we discuss the attention mechanism in Transformer neural network architectures and show that current random feature methods do not yield sufficiently good approximations in this case.

Chapters 3 and 5 are based on

- J. Wacker, M. Kanagawa, and M. Filippone. Improved random features for dot product kernels. *arXiv preprint arXiv:2201.08712*, 2022a

  submitted to the *Journal of Machine Learning Research*.

- J. Wacker and M. Filippone. Local random feature approximations of the gaussian kernel. In *Proceedings of the 26th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, Procedia Computer Science. Elsevier, 2021.

Chapter 4 is based on

- J. Wacker, R. Ohana, and M. Filippone. Complex-to-real random features for polynomial kernels. *arXiv preprint arXiv:2202.02031*, 2022b.

Chapter 6 is based on the following conference paper:

- R. Ohana, J. Wacker, J. Dong, S. Marmin, F. Krzakala, M. Filippone, and L. Daudet. Kernel computations from large-scale random features obtained by optical processing units. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 9294–9298, 2020.

The following additional conference paper was published during the course of writing this thesis:

- J. Wacker, M. Ladeira, and J. E. V. Nascimento. Transfer learning for brain tumor segmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries - 6th International Workshop, BrainLes 2020, Held in Conjunction with MICCAI 2020.*, volume 12658 of *Lecture Notes in Computer Science*, pages 241–251. Springer, 2020.

However, this contribution was made completely independently from the other works and we therefore consider it to be out of scope for this thesis.

# Mathematical Notation

We reserve italic letters (e.g., $a$) for scalars. We use lower case boldface letters (e.g., $\boldsymbol{a}$) for column vectors, and upper case boldface letters (e.g., $\boldsymbol{A}$) for matrices.

| | |
|---|---|
| $\boldsymbol{A}^H$ | Hermitian transpose of $\boldsymbol{A}$ |
| $\boldsymbol{A}^\top$ | transpose of $\boldsymbol{A}$ |
| $\|\boldsymbol{a}\|$ | Euclidean norm of $\boldsymbol{a}$ |
| $\|\boldsymbol{A}\|_{\mathrm{F}}$ | Frobenius norm of $\boldsymbol{A}$ |
| $(\boldsymbol{A})_{i,j}$ | $(i,j)$-th entry of $\boldsymbol{A}$ |
| $(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$ | horizontal concatenation of the column vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ into a matrix |
| $\{\boldsymbol{A}_i\}_{i=1}^N$ | set consisting of the matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_N$ |
| $\boldsymbol{I}_n$ | identity matrix of dimension $n$; the subscript may be omitted when no confusion arises |
| $\mathrm{tr}(\boldsymbol{A})$ | trace of $\boldsymbol{A}$ |
| $\mathrm{Re}\{\boldsymbol{A}\}$ | real part of $\boldsymbol{A}$ |
| $\mathrm{Im}\{\boldsymbol{A}\}$ | imaginary part of $\boldsymbol{A}$ |
| $\overline{\boldsymbol{A}}$ | conjugate of a matrix $\boldsymbol{A}$ |
| $\boldsymbol{e}_i$ | $i$-th standard basis vector, i.e., the $i$-th column of $\boldsymbol{I}$ |
| $\boldsymbol{0}$ | zero vector with proper dimension |
| $:=, =:$ | definitions from the right and from the left |
| $\mathrm{diag}(\boldsymbol{a})$ | diagonal matrix with the vector $\boldsymbol{a}$ being its diagonal |
| $\mathrm{vec}(\boldsymbol{A})$ | column vector obtained by stacking column-wise the elements of $\boldsymbol{A}$ |
| $\boldsymbol{A} \otimes \boldsymbol{B}$ | Kronecker product of two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ |
| $\boldsymbol{A} \odot \boldsymbol{B}$ | element-wise product of two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ |
| $\mathbb{E}[\boldsymbol{A}]$ | expection of $\boldsymbol{A}$ |
| $\mathbb{V}[a]$ | variance of $a$ |
| $\Pr\{\mathcal{E}\}$ | probability of an event $\mathcal{E}$ |
| $\mathcal{CN}(\boldsymbol{\mu}, \boldsymbol{K})$ | Circularly symmetric complex Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{K}$ |
| $\mathbb{R}$ | set of real numbers |
| $\mathbb{C}$ | set of complex numbers |
| $\mathbb{N}$ | set of natural numbers |
| $\mathcal{S}^{d-1}$ | $(d-1)$-dimensional unit sphere in $d$-dimensional space |

# Acronyms and Abbreviations

| | |
|---|---|
| CPU | Central Processing Unit |
| GPU | Graphical Processing Unit |
| OPU | Optical Processing Unit |
| NLP | Natural Language Processing |
| RF | Random Features |
| w.r.t. | with respect to |
| w.l.o.g. | without loss of generality |
| a.s. | almost surely |
| i.i.d. | independent and identically distributed |
| e.g. | exempli gratia, for example |
| i.e. | id est, that is |

# Chapter 1

# Introduction

The contents of this thesis are all related to random features for polynomial kernels and more general dot product kernels. The purpose of random features is to accelerate statistical models by constructing approximations of such kernels. Before delving into the details about random features, this introductory chapter serves to provide the reader with an intuitive understanding about the *benefits* of such approximations.

In the following, we define positive definite kernels including the subclasses of dot product kernels and shift-invariant kernels that are relevant to this thesis. We then discuss several example applications that suffer from a scalability bottleneck when exact kernel evaluations are used, and show how this bottleneck is resolved through the use of random features. Several of these examples are related to dot product kernels motivating the need for random feature approximations for this particular class of kernels. Lastly, we provide a brief introduction to the Optical Processing Unit (OPU), an optical hardware that we worked with, and from which we drew the inspiration for the main contents of this thesis.

For now, we treat random feature maps as black-box functions and postpone the introduction of first concrete examples to Chapter 2.

## 1.1 Positive Definite Kernels

*Positive definite kernels* are ubiquitous in machine learning. They are traditionally used in kernel methods (Scholkopf and Smola, 2002) and Gaussian processes (Rasmussen and Williams, 2006) to model non-linear phenomena in a theoretically principled way. Associated learning algorithms are often simple to solve and can therefore be equipped with theoretical guarantees. This is why traditional learning algorithms using positive definite kernels are often preferred in applications where the learning machine should behave in an anticipated manner, e.g., when high-stake decision-making is involved or when safety is required.

At the same time, many successful modern deep learning approaches embed kernel methods into neural network architectures to either yield richer feature rep-

resentations or to incorporate prior domain knowledge. We discuss such examples in the remainder of this chapter.

We now give a brief formal introduction to positive definite kernels. A more complete introduction can be found in, e.g., Scholkopf and Smola (2002, Chapter 2) and Steinwart and Christmann (2008, Chapter 4). The kernels we discuss in this work are generally real-valued, but their approximations can be complex-valued as we will see in Chapters 2 and 3. Therefore, we provide a more general definition for complex positive definite kernels in the following.

**Positive definite kernel.** Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ is a positive definite kernel if for every $x_1, \dots, x_m \in \mathcal{X}$ and $c_1, \dots, c_m \in \mathbb{C}$

$$\sum_{i=1}^{m} \sum_{j=1}^{m} c_i \overline{c_j} k(x_i, x_j) \geq 0 \tag{1.1}$$

holds, where $\bar{\cdot}$ denotes the complex conjugate.

We call the (positive semi-definite) matrix $\boldsymbol{K} \in \mathbb{C}^{m \times m}$ with elements $(\boldsymbol{K})_{i,j} = k(x_i, x_j)$ the Gram (or kernel) matrix of $k$ with respect to $x_1, \dots, x_m$. $\boldsymbol{K}$ is Hermitian, i.e., we have $k(x_i, x_j) = \overline{k(x_j, x_i)}$. For a real-valued kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, this simplifies to $k(x_i, x_j) = k(x_j, x_i)$. In this case, we use $c_1, \dots, c_m \in \mathbb{R}$ in Eq. (1.1).

**Feature map.** Kernels can equivalently be defined through feature maps. Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ is a positive definite kernel if there exists a $\mathbb{C}$-Hilbert space $\mathcal{H}$ and a map $\phi : \mathcal{X} \to \mathcal{H}$ such that for all $x, y \in \mathcal{X}$

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$$

holds, where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is an inner product in $\mathcal{H}$. We call $\phi$ a feature map and $\mathcal{H}$ a feature space of $k$.

In this work, we generally assume $\mathcal{X} \subseteq \mathbb{R}^d$. We denote vector-valued inputs by bold-faced letters. Hence, we write $k(\boldsymbol{x}, \boldsymbol{y})$ for some $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$. Depending on the kernel, $\mathcal{H}$ can be infinite dimensional.

The goal of this work is therefore to construct low-dimensional *random* feature maps yielding an *approximate* kernel $\hat{k}$. We denote these feature maps by $\Phi_{\mathcal{R}} : \mathbb{R}^d \to \mathbb{R}^D$ if they are real-valued and by $\Phi_{\mathcal{C}} : \mathbb{R}^d \to \mathbb{C}^D$ if they are complex-valued, and we drop the subscript when the context is clear. For unbiased random feature maps, the approximate kernel $\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y})$ or $\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{C}}(\boldsymbol{x})^\top \overline{\Phi_{\mathcal{C}}(\boldsymbol{y})}$ converges in probability to $k(\boldsymbol{x}, \boldsymbol{y})$ as $D$ increases. This leads to a trade-off between approximation quality and computational costs that both increase with $D$.

**Classes of kernel functions.** For the purpose of this work, we distinguish two major classes of kernels, *shift-invariant* or *stationary* kernels, and *dot product* kernels (see Rasmussen and Williams, 2006, Chapter 4). Shift-invariant kernels depend only on the lag vector $\boldsymbol{\tau} := \boldsymbol{x} - \boldsymbol{y} \in \mathbb{R}^d$, i.e., we can overload the notation and write $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{\tau})$ for two inputs $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$. A well-known example is the Gaussian kernel $\exp(-\|\boldsymbol{\tau}\|^2/(2l^2))$ with length scale parameter $l > 0$. Dot product kernels on the other hand depend only on the inner product $\boldsymbol{x}^\top \boldsymbol{y} \in \mathbb{R}$, so we write $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x}^\top \boldsymbol{y})$. Well-known examples include the polynomial kernel $(\gamma \boldsymbol{x}^\top \boldsymbol{y} + \nu)^p$ for some $\gamma, \nu \geq 0, p \in \mathbb{N}$ and the exponential dot product kernel $\exp(\boldsymbol{x}^\top \boldsymbol{y}/l^2)$ for some $l > 0$. We will cover random feature approximations for shift-invariant kernels and polynomial kernels in detail in Chapters 2, 3 and 4. General dot product kernels are covered later on in Chapter 5.

**Scalability bottleneck.** Statistical models such as kernel methods (Scholkopf and Smola, 2002) and Gaussian processes (Rasmussen and Williams, 2006) suffer from a scalability bottleneck when direct kernel evaluations are used. This is because the kernel needs to be evaluated for all pairs of inputs, which implies a time and memory complexity that is at least quadratic in the number of input points.

The scalability issue has been a focus of research since the earliest literature (Wahba, 1990, Chapter 7), and many approximation methods for reducing the computational costs have been developed (e.g., Williams and Seeger 2000; Rahimi and Recht 2007; Titsias 2009; Hensman et al. 2018). The *random features* that we discuss in this work, are a particularly popular approximation method that was originally introduced for *shift-invariant* kernels in the seminal work by Rahimi and Recht (2007). We follow this line of research and show in the following how such approximations can be used to improve the scalability of various statistical models that make use of kernels.

We start with the example of Gaussian process regression that is used in many of the numerical experiments that we carried out for this thesis.

## 1.2 Gaussian Process Regression

Gaussian process regression (GPR) (Rasmussen and Williams, 2006, Chapter 2) is a Bayesian non-parametric approach to supervised learning that makes use of positive-definite kernels. It is an attractive modelling choice as it provides uncertainty estimates through predictive variances next to the actual prediction. At the same time, the hyperparameters of the kernel function $k$ can be obtained through a gradient-based optimization of the *log marginal likelihood* (Rasmussen and Williams, 2006, Chapter 5.4) avoiding time-consuming cross-validation.

However, GPR does not scale well to a large number of training points due to the necessity for algebraic manipulations of the Gram matrix. We introduce GPR

here and show how its scalability bottleneck can be overcome by means of random feature approximations.

Suppose a training data set $\mathcal{D} := \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$ that we summarize in matrix notation as $\boldsymbol{X} := (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)^\top \in \mathbb{R}^{N \times d}$ and $\boldsymbol{y} := (y_1, \dots, y_N)^\top \in \mathbb{R}^N$. We assume that the observations $\boldsymbol{y}$ have been generated from $\boldsymbol{X}$ by an unknown latent function $f : \mathbb{R}^d \to \mathbb{R}$ that has been corrupted by independent Gaussian noise, i.e., $y_i = f(\boldsymbol{x}_i) + \epsilon_i$ with $\epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2_{\text{noise}})$ and $\sigma^2_{\text{noise}} > 0$.

In GPR, the vector of function evaluations $\mathbf{f} := (f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_N))^\top \in \mathbb{R}^N$ is assumed to have a joint Gaussian distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^N$ and covariance matrix $\boldsymbol{K}_{\mathbf{ff}} \in \mathbb{R}^{N \times N}$. We follow the standard approach and set $\boldsymbol{\mu}$ to zero here although more complex models exist (Rasmussen and Williams, 2006, Chapter 2.7). For now we assume to be working with real-valued GPs and introduce complex-valued GPs later on in this work. So the entries of $\boldsymbol{K}_{\mathbf{ff}}$ correspond to the evaluations of a real-valued positive definite kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ with $(\boldsymbol{K}_{\mathbf{ff}})_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ that determines the covariance of a pair of function values $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$.

The task of GPR is to predict the latent function value at a new test input $\boldsymbol{x}_* \in \mathbb{R}^d$ given a training set $\mathcal{D}$. The predictive distribution of $f(\boldsymbol{x}_*)|\mathcal{D} \in \mathbb{R}$ can be computed in closed form, and it is $\mathcal{N}(\mu_*, \sigma_*^2)$ with

$$\mu_* := \boldsymbol{k}_{\mathbf{f}*}^\top (\boldsymbol{K}_{\mathbf{ff}} + \sigma^2_{\text{noise}} \boldsymbol{I})^{-1} \boldsymbol{y} \quad \text{and} \quad \sigma_*^2 := k_{**} - \boldsymbol{k}_{\mathbf{f}*}^\top (\boldsymbol{K}_{\mathbf{ff}} + \sigma^2_{\text{noise}} \boldsymbol{I})^{-1} \boldsymbol{k}_{\mathbf{f}*}, \tag{1.2}$$

where $\boldsymbol{k}_{\mathbf{f}*} = (k(\boldsymbol{x}_1, \boldsymbol{x}_*), \dots, k(\boldsymbol{x}_N, \boldsymbol{x}_*))^\top \in \mathbb{R}^N$, $k_{**} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$ and $\boldsymbol{I} \in \mathbb{R}^{N \times N}$ is the identity matrix.

Although being available in closed form, the computation of the GPR predictor can be expensive in practice. The computational bottleneck here is to solve the linear systems in Eq. (1.2), which costs $\mathcal{O}(N^3)$ time. Even storing the matrix $\boldsymbol{K}_{\mathbf{ff}}$ requires $\mathcal{O}(N^2)$ memory and becomes infeasible when $N$ is large, typically greater than $10,000$ for a regular machine, so that approximations become necessary. We now present an alternative GPR formulation that is amenable to the random feature approximations discussed in this thesis.

**Explicit feature space formulation.** If there exists a finite-dimensional feature map $\Phi : \mathbb{R}^d \to \mathbb{R}^D$ such that $k(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})$, it can be shown (Rasmussen and Williams, 2006, Chapter 2) that Eq. (1.2) can be reformulated as

$$\mu_* := \Phi(\boldsymbol{x}_*)^\top \boldsymbol{A}^{-1} \Phi(\boldsymbol{X})^\top \boldsymbol{y} / \sigma^2_{\text{noise}} \quad \text{and} \quad \sigma_*^2 := \Phi(\boldsymbol{x}_*)^\top \boldsymbol{A}^{-1} \Phi(\boldsymbol{x}_*), \tag{1.3}$$

with $\boldsymbol{A} := \Phi(\boldsymbol{X})^\top \Phi(\boldsymbol{X}) / \sigma^2_{\text{noise}} + \boldsymbol{I}$ and $\Phi(\boldsymbol{X}) = (\Phi(\boldsymbol{x}_1), \dots, \Phi(\boldsymbol{x}_N))^\top \in \mathbb{R}^{N \times D}$.

The feature space representation (1.3) changes the computational cost from $\mathcal{O}(N^3)$ to $\mathcal{O}(ND^2)$, and thus improves the scaling of GPR drastically if $D \ll N$. Unfortunately, exact feature maps can be high-dimensional, as is the case for, e.g., polynomial kernels, or even infinite dimensional rendering an explicit feature space

formulation infeasible. In such cases, we can use the random feature maps studied in this thesis as a drop-in replacement for $\Phi$ in Eq. (1.3), where the dimension $D$ of $\Phi$ becomes a parameter determining the trade-off between computational costs and approximation quality of the exact GP predictor.

There exist further applications for positive definite kernels that, similar to GPR, suffer from a scalability bottleneck due to pairwise kernel evaluations. In the following, we introduce some example applications from the literature, which make use of dot product kernels and their random feature approximations in particular. Therefore, these applications motivate the necessity for random features for polynomial kernels and more general dot product kernels that are the focus of this manuscript.

## 1.3 Applications for Dot Product Kernels

While random features for shift-invariant kernels have been studied extensively in the past (see Liu et al. (2020) for a recent overview), there are relatively few works studying random features for the general class of dot product kernels. The purpose of this section is therefore to provide recent examples from the literature that benefit from such approximations.

As we will show in Chapter 5, any dot product kernel and even the Gaussian kernel can be formulated as a positively weighted sum of polynomial kernels. This turns polynomial kernels into very general function approximators to which we assign a great importance in this work. Polynomial kernels in turn model multiplicative interactions between the input features, which is why dot product kernels such as polynomial kernels are used in applications that make use of such interactions. Such applications include genomic data analysis (Aschard, 2016; Weissbrod et al., 2016), recommender systems (Rendle, 2010; Blondel et al., 2016), computer vision (Lin et al., 2018; Gao et al., 2016; Fukui et al., 2016), natural language processing (Yamada and Matsumoto, 2003; Chang et al., 2010; Vaswani et al., 2017), and evaluation metrics like the Kernel Inception Distance (KID) (Bińkowski et al., 2018) [1].

In the following, we focus on recent example applications in deep learning, namely bilinear pooling for fine-grained visual recognition (Lin et al., 2018), multimodal bilinear pooling for visual question answering (Fukui et al., 2016), and the dot-product attention mechanism used in Transformers (Vaswani et al., 2017).

### 1.3.1 Bilinear Pooling

We begin by discussing applications for the so-called *bilinear pooling* operation that is used in fine-grained visual recognition (Lin et al., 2018) and has been extended to a multi-modal context for visual question answering (Fukui et al., 2016). We

[1] We cover a random feature approximation of the KID in more detail in Chapter 4.

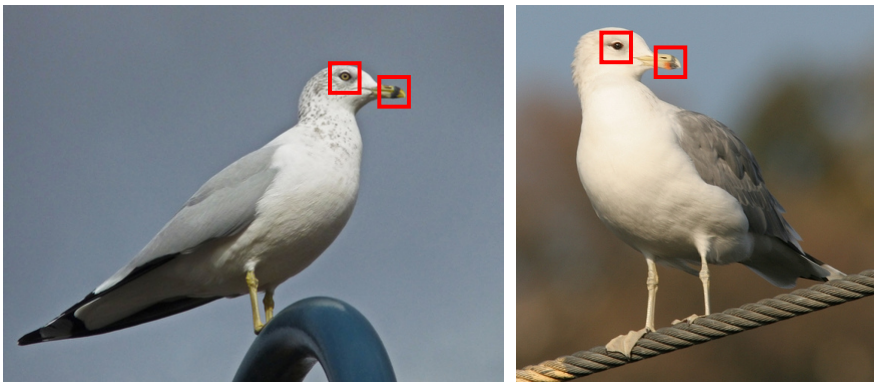


Figure 1.1: Comparing a Ring Billed Gull (left) to a California Gull (right) from the CUB-200 data set. The Ring Billed Gull has a yellow bill with a black ring around it, and a pale eye. The California Gull has a red spot on the lower bill and a dark eye.

cover the example of fine-grained visual recognition first, and show therein how bilinear pooling relates to explicit feature maps of a degree-2 polynomial kernel.

**Fine-grained visual recognition.** The task of fine-grained visual recognition is about the classification of pictures *within* a subordinate category. This classification usually requires the detection of fine-grained details. An example for such a task is the Caltech-UCSD Birds (CUB) 200 data set (Welinder et al., 2010) in which birds are categorized into 200 different species. Fig. 1.1 shows two example images of two similar bird species that can be distinguished by detecting details on their bills and eyes, respectively[2].

One way to solve this task would be to rely on anatomic annotations, locating different body parts of the bird on each picture. In this way, features could be extracted from each body part allowing for a detailed comparison. Although being provided in the CUB-200 data set, such annotations are expensive to obtain for a large number of images. This makes *holistic* models that globally process an image without annotations an attractive choice.

Lin et al. (2018) achieve remarkable improvements of holistic models on CUB-200 as well as other fine-grained visual recognition data sets, by applying a bilinear pooling operation to the outputs of a convolutional neural network. We define this operation and explain its scalability bottleneck in the following.

[2]See https://www.allaboutbirds.org/guide/Ring-billed_Gull/species-compare for an explanation. Accessed: 26/04/2022.

**Bilinear Pooling (Lin et al., 2018).** We consider a set of images $\mathcal{I}$, where an image $\boldsymbol{A} \in \mathcal{I}$ can be of arbitrary size. We then define a feature descriptor as a function $f : \mathcal{I} \times \mathcal{L} \to \mathbb{R}^d$ that describes an image $\boldsymbol{A}$ at a location $l \in \mathcal{L}$, where the set of locations $\mathcal{L}$ is defined by the feature descriptor and generally depends on $\boldsymbol{A}$.

Let $\boldsymbol{a} \otimes \boldsymbol{b} := \text{vec}(\boldsymbol{a}\boldsymbol{b}^\top) \in \mathbb{R}^{d_1 \cdot d_2}$ be the vectorized outer product of two vectors $\boldsymbol{a} \in \mathbb{R}^{d_1}$ and $\boldsymbol{b} \in \mathbb{R}^{d_2}$. The bilinear pooling operation is then defined as

$$\text{Bil}(f, \boldsymbol{A}) := \sum_{l \in \mathcal{L}} f(\boldsymbol{A}, l) \otimes f(\boldsymbol{A}, l) \in \mathbb{R}^{d^2}, \tag{1.4}$$

which forms a global image descriptor, since the sum over $\mathcal{L}$, referred to as a *pooling* operation, aggregates local image descriptions over image locations. $f$ could correspond to feature extractors such as SIFT (Lowe, 1999), HOG (Dalal and Triggs, 2005), or a convolutional neural network (Lecun et al., 1998).

The term $f(\boldsymbol{A}, l) \otimes f(\boldsymbol{A}, l)$ in Eq. (1.4) contains second-order multiplicative interactions of the coordinates of $f(\boldsymbol{A}, l)$. These interactions correspond to the explicit feature map of a degree-2 polynomial kernel as shown in Scholkopf and Smola (2002, Proposition 2.1). Taking the inner product of the bilinearly pooled features of two different images $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{I}$ thus yields a sum of degree-2 polynomial kernels:

$$\text{Bil}(f, \boldsymbol{A})^\top \text{Bil}(f, \boldsymbol{B}) = \sum_{l \in \mathcal{L}} \sum_{l' \in \mathcal{L}} (f(\boldsymbol{A}, l)^\top f(\boldsymbol{B}, l'))^2$$

The polynomial kernel here is defined as $k(\boldsymbol{x}, \boldsymbol{y}) = (\gamma \boldsymbol{x}^\top \boldsymbol{y} + \nu)^2$, where we set $\boldsymbol{x} = f(\boldsymbol{A}, l), \boldsymbol{y} = f(\boldsymbol{B}, l'), \gamma = 1$ and $\nu = 0$.

**Computational bottleneck.** The dimension of the bilinear features $\text{Bil}(f, \cdot)$ (1.4) is $d^2$, where $d$ is the dimension of the features extracted by $f$. The quadratic scaling in $d$ can render the bilinear pooling operation expensive. E.g., Lin et al. (2018) extract *relu5* outputs from a VGG-M network (Chatfield et al., 2014), which yields $d = 512$ dimensional outputs with $|\mathcal{L}| = 27^2$ for an RGB input image of size $448 \times 448 \times 3$ from the CUB-200 data set.

As there are 200 classes in the CUB-200 data set, a linear classifier used on top of the bilinear pooling features has $512^2 \cdot 200 \approx 52$ million parameters imposing high computational and memory costs. At the same time, storing $512^2$ dimensional features for subsequent image comparison and retrieval tasks requires one megabyte of storage per image, which is around 10 times as much as storing the original image in JPEG format.

Hence, Gao et al. (2016) propose *Compact Bilinear Pooling* to reduce the dimensionality of $\text{Bil}(f, \cdot)$ (1.4) using TensorSketch (Pham and Pagh, 2013). More precisely, they replace $f(\boldsymbol{A}, l) \otimes f(\boldsymbol{A}, l)$ in Eq. (1.4) by $\Phi(f(\boldsymbol{A}, l))$, where $\Phi : \mathbb{R}^d \to \mathbb{R}^D$ is a random feature map of the polynomial kernel. This can drastically reduce computational costs when $D \ll d^2$ holds. We will introduce such random feature maps as well as TensorSketch in Chapter 2.


Figure 1.2: Two question-answer pairs from the VQA data set (Antol et al., 2015) along with the corresponding images.

In the following, we provide an additional example application, where bilinear pooling is used to combine the features of two different modalities, namely image and text.

**Multi-modal bilinear pooling for visual question answering.** The task of Visual Question Answering (VQA) is to provide an answer to a question that relates to an image (see Fig. 1.2 for an example). The answers to be predicted can be multiple choice or open-ended, meaning that they need to be provided in the form of free text. Models solving the VQA task well could be used for instance to describe images to visually impaired people.

Answers in the VQA benchmark (Antol et al., 2015) take the form of short descriptive words instead of full sentences. They may contain only a single word, and many answers such as numbers, colors and objects appear frequently. Therefore, Fukui et al. (2016) model the VQA problem as a classification task, where the input to the model is an image and a question, and the output is the most probable answer among a provided list of frequently occurring answers in the data set.

The challenge in modelling this task is to combine the feature representations of two different modalities (image and text) for further processing. These representations then need to be classified into a large number of categories, e.g., Fukui et al. (2016) use 3000 of the most common answers as classes. They introduce the technique of *multi-modal compact bilinear pooling* to combine the features of both modalities in a low-dimensional representation that can be projected onto a 3000-dimensional class representation under a reasonable computational budget. This feature representation is an instance of random feature approximations for polynomial kernels. We therefore introduce multi-modal bilinear pooling in the following.

Let $f_{\text{img}} : \mathcal{I} \times \mathcal{L} \to \mathbb{R}^{d_1}$ be an image feature extraction function as introduced before for bilinear pooling. To convert a given question into a numerical represen-

tation, we further define a function $f_{\text{txt}} : \mathcal{T} \to \mathbb{R}^{d_2}$ that converts a text $t \in \mathcal{T}$ into a $d_2$-dimensional feature vector. For instance, this function could be a recurrent neural network or a Transformer (Vaswani et al., 2017).

We then extend Eq. (1.4) to a multi-modal context as follows:

$$\text{MulBil}(f_{\text{img}}, f_{\text{txt}}, \boldsymbol{A}, t) := \sum_{l \in \mathcal{L}} f_{\text{img}}(\boldsymbol{A}, l) \otimes f_{\text{txt}}(t) \in \mathbb{R}^{d_1 \cdot d_2} \tag{1.5}$$

Thus, the vectorized outer product is applied to the features of two different modalities instead of a feature map with itself, giving rise to a multi-modal feature vector.

This way of combining the features of different modalities provides a richer representation than taking their element-wise sum/product or simply concatenating them, since we consider multiplicative interactions between all *pairs* of elements. As for standard bilinear pooling, the caveat is that this feature representation is high-dimensional if $d_1$ and/or $d_2$ are large. E.g., $d_1 = d_2 = 2048$ in Fukui et al. (2016) lead to a subsequent classification layer with $2048^2 \cdot 3000 \approx 12.6$ billion parameters occupying around 48 gigabytes of memory when using 32 bits precision.

Similar to Gao et al. (2016), Fukui et al. (2016) propose to employ TensorSketch to obtain a low-dimensional sketch of $f_{\text{img}}(\boldsymbol{A}, l) \otimes f_{\text{txt}}(t)$ without ever constructing this vector explicitly. The sketching methods for polynomial kernels discussed in this thesis such as TensorSketch can be applied here despite the fact that the outer product $\otimes$ in Eq. (1.5) is not applied to a vector with itself. This is because random feature maps for polynomial kernels are generally constructed by combining independent linear projections of individual inputs as we will show in Chapter 2. In this way, Fukui et al. (2016) obtain a $16,000$-dimensional representation, which corresponds to around 0.4% of the original number of elements that would be obtained through Eq. (1.5). The storage of the classification layer is thus reduced to around 183 megabytes rendering the training of the model feasible. Efficiency improvements of random features for polynomial kernels that we discuss in this thesis could yield further computational savings for applications of bilinear pooling.

The applications discussed so far are connected to the use of polynomial kernels. We now provide an example application for the exponential dot product kernel.

### 1.3.2 Dot Product Attention for Transformers

Transformers (Vaswani et al., 2017) are neural network architectures that have been improving the state-of-the-art performance in numerous machine learning benchmarks over the last years. Examples include natural language processing (NLP) tasks like language generation (Brown et al., 2020) and machine translation (Banar et al., 2020), but also other tasks like image classification (Liu et al., 2021). Transformers gained particular popularity in NLP as they can be pretrained on large amounts of unstructured text available on the web, and be fine-tuned to the task at hand on a regular GPU affordable to the larger public (Devlin et al., 2019).

These architectures heavily rely on the attention mechanism (Vaswani et al., 2017) that models pairwise dependencies between the elements of an input sequence. E.g., in machine translation, the elements of an input sequence would be the tokens (word fragments like word stems or syllables) of a sentence to be translated. However, modelling pairwise interactions inside an input sequence introduces a quadratic time and memory complexity in the number $L$ of elements in the sequence, making it impossible to train and fine-tune such models for large $L \gg 512$. This is also because transformers commonly use multiple attention modules, e.g., BERT-large (Devlin et al., 2019) uses a total of 24 layers equipped with 16 attention heads leading to a total number of 384 attention modules. This computational bottleneck makes it difficult to capture long-range dependencies between different tokens. Such dependencies need to be captured in tasks like text summarization, where an input sequence could consist of thousands of tokens contained in a long text.

The computational bottleneck for the modelling of long input sequences has been addressed in numerous works (e.g. Xiong et al., 2021; Choromanski et al., 2021; Zaheer et al., 2020). We provide a brief introduction to the kernelized viewpoint here for which random feature approximations have been proposed (Choromanski et al., 2021). This viewpoint is of particular interest to this thesis as it shows that the attention matrix is a normalized Gram matrix of a dot product kernel that can be approximated through random features, enabling a linear instead of a quadratic time and memory complexity with respect to $L$.

**Dot-product attention.** The dot product self-attention mechanism (Vaswani et al., 2017) can be described as follows. Consider an input sequence of objects $\boldsymbol{x}_1, \dots, \boldsymbol{x}_L$ such as text tokens or image fragments, where each object $\{\boldsymbol{x}_i\}_{i=1}^L$ is represented numerically by a $d_{\text{emb}}$-dimensional feature vector, such that it can be processed by a neural network. We summarize the sequence in matrix form as $\boldsymbol{X} := (\boldsymbol{x}_1, \dots, \boldsymbol{x}_L)^\top \in \mathbb{R}^{L \times d_{\text{emb}}}$.

The self-attention mechanism is a *learnable* transformation that converts the input sequence $\boldsymbol{X}$ into an output sequence $\boldsymbol{Z} := (\boldsymbol{z}_1, \dots, \boldsymbol{z}_L)^\top \in \mathbb{R}^{L \times d}$, where the output dimension $d$ is determined by a set of trainable weight matrices $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{d_{\text{emb}} \times d}$. We define a set of linear projections $\boldsymbol{Q} := \boldsymbol{X}\boldsymbol{W}_Q, \boldsymbol{K} := \boldsymbol{X}\boldsymbol{W}_K, \boldsymbol{V} := \boldsymbol{X}\boldsymbol{W}_V$ and summarize the self-attention transformation in matrix-form as follows:

$$\begin{aligned} \boldsymbol{Z} =& \text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) := \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{V} \\ & \text{with} \quad \boldsymbol{A} := \exp(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d}) \quad \text{and} \quad \boldsymbol{D} := \text{diag}(\sum_{i=1}^{L} A_{1i}, \dots, \sum_{i=1}^{L} A_{Li}) \end{aligned}$$

Each element of $\boldsymbol{D}^{-1}\boldsymbol{A}$ hence corresponds to the evaluation of the *softmax*[3] func-

[3]Softmax is defined as a function $f : \mathbb{R}^L \to \mathbb{R}^L$ with $(f(\boldsymbol{x}))_j = \exp(x_j)/\sum_{j=1}^{L} \exp(x_j)$.

tion:

$$\alpha_{i,j} := (\boldsymbol{D}^{-1}\boldsymbol{A})_{i,j} = \frac{\exp((\boldsymbol{W}_Q\boldsymbol{x}_i)^\top(\boldsymbol{W}_K\boldsymbol{x}_j)/\sqrt{d})}{\sum_{j=1}^{L}\exp((\boldsymbol{W}_Q\boldsymbol{x}_i)^\top(\boldsymbol{W}_K\boldsymbol{x}_j)/\sqrt{d})}, \tag{1.6}$$

where $\boldsymbol{D}^{-1}$ takes the role of normalizing the rows of $\boldsymbol{A}$. $\alpha_{i,j} \in [0,1]$ for $i,j = 1,\dots,L$ are called *attention weights*. This means that each individual output $\{\boldsymbol{z}_i\}_{i=1}^{L}$ is computed as $\boldsymbol{z}_i = \sum_{j=1}^{L}(\boldsymbol{W}_V\boldsymbol{x}_j)\alpha_{i,j}$, and is hence a positively weighted average of the vectors $\{\boldsymbol{W}_V\boldsymbol{x}_j\}_{j=1}^{L}$.

The learnable attention weights $\alpha_{i,j}$ should be large for related embeddings and small for unrelated ones. For instance, in NLP tasks, these embeddings could encode semantic information, and a weighted average of semantically related tokens in an input sequence could yield a contextualized representation avoiding ambiguities. To give a concrete example, the word "Jaguar" can either relate to an animal or a car manufacturer and its numerical embedding may therefore be ambiguous. Obtaining a novel numerical representation by averaging out its embedding vector with the one for the word "car" that appears in the same input sequence would disambiguate its numerical representation. The attention weight between the embeddings of "Jaguar" and "car" should thus be large.

**Kernelized view.** We can see $\boldsymbol{A}$ as a Gram matrix consisting of evaluations of an exponential dot product kernel $k(\boldsymbol{x},\boldsymbol{y}) = \exp(\boldsymbol{x}^\top\boldsymbol{y})$ for some $\boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^d$. Setting $(\boldsymbol{A})_{i,j} = k(\boldsymbol{W}_Q\boldsymbol{x}_i/\sqrt[4]{d}, \boldsymbol{W}_K\boldsymbol{x}_j/\sqrt[4]{d})$ shows this correspondence. We will show in the following how random feature maps can accelerate the computation of the attention mechanism explained before.

There exist random feature approximations for the exponential dot product kernel (Kar and Karnick, 2012; Choromanski et al., 2021) and we derive novel random feature maps for it in Chapter 5. We can thus define the approximate kernel $\hat{k}(\boldsymbol{x},\boldsymbol{y}) := \Phi(\boldsymbol{x})^\top\Phi(\boldsymbol{y})$ for some feature map $\Phi : \mathbb{R}^d \to \mathbb{R}^D$ such that $\mathbb{E}[\hat{k}(\boldsymbol{x},\boldsymbol{y})] = k(\boldsymbol{x},\boldsymbol{y})$. Substituting $\hat{k}$ into Eq. (1.6) yields the approximate attention weights

$$\hat{\alpha}_{ij} = \frac{\Phi(\boldsymbol{W}_Q\boldsymbol{x}_i/\sqrt[4]{d})^\top\Phi(\boldsymbol{W}_K\boldsymbol{x}_j/\sqrt[4]{d})}{\sum_{j=1}^{L}\Phi(\boldsymbol{W}_Q\boldsymbol{x}_i/\sqrt[4]{d})^\top\Phi(\boldsymbol{W}_K\boldsymbol{x}_j/\sqrt[4]{d})}.$$

We can now efficiently compute each $\{\boldsymbol{z}_i\}_{i=1}^{L}$ as

$$\begin{aligned}\boldsymbol{z}_i = \sum_{j=1}^{L}(\boldsymbol{W}_V\boldsymbol{x}_j)\hat{\alpha}_{ij} &= \frac{\sum_{j=1}^{L}(\boldsymbol{W}_V\boldsymbol{x}_j)\Phi(\boldsymbol{W}_Q\boldsymbol{x}_i/\sqrt[4]{d})^\top\Phi(\boldsymbol{W}_K\boldsymbol{x}_j/\sqrt[4]{d})}{\Phi(\boldsymbol{W}_Q\boldsymbol{x}_i/\sqrt[4]{d})^\top\sum_{j=1}^{L}\Phi(\boldsymbol{W}_K\boldsymbol{x}_j/\sqrt[4]{d})} \\ &= \frac{\left(\sum_{j=1}^{L}(\boldsymbol{W}_V\boldsymbol{x}_j)\Phi(\boldsymbol{W}_K\boldsymbol{x}_j/\sqrt[4]{d})^\top\right)\Phi(\boldsymbol{W}_Q\boldsymbol{x}_i/\sqrt[4]{d})}{\Phi(\boldsymbol{W}_Q\boldsymbol{x}_i/\sqrt[4]{d})^\top\left(\sum_{j=1}^{L}\Phi(\boldsymbol{W}_K\boldsymbol{x}_j/\sqrt[4]{d})\right)},\end{aligned} \tag{1.7}$$

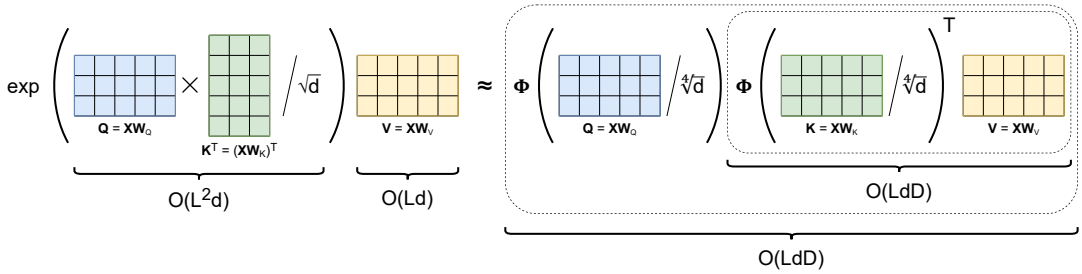



Figure 1.3: Exploiting the associativity of the matrix product to enable linear time complexity when computing random feature attention. The original unnormalized attention matrix $\exp(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d})$ (left) is factorized into the approximate matrix $\Phi(\boldsymbol{Q}/\sqrt[4]{d})\Phi(\boldsymbol{K}/\sqrt[4]{d})^\top$ (right) that can be efficiently multiplied with $\boldsymbol{V}$.

where the terms in parenthesis in Eq. (1.7) only need to be computed once in $\mathcal{O}(LdD)$ time as the terms are independent of $i$. Computing all $\{\boldsymbol{z}_i\}_{i=1}^L$ then takes $\mathcal{O}(LdD)$ again. Fig. 1.3 summarizes this computation in matrix form. We thus achieve $\mathcal{O}(LdD)$ instead of $\mathcal{O}(L^2d)$ computational complexity, where an advantage is achieved with random features if $D < L$.

A similar random feature approximation has been proposed to sample efficiently from the softmax distribution for classification problems with a large number of classes (Rawat et al., 2019).

The efficiency gains in Transformer architectures brought by random feature approximations could have a significant impact on computational as well as energy savings in the future. In fact, it was shown in Strubell et al. (2019) that training a simple and commonly used BERT-base (Devlin et al., 2019) language model on a GPU cloud platform creates the same amount of $CO_2$ emissions as a flight from New York to San Francisco for one passenger, while the price paid to the cloud platform lies between $3,700$ and $12,500$ US dollars. These amounts are orders of magnitude larger when including the necessary hyperparameter tuning and architecture search. Due to the widespread use of such models, efficient random feature approximations of the exponential kernel could thus contribute to more sustainable machine learning infrastructures in this domain.

Having motivated random feature approximations for dot product kernels through a set of example *applications*, we describe in the remainder of this chapter how such approximations *naturally occur* as the result of a physical process. The Optical Processing Unit is an optical hardware that makes use of this phenomenon to compute random features efficiently, both in terms of speed and power consumption. This device could potentially yield additional energy savings and working with it inspired the line of research that we pursue in this thesis.

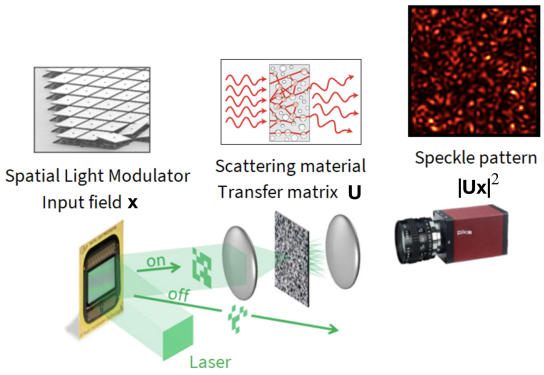



Figure 1.4: Experimental setup of the Optical Processing Unit (modified with permission from Saade et al. (2016)). The data vector is encoded in the coherent light from a laser using a DMD. Light then goes through a scattering medium and a speckle pattern is measured by a camera.

## 1.4 The Optical Processing Unit

The *Optical Processing Unit (OPU)* (Saade et al., 2016; Ohana et al., 2020) is an optical hardware that performs random projections literally at the speed of light, since these projections are the result of an optical scattering process. The principle of the random projections performed by the OPU is based on the use of a semi-transparent "scattering" material that refracts the laser light passing through it. As discussed in Liutkus et al. (2014), light going through such a scattering medium follows many complex paths that depend on the refractive index inhomogeneities at random positions. For a fixed scattering medium, the resulting process is still linear, deterministic, and reproducible making the features obtained by the OPU useful for machine learning.

Fig. 1.4 shows a schematic drawing of the physical apparatus of the OPU. We transform an arbitrary data point $\boldsymbol{x} \in \mathbb{R}^d$ into an analog representation by filtering a laser light beam at different positions using a *digital micromirror device (DMD)*. This encoded light then passes through the heterogeneous medium, implicitly performing a random matrix multiplication, where the random matrix is inherently determined by the scattering medium. The scattered light results in a so-called *speckle pattern* that is recorded by a camera, where the light intensity at each measuring point is the result of a superposition of the light beams filtered by the DMD. This superposition is mathematically modelled by a sum of the components of $\boldsymbol{x}$ weighted by random coefficients. Measuring the intensity of light at an arbitrary pixel location recorded by the camera induces a non-linear transformation of this sum. $D$ arbitrary pixel values taken together then constitute the random feature map

$$\Phi(\boldsymbol{x}) = |\boldsymbol{U}\boldsymbol{x}|^2/\sqrt{D}, \quad \text{with} \quad \boldsymbol{U} \in \mathbb{C}^{D\times d} \quad \text{and} \quad (\boldsymbol{U})_{i,j} \overset{i.i.d.}{\sim} \mathcal{CN}(0,1),$$

where $|\cdot|^2$ is the squared absolute value taken element-wise.

We showed in our work (Ohana et al., 2020) that the random projections computed and recorded by the OPU correspond to the random features of a kernel that is closely related to a degree-2 polynomial kernel. This is a remarkable result because it shows that the random features obtained by the aforementioned light scattering process are useful in machine learning applications where polynomial kernels are used.

We derive this result and further extensions in Chapter 6, where we also compare the energy consumption of the OPU against a recent GPU. Since the OPU has a lower power consumption (30 Watts) than a typical GPU (200-300 Watts) while potentially achieving faster projection times, these findings are interesting both from an ecological as well as from a performance perspective. Moreover, recent work (Gupta et al., 2019) has shown that the OPU can also be used to obtain purely *linear* random projections offering a wide range of additional applications in random feature approximations and randomized linear algebra.

# Chapter 2

# Random Feature Approximations of Shift-Invariant Kernels and Polynomial Kernels

In the previous chapter, we motivated the use of random feature approximations through a set of example applications, while treating random features as black-box functions. The purpose of this chapter is to provide a literature overview on random feature methods with a focus on random features for polynomial kernels, since these form the basis for the random feature methods developed in the subsequent chapters of this thesis.

At the same time, we cover random features for shift-invariant kernels, as they are the most widely used random feature approximations (see Liu et al. (2020) for a recent overview). Random feature constructions for shift-invariant kernels and polynomial kernels are conceptually different, but Pennington et al. (2015) established a connection between the two. As we compare against their method as well as against random Fourier features for the approximation of the Gaussian kernel in the empirical evaluations carried out for this thesis, we consider it important to discuss both lines of research in this chapter.

Random features were originally introduced as *random Fourier features* for shift-invariant kernels by Rahimi and Recht (2007), and several other works followed later on that studied random feature approximations for the polynomial kernel, in particular Kar and Karnick (2012); Pham and Pagh (2013); Hamid et al. (2014); Avron et al. (2014); Pennington et al. (2015); Ahle et al. (2020).

We therefore cover random Fourier features for shift-invariant kernels first.

## 2.1 Random Fourier Features for Shift-Invariant Kernels

In this section, we give a general introduction to random features for shift-invariant kernels. In addition, we discuss the subclass of isotropic kernels for which these random features take on a special form.

A shift-invariant (or stationary) kernel $k(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{C}$ is a positive definite function of $\boldsymbol{\tau} = \boldsymbol{x} - \boldsymbol{y} \in \mathbb{R}^d$, making it invariant to translations of the inputs. We will thus overload the notation and write $k(\boldsymbol{\tau})$ instead.

Shift-invariant kernels are characterized through Bochner's theorem (see Stein, 1999, Chapter 2.5):

**Theorem 2.1.1** (Bochner's theorem)**.** *A complex-valued shift-invariant kernel $k(\boldsymbol{\tau})$ on $\mathbb{R}^d$ is positive definite if and only if it can be written in the following form:*

$$k(\boldsymbol{\tau}) = \int_{\mathbb{R}^d} \exp(\mathrm{i}\boldsymbol{\omega}^\top \boldsymbol{\tau}) \mu(\boldsymbol{\omega}) d\boldsymbol{\omega}, \tag{2.1}$$

*where $\mu$ is a non-negative finite measure and $\mathrm{i} := \sqrt{-1}$.*

The construction of random Fourier features relies entirely on the above theorem and we introduce them in the following.

**Random Fourier features.** *Random Fourier features* were introduced in the seminal work by Rahimi and Recht (2007) who exploit the fact that the measure $\mu$ is proportional to a probability density $p$. To see this, let $\sigma^2 := k(\mathbf{0}) = \int_{\mathbb{R}^d} \mu(\boldsymbol{\omega}) d\boldsymbol{\omega} \geq 0$. We then obtain a density $p(\boldsymbol{\omega}) := \mu(\boldsymbol{\omega})/\sigma^2$ because $p(\boldsymbol{\omega}) \geq 0$ and $\int_{\mathbb{R}^d} p(\boldsymbol{\omega}) d\boldsymbol{\omega} = 1$ hold. W.l.o.g., we assume $\sigma^2 = 1$ from now on. (Stein, 1999, Chapter 2.5) call $p(\boldsymbol{\omega})$ the *spectral density*, which can be found for a given kernel through the following inversion formula:

$$p(\boldsymbol{\omega}) = \frac{1}{(2\pi)^d} \int_{\mathbb{R}^d} \exp(-\mathrm{i}\boldsymbol{\omega}^\top \boldsymbol{\tau}) k(\boldsymbol{\tau}) d\boldsymbol{\tau} \tag{2.2}$$

Therefore, there is a one-to-one correspondence between spectral densities and shift-invariant kernels.

Given a properly scaled shift-invariant kernel and its spectral density, we can rewrite Eq. (2.1) as:

$$\begin{aligned} k(\boldsymbol{\tau}) &= \int_{\mathbb{R}^d} \exp(\mathrm{i}\boldsymbol{\omega}^\top \boldsymbol{\tau}) p(\boldsymbol{\omega}) d\boldsymbol{\omega} = \mathbb{E}_{\boldsymbol{\omega} \sim p(\boldsymbol{\omega})} \left[ \exp(\mathrm{i}\boldsymbol{\omega}^\top \boldsymbol{\tau}) \right] \\ &= \mathbb{E}_{\boldsymbol{\omega} \sim p(\boldsymbol{\omega})} \left[ \exp(\mathrm{i}\boldsymbol{\omega}^\top \boldsymbol{x}) \, \overline{\exp(\mathrm{i}\boldsymbol{\omega}^\top \boldsymbol{y})} \right]. \end{aligned} \tag{2.3}$$

We can then obtain a Monte-Carlo estimate $\hat{k}_{\mathcal{C}}(\boldsymbol{\tau}) := \Phi_{\mathcal{C}}(\boldsymbol{x})^\top \overline{\Phi_{\mathcal{C}}(\boldsymbol{y})}$ of $k(\boldsymbol{\tau})$ with

$$\Phi_{\mathcal{C}}(\boldsymbol{x}) = \left( \exp(\mathrm{i}\boldsymbol{\omega}_1^\top \boldsymbol{x}), \ldots, \exp(\mathrm{i}\boldsymbol{\omega}_D^\top \boldsymbol{x}) \right)^\top / \sqrt{D} \in \mathbb{C}^D,$$

where $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D \overset{i.i.d.}{\sim} p(\boldsymbol{\omega})$. $\Phi_{\mathcal{C}} : \mathbb{R}^d \to \mathbb{C}^D$ is called a *random feature map* and we use the subscript $\mathcal{C}$ to emphasize the use of complex-valued basis functions. From now on, we drop the subscript $\boldsymbol{\omega} \sim p(\boldsymbol{\omega})$ of the expectation for brevity.

The dimension $D$ of the random feature map determines a trade-off between computational cost and approximation quality. We briefly derive the variances of random Fourier features to show this effect.

**Variance of the complex estimator.** The variance of $\hat{k}_{\mathcal{C}}(\boldsymbol{\tau})$ is:

$$\begin{aligned}\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{\tau})] &= \frac{1}{D^2}\sum_{\ell=1}^{D}\mathbb{V}[\exp(\mathrm{i}\boldsymbol{\omega}_\ell^\top\boldsymbol{\tau})] = \frac{1}{D^2}\sum_{\ell=1}^{D}\mathbb{E}\left[\left|\exp(\mathrm{i}\boldsymbol{\omega}_\ell^\top\boldsymbol{\tau})\right|^2\right] - \left|\mathbb{E}\left[\exp(\mathrm{i}\boldsymbol{\omega}_\ell^\top\boldsymbol{\tau})\right]\right|^2 \\ &= \frac{1}{D^2}\sum_{\ell=1}^{D}\mathbb{E}\left[\cos^2(\boldsymbol{\omega}_\ell^\top\boldsymbol{\tau}) + \sin^2(\boldsymbol{\omega}_\ell^\top\boldsymbol{\tau})\right] - k(\boldsymbol{\tau})^2 = \frac{1}{D}\left(1 - k(\boldsymbol{\tau})^2\right)\end{aligned}$$

As the $\{\boldsymbol{w}_\ell\}_{\ell=1}^D$ are i.i.d., we have $\mathbb{V}[\hat{k}(\boldsymbol{\tau})] \propto 1/D$ giving better kernel approximations for larger $D$. At the same time, the input $\boldsymbol{\tau}$ also affects the variance. E.g., if $\boldsymbol{\tau} = \mathbf{0}$, the variance is always zero regardless of $D$.

When the kernel of interest is real-valued, we can derive a real-valued feature map as follows.

**Real-valued feature maps.** For kernels $k(\boldsymbol{\tau}) \in \mathbb{R}$, we can rewrite Eq. (2.3) as:

$$\begin{aligned}k_{\mathcal{R}}(\boldsymbol{\tau}) &= \mathbb{E}\left[\exp(\mathrm{i}\boldsymbol{\omega}^\top\boldsymbol{\tau})\right] = \mathbb{E}\left[\cos(\boldsymbol{\omega}^\top\boldsymbol{\tau})\right] + \mathrm{i}\mathbb{E}\left[\sin(\boldsymbol{\omega}^\top\boldsymbol{\tau})\right] \\ &= \mathbb{E}\left[\cos(\boldsymbol{\omega}^\top\boldsymbol{\tau})\right] = \mathbb{E}\left[\cos(\boldsymbol{\omega}^\top\boldsymbol{x} - \boldsymbol{\omega}^\top\boldsymbol{y})\right] \\ &= \mathbb{E}\left[\cos(\boldsymbol{\omega}^\top\boldsymbol{x})\cos(\boldsymbol{\omega}^\top\boldsymbol{y}) + \sin(\boldsymbol{\omega}^\top\boldsymbol{x})\sin(\boldsymbol{\omega}^\top\boldsymbol{y})\right],\end{aligned}$$

where we use the trigonometric identity $\cos(\alpha-\beta) = \cos(\alpha)\cos(\beta)+\sin(\alpha)\sin(\beta)$. Thus, we can use an alternative random feature map $\Phi_{\mathcal{R}} : \mathbb{R}^d \to \mathbb{R}^{2D}$ for real-valued kernels $\hat{k}_{\mathcal{R}}(\boldsymbol{\tau}) := \Phi_{\mathcal{R}}(\boldsymbol{x})^\top\Phi_{\mathcal{R}}(\boldsymbol{y})$ with

$$\Phi_{\mathcal{R}}(\boldsymbol{x}) := \left[\cos(\boldsymbol{\omega}_1^\top\boldsymbol{x}), \sin(\boldsymbol{\omega}_1^\top\boldsymbol{x}), \ldots, \cos(\boldsymbol{\omega}_D^\top\boldsymbol{x}), \sin(\boldsymbol{\omega}_D^\top\boldsymbol{x})\right]^\top/\sqrt{D} \in \mathbb{R}^{2D}. \quad (2.4)$$

**Variance of the real estimator.** In this case the variance is

$$\begin{aligned}\mathbb{V}[\hat{k}_{\mathcal{R}}(\boldsymbol{\tau})] &= \frac{1}{D^2}\sum_{\ell=1}^{D}\mathbb{V}\left[\cos(\boldsymbol{\omega}_\ell^\top\boldsymbol{\tau})\right] = \frac{1}{D^2}\sum_{\ell=1}^{D}\mathbb{E}\left[\cos^2(\boldsymbol{\omega}_\ell^\top\boldsymbol{\tau})\right] - \mathbb{E}\left[\cos(\boldsymbol{\omega}_\ell^\top\boldsymbol{\tau})\right]^2 \\ &= \frac{1}{D}\left(\frac{1}{2} + \frac{1}{2}k(2\boldsymbol{\tau}) - k(\boldsymbol{\tau})^2\right) \leq \mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{\tau})],\end{aligned}$$

where we use the trigonometric identity $\cos(\alpha^2) = \frac{1}{2} + \frac{1}{2}\cos(2\alpha)$.

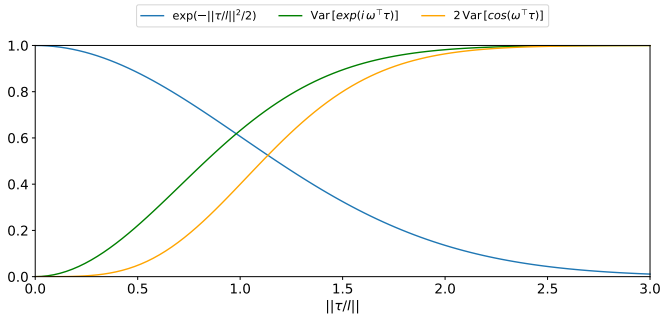



Figure 2.1: The Gaussian kernel (blue) and the variance of complex (green) and real-valued (yellow) kernel estimators. We double the variance of the real-valued estimator to compare feature maps of equal dimensions.

**Comparing the variance of real and complex kernel estimates.** Although we obtain a variance reduction w.r.t. $D$ when using the real-valued estimator, this comes at the expense of doubling the dimension of the feature map compared to the complex case. We therefore compare $\mathbb{V}[\hat{k}_{\mathcal{R}}(\boldsymbol{\tau})]$ and $\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{\tau})]$ on the example of the Gaussian kernel defined as $k(\boldsymbol{\tau}) := \exp(-\|\boldsymbol{\tau}\|^2/(2l^2))$ with length scale parameter $l > 0$ in the following.

Fig. 2.1 shows this comparison across different values of $\|\boldsymbol{\tau}/l\|$. In order to yield a fair comparison, we double the variance of the real-valued estimator since it requires $2D$ features (orange line). It can be seen that the real-valued estimator has a lower variance than its complex-valued counterpart (green line) for all values $\|\boldsymbol{\tau}/l\|$. Thus, for the Gaussian kernel, real-valued random Fourier features perform better despite doubling the feature map dimension. Furthermore, we can generally observe that variances increase as $\|\boldsymbol{\tau}/l\|$ increases, which indicates that more random features are required when $l$ is short compared to the scaling of the data itself. This happens in particular when the function to be modelled oscillates quickly, i.e., it uses high frequencies. We will empirically study this phenomenon in Chapter 5.

Although complex-valued feature maps perform worse in terms of the approximation variance in this example, we will show in Chapter 3 that the opposite can be the case for polynomial kernel approximations.

For further statistical guarantees on random Fourier features, we refer the reader to the following related works. Exponential concentration bounds are derived in the original work by Rahimi and Recht (2007) and are tightened by Sutherland and Schneider (2015). Statistical guarantees for kernel ridge regression are derived in (Avron et al., 2017; Rudi and Rosasco, 2017).

### 2.1.1 Spherical Random Features for Isotropic Kernels

A special class of random Fourier features are Spherical Random Features (SRF) introduced in Pennington et al. (2015) for isotropic kernels. They are of particular

importance to this thesis, as they will provide us with a baseline method when approximating high-degree polynomial kernels in Chapter 5.

We begin by characterizing isotropic kernels.

**Isotropic kernels.** Isotropic kernels are shift-invariant (or stationary) kernels of the form $k(\boldsymbol{x}, \boldsymbol{y}) = k(\|\boldsymbol{x} - \boldsymbol{y}\|)$ for some $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, where $\|\cdot\|$ denotes the Euclidean norm. This means that these kernels are also rotation-invariant. To see this, consider a rotation matrix $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ with $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I}_d$ being applied to both $\boldsymbol{x}$ and $\boldsymbol{y}$. Then we have:

$$\|\boldsymbol{R}\boldsymbol{x} - \boldsymbol{R}\boldsymbol{y}\|^2 = (\boldsymbol{R}\boldsymbol{x})^\top(\boldsymbol{R}\boldsymbol{x}) + (\boldsymbol{R}\boldsymbol{y})^\top(\boldsymbol{R}\boldsymbol{y}) - 2(\boldsymbol{R}\boldsymbol{x})^\top(\boldsymbol{R}\boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

We now show that all isotropic kernels are real-valued. Recall from the previous chapter that $\overline{k(\boldsymbol{x}, \boldsymbol{y})} = k(\boldsymbol{y}, \boldsymbol{x})$ holds for any kernel $k$. Therefore, we can write

$$\overline{k(\boldsymbol{x}, \boldsymbol{y})} = k(\boldsymbol{y}, \boldsymbol{x}) = k(\|-\boldsymbol{\tau}\|) = k(\|\boldsymbol{\tau}\|) = k(\boldsymbol{x}, \boldsymbol{y}),$$

which implies that $\operatorname{Im}\{k(\boldsymbol{\tau})\} = 0$, where $\operatorname{Im}\{\cdot\}$ denotes the imaginary part of a complex number.

Popular examples of isotropic kernels include Gaussian and Matérn kernels (see Rasmussen and Williams, 2006, Chapter 4.2). In the following, we derive random feature approximations for the specific case of isotropic kernels. For this purpose, we rely on a transformation of Bochner's theorem.

**Basis functions for isotropic kernels.** Bochner's theorem (2.1) can be reformulated to provide a specific spectral representation for the class of isotropic kernels. Let $k(\|\boldsymbol{\tau}\|)$ be an isotropic kernel with spectral density $p(\boldsymbol{\omega})$. As shown in Stein (1999, Chapter 2.10), the spectral density of isotropic kernels is radial, i.e., it only depends on $\|\boldsymbol{\omega}\|$. This allows to change the integration over the density $p(\boldsymbol{\omega})$ in Eq. (2.1) into an integration over the one-dimensional density $\hat{p}(\|\boldsymbol{\omega}\|)$, i.e., a density over the norm of $\boldsymbol{\omega}$, as follows:

$$k(\|\boldsymbol{\tau}\|) = \int_0^\infty d\,\|\boldsymbol{w}\|\,\hat{p}(\|\boldsymbol{\omega}\|)\,\underbrace{\Gamma\left(\frac{d}{2}\right)\left(\frac{2}{\|\boldsymbol{\omega}\|\,\|\boldsymbol{\tau}\|}\right)^{\frac{d}{2}-1} J_{\frac{d}{2}-1}(\|\boldsymbol{\omega}\|\,\|\boldsymbol{\tau}\|)}_{:=\,\Lambda_d(\|\boldsymbol{\omega}\|\|\boldsymbol{\tau}\|)}, \tag{2.5}$$

where $J_\nu$ with $\nu \in \mathbb{R}$ is the Bessel function of the first kind (Abramowitz and Stegun, 1974, Chapter 10) and $\Gamma(\cdot)$ is the Gamma function.

The one-dimensional density $\hat{p}(\|\boldsymbol{\omega}\|)$ can be derived from $p(\boldsymbol{\omega})$ by using an integration over spherical coordinates and the fact that both densities need to integrate to one:

$$\int_{\mathbb{R}^d} p(\boldsymbol{\omega})d\boldsymbol{w} = \int_0^\infty \underbrace{p(\boldsymbol{\omega})\,\|\boldsymbol{\omega}\|^{d-1}\,\frac{(2\pi)^{d/2}}{\Gamma(d/2)}}_{=:\,\hat{p}(\|\boldsymbol{\omega}\|)}\,d\,\|\boldsymbol{\omega}\| = 1 \tag{2.6}$$

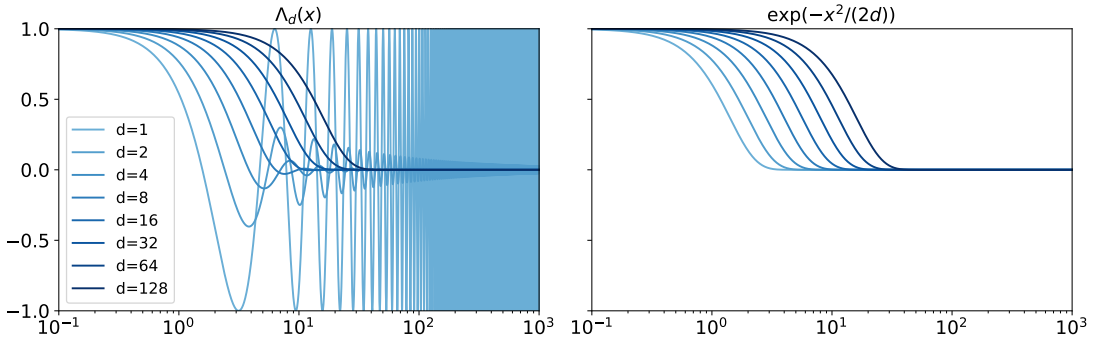



Figure 2.2: Comparing $\Lambda_d(x)$ on the left to $\exp(-x^2/(2d))$ on the right for different values of $d$. The functions are indistinguishable for large enough $d$.

In fact, the density $\hat{p}(\|\boldsymbol{\omega}\|)$ depends on the dimension $d$ of $\boldsymbol{\omega}$ and only exists if $k$ is a valid kernel, which is the case when $\int_0^\infty \|\boldsymbol{\tau}\|^{d-1} |k(\|\boldsymbol{\tau}\|)| \, d\|\boldsymbol{\tau}\| < \infty$ holds (see Stein, 1999, Chapter 2.10).

As noted in Genton (2002), the basis functions $\Lambda_d(\cdot)$ in Eq. (2.5) become less expressive and have an increasing lower bound for increasing $d$. Therefore, the class of valid isotropic kernels becomes more restrictive as $d$ increases and certain isotropic kernels are positive definite only up to a certain input dimension. Fig. 2.2 shows the basis functions $\Lambda_d(\cdot)$ in Eq. (2.5) for different values of $d$. It is interesting to note that $\Lambda_d(\|\boldsymbol{\omega}\| \, \|\boldsymbol{\tau}\|)$ approaches $\exp(-(\|\boldsymbol{\omega}\| \, \|\boldsymbol{\tau}\|)^2/(2d))$ as $d$ increases. This has the immediate consequence that any isotropic kernel that is positive definite for *all* $d$, can be modeled as a scale-mixture of Gaussians. This idea is formalized in Schoenberg's theorem:

**Theorem 2.1.2** (Schoenberg 1938)**.** *A continuous function $f : [0, \infty) \to \mathbb{R}$ is positive definite and radial on $\mathbb{R}^d$ if and only if it is of the form $f(r) = \int_0^\infty \exp(-r^2 t^2) d\mu(t)$, where $\mu$ is a finite non-negative Borel measure on $[0, \infty)$.*

Having learned that any positive definite isotropic kernel is directly defined by the one-dimensional density $\hat{p}(\|\boldsymbol{\omega}\|)$ allows us to adapt the sampling procedure of random Fourier features for the class of isotropic kernels. We refer to this novel sampling procedure as Spherical Random Features and explain it in the following.

**Spherical Random Features (SRF).** As noted earlier, the original spectral density $p(\boldsymbol{\omega})$ of isotropic kernels is radial and thus depends only on $\|\boldsymbol{\omega}\|$. This density must therefore be equal for all $\boldsymbol{\omega}$ with an equal radius $r := \|\boldsymbol{\omega}\| > 0$. This observation allows to sample from $p(\boldsymbol{\omega})$ by drawing uniform samples from the unit sphere $\mathcal{S}^{d-1} := \{\boldsymbol{\omega} \in \mathbb{R}^d : \|\boldsymbol{\omega}\| = 1\}$ and to scale them using samples from the norm density $\hat{p}(\|\boldsymbol{\omega}\|)$. Algorithm 1 summarizes this procedure.

Pennington et al. (2015) refer to the resulting random features as *Spherical Random Features*. Since isotropic kernels are always real-valued, we can replace

**Algorithm 1:** Spherical Random Features (SRF)

**Result:** An SRF map $\Phi_{\mathcal{C}}(\boldsymbol{x})$

Draw $D$ i.i.d. samples $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_D$ from $\mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$ ;

Draw $D$ i.i.d. samples $\omega_1, \omega_2, \ldots, \omega_D$ from $\hat{p}(\|\boldsymbol{\omega}\|)$ ;

Compute $\Phi_{\mathcal{C}}(\boldsymbol{x}) = \left(\exp\left(\mathrm{i}\frac{\omega_1}{\|\boldsymbol{s}_1\|}\boldsymbol{s}_1^\top \boldsymbol{x}\right), \ldots, \exp\left(\mathrm{i}\frac{\omega_D}{\|\boldsymbol{s}_D\|}\boldsymbol{s}_D^\top \boldsymbol{x}\right)\right)/\sqrt{D} \in \mathbb{C}^D$ ;

the complex-valued basis functions in Algorithm 1 with sines and cosines to obtain a real-valued feature map as we did in Eq. (2.4).

A crucial advantage of Algorithm 1 is that we can substitute i.i.d. samples from the unit-sphere $\mathcal{S}^{d-1}$ with orthogonal quasi-Monte-Carlo samples as described by Yu et al. (2016) to approximate the integral in Eq. (2.1) more efficiently, i.e., requiring less random features for the same kernel approximation error. An in-depth discussion when orthogonal samples are superior approximators for isotropic kernels can be found in Choromanski et al. (2018).

Spherical Random Features can also be used to approximate polynomial kernels on the unit sphere. However, such random feature approximations are biased. We therefore delay their introduction to the end of this chapter and cover unbiased approximations first.

## 2.2 Random Features for Polynomial Kernels

We study here random feature approximations of *polynomial kernels*, defined as

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\gamma \boldsymbol{x}^\top \boldsymbol{y} + \nu)^p, \tag{2.7}$$

where $\gamma, \nu \geq 0$ and $p \in \mathbb{N}$. We call such random features *polynomial sketches*[1]. Since polynomial kernels are not shift-invariant, widely known random Fourier features (Rahimi and Recht, 2007) cannot be applied for their unbiased approximation. Polynomial sketches are a fundamentally different approach, and can be understood as implicit linear random projections, or sketches (Woodruff, 2014), of the explicit high dimensional feature maps of polynomial kernels.

For simplicity, we focus on *homogeneous* polynomial kernels of the form

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y})^p, \tag{2.8}$$

i.e., we set $\gamma = 1, \nu = 0$ in Eq. (2.7). All other cases can be reduced to the homogeneous case, by appending $\sqrt{\nu}$ to the input vectors and rescaling them, i.e., by setting $\tilde{\boldsymbol{x}} := (\sqrt{\gamma}\boldsymbol{x}^\top, \sqrt{\nu})^\top \in \mathbb{R}^{d+1}$ and $\tilde{\boldsymbol{y}} := (\sqrt{\gamma}\boldsymbol{y}^\top, \sqrt{\nu})^\top \in \mathbb{R}^{d+1}$, we have

$$(\gamma \boldsymbol{x}^\top \boldsymbol{y} + \nu)^p = (\tilde{\boldsymbol{x}}^\top \tilde{\boldsymbol{y}})^p.$$

[1] We exclude Spherical Random Features (Pennington et al., 2015) from this terminology as they are not implicit high-dimensional sketches.

In this way, polynomial sketches for the homogeneous case can also be applied to the inhomogeneous case.

There exists an explicit feature map for polynomial kernels. To see this, we introduce some new notation. Recall from the previous chapter that we define the vectorized outer product between two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$ as $\boldsymbol{a} \otimes \boldsymbol{b} := \operatorname{vec}(\boldsymbol{a}\boldsymbol{b}^\top) \in \mathbb{R}^{d^2}$. We further define

$$\boldsymbol{a}^{(k)} := \underbrace{\boldsymbol{a} \otimes \cdots \otimes \boldsymbol{a}}_{k \text{ terms}} \in \mathbb{R}^{d^k} \tag{2.9}$$

for some $k \in \mathbb{N}$. Then the homogeneous polynomial kernel can be written as (Scholkopf and Smola, 2002, Proposition 2.1):

$$(\boldsymbol{x}^\top \boldsymbol{y})^p = (\boldsymbol{x}^{(p)})^\top \boldsymbol{y}^{(p)} \tag{2.10}$$

Thus, $\boldsymbol{x}^{(p)}$ and $\boldsymbol{y}^{(p)}$ are the exact feature maps of the input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. This exact feature expansion leads to $d^p$ dimensional vectors[2] that are infeasible to construct when $d$ and/or $p$ are moderately large. This justifies the need for randomized approximations of the polynomial kernel.

There are several different approaches for the randomized approximation of polynomial kernels in the literature. We identify four different construction types of such polynomial sketches that we call: (A) Product-Sketches, (B) Convolutions of CountSketches (a.k.a. TensorSketch), (C) Kronecker Product of SRHT sketches (a.k.a. TensorSRHT), and (D) Spherical Random Features for polynomial kernels. They all have in common that they can be constructed *without* forming the exact $d^p$-dimensional feature map of the polynomial kernel.

There exist further algorithms that can be used on top of the aforementioned random feature approximations to improve their efficiency. E.g., Hamid et al. (2014) propose to first create a high-dimensional projection that is down-projected in a second step and Ahle et al. (2020) propose a hierarchical random feature construction. Such algorithms are not the focus of this work and we will not discuss them further. The random features discussed in this thesis should be seen as base sketches that can be used inside such algorithms. The goal of this thesis is to improve the efficiency of these base sketches.

The contributions in this thesis are all made regarding Product-Sketches that were the first ones to be proposed in the literature. We cover them in the following and introduce random feature types (B)-(D) in the remainder of this chapter to give a context over the literature.

[2] By grouping up equal terms, we can reduce the dimensionality to $\binom{d+p-1}{p}$, which still leads to unrealistic dimensional feature vectors as soon as $d$ is large. For example, working with MNIST (Lecun et al., 1998) images of size 28x28 ($d = 784$) leads to 307,720 features for $p = 2$ and to 80,622,640 features for $p = 3$.

### 2.2.1 (A) Product-Sketches

We first study polynomial sketches proposed by Kar and Karnick (2012), which are also discussed in Hamid et al. (2014).

Let $D \in \mathbb{N}$ be the number of random features, and $p \in \mathbb{N}$ be the the degree of the polynomial kernel (2.8). Suppose we generate $p \times D$ i.i.d. random vectors

$$\boldsymbol{w}_{i,\ell} \in \mathbb{R}^d \quad \text{satisfying} \quad \mathbb{E}[\boldsymbol{w}_{i,\ell}\boldsymbol{w}_{i,\ell}^\top] = \boldsymbol{I}_d, \quad i \in \{1,\dots,p\}, \quad \ell \in \{1,\dots,D\}, \tag{2.11}$$

where $\boldsymbol{I}_d \in \mathbb{R}^{d\times d}$ denotes the identity matrix.

Then we define a random feature map as

$$\Phi(\boldsymbol{x}) := \frac{1}{\sqrt{D}}\left[(\prod_{i=1}^{p} \boldsymbol{w}_{i,1}^\top \boldsymbol{x}), \dots, (\prod_{i=1}^{p} \boldsymbol{w}_{i,D}^\top \boldsymbol{x})\right]^\top \in \mathbb{R}^D. \tag{2.12}$$

We call the specific form of the feature map in Eq. (2.12) a *Product-Sketch* since they are an element-wise product of individual linear random projections. The resulting approximation of the polynomial kernel (2.8) is given by

$$\hat{k}(\boldsymbol{x}, \boldsymbol{y}) := \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y}), \tag{2.13}$$

which is unbiased, as the expectation with respect to the random vectors (2.11) gives

$$\mathbb{E}\left[\Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})\right] = \frac{1}{D}\sum_{\ell=1}^{D}\prod_{i=1}^{p} \boldsymbol{x}^\top \mathbb{E}[\boldsymbol{w}_{i,\ell}\boldsymbol{w}_{i,\ell}^\top]\boldsymbol{y} = (\boldsymbol{x}^\top \boldsymbol{y})^p.$$

Kar and Karnick (2012) suggest to define random vectors in (2.11) using the Rademacher distribution: each element of $\boldsymbol{w}_{i,\ell}$ is independently drawn from $\{-1, 1\}$ with equal probability. Hamid et al. (2014) suggest to use Gaussian as well as structured Rademacher weights based on the subsampled randomized Hadamard transform (SRHT) (Tropp, 2011) instead. We study in Chapter 3 how the distribution of the random vectors affects the quality of the kernel approximation.

**Implicit sketching of high-dimensional features.** The random feature map (2.12) can be interpreted as a linear sketch (projection) of an explicit high-dimensional feature vector for the polynomial kernel. To describe this, consider the case $D = 1$ and let $\boldsymbol{w}_i = (w_{i,1}, \dots, w_{i,d})^\top \in \mathbb{R}^d$ with $i = 1, \dots, p$ be i.i.d. random vectors satisfying $\mathbb{E}[\boldsymbol{w}_i\boldsymbol{w}_i^\top] = \boldsymbol{I}_d$. Then, the random feature map $\Phi(\boldsymbol{x})$ (which is one dimensional in this case) for $\boldsymbol{x} := (x_1, \dots, x_d)^\top$ is given by

$$\Phi(\boldsymbol{x}) = \prod_{i=1}^{p} \boldsymbol{w}_i^\top \boldsymbol{x} = \prod_{i=1}^{p}\sum_{j=1}^{d} w_{i,j}x_j = \sum_{j_1=1,\dots,j_p=1}^{d} w_{1,j_1}x_{j_1}\cdots w_{p,j_p}x_{j_p} \tag{2.14}$$

$$= (\boldsymbol{w}_1 \otimes \cdots \otimes \boldsymbol{w}_p)^\top \boldsymbol{x}^{(p)}, \tag{2.15}$$

where $\boldsymbol{x}^{(p)} \in \mathbb{R}^{d^p}$ is defined as in Eq. (2.9). Therefore, for $D = 1$, the approximate kernel is given as

$$\hat{k}(\boldsymbol{x}, \boldsymbol{y}) := \Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{y}) = (\boldsymbol{w}_1 \otimes \cdots \otimes \boldsymbol{w}_p)^\top \boldsymbol{x}^{(p)} \cdot (\boldsymbol{w}_1 \otimes \cdots \otimes \boldsymbol{w}_p)^\top \boldsymbol{y}^{(p)} \tag{2.16}$$

As shown in Eq. (2.10), $\boldsymbol{x}^{(p)}$ and $\boldsymbol{y}^{(p)}$ are the exact feature maps of the polynomial kernel for the input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. Thus, the random feature map $\Phi(\boldsymbol{x}) \in \mathbb{R}$ in (2.15) is a projection of the exact feature map $\boldsymbol{x}^{(p)} \in \mathbb{R}^{d^p}$ onto $\mathbb{R}$.

Similarly, if $D > 1$, the random feature map $\Phi(\boldsymbol{x}) \in \mathbb{R}^D$ in (2.12) can be interpreted as a projection of the exact feature map $\boldsymbol{x}^{(p)}$ onto $\mathbb{R}^D$. A remarkable point of this random feature map is that it can be obtained without constructing the exact feature vector $\boldsymbol{x}^{(p)}$. Indeed, the computational complexity of constructing the random feature map $\Phi(\boldsymbol{x})$ is $\mathcal{O}(pdD)$, while the exact feature map $\boldsymbol{x}^{(p)}$ requires $\mathcal{O}(d^p)$.

The variances of the sketches introduced in this section have not been explored prior to our work. We will derive detailed variance formulas for several Product-Sketches in Chapters 3 and 4. However, there exist exponential concentration bounds similar to the ones derived in Rahimi and Recht (2007) for random Fourier features. These can be found in Kar and Karnick (2012); Hamid et al. (2014). We want to emphasize however, that these bounds serve to provide theoretical *guarantees* for the number of random features required for a certain approximation error. They should not be used to compare the statistical efficiency of these sketches. I.e., Hamid et al. (2014) obtain a tighter bound when using Gaussian weights in Eq. (2.11) than the one for Rademacher weights obtained by Kar and Karnick (2012). We will show in Chapter 3 however, that Rademacher sketches, and in particular structured ones yield much lower approximation variances than Gaussian sketches, which is also reflected in their empirical performances in downstream tasks in Chapter 4.

### 2.2.2 (B) Convolutions of CountSketches a.k.a. TensorSketch

*TensorSketch* is a structured polynomial sketch that was introduced by Pham and Pagh (2013). It can still be considered a state-of-the-art polynomial sketch in terms of kernel approximation quality and computation speed. Hence, it was used in Gao et al. (2016) and Fukui et al. (2016) to accelerate bilinear pooling operations as explained in the previous chapter. Since TensorSketch is a convolution of *CountSketches* (Charikar et al., 2002), we introduce CountSketches first.

**CountSketch (Charikar et al., 2002).** CountSketches were originally invented to maintain approximate counts of high-frequency elements in large data streams where it is infeasible to store the entire vocabulary of unique elements (Charikar et al., 2002). Weinberger et al. (2009); Woodruff (2014) show that CountSketches

can also be used as a dimensionality reduction technique leading to unbiased estimates of the linear kernel.

CountSketches are usually defined in terms of hash functions (Charikar et al., 2002; Pham and Pagh, 2013). However, in order to keep a consistent framework throughout this literature review, we define CountSketches in terms of randomized linear projections.

Let $\boldsymbol{C} \in \{-1, 0, 1\}^{D \times d}$ be a *sparse* projection matrix with elements

$$(\boldsymbol{C})_{i,j} = \delta(h_j = i)s_j, \quad \text{with} \quad i \in \{0, \dots, D-1\}, j \in \{1, \dots, d\}, \tag{2.17}$$

where $\boldsymbol{h} := (h_1, \dots, h_d)^\top \in \mathbb{R}^d$ has elements independently and uniformly drawn from $\{0, \dots, D-1\}$ and $\boldsymbol{s} := (s_1, \dots, s_d)^\top \in \{-1, 1\}^d$ is a vector with independent Rademacher samples. $\delta(h_j = i) = 1$ if $h_j = i$ and zero otherwise. The rows of $\boldsymbol{C}$ are indexed starting from zero as this will simplify our notation later on.

$\boldsymbol{C}$ is sparse because it has only one non-zero entry per column defined through $\boldsymbol{h}$. Thus, it has only $d$ non-zero entries in total allowing to compute the CountSketch $\boldsymbol{Cx} \in \mathbb{R}^D$ for a given $\boldsymbol{x} \in \mathbb{R}^d$ in $\mathcal{O}(d)$ time and memory. In fact, it is sufficient to only store $\boldsymbol{h}$ and $\boldsymbol{s}$ in the place of $\boldsymbol{C}$. The $i$-th element of $\boldsymbol{Cx}$ can then be efficiently computed as $(\boldsymbol{Cx})_i = \sum_{j:h_j=i} x_j s_j$.

We now show that $\boldsymbol{Cx}^\top \boldsymbol{Cy}$ is an unbiased estimate of the linear kernel:

$$\begin{aligned}
\mathbb{E}[\boldsymbol{Cx}^\top \boldsymbol{Cy}] &= \mathbb{E}\left[\sum_{i=0}^{D-1}\left(\sum_{j=1}^{d}\delta(h_j = i)s_j x_j\right)\left(\sum_{j'=1}^{d}\delta(h_{j'} = i)s_{j'} y_{j'}\right)\right] \\
&= \sum_{i=0}^{D-1}\sum_{j=1}^{d}\sum_{j'=1}^{d}\mathbb{E}[\delta(h_j = i)\delta(h_{j'} = i)]\mathbb{E}[s_j s_{j'}]x_j y_{j'} \\
&= \sum_{i=0}^{D-1}\sum_{j=1}^{d}\mathbb{E}[\delta(h_j = i)\delta(h_j = i)]x_j y_j = \sum_{i=0}^{D-1}\sum_{j=1}^{d}\frac{1}{D}x_j x_j = \boldsymbol{x}^\top \boldsymbol{y},
\end{aligned} \tag{2.18}$$

where we use the fact that $\mathbb{E}[s_j s_{j'}] = 1$ if $j = j'$ and $\mathbb{E}[s_j s_{j'}] = \mathbb{E}[s_j]\mathbb{E}[s_{j'}] = 0$ otherwise. Moreover, $\mathbb{E}[\delta(h_j = i)] = 1/D \cdot 1 + 0 = 1/D$.

In the following, we describe an efficient way to compute a CountSketch of $\boldsymbol{x}^{(p)}$ known as *TensorSketch* (Pham and Pagh, 2013) without ever constructing $\boldsymbol{x}^{(p)}$ itself. TensorSketch is thus a polynomial sketch and it is formed through a convolution of $p$ individual CountSketches.

**TensorSketch.** The convolution operation is equivalent to a product in the frequency domain. We therefore apply the discrete Fourier transform (DFT) to a CountSketch. Let $k \in \{0, \dots, D-1\}$ be the index over the elements of $\boldsymbol{Cx}$. The

$k$-th element of its DFT is computed as:

$$\begin{aligned}(\mathrm{DFT}[\boldsymbol{C}\boldsymbol{x}])_k &= \sum_{i=0}^{D-1} \exp\left(-\mathrm{i}\frac{2\pi}{D}ki\right) \boldsymbol{c}_i^\top \boldsymbol{x} = \sum_{j=1}^{d}\sum_{i=0}^{D-1} \exp\left(-\mathrm{i}\frac{2\pi}{D}ki\right) \delta(h_j = i) s_j x_j \\ &= \sum_{j=1}^{d} \exp\left(-\mathrm{i}\frac{2\pi}{D}kh_j\right) s_j x_j.\end{aligned}$$

Now we take the element-wise product of the Fourier transform of two independent CountSketches $\boldsymbol{C}_1\boldsymbol{x}$ and $\boldsymbol{C}_2\boldsymbol{x}$ defined through $\boldsymbol{h}_1, \boldsymbol{h}_2$ and $\boldsymbol{s}_1, \boldsymbol{s}_2$, respectively. The $k$-th element of this product then yields:

$$\begin{aligned}&(\mathrm{DFT}[\boldsymbol{C}_1\boldsymbol{x}])_k \cdot (\mathrm{DFT}[\boldsymbol{C}_2\boldsymbol{x}])_k \\ &= \left(\sum_{j=1}^{d} \exp\left(-\mathrm{i}\frac{2\pi}{D}kh_{1,j}\right) s_{1,j}x_j\right)\left(\sum_{j'=1}^{d} \exp\left(-\mathrm{i}\frac{2\pi}{D}kh_{2,j'}\right) s_{2,j'}x_{j'}\right) \\ &= \sum_{j=1}^{d}\sum_{j'=1}^{d} \exp\left(-\mathrm{i}\frac{2\pi}{D}k(h_{1,j} + h_{2,j'})\right) s_{1,j}s_{2,j'}x_j x_{j'} \\ &= \sum_{j=1}^{d}\sum_{j'=1}^{d} \exp\left(-\mathrm{i}\frac{2\pi}{D}k((h_{1,j} + h_{2,j'}) \bmod D)\right) s_{1,j}s_{2,j'}x_j x_{j'} = (\mathrm{DFT}[\boldsymbol{C}_{(1*2)}\boldsymbol{x}^{(2)}])_k,\end{aligned}$$

where $\boldsymbol{C}_{(1*2)} \in \{-1, 0, 1\}^{D\times d^2}$ is a novel CountSketch with $\boldsymbol{h}$ and $\boldsymbol{s}$ in Eq. (2.17) being replaced by

$$\boldsymbol{h}^{(1+2)} := \mathrm{vec}(\boldsymbol{h}_1 + \boldsymbol{h}_2^\top) \bmod D \in \mathbb{R}^{d^2} \quad \text{and} \quad \boldsymbol{s}^{(1\otimes 2)} := \boldsymbol{s}_1 \otimes \boldsymbol{s}_2 \in \mathbb{R}^{d^2}.$$

Note that the arithmetic modulus arises because $f(x) = \exp(-\mathrm{i}\frac{2\pi}{D}x)$ has a period of $D$.

In Appendix A.1, we show that $\boldsymbol{h}^{(1+2)}$ has elements drawn uniformly from $\{0, \ldots, D-1\}$. Moreover, $\boldsymbol{s}^{(1\otimes 2)}$ is a vector with elements drawn from the Rademacher distribution because the product of two independent Rademacher variables is again Rademacher distributed. This means that the convolution of two independent CountSketches of $\boldsymbol{x}$ yields a CountSketch of $\boldsymbol{x}^{(2)}$. Extending the derivation above to $p$ independent CountSketches $\{\boldsymbol{C}_i\boldsymbol{x}\}_{i=1}^{p}$ gives a new CountSketch $\boldsymbol{C}_{(1*\cdots*p)}$ of $\boldsymbol{x}^{(p)}$ that can be computed as:

$$\Phi(\boldsymbol{x}) := \mathrm{DFT}^{-1}\left[\mathrm{DFT}[\boldsymbol{C}_1\boldsymbol{x}] \cdots \mathrm{DFT}[\boldsymbol{C}_p\boldsymbol{x}]\right] = \boldsymbol{C}_{(1*\cdots*p)}\boldsymbol{x}^{(p)}, \tag{2.19}$$

where $\mathrm{DFT}^{-1}$ is the inverse discrete Fourier transform. We call $\Phi(\boldsymbol{x})$ (2.19) a TensorSketch of degree $p$. Using the Fast Fourier Transform, the total computational cost for obtaining $\Phi(\boldsymbol{x})$ is $\mathcal{O}(p(d+D\log D))$. This implies that we implicitly obtain a linear sketch in $d^p$ dimensions without ever having to compute $\boldsymbol{x}^{(p)}$ explicitly.

Using Eq. (2.18), we can show that the feature map $\Phi(\boldsymbol{x})$ leads to an unbiased estimate $\hat{k}(\boldsymbol{x}, \boldsymbol{y}) := \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})$ of the polynomial kernel:

$$\mathbb{E}[\hat{k}(\boldsymbol{x}, \boldsymbol{y})] = \boldsymbol{x}^{(p)\top} \boldsymbol{y}^{(p)} = (\boldsymbol{x}^\top \boldsymbol{y})^p$$

A caveat of TensorSketch is that the elements of the outer product $\boldsymbol{s}^{(1\otimes\cdots\otimes p)} = \boldsymbol{s}_1 \otimes \cdots \otimes \boldsymbol{s}_p$ are statistically dependent despite having pairwise covariances of zero. This is sufficient for TensorSketch to yield unbiased kernel estimates, but it influences higher order moments making it difficult to study its approximation variances for $p > 1$ as shown in Avron et al. (2014, Lemma 2 of the longer version) (the case $p = 1$ is covered in Weinberger et al. (2009)). There is thus no closed-form variance formula available in the literature[3].

Avron et al. (2014) derive only the following upper bound on the variance:

**Theorem 2.2.1** (Avron et al. (2014), Lemma 2)**.** *For any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, the variance of the TensorSketch kernel estimate $\hat{k}(\boldsymbol{x}, \boldsymbol{y}) := \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})$ with $\Phi$ defined as in Eq. (2.19) is upper-bounded as follows:*

$$\mathbb{V}\left[\hat{k}(\boldsymbol{x}, \boldsymbol{y})\right] \leq \frac{1}{D}(2 + 3^p) \|\boldsymbol{x}\|^{2p} \|\boldsymbol{y}\|^{2p}$$

We will show in Chapter 4 that TensorSketch performs better than Rademacher Product-Sketches empirically, in spite of the upper bound in Theorem 2.2.1 being larger than the one we obtain for the Rademacher and Gaussian Product-Sketches in Chapter 3. To the best of our knowledge, there are also no exponential concentration bounds for TensorSketch available in the literature, as it is the case for the Product-Sketches discussed in the previous section. We hypothesize that this is due to the fact that the rows of the matrix $\boldsymbol{C}_{(1*\cdots*p)}$ are not sampled independently obstructing a simple application of well-known concentration inequalities. Nonetheless, theoretical guarantees for downstream tasks like kernel ridge regression and kernel PCA are given in Avron et al. (2014); Ahle et al. (2020).

### 2.2.3 (C) Kronecker Product of SRHT Sketches a.k.a. TensorSRHT

In this section, we introduce structured polynomial sketches known as *TensorSRHT (Tensor Subsampled Randomized Hadamard Transform)*. Tropp (2011) studied TensorSRHT for $p = 1$ (linear case) called simply SRHT, and Hamid et al. (2014); Ahle et al. (2020) extended it[4] to arbitrary polynomial degrees $p$. Ahle et al. (2020) introduced the TensorSRHT sketch that we present in this section. The

[3]The original TensorSketch paper (Pham and Pagh, 2013) contains a variance formula, but makes the simplifying assumption that TensorSketch has the same variance properties as CountSketch (Charikar et al., 2002) applied to tensorized inputs.

[4]The sketches proposed by Hamid et al. (2014) and Ahle et al. (2020) are similar but not exactly the same. Hamid et al. (2014) uses $p \times B$ independent linear SRHT sketches (see Tropp,

prefix "Tensor" is used here to refer to the fact we form the Kronecker or Tensor product of $p$ independent sketches, which yields a new sketch that is an implicit linear projection of the tensorized input $\boldsymbol{x}^{(p)}$, as it was also the case for the other sketches described before. TensorSRHT is of great relevance to this thesis because we develop a closely related sketch in Chapter 3.

In order to define TensorSRHT, we need to cover some preliminaries first. More specifically, we need to define Kronecker products and Hadamard matrices.

**Kronecker Product.** We start by extending the notion of the outer product in Eq. (2.9) applied to vectors being applied to matrices instead. For this purpose, we define the Kronecker product of two matrices $\boldsymbol{A} \in \mathbb{R}^{m_1 \times n_1}, \boldsymbol{B} \in \mathbb{R}^{m_2 \times n_2}$ as

$$\boldsymbol{A} \otimes \boldsymbol{B} := \begin{bmatrix} (\boldsymbol{A})_{11}\boldsymbol{B} & \dots & (\boldsymbol{A})_{1n_1}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ (\boldsymbol{A})_{m_1 1}\boldsymbol{B} & \dots & (\boldsymbol{A})_{m_1 n_1}\boldsymbol{B} \end{bmatrix} \in \mathbb{R}^{(m_1 \cdot m_2) \times (n_1 \cdot n_2)}.$$

The Kronecker product has the *mixed-product property* defined as follows. For matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ of such sizes that one can compute $\boldsymbol{AC}$ and $\boldsymbol{BD}$, it holds that

$$(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{AC}) \otimes (\boldsymbol{BD}). \tag{2.20}$$

Note especially that we have $(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{x} \otimes \boldsymbol{y}) = \boldsymbol{Ax} \otimes \boldsymbol{By}$ for two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. We make use of this property for an efficient computation of TensorSRHT later.

We define structured Hadamard matrices next.

**Hadamard matrices.** Let $n := 2^m$ with $m \in \mathbb{N}$, and define $\boldsymbol{H}_n \in \{1, -1\}^{n \times n}$ to be the unnormalized Hadamard matrix, which is recursively defined as

$$\boldsymbol{H}_{2n} := \begin{bmatrix} \boldsymbol{H}_n & \boldsymbol{H}_n \\ \boldsymbol{H}_n & -\boldsymbol{H}_n \end{bmatrix}, \quad \text{with} \quad \boldsymbol{H}_2 := \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.21}$$

From now on, we always use $\boldsymbol{H}_d \in \{1, -1\}^{d \times d}$ with $d$ being the dimensionality of input vectors, assuming $d = 2^m$ for some $m \in \mathbb{N}$. If $d \neq 2^m$ for any $m \in \mathbb{N}$, we pad the input vectors with zeros until their dimensionality becomes $2^m$ for some $m \in \mathbb{N}$. For $i = 1, \dots, d$, let $\boldsymbol{h}_i \in \{1, -1\}^d$ be the $i$-th column of $\boldsymbol{H}_d$, i.e.,

$$\boldsymbol{H}_d = (\boldsymbol{h}_1, \dots, \boldsymbol{h}_d) \in \{1, -1\}^{d \times d}.$$

$\boldsymbol{H}_d$ has very useful properties that we exploit for the construction of several polynomial sketches in this thesis. The first useful property is $\boldsymbol{H}_d \boldsymbol{H}_d^\top = \boldsymbol{H}_d^\top \boldsymbol{H}_d =$

2011), where $B := \lceil \frac{D}{d} \rceil$ is the number of SRHT blocks per degree. The elements of these sketches are then shuffled across degress and blocks and the blocks are multiplied elementwise over $p$. Ahle et al. (2020) compute only $p$ independent sketches and subsample from their tensor product instead.

$d\boldsymbol{I}_d$, which implies that distinct columns (and rows) of $\boldsymbol{H}_d$ are orthogonal to each other, i.e., $\boldsymbol{h}_i^\top \boldsymbol{h}_j = 0$ for $i \neq j$. Another useful property is that $\boldsymbol{H}_{d^2} = \boldsymbol{H}_d \otimes \boldsymbol{H}_d$ holds, which will be needed for the construction of the TensorSRHT sketch in the following. Lastly, the recursive definition of $\boldsymbol{H}_d$ gives rise to the Fast Walsh-Hadamard Transform (FWHT) (Fino and Algazi, 1976) that multiplies $\boldsymbol{H}_d$ with a vector $\boldsymbol{a} \in \mathbb{R}^d$ in $\mathcal{O}(d \log d)$ instead of $\mathcal{O}(d^2)$ time while $\boldsymbol{H}_d$ does not need to be stored in memory.

We now move on to define SRHT, which is a linear random projection that forms the basis for the construction of TensorSRHT.

**Subsampled Randomized Hadamard Transform (SRHT).** SRHT (Tropp, 2011) is a linear random projection that is particularly fast due to the structured nature of $\boldsymbol{H}_d$.

Let $\boldsymbol{D} := \operatorname{diag}(\boldsymbol{w}) \in \mathbb{R}^{d \times d}$ be a diagonal matrix whose diagonal entries $\boldsymbol{w} := (w_1, \ldots, w_d)^\top \in \{1, -1\}^d$ are i.i.d. Rademacher random variables. Further consider a random permutation of the indices $\pi : \{1, \ldots, d\} \to \{1, \ldots, d\}$, and let $\pi(1), \ldots, \pi(d)$ be the permuted indices. We define a sampling matrix $\boldsymbol{P}_\pi := (\boldsymbol{e}_{\pi(1)}, \ldots, \boldsymbol{e}_{\pi(D)})^\top \in \mathbb{R}^{D \times d}$ with $D \leq d$, where $\boldsymbol{e}_{\pi(\ell)} \in \mathbb{R}^d$ is a vector whose $\pi(\ell)$-th element is 1 and all other elements are 0.

The SRHT is then defined as

$$\Phi(\boldsymbol{x}) := \boldsymbol{P}_\pi \boldsymbol{H}_d \boldsymbol{D} \boldsymbol{x} / \sqrt{D} = \boldsymbol{P}_\pi (\boldsymbol{H}_d (\boldsymbol{w} \odot \boldsymbol{x})) / \sqrt{D} \in \mathbb{R}^D, \tag{2.22}$$

where $\odot$ denotes the element-wise (Hadamard) product.

The parenthesis on the right-hand-side of Eq. (2.22) indicate the efficient order of computation to leverage the Fast Walsh-Hadamard Transform. $\Phi(\boldsymbol{x})$ can thus be computed in $\mathcal{O}(d \log d)$ instead of $\mathcal{O}(d^2)$ time, and using only $\mathcal{O}(d)$ memory since $\boldsymbol{H}_d$ does not need to be stored.

The sketch (2.22) leads to an unbiased estimate $\hat{k}(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})$ because

$$\begin{aligned}
\mathbb{E}[\Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})] &= \frac{1}{D} \sum_{\ell=1}^{D} \mathbb{E}[\boldsymbol{h}_{\pi(\ell)}^\top (\boldsymbol{w} \odot \boldsymbol{x}) \cdot \boldsymbol{h}_{\pi(\ell)}^\top (\boldsymbol{w} \odot \boldsymbol{y})] \\
&= \frac{1}{D} \sum_{\ell=1}^{D} \sum_{j=1}^{d} \sum_{j'=1}^{d} \mathbb{E}[h_{\pi(\ell),j} h_{\pi(\ell),j'}] \mathbb{E}[w_j w_{j'}] x_j y_{j'} \\
&= \frac{1}{D} \sum_{\ell=1}^{D} \sum_{j=1}^{d} \underbrace{h_{\pi(\ell),j}^2 \mathbb{E}[w_j^2]}_{=\,1} x_j y_j = \boldsymbol{x}^\top \boldsymbol{y},
\end{aligned}$$

where the expectation is taken w.r.t. both the random permutation $\pi$ as well as $\boldsymbol{w}$. We have $h_{\pi(\ell),j}^2 = 1$ for all $\{h_{\pi(\ell),j}\}_{j=1}^d$ because the elements of $\boldsymbol{h}_{\pi(\ell)}$ are either $-1$ or $+1$.

The goal of TensorSRHT is to apply SRHT to the tensorized input $\boldsymbol{x}^{(p)}$ to obtain an unbiased estimate of the polynomial kernel without ever constructing $\boldsymbol{x}^{(p)}$ explicitly.

**TensorSRHT.** The sketch defined by Ahle et al. (2020, Definition 15) exploits the mixed-product property (2.20) of the Kronecker product. This property allows for an efficient construction of a $d^p$-dimensional SRHT sketch, while working only with random projections in $\mathbb{R}^d$.

Let $\{\boldsymbol{D}_i\}_{i=1}^p$ be independently sampled diagonal matrices of size $\mathbb{R}^{d\times d}$ with their diagonal elements being defined through the independent Rademacher vectors $\{\boldsymbol{w}_i\}_{i=1}^p$ of size $\mathbb{R}^d$, respectively. We further define a random sampling matrix $\boldsymbol{P}_\pi := (\boldsymbol{e}_{\pi(1)}, \dots, \boldsymbol{e}_{\pi(D)})^\top \in \mathbb{R}^{D\times d^p}$ with $D \leq d^p$ as before.

We now show how to transform $p$ individual SRHT sketches $\{\boldsymbol{H}\boldsymbol{D}_i\boldsymbol{x}\}_{i=1}^p$ into a single SRHT sketch of an input point $\boldsymbol{x}^{(p)}$. Fur this purpose, we apply the mixed-product property (2.20) twice and use $\boldsymbol{H}_d \otimes \boldsymbol{H}_d = \boldsymbol{H}_{d^2}$ to show the following key relationship used in TensorSRHT:

$$\begin{aligned}\Phi(\boldsymbol{x}) &:= \boldsymbol{P}_\pi(\boldsymbol{H}_d\boldsymbol{D}_1\boldsymbol{x} \otimes \cdots \otimes \boldsymbol{H}_d\boldsymbol{D}_p\boldsymbol{x})/\sqrt{D} && (2.23)\\ &= \boldsymbol{P}_\pi(\boldsymbol{H}_d\boldsymbol{D}_1 \otimes \cdots \otimes \boldsymbol{H}_d\boldsymbol{D}_p)(\boldsymbol{x} \otimes \cdots \otimes \boldsymbol{x})/\sqrt{D}\\ &= \boldsymbol{P}_\pi(\boldsymbol{H}_d \otimes \cdots \otimes \boldsymbol{H}_d)(\boldsymbol{D}_1 \otimes \cdots \otimes \boldsymbol{D}_p)\boldsymbol{x}^{(p)}/\sqrt{D}\\ &= \boldsymbol{P}_\pi\boldsymbol{H}_{d^p}(\boldsymbol{D}_1 \otimes \cdots \otimes \boldsymbol{D}_p)\boldsymbol{x}^{(p)}/\sqrt{D} && (2.24)\end{aligned}$$

$(\boldsymbol{D}_1 \otimes \cdots \otimes \boldsymbol{D}_p)$ in Eq. (2.24) is a diagonal matrix of size $\mathbb{R}^{d^p\times d^p}$ with its diagonal elements being $\boldsymbol{w}_1 \otimes \cdots \otimes \boldsymbol{w}_p \in \mathbb{R}^{d^p}$. As noted for TensorSketch before, these elements are Rademacher distributed with zero covariance. $\Phi(\boldsymbol{x})$ (2.24) thus corresponds to SRHT applied to $\boldsymbol{x}^{(p)}$.

The equivalent formulation in Eq. (2.23) allows us to efficiently compute this sketch.

**Efficient computation.** We can efficiently compute $\Phi(\boldsymbol{x})$ by first computing $\{\boldsymbol{H}\boldsymbol{D}_i\boldsymbol{x}\}_{i=1}^p$ via the FWHT as shown in Eq. (2.23). Then we sample $D$ index tuples $\{(i_{1,\ell}, \dots, i_{p,\ell})\}_{\ell=1}^D$ without replacement, where $i_{1,\ell}, \dots, i_{p,\ell}$ are uniformly sampled from $\{1, \dots, d\}$. The tuples $\{(i_{1,\ell}, \dots, i_{p,\ell})\}_{\ell=1}^D$ can be understood as $D$ single indices sampled without replacement from $\{1, \dots, d^p\}$.

We then compute each element $\{\Phi(\boldsymbol{x})_\ell\}_{\ell=1}^D$ of the final TensorSRHT sketch as

$$\Phi(\boldsymbol{x})_\ell = (\boldsymbol{H}\boldsymbol{D}_1\boldsymbol{x})_{i_{1,\ell}} \cdots (\boldsymbol{H}\boldsymbol{D}_p\boldsymbol{x})_{i_{p,\ell}}/\sqrt{D}, \tag{2.25}$$

where $\Phi(\boldsymbol{x})_\ell$ corresponds to a randomly sampled element (without replacement) from the $d^p$-dimensional vector $\boldsymbol{H}_{d^p}(\boldsymbol{D}_1 \otimes \cdots \otimes \boldsymbol{D}_p)\boldsymbol{x}^{(p)}/\sqrt{D}$.

The computational cost of TensorSRHT is thus $\mathcal{O}(p(d\log d + D))$, since computing $\{\boldsymbol{H}\boldsymbol{D}_i\boldsymbol{x}\}_{i=1}^p$ costs $\mathcal{O}(pd\log d)$ and taking the products in Eq. (2.25) for $\ell = 1, \dots, D$ costs an additional $\mathcal{O}(pD)$. Recall that TensorSketch discussed in the section before has a cost of $\mathcal{O}(p(D\log D + d))$. This makes TensorSRHT faster than TensorSketch if $D > d$.

**Statistical guarantees.** As for TensorSketch, the variance of the TensorSRHT sketch discussed here is not available in closed form in the literature as soon as $p > 1$ (a result for $p = 1$ is available in Choromanski et al. (2017)). Once again, we hypothesize that this is because the entries of $(\boldsymbol{D}_1 \otimes \cdots \otimes \boldsymbol{D}_p)$ in Eq. (2.24) are statistically dependent despite having zero covariance. Moreover, the elements of $\Phi(\boldsymbol{x})$ (2.24) additionally have *non-zero* covariances since every row of $\boldsymbol{H}_{d^p}$ is multiplied by the same matrix $(\boldsymbol{D}_1 \otimes \cdots \otimes \boldsymbol{D}_p)$.

One of the main contributions of this thesis is to derive a closely related structured sketch in Chapter 3 for which we obtain closed form variances. While Ahle et al. (2020) focus on statistical guarantees that do not take the data distribution into account, we elucidate under which conditions of the input data our structured sketches perform better than non-structured sketches in terms of variance comparisons. In particular, we show that our structured sketches yield lower variances than unstructured analogs for odd degrees of the polynomial kernel, with the additional advantage of being faster. We are not aware of any other work in the literature that theoretically proves such statistical advantages for structured polynomial sketches. Nonetheless, theoretical guarantees for TensorSRHT for downstream tasks can be found in Ahle et al. (2020).

### 2.2.4 (D) Spherical Random Features for Polynomial Kernels

Pennington et al. (2015) propose a fundamentally different approach to polynomial kernel approximation from the previous ones presented in this chapter. Instead of constructing an unbiased implicit sketch in $d^p$ dimensions, the authors propose to approximate polynomial kernels through isotropic kernels and random Fourier features (see Section 2.1.1). We review this approach here as it can be considered the state-of-the-art for the approximation of *high-degree* polynomial kernels on the unit sphere. It also serves as an important baseline in our work and we found it rather difficult to implement it in practice, which is why this section provides a comprehensive introduction and discusses some limitations of the method.

The approach by Pennington et al. (2015) is motivated by the fact that polynomial kernels with data lying on the unit-sphere can be expressed as isotropic kernels. The authors consider the approximation of a polynomial kernel $k : \mathcal{S}^{d-1} \times \mathcal{S}^{d-1} \to \mathbb{R}$ being parameterized as follows:

$$k(\boldsymbol{x}, \boldsymbol{y}) = \left(1 - \frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{a^2}\right)^p = \left(\nu + \gamma(\boldsymbol{x}^\top \boldsymbol{y})\right)^p, \tag{2.26}$$

where $\nu = 1 - 2/a^2$ and $\gamma = 2/a^2$ for some $a \geq 2$.

Eq. (2.26) shows the correspondence between a shift-invariant kernel (left-hand-side) and a polynomial kernel (right-hand-side) valid for inputs lying on the unit-sphere.

However, the condition $a \geq 2$ makes it impossible to model homogeneous polynomial kernels for which we have $\nu = 0$ and thus $a = \sqrt{2}$. Furthermore, the offset $\nu$ and the scaling term $\gamma$ are inter-dependent through a common dependency on $a$, which limits the class of polynomial kernels being modelled.

Since Eq. (2.26) takes the form of an isotropic kernel, one could hope to approximate $k$ with random Fourier features as shown in Section 2.1. Yet, for $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}^{d-1}$, we have $\|\boldsymbol{\tau}\| = \|\boldsymbol{x} - \boldsymbol{y}\| = \sqrt{2 - 2\cos\theta} \in [0, 2]$, where $\theta$ is the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$. Therefore, the support of $\boldsymbol{\tau}$ is not defined over the entire space $\mathbb{R}^d$ and Bochner's theorem (2.1) does not apply. Pennington et al. (2015) further prove that the support of $\boldsymbol{\tau}$ cannot be artificially extended such that the kernel function (2.26) remains positive definite. Hence, there exists no spectral density for this kernel, which obstructs the use of random Fourier features.

The authors therefore consider a *biased* approximation by finding a closely related isotropic kernel defined on $\mathbb{R}^d$, i.e., one for which random Fourier features can be obtained.

**Finding an approximate isotropic kernel.** Recall from Section 2.1 that any isotropic kernel defined on $\mathbb{R}^d$ for some $d \in \mathbb{N}$ can be expressed using Eq. (2.5) as

$$\hat{k}(\|\boldsymbol{\tau}\|) = \int_0^\infty d\,\|\boldsymbol{\omega}\|\;\hat{p}(\|\boldsymbol{\omega}\|)\;\Lambda_d(\|\boldsymbol{\omega}\|\,\|\boldsymbol{\tau}\|), \tag{2.27}$$

where $\hat{p}(\|\boldsymbol{\omega}\|)$ is a one-dimensional density and $\Lambda_d(\cdot)$ is defined through Eq. (2.5). Pennington et al. (2015) propose to optimize $\hat{p}(\|\boldsymbol{\omega}\|)$ so as to minimize the mean squared error

$$L = \frac{1}{2}\int_0^2 d\,\|\boldsymbol{\tau}\|\,\Big(k(\|\boldsymbol{\tau}\|) - \hat{k}(\|\boldsymbol{\tau}\|)\Big)^2 \tag{2.28}$$

with $k$ being defined as in Eq. (2.26). In practice, $L$ is evaluated on a grid over $\|\boldsymbol{\tau}\| \in [0, 2]$. Here, the density is parameterized as

$$\hat{p}(\|\boldsymbol{\omega}\|) \propto \|\boldsymbol{\omega}\|^{d-1} \max\left(0, \sum_{i=1}^N c_i \left(\frac{1}{\sqrt{2}\sigma_i}\right)^d \exp(-\|\boldsymbol{\omega}\|^2 / 4\sigma_i^2)\right) \tag{2.29}$$

for some $\{c_i\}_{i=1}^N, \{\sigma_i\}_{i=1}^N$ with $c_i \in \mathbb{R}$ and $\sigma_i > 0$.

The formulation of $\hat{p}(\|\boldsymbol{\omega}\|)$ is motivated by Theorem 2.1.2 stating that any isotropic kernel defined on arbitrary domains $\mathbb{R}^d$ can be modelled as a scale-mixture of Gaussian functions. Eq. (2.29) is obtained by choosing a scale-mixture over Gaussian distributions for $p(\boldsymbol{\omega})$ (2.2) and transforming it using Eq. (2.6). Pennington et al. (2015) further modify the resulting density by adding $\max(0, \cdot)$ and allowing negative values for $\{c_i\}_{i=1}^N$ in Eq. (2.29) for greater numerical flexibility. The normalization constant in Eq. (2.29) is found through a numerical integration on a grid over $\|\boldsymbol{\omega}\| \in \mathbb{R}_+$, which is also done for the evaluation of $\hat{k}$ (2.27) itself. $\{c_i\}_{i=1}^N, \{\sigma_i\}_{i=1}^N$ in Eq. (2.29) are then found using a gradient-based optimizer. Once they are obtained, one can sample from $\hat{p}(\|\boldsymbol{\omega}\|)$ using inverse-transform-sampling to eventually sample Spherical Random Features using Algorithm 1.

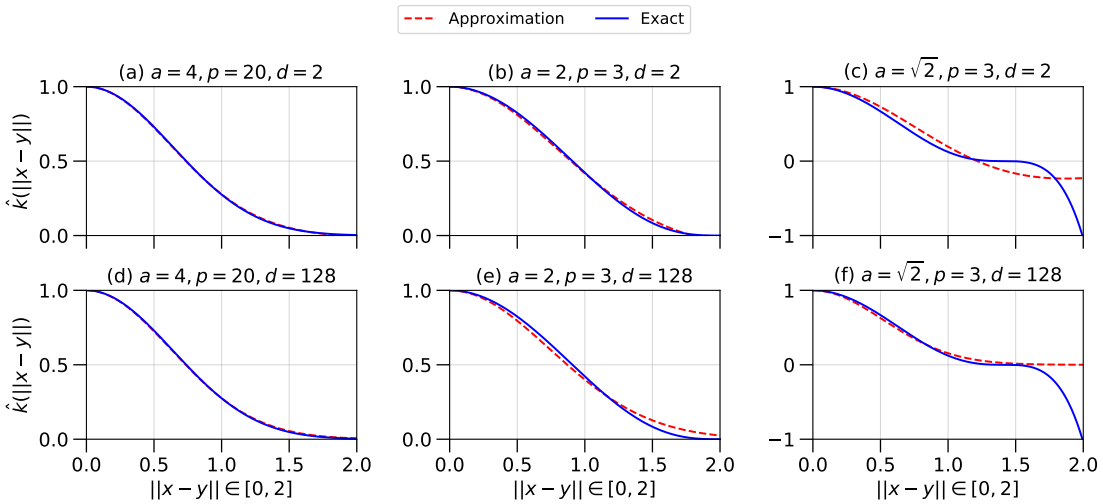



Figure 2.3: Approximating polynomial kernels on the unit-sphere through isotropic kernels. The first row shows approximations for $d = 2$ while the bottom row shows them for $d = 128$.

**Limits of the approximation.** Unlike the polynomial sketches presented earlier, the approximation (2.27) of $k$ (2.26) is biased. We show some characteristic approximation functions $\hat{k}(\|\boldsymbol{\tau}\|)$ (2.27) in Fig. 2.3 that are obtained through the optimization of Eq. (2.28). Note that these functions are themselves approximated using Spherical Random Features in a second step. Plots (a),(b),(d),(e) show kernel parameterizations with $a \geq 2$ as desired by Pennington et al. (2015). Plots (c) and (f) explicitly show homogeneous polynomial kernels ($a = \sqrt{2}$), and it becomes immediately obvious that they cannot be matched through $\hat{k}$ (2.27) exposing an important limitation of this method.

Furthermore, we note that high-degree ($p = 20$) polynomial kernels are better approximated than low-degree ($p = 3$) ones confirming the findings of the authors, who argue that their method is particularly suitable for large $p$ leading to a lower bias. They further show that Spherical Random Features can outperform previous approaches presented in Sections 2.2.1 and 2.2.2 for data lying on the unit-sphere in this case.

Although performing well for large $p$, we want to emphasize that the method by Pennington et al. (2015) is challenging to implement in practice. We therefore provide some insights on the numerical stability of the method that can help to simplify the implementation.

**Numerical stability.** The issue of numerical stability is not discussed in Pennington et al. (2015) but quite relevant. The evaluation of the Bessel function of the first kind $J_{d/2-1}(\cdot)$ as part of $\Lambda_d(\cdot)$ in Eq. (2.27) yields small values and tends to

numerically underflow for large $d$. The original implementation[5] proposed by the authors solves this issue by using different series approximations for different values of $d$. Instead, we propose to use $\exp(-x^2/(2d))$ instead of $\Lambda_d(x)$ from $d > 128$ to simplify the implementation. This gives a reasonably good approximation of $\Lambda_d(x)$ as shown in Fig. 2.2.

[5] https://github.com/felixyu/SRF

# Chapter 3

# Analysis and Extension of Product-Sketches

In this chapter, we focus on the unbiased approximation of polynomial kernels by means of Product-Sketches that we introduced in Chapter 2. As we will learn in Chapter 5, polynomial sketches can be used to construct random feature maps for arbitrary dot product kernels. This gives the study of the efficient approximation of polynomial sketches a large importance. Therefore, we carry out an in-depth variance analysis of Product-Sketches in this chapter and propose structured and complex extensions that improve these variances.

Product-Sketches were used in Kar and Karnick (2012) and Hamid et al. (2014), who proposed to sample corresponding weight vectors from Gaussian and Rademacher distributions, and derived concentration results for both. The bound for Gaussian Product-Sketches derived in Hamid et al. (2014, Lemma 3.1) guarantees a given pointwise kernel approximation error for fewer random features than the bound by Kar and Karnick (2012) for Rademacher sketches. However, this result does not reflect the empirical performance of these two sketches as we show later in this work.

In this chapter, we carry out a variance analysis of Product-sketches and we show that Rademacher sketches achieve a variance lower bound under the assumption that the corresponding weight vectors are i.i.d. We further prove that even lower variances can be attained when lifting the i.i.d. assumption and using structured Rademacher sketches.

To the best of our knowledge, such an analysis is novel to the literature and it elucidates under which conditions of the input data one sketch should be preferred over another. This is a key contribution of this chapter, which complements existing theoretical results that rather focus on the question *how many* random features are needed to provide a certain approximation error at a given probability. Unlike our analysis, such statistical guarantees can make it difficult to compare the *average-case* efficiency of different sketches.

The second major contribution of this chapter is to derive *complex-valued* gen-

eralizations of Product-Sketches and to study under which conditions they achieve lower variances than real-valued analogs. We show that variances generally decrease considerably for positively-valued data and large polynomial degrees.

## 3.1 Variance Analysis for Real-Valued Product-Sketches

We begin by studying the variance properties of existing real-valued Product-Sketches, namely the Gaussian and Rademacher sketches introduced in Kar and Karnick (2012); Hamid et al. (2014).

Recall from Chapter 2 that a Product-Sketch is defined as

$$\Phi_{\mathcal{R}}(\boldsymbol{x}) := \frac{1}{\sqrt{D}} \left[ (\prod_{i=1}^{p} \boldsymbol{w}_{i,1}^\top \boldsymbol{x}), \dots, (\prod_{i=1}^{p} \boldsymbol{w}_{i,D}^\top \boldsymbol{x}) \right]^\top \in \mathbb{R}^D, \tag{3.1}$$

for some weight vectors $\boldsymbol{w}_{i,\ell}$ with $i = 1, \dots, p$ and $\ell = 1, \dots, D$, where the subscript $\mathcal{R}$ emphasizes the use of real-valued weight vectors here. We construct complex-valued generalizations later on.

The resulting approximation of the polynomial kernel $(\boldsymbol{x}^\top \boldsymbol{y})^p$ is then given by

$$\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) := \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y}). \tag{3.2}$$

For ease of presentation, we focus on the case $D = 1$ (a single random feature), and consider $p$ i.i.d. weight vectors

$$\boldsymbol{w}_i = (w_{i,1}, \dots, w_{i,d})^\top \in \mathbb{R}^d \quad \text{satisfying} \quad \mathbb{E}[\boldsymbol{w}_i] = \boldsymbol{0}, \quad \mathbb{E}[\boldsymbol{w}_i \boldsymbol{w}_i^\top] = \boldsymbol{I}_d, \tag{3.3}$$

with $i = 1, \dots, p$, where the elements $w_{i,1}, \dots, w_{i,d}$ are themselves i.i.d. Then the resulting approximate kernel (3.2) is given by

$$\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{p} \boldsymbol{w}_i^\top \boldsymbol{x} \boldsymbol{w}_i^\top \boldsymbol{y}. \tag{3.4}$$

The variances for the case $D > 1$ can be simply obtained by dividing the variance for $D = 1$ by $D$, since the approximate kernel (3.2) for $D > 1$ is the average of $D$ i.i.d. copies of the approximate kernel (3.4) for $D = 1$.

Since the polynomial sketches discussed here are unbiased, the resulting variance of the approximate kernel $\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y})$ in (3.2) is the *mean-square error* with respect to the true polynomial kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}, \boldsymbol{y})^p$, thus serving as a quality metric:

$$\mathbb{V}[\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y})] := \mathbb{E}[(\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) - \mathbb{E}[\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y})])^2] = \mathbb{E}[(\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) - k(\boldsymbol{x}, \boldsymbol{y}))^2],$$

where the expectation is with respect to the weight vectors $\boldsymbol{w}_i$ for $i = 1, \ldots, p$. Thus, the variance analysis also enables understanding how the distribution of the weight vectors affects the quality of the kernel approximation.

Theorem 3.1.1 below provides a closed form expression of the variance of the approximate kernel and its lower bound. To the best of our knowledge, this result is a novel contribution to the literature. It is useful in understanding how the variance depends on the distribution of the weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_p$ and the properties of input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. Since this result is a special case of a more general result presented later in Theorem 3.3.1 regarding complex polynomial sketches, we omit its proof.

**Theorem 3.1.1.** *Let $\boldsymbol{x} := (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$ and $\boldsymbol{y} := (y_1, \ldots, y_d)^\top \in \mathbb{R}^d$ be any input vectors. Let $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_p \in \mathbb{R}^d$ be i.i.d. random vectors satisfying (3.3), such that elements $w_{i1}, \ldots, w_{id}$ of each vector $\boldsymbol{w}_i = (w_{i1}, \ldots, w_{id})^\top$ are themselves i.i.d. Let $\boldsymbol{w} = (w_1, \ldots, w_d)^\top \in \mathbb{R}^d$ be a random vector independently and identically distributed as $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_p$. Then, for the variance of the approximate kernel (3.4), we have*

$$\mathbb{V}\left[\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y})\right] = \left(\sum_{k=1}^{d} \mathbb{E}[w_k^4] x_k^2 y_k^2 + \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 - 3\sum_{k=1}^{d} x_k^2 y_k^2 + 2(\boldsymbol{x}^\top \boldsymbol{y})^2\right)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \tag{3.5}$$

$$\geq \left(\|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + 2\left[(\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{k=1}^{d} x_k^2 y_k^2\right]\right)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \tag{3.6}$$

Eq. (3.5) of Theorem 3.1.1 shows that the distribution of the i.i.d. weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_p$ affects the variance of the approximate kernel only through the 4-th moments $\mathbb{E}[w_k^4]$, provided that the distribution satisfies the required conditions in Theorem 3.1.1. In particular, the smaller these 4-th moments are, the smaller the variance becomes. This observation enables discussing the optimal choice for the distribution of the weight vectors. The lower bound in Eq. (3.6) is the lowest possible variance, since we have $\mathbb{E}[w_k^4] \geq (\mathbb{E}[w_k^2])^2 = 1$ by Jensen's inequality, where $\mathbb{E}[w_k^2] = 1$ follows from Eq. (3.3).

Let us discuss two choices for the distribution of the weight vectors in Eq. (3.3): Rademacher and Gaussian distributions. For the Rademacher distribution, we have $\mathbb{E}[w_k^4] = 0.5 \cdot 1^4 + 0.5 \cdot (-1)^4 = 1$. Therefore, the variance in Eq. (3.5) simplifies to:

$$\textbf{(Var. Rademacher)} \quad \left(\|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + 2\left[(\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{k=1}^{d} x_k^2 y_k^2\right]\right)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p}, \tag{3.7}$$

which is equal to the lower bound in Eq. (3.6). Therefore, Theorem 3.1.1 shows the optimality of the Rademacher distribution for the approximate kernel (3.4).

As an alternative, one can consider the standard Gaussian distribution, that is, $w_k \sim \mathcal{N}(0,1)$. Then we have $\mathbb{E}[w_k^4] = 3$ (the fourth moment of a standard Gaussian random variable), and the variance in Eq. (3.5) simplifies to:

$$\textbf{(Var. Gaussian)} \quad \big(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + 2(\boldsymbol{x}^\top\boldsymbol{y})^2\big)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}. \tag{3.8}$$

This is larger than the variance of the Rademacher sketch in Eq. (3.7). Thus, in terms of the variance of the approximate kernel, the Gaussian sketch is less efficient than the Rademacher sketch. These arguments suggest that the Rademacher sketch should be a preferred choice. This is in contrast to (Hamid et al., 2014, Lemma 3.1) who obtain better statistical guarantees for Gaussian sketches than the ones for Rademacher sketches derived in (Kar and Karnick, 2012). This is most likely because the bound by Hamid et al. (2014) exploits upper bounds on higher order moments and is probably tighter than the one given by (Kar and Karnick, 2012) that does not exploit such information.

## 3.2 Complex-Valued Product-Sketches

We now introduce complex-valued polynomial sketches, one of our novel contributions. We do this by extending the analysis of Choromanski et al. (2017) for linear sketches to polynomial sketches.[1]

As before, without loss of generality, we focus on approximating the homogeneous polynomial kernel $k(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}^\top\boldsymbol{y})^p$ of degree $p \in \mathbb{N}$. Let $D \in \mathbb{N}$. Suppose we generate $p \times D$ *complex-valued* random vectors satisfying

$$\boldsymbol{z}_{i,\ell} \in \mathbb{C}^d \quad \text{satisfying} \quad \mathbb{E}[\boldsymbol{z}_{i,\ell}\overline{\boldsymbol{z}_{i,\ell}}^\top] = \boldsymbol{I}_d, \quad i \in \{1,\dots,p\}, \quad \ell \in \{1,\dots,D\} \tag{3.9}$$

We then define a *complex-valued random feature map* as

$$\Phi_{\mathcal{C}}(\boldsymbol{x}) := \frac{1}{\sqrt{D}}\left[(\prod_{i=1}^{p}\boldsymbol{z}_{i,1}^\top\boldsymbol{x}),\dots,(\prod_{i=1}^{p}\boldsymbol{z}_{i,D}^\top\boldsymbol{x})\right]^\top \in \mathbb{C}^D, \quad \boldsymbol{x} \in \mathbb{R}^d, \tag{3.10}$$

and the resulting approximate kernel as

$$\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y}) := \Phi_{\mathcal{C}}(\boldsymbol{x})^\top\overline{\Phi_{\mathcal{C}}(\boldsymbol{y})} = \frac{1}{D}\sum_{\ell=1}^{D}\prod_{i=1}^{p}(\boldsymbol{z}_{i,\ell}^\top\boldsymbol{x})\overline{(\boldsymbol{z}_{i,\ell}^\top\boldsymbol{y})}, \quad \boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^d. \tag{3.11}$$

Eq. (3.11) is a generalization of the approximate kernel (3.2) with real-valued features, as Eq. (3.2) can be recovered by defining the complex random vectors $\boldsymbol{z}_{i,\ell}$

[1] More specifically, Choromanski et al. (2017) analyze the variance of the *real part* of the approximate complex-valued kernel in Eq. (3.11) for $p = 1$. In contrast, we study Eq. (3.11) with generic $p \in \mathbb{N}$, and analyze the variance of Eq. (3.11) itself, including both the real and imaginary parts.

in Eq. (3.9) as real random vectors $\boldsymbol{w}_{i,\ell}$ in Eq. (3.3); in this case the requirement $\mathbb{E}[\boldsymbol{z}_{i,\ell}\overline{\boldsymbol{z}_{i,\ell}}^\top] = \mathbb{E}[\boldsymbol{w}_{i,\ell}\boldsymbol{w}_{i,\ell}^\top] = \boldsymbol{I}_d$ is satisfied.

For example, complex-valued random vectors $\boldsymbol{z}_{i,\ell}$ satisfying Eq. (3.9) can be generated as follows.

**Example 1.** *Suppose we generate $2\times p\times D$ independent real-valued random vectors*

$$\boldsymbol{v}_{i,\ell},\ \boldsymbol{w}_{i,\ell} \in \mathbb{R}^d \quad \textit{satisfying} \quad \mathbb{E}[\boldsymbol{v}_{i,\ell}] = \mathbb{E}[\boldsymbol{w}_{i,\ell}] = \boldsymbol{0}, \quad \mathbb{E}[\boldsymbol{v}_{i,\ell}\boldsymbol{v}_{i,\ell}^\top] = \mathbb{E}[\boldsymbol{w}_{i,\ell}\boldsymbol{w}_{i,\ell}^\top] = \boldsymbol{I}_d \tag{3.12}$$

*for $i \in \{1,\dots,p\}$, $\ell \in \{1,\dots,D\}$. Then one can define complex-valued random vectors (3.9) as*

$$\boldsymbol{z}_{i,\ell} := \sqrt{\frac{1}{2}}(\boldsymbol{v}_{i,\ell} + \mathrm{i}\boldsymbol{w}_{i,\ell}) \in \mathbb{C}^d, \quad i \in \{1,\dots,p\}, \quad \ell \in \{1,\dots,D\}. \tag{3.13}$$

The following two examples are specific cases of Example 1 and are complex versions of the real-valued Rademacher and Gaussian sketches discussed previously.

**Example 2** (Complex Rademacher Sketch)**.** *In Example 1, suppose that elements of random vectors $\boldsymbol{v}_{i,\ell}$ and $\boldsymbol{w}_{i,\ell}$ are independently sampled from the Rademacher distribution, i.e., sampled uniformly from $\{1,-1\}$. Then the resulting random vectors $\boldsymbol{v}_{i,\ell}$, $\boldsymbol{w}_{i,\ell}$ satisfy the conditions in Eq. (3.12) and thus the complex random vectors in Eq. (3.13) satisfy the condition Eq. (3.9).*

**Example 3** (Complex Gaussian Sketch)**.** *In Example 1, suppose that elements of random vectors $\boldsymbol{v}_{i,\ell}$ and $\boldsymbol{w}_{i,\ell}$ are independently sampled from the standard Gaussian distribution, $\mathcal{N}(0,1)$. Then the resulting random vectors $\boldsymbol{v}_{i,\ell}$, $\boldsymbol{w}_{i,\ell}$ satisfy the conditions in Eq. (3.12) and thus the complex random vectors in Eq. (3.13) satisfy the condition Eq. (3.9).*

**Example 4.** *Suppose the elements of each random vector $\boldsymbol{z}_{i,\ell} \in \mathbb{C}^d$ are independently sampled from the uniform distribution on $\{1,-1,\mathrm{i},-\mathrm{i}\}$. Then the requirement in Eq. (3.9) is satisfied.*

Example 4 is essentially identical to the complex Rademacher sketch in Example 2, in that each element of $\boldsymbol{z}_{i,\ell}$ in Example 4 can be obtained by multiplying $e^{\mathrm{i}\pi/4}$ to an element of $\boldsymbol{z}_{i,\ell}$ in Example 2, and vice versa. The multiplication by $e^{\mathrm{i}\pi/4}$ is equivalent to rotating an element counter-clockwise by 45 degrees. See Fig. 3.1 for an illustration. One can see that this multiplication by $e^{\mathrm{i}\pi/4}$ does not change the resulting approximate kernel (3.11). In this sense, the constructions of Example 2 and Example 4 are equivalent. However, the sketch in Example 4 gives a computational advantage over Example 2: Since every element of each random vector $\boldsymbol{z}_{i,\ell}$ is either real *or* imaginary, the inner products $\boldsymbol{z}_{i,\ell}^\top\boldsymbol{x}$ in Eq. (3.10) can be computed at the same cost as for real polynomial sketches.

We show in the following proposition that the approximate kernel (3.11) is an unbiased estimator of the polynomial kernel $(\boldsymbol{x}^\top\boldsymbol{y})^p$.

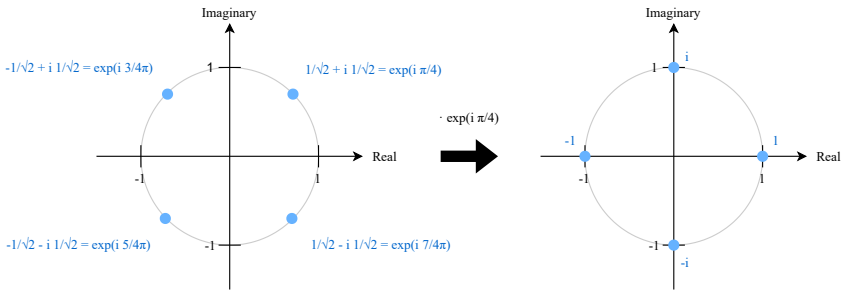



Figure 3.1: Multiplying each element of a random vector $\boldsymbol{z}_{i,\ell}$ in Example 2 by $\exp(\mathrm{i}\frac{\pi}{4})$ corresponds to a counter-clockwise rotation of that element by 45 degrees on the complex plane. The support of the resulting elements is $\{1, -1, \mathrm{i}, -\mathrm{i}\}$ and the construction of Example 4 is obtained.

**Proposition 3.2.1.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ be arbitrary, and $\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})$ be the approximate kernel in (3.11). Then we have*

$$\mathbb{E}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = (\boldsymbol{x}^\top \boldsymbol{y})^p$$

*Proof.* Since Eq. (3.11) is the empirical average of $D$ terms, it is sufficient to show the unbiasedness of each term. To this end, we consider here the case $D = 1$ and drop the index $\ell$. We have

$$\mathbb{E}\left[\prod_{i=1}^{p} \boldsymbol{z}_i^\top \boldsymbol{x} \overline{\boldsymbol{z}_i^\top \boldsymbol{y}}\right] = \prod_{i=1}^{p} \mathbb{E}\left[\boldsymbol{z}_i^\top \boldsymbol{x} \overline{\boldsymbol{z}_i^\top \boldsymbol{y}}\right] = \prod_{i=1}^{p} \boldsymbol{x}^\top \mathbb{E}\left[\boldsymbol{z}_i \overline{\boldsymbol{z}_i}^\top\right] \boldsymbol{y} = (\boldsymbol{x}^\top \boldsymbol{y})^p.$$

where we used Eq. (3.9) in the last identity. ∎

## 3.3 Variance of Complex-Valued Product-Sketches

We now study the variance of the approximate kernel (3.11) with the complex-valued random feature map (3.10). As for the real-valued case, we consider the case $D = 1$ and drop the index $\ell$:

$$\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{p} \boldsymbol{z}_i^\top \boldsymbol{x} \overline{\boldsymbol{z}_i^\top \boldsymbol{y}}. \tag{3.14}$$

The variance of the case $D > 1$ can be obtained by dividing the variance of Eq. (3.14) by $D$, since the approximate kernel (3.11) is the average of $D$ i.i.d. copies of Eq. (3.14). We denote by $z_{i,k}$ the $k$-th element of $\boldsymbol{z}_i$.

Note that the variance of a complex random variable $Z \in \mathbb{C}$ is defined by

$$\mathbb{V}[Z] := \mathbb{E}[|Z - \mathbb{E}[Z]|^2] = \mathbb{E}[(Z - \mathbb{E}[Z])\overline{(Z - \mathbb{E}[Z])}] = \mathbb{E}[|Z|^2] - |\mathbb{E}[Z]|^2$$

Theorem 3.3.1 below characterizes the variance in terms of the input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and the distribution of the complex weight vectors (3.9). The proof is given in Appendix B.1.1.

**Theorem 3.3.1.** *Let $\boldsymbol{x} := (x_1, \dots, x_d)^\top \in \mathbb{R}^d$ and $\boldsymbol{y} := (y_1, \dots, y_d)^\top \in \mathbb{R}^d$ be any input vectors. Let $\boldsymbol{z}_1, \dots, \boldsymbol{z}_p \in \mathbb{C}^d$ be i.i.d. random vectors satisfying (3.9), such that elements $z_{i1}, \dots, z_{id}$ of each vector $\boldsymbol{z}_i = (z_{i1}, \dots, z_{id})^\top$ are themselves i.i.d. Let $\boldsymbol{z} = (z_1, \dots, z_d)^\top \in \mathbb{C}^d$ be a random vector independently and identically distributed as $\boldsymbol{z}_1, \dots, \boldsymbol{z}_p$, and write $z_k = a_k + \mathrm{i}b_k$ with $a_k, b_k \in \mathbb{R}$. Suppose*

$$\mathbb{E}[a_k b_k] = 0, \quad \mathbb{E}[a_k^2] = q, \quad \mathbb{E}[b_k^2] = 1 - q \quad \text{where} \quad 0 \le q \le 1. \tag{3.15}$$

*Then, for the approximate kernel (3.14), we have*

$$\begin{aligned}\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = \Bigg( &\sum_{k=1}^{d} \mathbb{E}[|z_k|^4] x_k^2 y_k^2 + \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 - \sum_{k=1}^{d} x_k^2 y_k^2 \\ &+ \left((2q-1)^2 + 1\right) \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{k=1}^{d} x_k^2 y_k^2\right) \Bigg)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \qquad (3.16) \\ \ge \Bigg( &\|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + \left((2q-1)^2 + 1\right) \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{k=1}^{d} x_k^2 y_k^2\right) \Bigg)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p}. \qquad (3.17)\end{aligned}$$

Theorem 3.3.1 is a generalization of Theorem 3.1.1 as it applies to a spectrum of complex polynomial sketches in terms of $q$, where the case $q = 1$ recovers Theorem 3.1.1 for real-valued polynomial sketches. The key condition in Theorem 3.3.1 is Eq. (3.15),[2] where the constant $q$ is the average length of the real part $a_k$ of each random element $z_k = a_k + \mathrm{i}b_k$. Note that $\mathbb{E}[b_k^2] = 1 - q$ follows from $\mathbb{E}[a_k^2] = q$ since $1 = \mathbb{E}[|z_k|^2] = a_k^2 + b_k^2$. Eq. (3.15) is satisfied for Examples 2, 3 and 4 with $q = 1/2$ and for the real-valued Rademacher and Gaussian sketches with $q = 1$. If $z_k$ is sampled uniformly from $\{\mathrm{i}, -\mathrm{i}\}$, which is eligible as it satisfies Eq. (3.9), then $q = 0$. If $z_k$ is sampled uniformly from $\{1, -1\}$ with probability $q$ and from $\{\mathrm{i}, -\mathrm{i}\}$ with probability $1 - q$, then Eq. (3.15) is satisfied with this $q$.

Eq. (3.17) is the smallest possible variance attainable by complex polynomial sketches satisfying the conditions in Theorem 3.3.1. For $q = 1/2$, this lower bound is attained by the complex Rademacher sketch (Example 2) and its equivalent construction (Example 4), for which we have $\mathbb{E}[|z_k|^4] = 1$:

$$\textbf{(Comp. Rademacher)} \quad \left( \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{k=1}^{d} x_k^2 y_k^2 \right)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \tag{3.18}$$

On the other hand, for the complex Gaussian sketch (Example 3) we have $\mathbb{E}[|z_k|^4] = \mathbb{E}[(a_k^2 + b_k^2)^2] = 2$, and the variance is given by

$$\textbf{(Comp. Gaussian)} \quad \left(\|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top \boldsymbol{y})^2\right)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \tag{3.19}$$

[2] Eq. (3.15) implies that $z_k$ is a *proper* complex random variable (Neeser and Massey, 1993).

**Comparing the Real and Complex Polynomial Sketches.** Let us now compare the variances of real ($q = 1$) and complex ($q \neq 1$) polynomial sketches. First, it is easy to see that the variance of the complex Gaussian polynomial sketch (Eq. (3.19)) is upper-bounded by the variance of the real Gaussian sketch (Eq. (3.8)). For the lower bound in Eq. (3.17), which is attained by the Rademacher sketches, a more detailed analysis is needed. To this end, consider the term that depends on $q$:

$$\left((2q-1)^2+1\right)\left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{k=1}^{d} x_k^2 y_k^2\right)$$

The variance in Eq. (3.16) is a monotonically increasing function of this term. Suppose

$$(\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{k=1}^{d} x_k^2 y_k^2 = \sum_{i=1}^{d} \sum_{\substack{j=1 \\ j\neq i}}^{d} x_i x_j y_i y_j \geq 0 \tag{3.20}$$

Then $q = 1/2$ (e.g., complex sketches in Examples 2, 3 and 4) makes the term the smallest, while $q = 1$ and $q = 0$ (purely real and imaginary polynomial sketches) makes it the largest. In other words, for input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ satisfying Eq. (3.20), complex-valued sketches with $q = 1/2$ result in a lower variance than the real-valued counterparts with $q = 1$. On the other hand, if Eq. (3.20) does not hold, real-valued sketches result in a lower variance than the complex-valued counterparts.

Therefore, whether complex-valued Rademacher sketches ($q = 1/2$) yield a lower variance than real-valued Rademacher sketches ($q = 1$) depends on whether Eq. (3.20) holds. For example, Eq. (3.20) holds true if input vectors $\boldsymbol{x} = (x_1, \ldots, x_d)^\top$ and $\boldsymbol{y} = (y_1, \ldots, y_d)^\top$ are *nonnegative*: $x_1, ..., x_d \geq 0$ and $y_1, \ldots, y_d \geq 0$. Nonnegative input vectors are ubiquitous in real-world applications, e.g., where each input feature represents the amount of a certain quantity, where input vectors are given by bag-of-words representations, one-hot encoding (categorical data), or min-max feature scaling, and where they are outputs of a ReLU neural network[3]. For such applications with nonnegative input vectors, complex-valued polynomial sketches always yield a smaller variance than the real-valued counterparts.

## 3.4 Probabilistic Error Bounds for Rademacher Sketches

We present here probabilistic error bounds for the approximate kernel in Eq. (3.11) in terms of the number $D$ of random features, using the variance formula obtained in the previous section and focusing on Rademacher sketches. The proof of the following result is given in Appendix B.1.2.

[3] c.f. DeepFried Convnets (Yang et al., 2015) and fine-grained image recognition (Gao et al., 2016).

**Theorem 3.4.1.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ be arbitrary input vectors. For $0 \leq q \leq 1$, consider a polynomial sketch in Theorem 3.3.1 such that $\mathbb{E}[|z_k|^4] = 1$ and thus attains the variance in Eq. (3.17). Define a constant $\sigma^2 \geq 0$ by*

$$\sigma^2 := \frac{1}{\|\boldsymbol{x}\|^{2p}\|\boldsymbol{y}\|^{2p}} \left[ \left( \|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + \left((2q-1)^2+1\right)\left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{k=1}^{d} x_k^2 y_k^2\right) \right)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \right]$$

*Let $\epsilon, \delta > 0$ be arbitrary, and $D \in \mathbb{N}$ be such that*

$$D \geq 2\left(\frac{2}{3\epsilon} + \frac{\sigma^2}{\epsilon^2}\right)\log\left(\frac{2}{\delta}\right). \tag{3.21}$$

*Then, for the approximate kernel $\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})$ in Eq. (3.11), we have*

$$\Pr\left[\left|\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}) - (\boldsymbol{x}^\top \boldsymbol{y})^p\right| \leq \epsilon \|\boldsymbol{x}\|_1^p \|\boldsymbol{y}\|_1^p\right] \geq 1 - \delta.$$

Eq. (3.21) shows that the required number $D$ of random features to achieve the relative accuracy of $\varepsilon$ (where the "relative" is with respect to $\|\boldsymbol{x}\|_1^p \|\boldsymbol{y}\|_1^p$) with probability at least $1 - \delta$. For small $\epsilon$, the second term $\sigma^2/\epsilon^2$ dominates the first term $2/(3\epsilon)$. This second term depends on $\sigma^2$, which is a scaled version of the variance in Eq. (3.17) of the approximate kernel for $D = 1$. Thus, if the variance in Eq. (3.17) is smaller (resp. larger), one needs a smaller (resp. larger) number of random features to achieve the relative accuracy of $\epsilon$.

Let us now compare the real-valued ($q = 1$) and complex-valued ($q = 1/2$) Rademacher sketches. As discussed earlier, the complex Rademacher sketch has a smaller variance than the real Rademacher sketch when the inequality in Eq. (3.20) holds for the two input vectors $\boldsymbol{x}, \boldsymbol{y}$. In particular, this inequality always holds when the input vectors are nonnegative. Therefore, in this case, the complex Rademacher sketch requires a smaller number of random features than the real Rademacher sketch to achieve a given accuracy.

When the inequality in Eq. (3.20) holds, the advantage of the complex Rademacher sketch becomes more significant for larger $p$. We illustrate this in Fig. 3.2, where $\boldsymbol{x} = \boldsymbol{y} = (1, \ldots, 1)^\top/\sqrt{d} \in \mathbb{R}^d$. In this case, we have $\sigma^2 = (2 - 1/d)^p - 1$ for the complex Rademacher sketch ($q = 1/2$), while we have $\sigma^2 = (3-2/d)^p - 1$ for the real Rademacher sketch ($q = 1$). For larger $p$, the value of $\sigma^2$ for the real Rademacher sketch becomes relatively larger than that for the complex Rademacher sketch, implying that the complex Rademacher sketch is more efficient. Fig. 3.2 empirically supports this observation.

## 3.5 Structured Product-Sketches

We study here *structured* polynomial sketches and their extensions with complex features. In Section 3.1, we studied polynomial sketches in Eq. (3.4) (or Eq. (3.10)

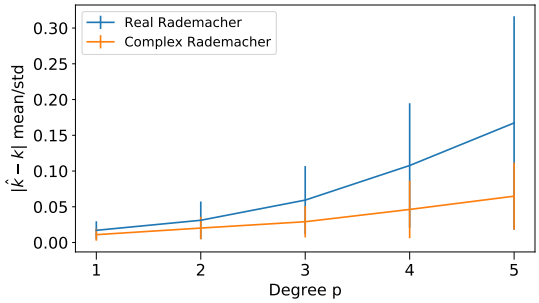



Figure 3.2: This plot shows the mean absolute error $\mathbb{E}[|\hat{k}(\boldsymbol{x}, \boldsymbol{y}) - (\boldsymbol{x}^\top \boldsymbol{y})^p|]$ for different values of the degree $p$, where the mean is taken over 100 independent constructions of the approximate kernel $\hat{k}(\boldsymbol{x}, \boldsymbol{y})$, for both the real and complex Rademacher sketches. The number of random features is $D = 5,000$ and the dimensionality of input vectors is $d = 1,000$. The input vectors are $\boldsymbol{x} = \boldsymbol{y} = (1, \ldots, 1)^\top / \sqrt{d} \in \mathbb{R}^d$.

for complex extensions), where the $p \times D$ random vectors $\boldsymbol{w}_{i,\ell} \in \mathbb{R}^d$ $(i = 1, \ldots, p, \ell = 1, \ldots, D)$ are generated in an i.i.d. manner. By putting a structural constraint on these vectors, one can construct more efficient random features with a lower variance. Moreover, such a structural constraint leads to a computational advantage, as the imposed structure may be used for implementing an efficient algorithm for fast matrix multiplication.

We develop two novel structured polynomial sketches that are closely related to *TensorSRHT* (Ahle et al., 2020) introduced in Chapter 2. Just like the original version by Ahle et al. (2020), our sketches apply the Subsampled Randomized Hadamard Transform (SRHT) to an input $\boldsymbol{x} \in \mathbb{R}^d$ in order to then form an implicit sketch of $\boldsymbol{x}^{(p)}$. The key difference is that our sketches are embedded into the Product-Sketch framework and permit us to derive closed-form variances for them. Currently there are only upper bounds on the variances for structured polynomial sketches like TensorSRHT and TensorSketch available in the literature, which is why statistical advantages have only been shown empirically up until now. Our contribution is to theoretically characterize statistical advantages over non-structured polynomial sketches by means of closed-form variance derivations and comparisons. Since our two proposed sketches are closely related to TensorSRHT by Ahle et al. (2020), we will stick to the term TensorSRHT and add further precisions when necessary.

Our second main contribution in this section is to develop a complex extension of the aforementioned structured sketches and to study their variance advantage empirically. In Section 3.5.1, we introduce our version of TensorSRHT with real features, and present its extension using complex features in Section 3.5.2. We then make a comparison between the real and complex versions in Section 3.5.3.

### 3.5.1 Our Version of TensorSRHT

Recall from Chapter 2 that TensorSRHT by Ahle et al. (2020) is defined as

$$\begin{aligned}\Phi(\boldsymbol{x}) &= \boldsymbol{P}_\pi(\boldsymbol{H}_d\boldsymbol{D}_1\boldsymbol{x} \otimes \cdots \otimes \boldsymbol{H}_d\boldsymbol{D}_p\boldsymbol{x})/\sqrt{D} \\ &= \boldsymbol{P}_\pi\boldsymbol{H}_{d^p}(\boldsymbol{D}_1 \otimes \cdots \otimes \boldsymbol{D}_p)\boldsymbol{x}^{(p)}/\sqrt{D},\end{aligned} \tag{3.22}$$

where $\boldsymbol{P}_\pi \in \{0,1\}^{D\times d^p}$ is a sampling matrix, $\boldsymbol{H}_n$ is the unnormalized Hadamard matrix of size $\{0,1\}^{n\times n}$ for some $n = 2^m, m \in \mathbb{N}$ and $\{\boldsymbol{D}_i\}_{i=1}^p$ are diagonal Rademacher matrices of size $\mathbb{R}^{d\times d}$.

By looking at Eq. (3.22), one can see that TensorSRHT is constructed by forming $D$ distinct combinations of the elements of $\{\boldsymbol{H}\boldsymbol{D}_i\boldsymbol{x}\}_{i=1}^p$. More precisely, we sample the index tuples $\{(i_{1,\ell},\dots,i_{p,\ell})\}_{\ell=1}^D$ without replacement, where $i_{1,\ell},\dots,i_{p,\ell}$ are uniformly sampled from $\{1,\dots,d\}$. Each element $\{\Phi(\boldsymbol{x})_\ell\}_{\ell=1}^D$ is then computed as

$$\Phi(\boldsymbol{x})_\ell = (\boldsymbol{H}_d\boldsymbol{D}_1\boldsymbol{x})_{i_{1,\ell}} \cdots (\boldsymbol{H}_d\boldsymbol{D}_p\boldsymbol{x})_{i_{p,\ell}}/\sqrt{D}. \tag{3.23}$$

In the following, we propose two modifications of TensorSRHT that can be represented in terms of an equivalent Product-Sketch (3.4). The advantage of our proposed sketches is that we can compute their variances in closed form making it possible to study the statistical advantages of our TensorSRHT modifications compared to non-structured Product-Sketches.

We call our modified TensorSRHT sketches *stacked TensorSRHT* and *upsampled TensorSRHT*, respectively. We developed stacked TensorSRHT in our earlier work (Wacker et al., 2022a) since we could easily derive a convex surrogate function of its variance over $D$, as it is needed for our optimized Maclaurin method that we propose in Chapter 5. In our later work (Wacker et al., 2022b), we developed the upsampled TensorSRHT that reduces the time complexity from $\mathcal{O}(p(D\log d + D))$ of stacked TensorSRHT to $\mathcal{O}(p(d\log d + D))$ to project a single data point, and is thus faster when $D > d$. Upsampled TensorSRHT has the same time complexity as TensorSRHT by Ahle et al. (2020) that can be considered the state-of-the-art in terms of computation time. We start by defining stacked TensorSRHT and introduce upsampled TensorSRHT afterwards.

#### Stacked TensorSRHT

To simplify the definition of stacked TensorSRHT, we distinguish two cases for $D$, namely $D \leq d$ and $D > d$. We now cover the first case.

**Case $D \leq d$.** For now we assume that the number $D$ of random features is less or equal to the dimensionality $d$ of the input vectors: $D \leq d$. For $i = 1,\dots,p$, define $\boldsymbol{w}_i \in \mathbb{R}^d$ as a random vector whose elements are i.i.d. Rademacher random variables:

$$\boldsymbol{w}_i := (w_{i,1},\dots,w_{i,d})^\top \in \mathbb{R}^d, \quad w_{i,j} \overset{i.i.d.}{\sim} \text{unif}(\{1,-1\}) \quad (j = 1,\dots,d)$$

Consider a random permutation of the indices $\pi_i : \{1, \dots, d\} \to \{1, \dots, d\}$ for every $i = 1, \dots, p$, and let

$$\pi_i(1), \dots, \pi_i(d)$$

be the permuted indices. For $i = 1, \dots, p$ and $\ell = 1, \dots, D$, we then define a random vector $\boldsymbol{s}_{i,\ell} \in \mathbb{R}^d$ as the Hadamard product (i.e., element-wise product) of the Rademacher vector $\boldsymbol{w}_i$ and the permuted column $\boldsymbol{h}_{\pi_i(\ell)}$ of the Hadamard matrix:

$$\boldsymbol{s}_{i,\ell} := \boldsymbol{w}_i \circ \boldsymbol{h}_{\pi_i(\ell)} = (w_{i,1} h_{\pi_i(\ell),1}, \dots, w_{i,d} h_{\pi_i(\ell),d})^\top \in \mathbb{R}^d, \tag{3.24}$$

where $h_{\pi_i(\ell),j}$ denotes the $j$-th element of $\boldsymbol{h}_{\pi_i(\ell)}$.

Because of the orthogonality of the columns $\boldsymbol{h}_1, \dots, \boldsymbol{h}_d$ of the Hadamard matrix $\boldsymbol{H}_d$, the random weight vectors $\boldsymbol{s}_{i,1}, \dots, \boldsymbol{s}_{i,d}$ are orthogonal to each other almost surely: for $\ell \neq m$, we have

$$\boldsymbol{s}_{i,\ell}^\top \boldsymbol{s}_{i,m} = (\boldsymbol{w}_i \circ \boldsymbol{h}_{\pi_i(\ell)})^\top (\boldsymbol{w}_i \circ \boldsymbol{h}_{\pi_i(m)}) = \sum_{j=1}^{d} w_{i,j}^2 h_{\pi_i(\ell),j} h_{\pi_i(m),j} = h_{\pi_i(\ell)}^\top h_{\pi_i(m)} = 0.$$

Note also that, given the permutation $\pi_i(1), \dots, \pi_i(d)$, the elements of each random vector $\boldsymbol{s}_{i,\ell}$ in (3.24) are i.i.d. Rademacher variables.

Finally, we define a random feature map $\Phi_{\mathcal{R}}(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}^D$ for the case $D \leq d$ as

$$\Phi_{\mathcal{R}}(\boldsymbol{x}) := \frac{1}{\sqrt{D}} \left[ (\prod_{i=1}^{p} \boldsymbol{s}_{i,1}^\top \boldsymbol{x}), \dots, (\prod_{i=1}^{p} \boldsymbol{s}_{i,D}^\top \boldsymbol{x}) \right]^\top \in \mathbb{R}^D, \tag{3.25}$$

which defines an approximate kernel as

$$\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) := \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y}) = \frac{1}{D} \sum_{\ell=1}^{D} \Phi_{\mathcal{R}}(\boldsymbol{x})_\ell \Phi_{\mathcal{R}}(\boldsymbol{y})_\ell$$

where $\Phi_{\mathcal{R}}(\cdot)_\ell$ denotes the $\ell$-th element of $\Phi_{\mathcal{R}}(\cdot)$.

The orthogonality of the weight vectors in Eq. (3.24) leads to *negative covariances* between the terms $\Phi_{\mathcal{R}}(\boldsymbol{x})_\ell \Phi_{\mathcal{R}}(\boldsymbol{y})_\ell$ and $\Phi_{\mathcal{R}}(\boldsymbol{x})_m \Phi_{\mathcal{R}}(\boldsymbol{y})_m$ with distinct indices $\ell \neq m$ in the approximate kernel. These negative covariances decrease the overall variance of the approximate kernel, as we will show later in Theorem 3.5.2.

For $D \leq d$, our version of TensorSRHT (3.25) and the one by Ahle et al. (2020) in Eq. (3.22) are almost equivalent. To see this, let $(\boldsymbol{s}_{i,1}, \dots, \boldsymbol{s}_{i,d})^\top \boldsymbol{x} = \boldsymbol{P}_{\pi_i} \boldsymbol{H}_d \boldsymbol{D}_i \boldsymbol{x}$ with $\boldsymbol{D}_i := \mathrm{diag}(\boldsymbol{w}_i)$ and $\boldsymbol{P}_{\pi_i} := (\boldsymbol{e}_{\pi_i(1)}, \dots, \boldsymbol{e}_{\pi_i(d)})^\top \in \mathbb{R}^{d \times d}$, where $\boldsymbol{e}_{\pi_i(\ell)} \in \mathbb{R}^d$ is a vector whose $\pi_i(\ell)$-th element is 1 and other elements are 0 ($\ell = 1, \dots, d$). $\boldsymbol{P}_{\pi_i}$ has the role of shuffling the elements of $\boldsymbol{H}_d \boldsymbol{D}_i \boldsymbol{x}$ at random. As a result, the $\ell$-th element of $\Phi_{\mathcal{R}}(\boldsymbol{x})$ (3.25) can be expressed as

$$\Phi_{\mathcal{R}}(\boldsymbol{x})_\ell = (\boldsymbol{H}_d \boldsymbol{D}_1 \boldsymbol{x})_{\pi_1(\ell)} \cdots (\boldsymbol{H}_d \boldsymbol{D}_p \boldsymbol{x})_{\pi_p(\ell)} / \sqrt{D}. \tag{3.26}$$

Each permutation index $\{\pi_i(\ell)\}_{\ell=1}^D$ is sampled from $\{1,\dots,d\}$ without replacement for every $i = 1,\dots,p$ due to the shuffling operation. From this it follows that the tuples $\{(\pi_1(\ell),\dots,\pi_p(\ell))\}_{\ell=1}^D$ are sampled without replacement from $\{1,\dots,d\}^p$. However, it is not possible to obtain two tuples that have equal indices $\pi_i(\ell) = \pi_i(\ell')$ for some $\ell \neq \ell'$ and $i \in 1,\dots,p$, which shows a minor difference between our version of TensorSRHT (3.26) and TensorSRHT by Ahle et al. (2020) (3.23).

**Case $D > d$.** We now explain how to draw $D > d$ random feature samples, which requires an extension of the sketch in Eq. (3.25) described so far. In the case of stacked TensorSRHT, we independently generate the feature map in Eq. (3.25) a total number of $B := \lceil \frac{D}{d} \rceil$ times and concatenate the resulting $B$ vectors to obtain a $Bd$-dimensional feature map, and then discard the redundant last $Bd - D$ components of it to obtain a $D$-dimensional feature map. In this way, we can obtain a $D$-dimensional feature map for arbitrary $D \in \mathbb{N}$, which we can still write as

$$\Phi_{\mathcal{R}}(\boldsymbol{x}) := \frac{1}{\sqrt{D}} \left[ (\prod_{i=1}^{p} \boldsymbol{s}_{i,1}^\top \boldsymbol{x}), \dots, (\prod_{i=1}^{p} \boldsymbol{s}_{i,D}^\top \boldsymbol{x}) \right]^\top \in \mathbb{R}^D. \tag{3.27}$$

as in Eq. (3.25), but with $\{\boldsymbol{s}_{i,\ell}\}_{\ell=1}^D$ corresponding to our stacked construction. The entire procedure for constructing the structured polynomial sketch in Eq. (3.27) is outlined in Algorithm 2, where we also cover the complex-valued case discussed later.

In Algorithm 2, we use the equivalent matrix formulation introduced in Eq. (3.26), since it enables the Fast Walsh-Hadamard transform by employing the associativity, and thus the feature map can be computed much faster. To explain this more precisely, we can compute

$$(\boldsymbol{s}_{i,1}^\top \boldsymbol{x}, \dots, \boldsymbol{s}_{i,d}^\top \boldsymbol{x}) = \boldsymbol{x}^\top (\boldsymbol{s}_{i,1}, \dots, \boldsymbol{s}_{i,d}) = \boldsymbol{x}^\top (\boldsymbol{D}_i \boldsymbol{H}_d \boldsymbol{P}_{\pi_i}) = \left( (\boldsymbol{x}^\top \boldsymbol{D}_i) \boldsymbol{H}_d \right) \boldsymbol{P}_{\pi_i}$$

by 1) first computing $\boldsymbol{x}^\top \boldsymbol{D}_i$, 2) then multiplying the Hadamard matrix $\boldsymbol{H}_d$ using the Fast Walsh-Hadamard transform, and 3) lastly multiplying the permutation matrix $\boldsymbol{P}_{\pi_i}$, which is more efficient than first precomputing $\boldsymbol{D}_i \boldsymbol{H}_d \boldsymbol{P}_{\pi_i}$ and then multiplying $\boldsymbol{x}^\top$. In this way, thanks to the Fast Walsh-Hadamard transform, $(\boldsymbol{s}_{i,1}^\top \boldsymbol{x}, \dots, \boldsymbol{s}_{i,d}^\top \boldsymbol{x})$ can be computed in $\mathcal{O}(d \log d)$ instead of $\mathcal{O}(d^2)$ (Fino and Algazi, 1976). The total computational complexity is therefore $\mathcal{O}(p(D \log d + D))$ and the memory requirement is $\mathcal{O}(pD)$, and this is a computational advantage over the non-structured i.i.d. estimator in Eq. (3.4).

The feature map in Eq. (3.27) induces an approximate kernel $\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y})$. The following proposition summarizes that this approximate kernel is unbiased with respect to the target polynomial kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y})^p$. As mentioned earlier, stacked TensorSRHT discussed here is slightly different from TensorSRHT by Ahle et al. (2020). Therefore, this result is novel in its own right. The result follows from Proposition 3.5.8 in the next subsection, so we omit the proof.

**Algorithm 2:** Real and Complex stacked TensorSRHT

**Result:** A feature map $\Phi_{\mathcal{R}/\mathcal{C}}(\boldsymbol{x})$
Pad $\boldsymbol{x}$ with zeros so that $d$ becomes a power of 2 ;
Let $B = \lceil \frac{D}{d} \rceil$ be the number of stacked projection blocks ;
**forall** $b \in \{1, \dots, B\}$ **do**
  **forall** $i \in \{1, \dots, p\}$ **do**
    **Real case** Generate a random vector $\mathbf{w}_i = (w_{i,1}, \dots, w_{i,d})^\top \in \mathbb{R}^d$ as $w_{i,1}, \dots, w_{i,d} \overset{i.i.d}{\sim} \mathrm{unif}(\{1, -1\})$, and define a diagonal matrix $\boldsymbol{D}_i := \mathrm{diag}(\boldsymbol{w}_i) \in \mathbb{R}^{d \times d}$;
    **Complex case** Generate a random vector $\mathbf{z}_i = (z_{i,1}, \dots, z_{i,d})^\top \in \mathbb{C}^d$ as $z_{i,1}, \dots, z_{i,d} \overset{i.i.d}{\sim} \mathrm{unif}(\{1, -1, \mathrm{i}, -\mathrm{i}\})$, and define a diagonal matrix $\boldsymbol{D}_i := \mathrm{diag}(\boldsymbol{w}_i) \in \mathbb{C}^{d \times d}$ ;
    Randomly permute the indices $1, \dots, d$ to $\pi_i(1), \dots, \pi_i(d)$ ;
    Let $\boldsymbol{P}_\pi := (\boldsymbol{e}_{\pi_i(1)}, \dots, \boldsymbol{e}_{\pi_i(d)}) \in \mathbb{R}^{d \times d}$, where $\boldsymbol{e}_{\pi_i(\ell)} \in \mathbb{R}^d$ is a vector whose $\pi_i(\ell)$-th element is 1 and other elements are 0 $(\ell = 1, \dots, d)$ ;
    Let $(\boldsymbol{s}_{i,1}, \dots, \boldsymbol{s}_{i,d}) := \boldsymbol{D}_i \boldsymbol{H}_d \boldsymbol{P}_{\pi_i}$ ;
  **end**
  Compute $\Phi_b(\boldsymbol{x}) := \sqrt{1/D}[(\prod_{i=1}^p \boldsymbol{s}_{i,1}^\top \boldsymbol{x}), \dots, (\prod_{i=1}^p \boldsymbol{s}_{i,d}^\top \boldsymbol{x})]^\top$ ;
**end**
Concatenate the elements of $\Phi_1(\boldsymbol{x}), \dots, \Phi_B(\boldsymbol{x})$ to yield a single projection vector $\Phi_{\mathcal{R}/\mathcal{C}}(\boldsymbol{x})$ and keep the first $D$ entries ;

**Proposition 3.5.1.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ be arbitrary, and $\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y})$ be the approximate kernel with $\Phi_{\mathcal{R}}(\boldsymbol{x}), \Phi_{\mathcal{R}}(\boldsymbol{y}) \in \mathbb{R}^D$ given by the random feature map in Eq. (3.27). Then we have $\mathbb{E}[\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y})] = (\boldsymbol{x}^\top \boldsymbol{y})^p$.*

We next study the variance of the approximate kernel given by stacked TensorSRHT, which is the mean squared error of the approximate kernel since it is unbiased as shown above. The following theorem provides a closed form expression for the variance, whose proof is given for the more general complex case in Appendix B.2.2. It is a novel result and extends Choromanski et al. (2017, Theorem 3.3) to the setting $p > 1$ and $D > d$.

**Theorem 3.5.2** (Variance of Real Stacked TensorSRHT)**.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ be arbitrary, and $\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y})$ be the approximate kernel with $\Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \in \mathbb{R}^D$ given by the random feature map in Eq. (3.27). Then we have*

$$\mathbb{V}\left[\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y})\right] = \underbrace{\frac{V_{\mathrm{Rad}}^{(p)}}{D}}_{(A)} - \underbrace{\frac{c(D,d)}{D^2}\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_{\mathrm{Rad}}^{(1)}}{d-1}\right)^p\right]}_{(B)}, \tag{3.28}$$

*where* $V_{\mathrm{Rad}}^{(p)} \geq 0$ *and* $V_{\mathrm{Rad}}^{(1)} \geq 0$ *are the variances of the real Rademacher sketch with a single feature in Eq. (3.7) with generic* $p \in \mathbb{N}$ *and* $p = 1$*, respectively, and* $c(D, d) \in \mathbb{N}$ *is defined by*

$$c(D, d) := \lfloor D/d \rfloor d(d-1) + \mathrm{mod}(D, d)(\mathrm{mod}(D, d) - 1). \tag{3.29}$$

**Remark 3.5.3.** *The constant* $c(D, d)$ *in Eq. (3.29) is the number of pairs of indices* $\ell, \ell' = 1, \ldots, D$ *with* $\ell \neq \ell'$ *for which the covariance of the weight vectors* $\boldsymbol{s}_{i,\ell}$ *and* $\boldsymbol{s}_{i,\ell'}$ *in Eq. (3.27) is non-zero (see the proof in Appendix B.2.2 for details). If* $D = Bd$ *for some* $B \in \mathbb{N}$*, this constant simplifies to* $c(D, d) = Bd(d-1)$*, and the variance in Eq. (3.28) becomes*

$$\mathbb{V}\left[\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y})\right] = \frac{1}{D} V_{\mathrm{Rad}}^{(p)} - \frac{d-1}{D}\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_{\mathrm{Rad}}^{(1)}}{d-1}\right)^p\right].$$

*An interesting subcase is* $p = 1$*, for which the variance becomes zero. Thus, setting* $D \in \{kd \mid k \in \mathbb{N}\}$ *for* $p = 1$ *is equivalent to using the linear kernel with the original inputs.*

Theorem 3.5.2 enables understanding the condition under which stacked TensorSRHT has a smaller variance than the unstructured Rademacher sketch in Eq. (3.1). Note that the term $(A)$ in Eq. (3.28) is the variance of the approximate kernel with the Rademacher sketch with $D$ features. On the other hand, the term $(B)$ in Eq. (3.28) can be interpreted as the effect of the structured sketch. The term $(B)$ always becomes non-negative when $p$ is odd, and thus the overall variance of stacked TensorSRHT becomes smaller than the Rademacher sketch, as summarized in the following corollary. Thus, when $p$ is odd, stacked TensorSRHT should be preferred over the Rademacher sketch.

**Corollary 3.5.4.** *Let* $p \in \mathbb{N}$ *be odd. Then, for all input vectors* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$*, the variance of the approximate kernel with stacked TensorSRHT in Eq. (3.28) is smaller or equal to the variance of the approximate kernel with the Rademacher sketch:*

$$\frac{V_{\mathrm{Rad}}^{(p)}}{D} - \frac{c(D, d)}{D^2}\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_{\mathrm{Rad}}^{(1)}}{d-1}\right)^p\right] \leq \frac{V_{\mathrm{Rad}}^{(p)}}{D}$$

*Proof.* Since, $V_{\mathrm{Rad}}^{(1)} \geq 0$, we have $(\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{1}{d-1} V_{\mathrm{Rad}}^{(1)} \leq (\boldsymbol{x}^\top \boldsymbol{y})^2$. For odd $p$ this leads to $\left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{1}{d-1} V_{\mathrm{Rad}}^{(1)}\right)^p \leq (\boldsymbol{x}^\top \boldsymbol{y})^{2p}$. The assertion immediately follows. ∎

If $p$ is even, on the other hand, the variance of stacked TensorSRHT can be larger than the Rademacher sketch for certain input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$. For instance, if $\boldsymbol{x}$ and $\boldsymbol{y}$ are orthogonal, i.e., $\boldsymbol{x}^\top \boldsymbol{y} = 0$, then the variance of TensorSRHT in Eq. (3.28) is

$$\text{Eq. (3.28)} = \frac{V_{\mathrm{Rad}}^{(p)}}{D} + \frac{c(D, d)}{D^2}\left(\frac{V_{\mathrm{Rad}}^{(1)}}{d-1}\right)^p \geq \frac{V_{\mathrm{Rad}}^{(p)}}{D}.$$

Therefore, for even $p$, we do not have a theoretical guarantee for the advantage of stacked TensorSRHT over the Rademacher sketch in terms of their variances. In practice, however, stacked TensorSRHT has often a smaller variance than the Rademacher sketch also for even $p$, as demonstrated in our experiments described later. Moreover, stacked TensorSRHT has a computational advantage over the Rademacher sketch, thanks to the fast Walsh-Hadamard transform.

**Remark 3.5.5.** *One can straightforwardly derive a probabilistic error bound for stacked TensorSRHT by using Theorem 3.5.2 and Chebyshev's inequality. However, deriving an exponential tail bound for stacked TensorSRHT is more involved despite our derived closed form variance formula, because different features in the feature map $\Phi_{\mathcal{R}}(\boldsymbol{x})$ in Eq. (3.25) are dependent for stacked TensorSRHT and thus applying Bersnstein's inequality is not straightforward. One can find an exponential tail bound for TensorSRHT in Ahle et al. (2020, Lemma 33 in the longer version), while they analyze a slightly different version of TensorSRHT from ours and their bound is a uniform upper bound that holds for all input vectors simultaneously.*

Our variance formula in Eq. (3.28), which is a novel contribution to the literature, provides a precise characterization of how the variance of the approximate kernel depends on the input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, and shows when stacked TensorSRHT is more advantageous than the Rademacher sketch. Moreover, as the variance formula can be computed in practice, it can be used for designing an objective function for a certain optimization problem, as we do in Chapter 5 for designing a data-driven approach to feature construction.

### Upsampled TensorSRHT

We now briefly discuss our upsampled TensorSRHT modification that is faster than stacked TensorSRHT if $D > d$ and more closely related to the original construction in Ahle et al. (2020) while still providing closed form variances. The disadvantage of upsampled TensorSRHT is that we have not derived a convex surrogate function for its variance, obstructing its use for the optimized Maclaurin method in Chapter 5.

For $D \leq d$, upsampled TensorSRHT and stacked TensorSRHT are equivalent. We will thus directly focus on the case $D > d$. Instead of generating the feature map in Eq. (3.25) a total number of $B = \lceil D/d \rceil$ times independently as for stacked TensorSRHT, we keep the construction in Eq. (3.25) and change only the sampling matrices $\{\boldsymbol{P}_{\pi_i}\}_{i=1}^p$.

For this purpose, we concatenate the index vector $(1, \ldots, d)^\top \in \mathbb{R}^d$ a total number of $B$ times with itself and call the resulting vector $\boldsymbol{p}_{\text{base}} \in \mathbb{R}^{\lceil D/d \rceil d}$. We then shuffle the elements of $\boldsymbol{p}_{\text{base}}$ for each $i \in \{1, \ldots, p\}$ independently at random and call the shuffled vectors $\{\boldsymbol{p}_i\}_{i=1}^p$. Each vector $\boldsymbol{p}_i$ defines a sampling matrix $\boldsymbol{P}_i = (\boldsymbol{e}_{p_{i,1}}, \ldots, \boldsymbol{e}_{p_{i,D}})^\top \in \{0,1\}^{D \times d}$. We keep only the first $D$ elements of every $\boldsymbol{p}_i$ to create $D$ random feature samples. Now we let $(\boldsymbol{s}_{i,1}, \ldots, \boldsymbol{s}_{i,D}) := \boldsymbol{D}_i \boldsymbol{H}_d \boldsymbol{P}_i$.

Using $\{\boldsymbol{s}_{i,1}, \ldots, \boldsymbol{s}_{i,D}\}_{i=1}^{p}$ as part of Eq. (3.25) defines our upsampled TensorSRHT sketch.

The computation of this sketch can be equivalently written as

$$\Phi_{\mathcal{R}}(\boldsymbol{x}) := (\boldsymbol{P}_1 \boldsymbol{H}_d \boldsymbol{D}_1 \boldsymbol{x}) \odot \cdots \odot (\boldsymbol{P}_p \boldsymbol{H}_d \boldsymbol{D}_p \boldsymbol{x})/\sqrt{D}, \tag{3.30}$$

which allows for a computation in $\mathcal{O}(p(d \log d + D))$ time.

The unbiasedness of upsampled TensorSRHT can be shown by following the same arguments as in Proposition 3.5.8. The following theorem provides a closed form expression for the variance, whose proof is given for the more general complex case in Appendix B.2.2.

**Theorem 3.5.6** (Variance of Real Upsampled TensorSRHT)**.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ be arbitrary, and $\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y})$ be the approximate kernel with $\Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \in \mathbb{R}^D$ given by the random feature map in Eq. (3.30). Then we have*

$$\mathbb{V}\left[\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y})\right] = \underbrace{\frac{V_{\mathrm{Rad}}^{(p)}}{D}}_{(A)} - \underbrace{\left(1 - \frac{1}{D}\right)\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_{\mathrm{Rad}}^{(1)}}{\lceil D/d \rceil d - 1}\right)^p\right]}_{(B)}, \tag{3.31}$$

*where $V_{\mathrm{Rad}}^{(p)} \geq 0$ and $V_{\mathrm{Rad}}^{(1)} \geq 0$ are the variances of the real Rademacher sketch with a single feature in Eq. (3.7) with generic $p \in \mathbb{N}$ and $p = 1$, respectively, as before.*

**Remark 3.5.7.** *$c(D, d)$ in Eq. (3.28) has been replaced by $(1 - 1/D)$ in Eq. (3.31) because all weight vectors $\boldsymbol{s}_{i,\ell}$ in Eq. (3.25) have a non-zero covariance for upsampled TensorSRHT. At the same time, $\frac{V_{\mathrm{Rad}}^{(1)}}{d-1}$ is replaced by $\frac{V_{\mathrm{Rad}}^{(1)}}{\lceil D/d \rceil d - 1}$, which counterbalances the aforementioned upscaling from $c(D, d)$ to $(1 - 1/D)$. We observed that upsampled TensorSRHT and stacked TensorSRHT perform similarly well in preliminary experiments although we did not carry out a detailed comparison since we developed both sketches in separate works and for a different purpose.*

### 3.5.2 Complex-Valued TensorSRHT

We present here a generalization of stacked and upsampled TensorSRHT by allowing for complex features. To this end, let $z \in \mathbb{C}$ be a random variable such that (i) $|z| = 1$ almost surely, (ii) $\mathbb{E}[z] = 0$ and (iii) $z$ is symmetric, i.e., the distributions of $z$ and $-z$ are the same. Define then $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_p \in \mathbb{C}^d$ as i.i.d. complex random vectors such that elements of each random vector $\boldsymbol{z}_i$ are i.i.d. realizations of $z$:

$$\boldsymbol{z}_i = (z_{i,1}, \ldots, z_{i,d})^\top \in \mathbb{C}^d, \quad z_{i,j} \overset{i.i.d.}{\sim} P_z \quad (j = 1, \ldots, d), \tag{3.32}$$

where $P_z$ denotes the probability distribution of $z$.

We propose the complex version of stacked TensorSRHT in detail, while the complex version of upsampled TensorSRHT is obtained using the same modifications. Let $\pi : \{1, \dots, d\} \to \{1, \dots, d\}$ be a random permutation of indices $1, \dots, d$. For $i = 1, \dots, p$ and $\ell = 1, \dots, D$, we then define a random vector $\boldsymbol{s}_{i,\ell} \in \mathbb{C}^d$ as the Hadamard product of the random vector $\boldsymbol{z}_i$ in (3.32) and the permuted column $\boldsymbol{h}_{\pi(\ell)}$ of the Hadamard matrix $\boldsymbol{H}_d$:

$$\boldsymbol{s}_{i,\ell} := \boldsymbol{z}_i \circ \boldsymbol{h}_{\pi(\ell)} = (z_{i,1} h_{\pi(\ell),1}, \dots, z_{i,d} h_{\pi(\ell),d})^\top \in \mathbb{C}^d, \tag{3.33}$$

With these weight vectors $\boldsymbol{s}_{i,\ell}$, we define a random feature map exactly in the same way as the feature map in Eq. (3.27) for the real stacked TensorSRHT in Section 3.5.1. We define the resulting feature map $\Phi_{\mathcal{C}} : \mathbb{R}^d \to \mathbb{C}^D$ by

$$\Phi_{\mathcal{C}}(\boldsymbol{x}) := \frac{1}{\sqrt{D}} \left[ (\prod_{i=1}^{p} \boldsymbol{s}_{i,1}^\top \boldsymbol{x}), \dots, (\prod_{i=1}^{p} \boldsymbol{s}_{i,D}^\top \boldsymbol{x}) \right]^\top \in \mathbb{C}^D. \tag{3.34}$$

We call this feature construction *complex stacked TensorSRHT*. Substituting the diagonal real Rademacher matrices $\{\boldsymbol{D}_i\}_{i=1}^p$ in Eq. (3.30) by the matrices $\{\mathrm{diag}(\boldsymbol{z}_i)\}_{i=1}^p$ instead results in the complex version of upsampled TensorSRHT.

Admissible examples of the distribution $P_z$ in Eq. (3.32) include: (1) the uniform distribution on $\{1, -1\}$; (2) the uniform distribution on $\{1, -1, \mathrm{i}, -\mathrm{i}\}$; and (3) the uniform distribution on the unit circle in $\mathbb{C}^d$. The example (1) is where $z$ is a real Rademacher random variable, and in this case the complex TensorSRHT coincides with the real TensorSRHT. Thus, the complex TensorSRHT is a strict generalization of the real TensorSRHT.

We first show that the stacked complex TensorSRHT provides an unbiased approximation of the polynomial kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y})^p$. Since the real TensorSRHT is a special case, its unbiasedness follows from this result.

**Proposition 3.5.8.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ be arbitrary, and $\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{C}}(\boldsymbol{x})^\top \overline{\Phi_{\mathcal{C}}(\boldsymbol{y})}$ be the approximate kernel with $\Phi_{\mathcal{C}}(\boldsymbol{x}), \Phi_{\mathcal{C}}(\boldsymbol{y}) \in \mathbb{C}^D$ given by the random feature map in Eq. (3.34). Then we have $\mathbb{E}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = (\boldsymbol{x}^\top \boldsymbol{y})^p$.*

*Proof.* We first show $\mathbb{E}[\boldsymbol{s}_{i,\ell} \overline{\boldsymbol{s}_{i,\ell}}^\top] = \boldsymbol{I}_d$ for all $i = 1, \dots, p$ and $\ell = 1, \dots, D$. This follows from the fact that, for all $t, u = 1, \dots, d$, we have

$$\mathbb{E}[(\boldsymbol{s}_{i,\ell} \overline{\boldsymbol{s}_{i,\ell}}^\top)_{tu}] = \mathbb{E}[z_{i,t} h_{\pi(\ell),t} \overline{z_{i,u}} h_{\pi(\ell),u}] = \begin{cases} \mathbb{E}[|z_{i,t}|^2] \mathbb{E}[h_{\pi(\ell),t}^2] = 1 & (\text{if } \ t = u), \\ \mathbb{E}[z_{i,t}] \mathbb{E}[\overline{z_{i,u}}] \mathbb{E}[h_{\pi(\ell),t} h_{\pi(\ell),u}] = 0 & (\text{if } \ t \neq u), \end{cases}.$$

Using this, we have

$$\mathbb{E}\left[\Phi_{\mathcal{C}}(\boldsymbol{x})^\top \overline{\Phi_{\mathcal{C}}(\boldsymbol{y})}\right] = \mathbb{E}\left[\frac{1}{D} \sum_{\ell=1}^{D} \prod_{i=1}^{p} \boldsymbol{x}^\top \boldsymbol{s}_{i,\ell} \overline{\boldsymbol{s}_{i,\ell}}^\top \boldsymbol{y}\right] = \frac{1}{D} \sum_{\ell=1}^{D} \prod_{i=1}^{p} \boldsymbol{x}^\top \mathbb{E}[\boldsymbol{s}_{i,\ell} \overline{\boldsymbol{s}_{i,\ell}}^\top] \boldsymbol{y} = (\boldsymbol{x}^\top \boldsymbol{y})^p$$

■

The proof for the unbiasedness of complex upsampled TensorSRHT is analogous.

We now study the variance of the approximate kernel given by complex stacked TensorSRHT in Eq. (3.34) as well as complex upsampled TensorSRHT. To this end, we use the same notation as Theorem 3.3.1 to write the real and imaginary parts of the random variable $z$ as $z = a + \mathrm{i}b$ with real-valued random variables $a, b \in \mathbb{R}$. The proof of the following theorem is provided in Appendix B.2.2.

**Theorem 3.5.9** (Variance of Complex Stacked TensorSRHT)**.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ be arbitrary, and $\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{C}}(\boldsymbol{x})^\top \overline{\Phi_{\mathcal{C}}(\boldsymbol{y})}$ be the approximate kernel with $\Phi_{\mathcal{C}}(\boldsymbol{x}), \Phi_{\mathcal{C}}(\boldsymbol{y}) \in \mathbb{C}^D$ given by the complex random feature map in Eq. (3.34). For the random variable $z$ defining Eq. (3.32), write $z = a + \mathrm{i}b$ with $a, b \in \mathbb{R}$, and suppose that*

$$\mathbb{E}[ab] = 0, \quad \mathbb{E}[a^2] = q, \quad \mathbb{E}[b^2] = 1 - q \quad \textit{where} \quad 0 \leq q \leq 1.$$

*Then we have*

$$\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = \underbrace{\frac{V_q^{(p)}}{D}}_{(A)} - \underbrace{\frac{c(D,d)}{D^2}\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1}\right)^p\right]}_{(B)}, \tag{3.35}$$

*for complex stacked TensorSRHT, where $V_q^{(p)} \geq 0$ and $V_q^{(1)} \geq 0$ are Eq. (3.17) with the considered value of $p$ and $p = 1$, respectively, and $c(D, d) \in \mathbb{N}$ is defined in (3.29).*

*For complex upsampled TensorSRHT, we have*

$$\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = \underbrace{\frac{V_q^{(p)}}{D}}_{(A)} - \underbrace{\left(1 - \frac{1}{D}\right)\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_q^{(1)}}{\lceil D/d \rceil d - 1}\right)^p\right]}_{(B)}. \tag{3.36}$$

Regarding Theorem 3.5.9, we make the following observations.

- The case $q = 1$ recovers Theorem 3.5.2 and Theorem 3.5.6 on the real TensorSRHT, where $z \in \{1, -1\}$ is a Rademacher random variable. The case $q = 1/2$ is the complex TensorSRHT with, for instance, $P_z$ being the uniform distribution on $\{1, -1, \mathrm{i}, -\mathrm{i}\}$ or on the unit circle in $\mathbb{C}$. Other values of $q \in [0, 1]$ can also be considered, but we do not discuss them further.

- The first term $(A)$ in Eq. (3.35) and Eq. (3.36) is the variance of the unstructured polynomial sketch in Eq. (3.10) with $D$ features, since $V_q^{(p)}$ is its variance with a single feature ($D = 1$) in Eq. (3.17). The second term $(B)$ in Eq. (3.35) and Eq. (3.36) is the effect of using the structured sketch. The quantity $V_q^{(1)}$ is the variance of the unstructured sketch in Eq. (3.10) with a single feature in Eq. (3.17) with $p = 1$.

- As for the real case, the variance (3.35) becomes zero when $p = 1$ and $D \in \{kd \mid k \in \mathbb{N}\}$.

As we discussed for the real TensorSRHT in Corollary 3.5.4, Theorem 3.5.9 enables understanding a condition under which the complex TensorSRHT is advantageous over the corresponding unstructured complex sketch in Eq. (3.10). As for the real case, the condition is that the degree $p$ of the polynomial kernel is *odd*, as stated in the following.

**Corollary 3.5.10.** *Let $p \in \mathbb{N}$ be odd. Then, for all input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, the variance of the approximate kernel with the complex stacked TensorSRHT in Eq. (3.35) is smaller or equal to the variance of the approximate kernel with the corresponding unstructured polynomial sketch:*

$$\frac{V_q^{(p)}}{D} - \frac{c(D,d)}{D^2}\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1}\right)^p\right] \leq \frac{V_q^{(p)}}{D}$$

*Proof.* Since, $V_q^{(1)} \geq 0$, we have $(\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{1}{d-1}V_q^{(1)} \leq (\boldsymbol{x}^\top \boldsymbol{y})^2$. For odd $p$ this leads to $\left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{1}{d-1}V_q^{(1)}\right)^p \leq (\boldsymbol{x}^\top \boldsymbol{y})^{2p}$. The assertion immediately follows. ∎

As discussed for the real case, if $p$ is even, the variance of the complex TensorSRHT can be larger than the corresponding unstructured sketch for certain input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ (e.g., when $\boldsymbol{x}^\top \boldsymbol{y} = 0$). Empirically, however, the complex TensorSRHT often has a smaller variance also for even $p$, as we demonstrate later. Moreover, the arguments of Corollary 3.5.10 can also be applied to upsampled TensorSRHT.

### 3.5.3 Comparing the Real and Complex TensorSRHT

Let us now compare the real and complex TensorSRHT. We focus on a comparison of stacked TensorSRHT here although the same arguments can be applied to upsampled TensorSRHT. To make the discussion clearer, suppose that the number of random features satisfies $D = Bd$ for some $B \in \mathbb{N}$, as in Remark 3.5.3. Then the variance formula in Eq. (3.35) simplifies to

$$\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = \underbrace{\frac{V_q^{(p)}}{D}}_{(A)} - \underbrace{\frac{d-1}{D}\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1}\right)^p\right]}_{(B)}. \tag{3.37}$$

Recall that setting $q = 1$ and $q = 1/2$ recover the variances of real and complex TensorSRHT, respectively. Thus, let us compare these two cases. We make the following observations:

- As discussed in Section 3.3, it holds that $V_{1/2}^{(p)} \leq V_1^{(p)}$ and $V_{1/2}^{(1)} \leq V_1^{(1)}$ given that the input vectors $\boldsymbol{x} = (x_1, \ldots, x_d), \boldsymbol{y} = (y_1, \ldots, y_d)$ satisfy the inequality in Eq. (3.20), i.e., $\sum_{i \neq j} x_i x_j y_i y_j \geq 0$, which is satisfied when $\boldsymbol{x}$ and $\boldsymbol{y}$ are non-negative vectors.

- Thus, if Eq. (3.20) is satisfied, the first term $(A)$ becomes smaller for $q = 1/2$ (complex case) than $q = 1$ (real case). On the other hand, if $p$ is odd, the second term $(B)$ becomes smaller for $q = 1/2$ than $q = 1$; thus, the variance reduction (i.e., $-(B)$) is smaller for $q = 1/2$ than $q = 1$.

The above observations suggest that, even when Eq. (3.20) is satisfied, whether the complex TensorSRHT ($q = 1/2$) has a smaller variance than the real TensorSRHT ($q = 1$) depends on the balance between the two terms $(A)$ and $(B)$ and on the properties of the input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$. We have not been able to provide a theoretical characterization of exact situations where the complex TensorSRHT has a smaller variance than the real TensorSRHT.

To complement the lack of a theoretical characterization, we performed experiments to compare the variances of real and complex TensorSRHT, whose results are shown in Fig. 3.3. We evaluated the variance formula in Eq. (3.37) for $q = 1$ (real) and $q = 1/2$ (complex), for 1000 pairs of input vectors $\boldsymbol{x}, \boldsymbol{y}$ randomly sampled from a given dataset (EEG, CIFAR 10 ResNet34 features, MNIST and Gisette). For each pair $\boldsymbol{x}, \boldsymbol{y}$, we computed the ratio of Eq. (3.37) with $q = 1/2$ divided by Eq. (3.37) with $q = 1$, and Fig. 3.3 shows the empirical cumulative distribution function of this ratio for the 4 datasets. In these datasets, the input vectors are nonnegative.

Fig. 3.3 shows that, for 100%, 100%, 97.8%, 100% of the cases of the 4 datasets, respectively, the variance of the complex TensorSRHT is smaller than that of the real TensorSRHT. Moreover, the ratio of the variances tends to be even smaller for a larger value of $p$. These results suggest that the complex TensorSRHT is effective in reducing the variance of the real TensorSRHT, and the variance reduction is more significant for a larger value $p$ of the polynomial degree. We leave a theoretical analysis for explaining this improvement of the complex TensorSRHT for a future work.

So far, we have compared the variances of real and complex Product-Sketches. In the following, we compare complex Rademacher and TensorSRHT sketches against their real analogs in terms of their *downstream task performance* for the task of Gaussian process classification.

### 3.5.4 Wall-Clock Time Comparison of Real and Complex Random Features in GP Classification

In this section, we empirically compare the complex Rademacher and TensorSRHT sketches that we introduced in this chapter against their real analogs, where we measure classification error against wall-clock time for the task of Gaussian process

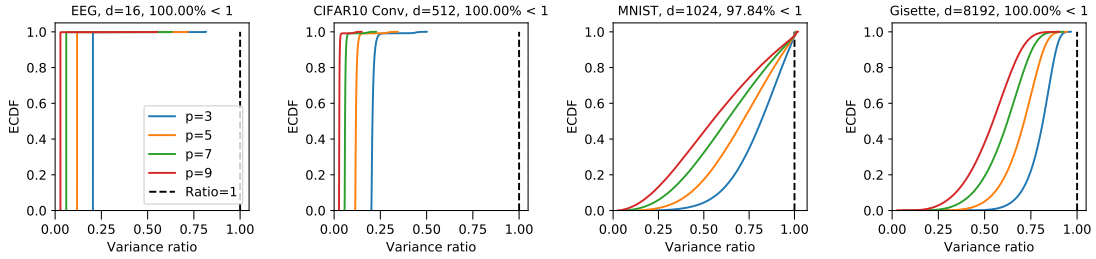



Figure 3.3: Empirical cumulative distribution of pairwise ratios Var(Compl. TensorSRHT) / Var(Real TensorSRHT) on a subsample (1000 samples) of four different datasets (EEG, CIFAR10 ResNet34 features, MNIST, Gisette) with unit-normalized data where $D = d$. The datasets are not zero-centered and therefore entirely positive.

classification. This comparison is relevant because complex-valued random feature maps may incur a higher computational cost for the downstream task. We therefore investigate whether they lead to a better downstream task performance using the *same* computational budget as their real-valued counterparts. For completeness, we explain how to use complex-valued random features in Gaussian process inference and discuss the resulting computational costs in Appendix B.3[4].

We consider Gaussian process classification using the polynomial kernel

$$k(\boldsymbol{x}, \boldsymbol{y}) = \sigma^2 \left( \left(1 - \frac{2}{a^2}\right) + \frac{2}{a^2} \boldsymbol{x}^\top \boldsymbol{y} \right)^p = \sigma^2 \left(1 - \frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{a^2}\right)^p \tag{3.38}$$

with $p \in \mathbb{N}$, $a = 4$, $\sigma^2 > 0$, and $\|\boldsymbol{x}\| = \|\boldsymbol{y}\| = 1$.

**Setting.** We use the Rademacher sketch and stacked TensorSRHT, and their respective complex versions. For each polynomial sketch, we compute the KL divergence (B.23) between the approximate and exact GP posteriors (see Appendix B.3 for details), and record wall-clock time (in seconds) spent on constructing random features and on computing the approximate GP posterior.[5] We use FashionMNIST (Xiao et al., 2017) for this experiment.

**Results.** Fig. 3.4 describes the results. The approximate GPs using complex random features achieve equal or lower KL-divergences than those using real fea-

[4]Specifically, if one uses $D$ complex features, then the inversion of the matrix in Eq. (B.20) requires 4 times as many floating point operations as the case of using $D$ real features. Note that, if one instead uses $2D$ real features, then the inversion of the matrix in Eq. (B.20) requires 8 times as many operations as the case of using $D$ real features. Thus, doubling the number of real features is 2 times more expensive than using complex features. See Appendix B.3.3 for details.

[5]We recorded the time measurements on an NVIDIA P100 GPU and PyTorch version 1.10 with native complex linear algebra support.

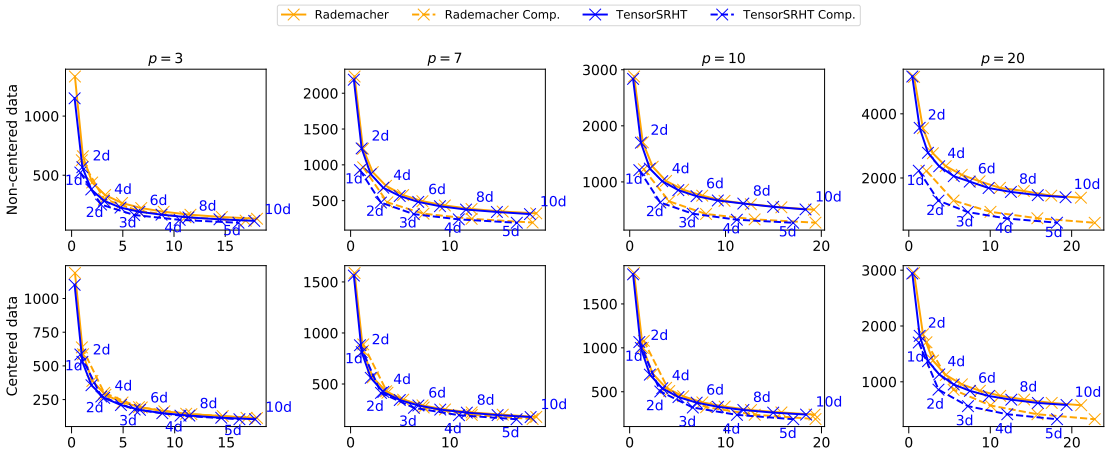



Figure 3.4: Results of the experiments in Section 3.5.4 on wall-clock time comparison of real and complex random features in GP classification on FashionMNIST. In each plot, the vertical axis shows the KL divergence (B.23) between the approximate and the exact GP posteriors for each polynomial sketch, and the horizontal axis is wall-clock time (in seconds) spent on constructing random features and on computing the approximate GP posterior. Each column corresponds to a different degree $p \in \{3, 7, 10, 20\}$ of the polynomial kernel in Eq. (3.38). The top row shows results on the non-centered (thus non-negative) data, and the bottom row to those on the zero-centered data. The number of random features is $D \in \{1d, \ldots, 10d\}$ for real features, and $D \in \{1d, \ldots, 5d\}$ for complex features, as annotated next to the respective measurements in each plot.

tures of the same computation time, for all the cases. In particular, the improvements of complex features are larger for higher polynomial degrees $p$ and for the non-centered (and thus non-negative) data. This observations agrees with the discussion in Section 3.3 on when complex features yield lower variances than real features.

# Chapter 4

# Complex-to-Real Product-Sketches

In the previous chapter, we introduced complex Product-Sketches and analyzed their implications on the variances of the corresponding kernel estimate. Although complex polynomial sketches can yield much lower variances than real-valued analogs, they require the downstream model to handle complex data, which usually leads to an increased computational cost as we show for the example of Gaussian processes in Section B.3.

In this chapter, we show how to transform complex polynomial sketches into real ones that we call *Complex-to-Real (CtR)* polynomial sketches from now on. CtR-Sketches do not require the downstream model to handle complex data and can be used as a drop-in replacement inside any model that makes use of random feature approximations. We show that CtR-sketches maintain the variance reduction properties of complex Product-Sketches, thus leading to an even better downstream performance under a fixed computational budget.

## 4.1 Transforming Complex into Real Sketches

In the following, we show how to convert complex polynomial sketches into real-valued CtR-sketches. Let $\Phi_{\mathcal{C}} : \mathbb{R}^d \to \mathbb{C}^D$ be a complex polynomial sketch and $\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{C}}(\boldsymbol{x})^\top \overline{\Phi_{\mathcal{C}}(\boldsymbol{y})}$ the approximate kernel as defined in Chapter 3, where we make the use of complex projections explicit through the subscript $\mathcal{C}$. We further denote by $\mathrm{Re}\{\cdot\}$ and $\mathrm{Im}\{\cdot\}$ the real and imaginary parts of a complex vector.

$\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})$ is generally complex-valued and can hence be written as

$$\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{Re}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})\} + \mathrm{i} \cdot \mathrm{Im}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})\}.$$

For an unbiased polynomial sketch, we have $\mathbb{E}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = k(\boldsymbol{x}, \boldsymbol{y}) + 0 \cdot \mathrm{i}$. From this it follows that $\mathbb{E}[\mathrm{Re}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})\}] = k(\boldsymbol{x}, \boldsymbol{y})$ and $\mathbb{E}[\mathrm{Im}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})\}] = 0$ through the linearity of the expectation.

We thus define $\hat{k}_{\mathrm{CtR}}(\boldsymbol{x}, \boldsymbol{y}) := \mathrm{Re}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})\}$ to be our novel kernel estimate which, by expanding the real part of $\Phi_{\mathcal{C}}(\boldsymbol{x})^\top \overline{\Phi_{\mathcal{C}}(\boldsymbol{y})}$, can be written as

$$\hat{k}_{\mathrm{CtR}}(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{Re}\{\Phi_{\mathcal{C}}(\boldsymbol{x})\}^\top \mathrm{Re}\{\Phi_{\mathcal{C}}(\boldsymbol{y})\} + \mathrm{Im}\{\Phi_{\mathcal{C}}(\boldsymbol{x})\}^\top \mathrm{Im}\{\Phi_{\mathcal{C}}(\boldsymbol{y})\}.$$

Note that $\mathrm{Re}\{\Phi_{\mathcal{C}}(\boldsymbol{x})\}, \mathrm{Im}\{\Phi_{\mathcal{C}}(\boldsymbol{x})\} \in \mathbb{R}^D$, which allows us to define a $2D$-dimensional *real-valued* polynomial sketch

$$\begin{aligned}\Phi_{\mathrm{CtR}}(\boldsymbol{x}) := \big(&\mathrm{Re}\{\Phi_{\mathcal{C}}(\boldsymbol{x})_1\}, \ldots, \mathrm{Re}\{\Phi_{\mathcal{C}}(\boldsymbol{x})_D\}, \\ &\mathrm{Im}\{\Phi_{\mathcal{C}}(\boldsymbol{x})_1\}, \ldots, \mathrm{Im}\{\Phi_{\mathcal{C}}(\boldsymbol{x})_D\}\big)^\top \in \mathbb{R}^{2D},\end{aligned} \tag{4.1}$$

for which we have

$$\Phi_{\mathrm{CtR}}(\boldsymbol{x})^\top \Phi_{\mathrm{CtR}}(\boldsymbol{y}) = \mathrm{Re}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})\} = \hat{k}_{\mathrm{CtR}}(\boldsymbol{x}, \boldsymbol{y}). \tag{4.2}$$

We call $\Phi_{\mathrm{CtR}}$ a *Complex-to-Real (CtR)* polynomial sketch and summarize its construction in Algorithm 3, where we assume $\Phi_{\mathcal{C}}$ to be one of the Product-Sketches defined in Chapter 3.

The dimension of the feature map $\Phi_{\mathrm{CtR}}$ (4.1) is $2D$ for a given $D$ random samples, while the dimension of the real and complex feature maps, $\Phi_{\mathcal{R}}$ and $\Phi_{\mathcal{C}}$, discussed in the previous chapter is only $D$. Feature maps with higher dimensions usually render the downstream task more expensive and the key question that we address in this chapter is thus: Does the CtR estimator in Eq. (4.2) yield lower variances than $\hat{k}_{\mathcal{R}}(\boldsymbol{x}, \boldsymbol{y}) = \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y})$, where $\Phi_{\mathcal{R}}$ is the corresponding real-valued Product-Sketch defined in Chapter 3, if $\Phi_{\mathrm{CtR}}$ and $\Phi_{\mathcal{R}}$ have the *same* output dimension?

We will show in Section 4.2.1 that this is indeed the case meaning that CtR-sketches inherit the variance reduction properties of complex Product-Sketches. This result implies that, under the conditions provided in this work, CtR-Product-Sketches should be preferred over traditional real-valued Product-Sketches contained in (Kar and Karnick, 2012; Hamid et al., 2014). Unlike the complex sketches discussed in the previous chapter, CtR-sketches are real-valued and do not require the downstream model to handle complex data.

## 4.2 Variances of CtR Product-Sketches

We will now study the variances of CtR-Product-Sketches. Obtaining a closed form expression for their variances seems challenging at first sight, as we need to evaluate

$$\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x}, \boldsymbol{y})] = \mathbb{V}\left[\frac{1}{D}\sum_{\ell=1}^{D} \mathrm{Re}\left\{\prod_{i=1}^{p} (\boldsymbol{w}_{i,\ell}^\top \boldsymbol{x})\overline{(\boldsymbol{w}_{i,\ell}^\top \boldsymbol{y})}\right\}\right],$$

where the weights $\boldsymbol{W}_i = (\boldsymbol{w}_{i,1}, \ldots, \boldsymbol{w}_{i,D})^\top \in \mathbb{C}^{D \times d}$ are defined through the respective sketch as shown in Algorithm 3.

**Algorithm 3:** Complex-to-Real (CtR) Random Features

**Result:** A feature map $\Phi_{\mathrm{CtR}}(\boldsymbol{x})$
**Input:** Datapoint $\boldsymbol{x} \in \mathbb{R}^d$, projection dimension $D \in \mathbb{N}$, degree $p \in \mathbb{N}$ ;
Sample $\{\boldsymbol{W}_i\}_{i=1}^p$ with $\boldsymbol{W}_i \in \mathbb{C}^{D\times d}$ independently according to one of the following sketches:

- Gaussian: $(\boldsymbol{W}_i)_{\ell,k} \overset{i.i.d.}{\sim} \mathcal{CN}(0,1)$
- Rademacher: $(\boldsymbol{W}_i)_{\ell,k} \overset{i.i.d.}{\sim} \mathrm{Unif}(\{1,-1,\mathrm{i},-\mathrm{i}\})$
- Stacked/Upsampled TensorSRHT: $\boldsymbol{W}_i = (\boldsymbol{s}_{i,1},\dots,\boldsymbol{s}_{i,D})^\top$ with $\{\boldsymbol{s}_{i,\ell}\}_{\ell=1}^D$ defined in Eq. (3.34) in Section 3

Compute $\Phi_{\mathcal{C}}(\boldsymbol{x}) := (\boldsymbol{W}_1\boldsymbol{x} \odot \cdots \odot \boldsymbol{W}_p\boldsymbol{x})/\sqrt{D}$ ;
Set $\Phi_{\mathrm{CtR}}(\boldsymbol{x}) := (\mathrm{Re}\{\Phi_{\mathcal{C}}(\boldsymbol{x})_1\},\dots,\mathrm{Re}\{\Phi_{\mathcal{C}}(\boldsymbol{x})_D\},$
$\mathrm{Im}\{\Phi_{\mathcal{C}}(\boldsymbol{x})_1\},\dots,\mathrm{Im}\{\Phi_{\mathcal{C}}(\boldsymbol{x})_D\})^\top \in \mathbb{R}^{2D}$ ;

The challenge here is that the real part of the product of complex numbers corresponds to a large sum and we are not allowed to "swap" $\mathrm{Re}(\cdot)$ with this product. However, we prove the following proposition that simplifies this problem and allows us to make use of the variances for complex Product-Sketches that we derived previously in Chapter 3.

**Proposition 4.2.1.** *Let $\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y}) = \mathrm{Re}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})\}$ for some complex-valued kernel estimate $\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})$ with $\mathbb{E}[\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})] = k(\boldsymbol{x},\boldsymbol{y})$. Then the following holds:*

$$\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] = \frac{1}{2}\mathrm{Re}\left\{\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] + \mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]\right\} \tag{4.3}$$
$$\text{with} \quad \mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] := \mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}$$

*being the* pseudo-variance *of $\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})$ and $\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$ its variance.*

*Proof.* For a complex random variable $z = a + \mathrm{i}\,b$ with $a,b \in \mathbb{R}$, we have $|z|^2 = a^2+b^2$ and $\mathrm{Re}\{z^2\} = a^2-b^2$. Combining both equations gives $a^2 = \frac{1}{2}(|z|^2+\mathrm{Re}\{z^2\})$. The scalar $a$ is real-valued and its variance $\mathbb{V}[a] = \mathbb{E}[a^2] - \mathbb{E}[a]^2$ is thus

$$\mathbb{V}[a] = \frac{1}{2}\mathrm{Re}\{\mathbb{E}[|z|^2] + \mathbb{E}[z^2] - 2\mathbb{E}[a]^2\}. \tag{4.4}$$

Now we set $z = \hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})$, from which it follows that $a = \mathrm{Re}\{z\} = \hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})$. Eq. (4.4) can then be rewritten as

$$\begin{aligned}
\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] &= \frac{1}{2}\mathrm{Re}\{\mathbb{E}[|\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})|^2] + \mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] - 2\mathbb{E}[\mathrm{Re}\{\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})\}]^2\} \\
&= \frac{1}{2}\mathrm{Re}\{\mathbb{E}[|\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})|^2] + \mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] - 2\mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]^2\} \\
&= \frac{1}{2}\mathrm{Re}\{\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] + \mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]\},
\end{aligned}$$

where $\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] := \mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] - \mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] \in \mathbb{C}$ is called the *pseudo-variance* of $\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})$ (Park, 2018, Chapter 5). ■

**Remark 4.2.2.** *In fact, we show next that* $\mathrm{Im}\{\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]\} = 0$ *for all the polynomial sketches discussed in this work. Hence, we can also write* $\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] = \frac{1}{2}(\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] + \mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})])$ *for them since* $\mathbb{V}[z] \in \mathbb{R}$ *for any* $z \in \mathbb{C}$.

In order to determine $\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})]$, we thus only need to know $\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$ and $\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$ for Gaussian, Rademacher and TensorSRHT sketches discussed in Algorithm 3. As we already derived $\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$ for all these sketches in the previous chapter, the remaining task is to derive the pseudo-variances $\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$, which can be easily done using very similar techniques that we already used for the derivation of the variances.

We derive the Pseudo-Variances of Gaussian and Rademacher sketches in Section C.1 and the ones for upsampled and stacked TensorSRHT in Section C.3. We summarize these novel results in the lower half of Table 4.1. For completeness, we add the variances of real and complex polynomial sketches from Chapter 3 $\Phi_{\mathcal{R}}$ to the upper half of Table 4.1. The variance of any CtR-Sketch discussed in this work can thus be obtained by substituting the corresponding variance and pseudo-variance expressions from Table 4.1 into Eq. (4.3).

Having worked out the variances of CtR-Product-Sketches allows us to compare these sketches to their real-valued analogs in the following and to draw conclusions when which sketch should be preferred.

### 4.2.1 Variance Comparison for Gaussian and Rademacher CtR-Sketches

We begin by studying the variance reduction properties of Rademacher and Gaussian CtR-sketches over their real-valued analogs that were originally proposed by Kar and Karnick (2012); Hamid et al. (2014).

Let $\Phi_{\mathcal{R}} : \mathbb{R}^d \to \mathbb{R}^{2D}$ (using $2D$ random features) be a real polynomial sketch as defined in Chapter 3 and $\Phi_{\mathrm{CtR}} : \mathbb{R}^d \to \mathbb{R}^{2D}$ a CtR-sketch (using only $D$ random features) as defined in Eq. (4.1). Let $\hat{k}_{\mathcal{R}}(\boldsymbol{x},\boldsymbol{y})$ and $\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})$ be the respective approximate kernels for some $\boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^d$. Then we can provide the following theorem for the case of Rademacher sketches.

**Theorem 4.2.3** (CtR-Rademacher advantage)**.** *Let* $a := \sum_{i=1}^{d}\sum_{j\neq i}^{d} x_i x_j y_i y_j$ *and* $b(j) := \|\boldsymbol{x}\|^{2j}\|\boldsymbol{y}\|^{2j} - (\sum_{i=1}^{d} x_i^2 y_i^2)^j \geq 0$. *Then* $\mathbb{V}[\hat{k}_{\mathcal{R}}(\boldsymbol{x},\boldsymbol{y})] - \mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})]$ *yields*

$$\frac{1}{2D}\sum_{k=2}^{p}\sum_{j=0}^{k-1}\binom{p}{k}\binom{k}{j} b(j)\; a^{p-j} \geq 0 \quad \textit{if} \quad a \geq 0.$$

*Furthermore, CtR-Rademacher sketches achieve the lowest possible variance for* $\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y}) = \mathrm{Re}\{\Phi_{\mathrm{C}}(\boldsymbol{x})^{\top}\overline{\Phi_{\mathrm{C}}(\boldsymbol{y})}\}$ *with* $\Phi_{\mathrm{C}}$ *being defined through Eq. (3.10) in Chapter 3.*

| **Sketch** | **Variance** $\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$ $(q=1)$ and $\mathbb{V}[\hat{k}_{\mathcal{R}}(\boldsymbol{x},\boldsymbol{y})]$ $(q=2)$ |
|---|---|
| Gaussian | $D^{-1}[(\lVert\boldsymbol{x}\rVert^2 \lVert\boldsymbol{y}\rVert^2 + q(\boldsymbol{x}^\top\boldsymbol{y})^2)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}]$ |
| Rademacher | $D^{-1}[(\lVert\boldsymbol{x}\rVert^2 \lVert\boldsymbol{y}\rVert^2 + q((\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2))^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}]$ |
| TensorSRHT (stacked) | $\mathbb{V}^{(p)}_{\text{Rad.}} - (c(D,d)/D^2)\cdot[(\boldsymbol{x}^\top\boldsymbol{y})^{2p} - (\mathrm{C}_{\text{Var.}})^p]$ with $\mathrm{C}_{\text{Var.}} = (\boldsymbol{x}^\top\boldsymbol{y})^2 - (d-1)^{-1}\,\mathbb{V}^{(1)}_{\text{Rad.}}$ |
| TensorSRHT (upsampled) | $\mathbb{V}^{(p)}_{\text{Rad.}} - (1-1/D)\cdot[(\boldsymbol{x}^\top\boldsymbol{y})^{2p} - (\mathrm{C}_{\text{Var.}})^p]$ with $\mathrm{C}_{\text{Var.}} = (\boldsymbol{x}^\top\boldsymbol{y})^2 - (\lceil D/d\rceil d-1)^{-1}\,\mathbb{V}^{(1)}_{\text{Rad.}}$ |
| **Sketch** | **Pseudo-Variance** $\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$ |
| Gaussian | $D^{-1}[(2(\boldsymbol{x}^\top\boldsymbol{y})^2)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}]$ |
| Rademacher | $D^{-1}[(2(\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}]$ |
| TensorSRHT (stacked) | $\mathbb{PV}^{(p)}_{\text{Rad.}} - (c(D,d)/D^2)\cdot[(\boldsymbol{x}^\top\boldsymbol{y})^{2p} - (\mathrm{C}_{\text{PVar.}})^p]$ with $\mathrm{C}_{\text{PVar.}} = (\boldsymbol{x}^\top\boldsymbol{y})^2 - (d-1)^{-1}\,\mathbb{PV}^{(1)}_{\text{Rad.}}$ |
| TensorSRHT (upsampled) | $\mathbb{PV}^{(p)}_{\text{Rad.}} - (1-1/D)\cdot[(\boldsymbol{x}^\top\boldsymbol{y})^{2p} - (\mathrm{C}_{\text{PVar.}})^p]$ with $\mathrm{C}_{\text{PVar.}} = (\boldsymbol{x}^\top\boldsymbol{y})^2 - (\lceil D/d\rceil d-1)^{-1}\,\mathbb{PV}^{(1)}_{\text{Rad.}}$ |

Table 4.1: Variances of complex $\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})$ and real $\hat{k}_{\mathcal{R}}(\boldsymbol{x},\boldsymbol{y})$ as well as pseudo-variances of complex $\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})$ are shown.
$\mathbb{V}^{(p)}_{\text{Rad.}}, \mathbb{V}^{(1)}_{\text{Rad.}}$ and $\mathbb{PV}^{(p)}_{\text{Rad.}}, \mathbb{PV}^{(1)}_{\text{Rad.}}$ are the Rademacher variances and pseudo-variances for a given degree $p$ and $p=1$, respectively.
$c(D,d) := \lfloor D/d\rfloor d(d-1) + (D \mod d)(D \mod d-1)$.

*Proof.* The variance reduction property is proven in Section C.2 and we show in Section C.1 that Rademacher sketches achieve the lowest possible pseudo-variance for the estimator defined through Eq. (3.10). Since they also achieve the lowest possible variance as shown in Theorem 3.3.1, this implies that they achieve the lowest CtR-variance. ∎

Theorem 4.2.3 tells us that $\Phi_{\mathrm{CtR}}$ should be preferred over $\Phi_{\mathcal{R}}$ when $a \geq 0$ for two given inputs $\boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^d$ and the variance gap increases as $p$ increases. This is the same condition under which complex Rademacher sketches have lower variances than real ones (see Eq. (3.20) in Chapter 3), which holds, e.g., when $\boldsymbol{x},\boldsymbol{y}$ are non-negative. Even when this condition is not always met, CtR-Rademacher sketches perform similarly or better than real ones in our experiments in Section 4.3.1.

We can additionally provide the following theorem for Gaussian polynomial sketches proved in Section C.2.

**Theorem 4.2.4** (CtR-Gaussian advantage)**.** *For any* $\boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^d$, $\mathbb{V}[\hat{k}_{\mathcal{R}}(\boldsymbol{x},\boldsymbol{y})] -$

$\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x}, \boldsymbol{y})]$ *yields*

$$\frac{1}{2D} \sum_{k=0}^{p-1} \binom{p}{k} (2^k - 1)(\boldsymbol{x}^\top \boldsymbol{y})^{2k} \left( \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 \right)^{p-k} \geq 0.$$

Thus, regardless of the input data, $\Phi_{\mathrm{CtR}}$ should be preferred over $\Phi_{\mathcal{R}}$ when using Gaussian polynomial sketches. The advantage again increases with $p$.

The results of Theorems 4.2.3 and 4.2.4 are remarkable because they show conditions under which the CtR-Gaussian and Rademacher sketches outperform the corresponding real analogs defined through Eq. (3.1) in Chapter 3, when both sketches use the *same* feature map dimension. CtR-Rademacher can thus be more efficient than Rademacher sketches with real-valued weights that already achieve a variance lower bound in Theorem 3.1.1. We now give an intuition for this variance reduction.

**Intuition for the variance reduction.** To obtain an intuition for the variance reduction of CtR Product-Sketches, we look at the variance structure of complex Product-Sketches first. We assume a single random feature $D = 1$ for simplicity here. The variance of the approximate kernel $\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})$ (3.11) can be written as:

$$\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = \mathbb{V}\left[ \prod_{i=1}^{p} (\boldsymbol{w}_i^\top \boldsymbol{x})(\boldsymbol{w}_i^\top \boldsymbol{y}) \right] = \prod_{i=1}^{p} \underbrace{\mathbb{E}\left[ |(\boldsymbol{w}_i^\top \boldsymbol{x})(\boldsymbol{w}_i^\top \boldsymbol{y})|^2 \right]}_{=:\ \mu_{2,\mathcal{C}}} - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \tag{4.5}$$

The value of $\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})]$ thus entirely depends on the second complex moment $\mu_{2,\mathcal{C}} \geq 0$. As shown in Section B.1.1, this second moment is smaller for complex Rademacher $\{\boldsymbol{w}_i\}_{i=1}^p$ than for real ones if $\sum_{i=1}^d \sum_{j=1}^d x_i x_j y_i y_j \geq 0$ holds. If we thus let $\mu_{2,\mathcal{C}} = c\mu_{2,\mathcal{R}}$ for some $c < 1$ in this case, we get $\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = c^p \mu_{2,\mathcal{R}} - (\boldsymbol{x}^\top \boldsymbol{y})^{2p}$, where $\mu_{2,\mathcal{R}}$ denotes the respective second moment when $\{\boldsymbol{w}_i\}_{i=1}^p$ are real-valued. We thus obtain a variance advantage that increases with $p$.

This advantage is maintained for CtR-sketches because for any complex random variable, the relationship

$$\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = \mathbb{V}[\mathrm{Re}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})\}] + \mathbb{V}[\mathrm{Im}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})\}]$$

and therefore $\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x}, \boldsymbol{y})] \leq \mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})]$ hold.

Interestingly, when $p = 1$, CtR-sketches are equivalent to doubling the number of random features of the analogous sketch with real-valued weights. This can be seen when looking at Eq. (4.1) and recalling that the real and imaginary parts of $\{\boldsymbol{w}_i\}_{i=1}^p$ are sampled independently in our construction. As the resulting CtR feature map is $2D$-dimensional, there is no statistical advantage w.r.t. the feature map dimension in this case. However, the product in Eq. (4.5) *increases* the variance advantage, giving CtR-sketches a variance improvement from $p \geq 2$.

We study the variance reduction properties of CtR-TensorSRHT next.

### 4.2.2 Variance of CtR-TensorSRHT

As we already showed in Corollary 3.5.4 for real TensorSRHT and Corollary 3.5.10 for complex TensorSRHT, we will now show that CtR-TensorSRHT can achieve a variance reduction over their non-structured counterpart (CtR-Rademacher sketches) for odd degrees $p$ of the polynomial kernel. The analysis is almost the same as in Corollary 3.5.10.

**Corollary 4.2.5.** *Let $p \in \mathbb{N}$ be odd. Then, for all input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ with $\sum_{i=1}^{d} \sum_{j \neq i}^{d} x_i x_j y_i y_j \geq 0$, the variance of the approximate kernel with the CtR-TensorSRHT sketch in Table 4.1 is smaller or equal to the variance of the approximate kernel with the corresponding unstructured polynomial sketch.*

*Proof.* First recall from Eq. (4.3) that the variance of CtR-sketches is given by the average $\frac{1}{2}\mathrm{Re}\{\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] + \mathbb{PV}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})]\}$, where we can drop $\mathrm{Re}\{\cdot\}$ because both the variances and pseudo-variances in Table 4.1 are already real-valued.

Next, we identify the following reduction term

$$R_{\text{Var./Pvar.}} := (1 - 1/D)\left((\boldsymbol{x}^\top \boldsymbol{y})^{2p} - (C_{\text{Var./PVar.}})^p\right) \tag{4.6}$$

that is subtracted from the Rademacher variance $\mathbb{V}^{(p)}_{\text{Rad.}}$ and pseudo-variance $\mathbb{PV}^{(p)}_{\text{Rad.}}$ in Table 4.1, respectively, where we chose upsampled TensorSRHT here. Stacked TensorSRHT is analogous with $(1 - 1/D)$ being substituted by $c(D, d)/D^2$.

If $p$ is odd, we have $(\mathrm{C}_{\text{Var.}})^p \leq (\boldsymbol{x}^\top \boldsymbol{y})^{2p}$ because $\mathbb{V}^{(1)}_{\text{Rad.}} \geq 0$ and therefore $R_{\text{Var.}} \geq 0$ holds. In this case, the variance of complex/real TensorSRHT is upper-bounded by the complex/real Rademacher variance $\mathbb{V}^{(p)}_{\text{Rad.}} \geq 0$.

If we further have $\mathbb{PV}^{(1)}_{\text{Rad.}} = \sum_{i=1}^{d} \sum_{j \neq i}^{d} x_i x_j y_i y_j \geq 0$, the pseudo-variance of complex TensorSRHT is also upper-bounded by the Rademacher pseudo-variance. This is because $0 \leq (\mathrm{C}_{\text{PVar.}})^p \leq (\boldsymbol{x}^\top \boldsymbol{y})^{2p}$ and $R_{\text{PVar.}} \geq 0$ hold. This completes the proof. ∎

The condition $\sum_{i=1}^{d} \sum_{j \neq i} x_i x_j y_i y_j \geq 0$ is the same for which CtR-Rademacher sketches achieve lower variances than real Rademacher sketches, and we will see empirically in the following, that CtR-TensorSRHT sketches outperform real TensorSRHT sketches under this same condition. Therefore, the conditions for structured sketches to outperform non-structured ones and CtR-sketches outperforming their real analogs are well aligned. Moreover, CtR-TensorSRHT inherits the variance reduction of CtR-Rademacher sketches over their real analogs because the Rademacher variance and pseudo-variance are both included in the ones of complex TensorSRHT (see Table 4.1.

We now move on to comparing the variance of CtR-TensorSRHT against its real-valued analog.

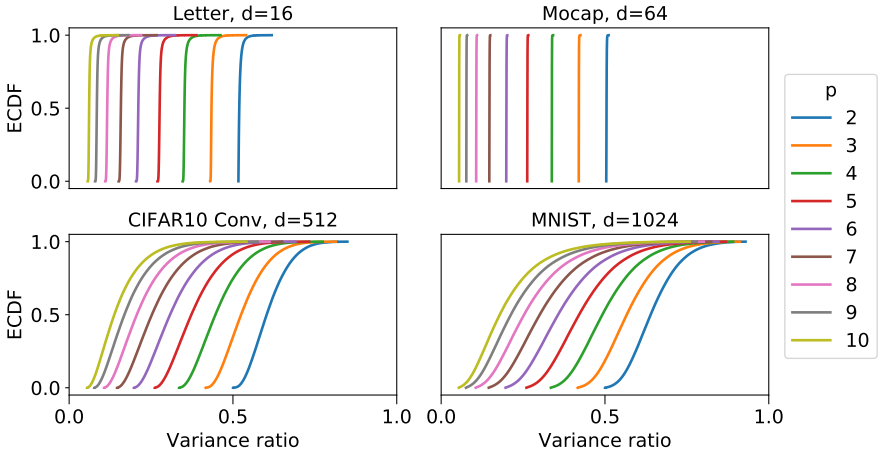



Figure 4.1: Var(CtR-TensorSRHT) / Var(TensorSRHT). The target kernel is $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^p$.

**Comparing the variances of CtR-TensorSRHT against TensorSRHT.**

Analyzing the variance reduction properties of CtR-TensorSRHT over its real analog is very involved as it has already been the case for complex TensorSRHT as we showed in Section 3.5.3. Therefore, we will follow the example of Section 3.5.3 and resort to an empirical comparison here.

**Experimental setting.** Since we obtained the variances of CtR-TensorSRHT and real TensorSRHT in closed form as shown in Table 4.1, we can carry out an exact empirical comparison of their variance ratio on pairs of data points contained in four example data sets (Letter and Mocap (Dua and Graff, 2017), CIFAR-10[1] (Krizhevsky et al., 2009) and MNIST (Lecun et al., 1998)). All data sets except for Mocap contain only non-negative inputs. For Mocap, we subtract the minimum value of each input dimension $\{x_i\}_{i=1}^d$ of $\boldsymbol{x} \in \mathbb{R}^d$ across the data set to achieve non-negativity. We also make sure that both feature maps, CtR-TensorSRHT and TensorSRHT use the same feature map dimension $D = 2d$ in this comparison, and we choose upsampled TensorSRHT here.

**Results.** Fig. 4.1 shows the results of the comparison through empirical cumulative distribution functions of the variance ratios evaluated on all input pairs inside 1000 random samples of each dataset. We observe that CtR-TensorSRHT outperforms real TensorSRHT (all variance ratios $< 1$) and the improvement increases with $p$.

Since the variance of TensorSRHT is closely related to Rademacher variances as mentioned before, we study the importance of the non-negativity condition of

[1] We use the the convolutional outputs of a pretrained ResNet34 (He et al., 2016) on ImageNet (Russakovsky et al., 2015).

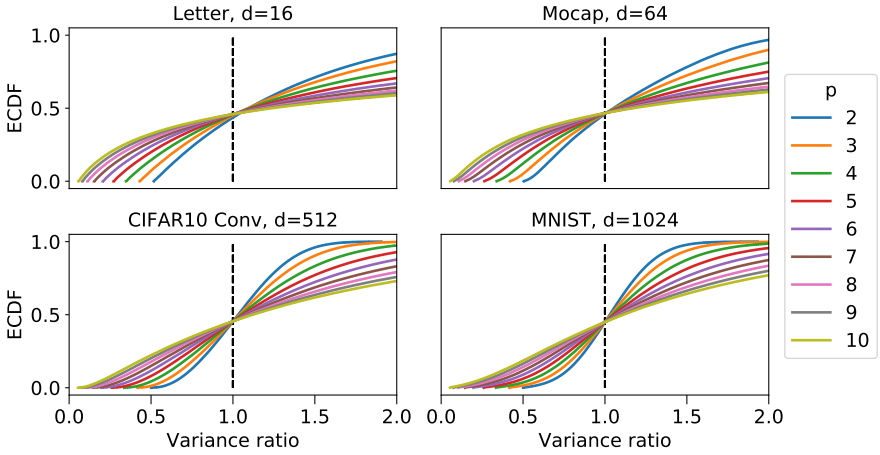



Figure 4.2: Var(CtR-TensorSRHT) / Var(TensorSRHT) on zero-centered data. The target kernel is $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^p$.

the data in Theorem 4.2.3 for the case of CtR-TensorSRHT. For this purpose, we repeat the same experiment as before for zero-centered data that breaks the non-negativity condition. Results are shown in Fig. 4.2. We see that CtR-TensorSRHT performs similarly as its real analog in this case (around half of the variance ratios are above and below 1). Therefore, a variance reduction happens in particular for non-negative data as is the case for CtR-Rademacher sketches. This makes sense because both the variance and the pseudo-variance of TensorSRHT are closely related to the ones of Rademacher sketches as shown in Table 4.1.

So far, we have compared both complex and Complex-to-Real Product-Sketches against real Product-Sketches. The last open question is how Complex-to-Real sketches compare against complex sketches. To answer this question, we do *not* focus on a variance comparison between the two, but we consider studying the implications on the downstream task to be more important. This is because complex random feature maps require the downstream task to handle complex data incurring higher computational costs as shown previously in Section 3.5.3 in Chapter 3. We will therefore extend the Gaussian process classification example from Section 3.5.3, and compare real, complex, and Complex-to-Real TensorSRHT sketches on this downstream task.

**Comparing Real, Complex and CtR-Sketches.** We compare the convergence of a GP classifier using real, complex and CtR-TensorSRHT towards one using the exact kernel on MNIST under a given time budget in Fig. 4.3. We can see that a complex GP classifier using complex TensorSRHT needs roughly the same time for a projection dimension $D$ as real and CtR-TensorSRHT for dimension $2D$. This is because complex TensorSRHT requires solving a complex GP model that takes more operations to solve for the same feature dimension as a real GP as explained Appendix B.3.

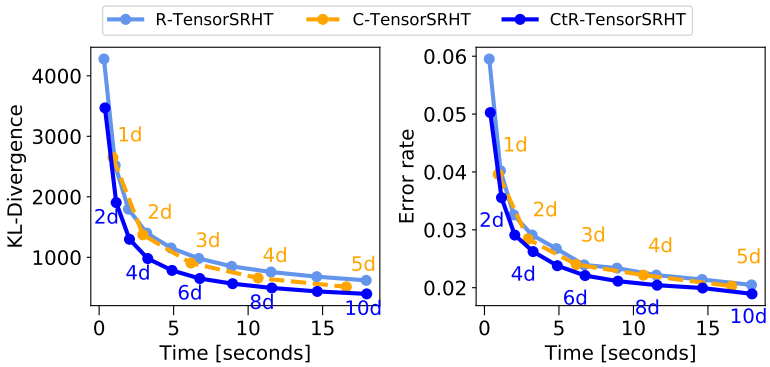



Figure 4.3: KL-divergence (left) and test error (right) for a given computation time budget computed on MNIST ($d = 1024$). Projection dimensions are annotated next to the respective measurements. We used the kernel $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^6$.

This gives a clear advantage to the CtR-TensorSRHT sketch that achieves the lowest KL-divergence and test errors in the same amount of time. Therefore, we recommend the use of CtR-TensorSRHT over real and complex TensorSRHT. Here we used the kernel $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^6$ and timings were measured on an NVIDIA P100 graphics card and PyTorch 1.10 (Paszke et al., 2019) with native complex linear algebra support.

## 4.3 Comparison against the State-of-the-Art

In this section, we compare our most efficient Product-Sketch, upsampled CtR-TensorSRHT, against TensorSketch (Pham and Pagh, 2013) that we introduced in Chapter 2. TensorSketch can be seen as the state-of-the-art *unbiased*[2] polynomial sketch. Unlike for CtR-TensorSRHT, the variance of TensorSketch is not available in the literature. We will therefore carry out an empirical variance comparison here. Furthermore, we add a feature construction time benchmark to this comparison, since TensorSketch leverages the fast Fourier transform and is therefore considered to be a particularly fast sketch.

**Variance Comparison.** We repeat the same experiment as in Fig. 4.1, but this time we evaluate the variance ratios of upsampled CtR-TensorSRHT against TensorSketch. As the variance of TensorSketch is not available in closed form, we

[2]For high-degree polynomial kernels on the unit-sphere it is still outperformed by biased Spherical Random Features (SRF) (Pennington et al., 2015). However, SRF require a preprocessing step for every polynomial kernel parameterization and only work for data lying on the unit-sphere, obstructing a straightforward application, e.g., in deep learning. We will carry out empirical evaluations against SRF in Chapter 5.

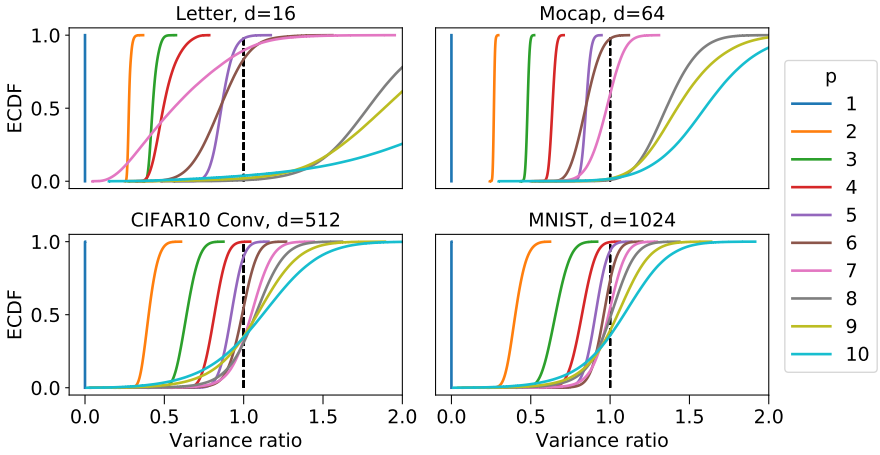



Figure 4.4: Var(CtR-TensorSRHT) / Var(TensorSketch). Empirical cumulative distribution function of variance ratios for feature maps with equal dimension $D = 2d$. The target kernel is $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^p$.

estimate it using 1000 Monte-Carlo samples of the kernel estimate. Fig. 4.4 shows a comparison of the variance ratios.

In general, upsampled CtR-TensorSRHT improves over TensorSketch for $p \leq 5$ (more than half of the ratios are less than one). This is an important advantage as such degrees are commonly used in practice (Pham and Pagh, 2013; Gao et al., 2016; Fukui et al., 2016). For $p = 1$, TensorSRHT always has a variance of zero as soon as $D = kd$ for $k \in \mathbb{N}$, which can be derived from the TensorSRHT variance equation in Table 4.1.

For large $p$, TensorSketch and upsampled CtR-TensorSRHT perform similarly well (around half the variance ratios are smaller/greater than 1) except for some cases with low-dimensional data where TensorSketch performs better. The ratios would be much worse if we used the real TensorSRHT instead as shown in Fig. 4.5, which underlines the relative improvement of the CtR-extension.

The variance comparison of CtR-TensorSRHT against TensorSketch is impressive given the fact that (upsampled) CtR-TensorSRHT has a faster running time, which we will compare next.

**Feature construction time comparison.** In the following, we carry out a feature construction time comparison of upsampled CtR-TensorSRHT against TensorSketch that has a time complexity $\mathcal{O}(p(D \log D + D))$ with $D$ being the projection dimension.

Recall that our proposed upsampled TensorSRHT approach introduced in Section 3.5.1 in Chapter 3 has a time complexity of $\mathcal{O}(p(d \log d + D))$ and is thus faster in theory when $D > d$.

The left plot in Fig. 4.6 shows the results of this comparison, where we added

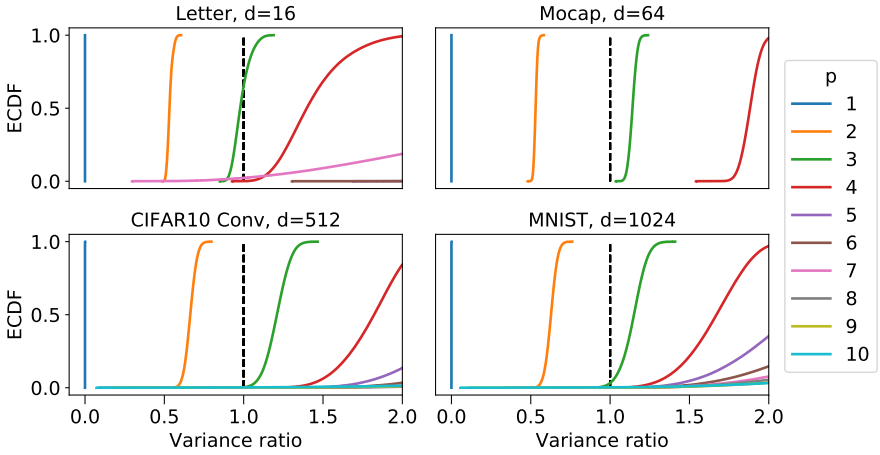



Figure 4.5: Var(TensorSRHT) / Var(TensorSketch). Empirical cumulative distribution function of variance ratios for feature maps with equal dimension $D = 2d$. The target kernel is $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^p$.

real Rademacher and upsampled TensorSRHT Product-Sketches to the plot to give a more complete picture. The construction times of real TensorSRHT and CtR-TensorSRHT have a smaller slope with respect to $D$ than the other sketches leading to the lowest feature construction times, in particular when $D \gg d$. There is a small computational overhead for CtR-TensorSRHT compared to real TensorSRHT because CtR-TensorSRHT initially requires two Hadamard-projections (real and imaginary parts), but uses the same sampling matrix leading to the same scaling property with respect to $D$.

The right plot of Fig. Fig. 4.6 shows the kernel approximation error (relative Frobenius norm error) under a computational budget. We can see that our proposed CtR-TensorSRHT gives the best kernel approximations in the shortest amount of time when $D$ is sufficiently large. As before, timings were measured on an NVIDIA P100 GPU and PyTorch 1.10 (Paszke et al., 2019) with native complex linear algebra support.

One may argue that the feature construction time is not the computational bottleneck in downstream tasks like closed form GP regression, where the bottleneck is the construction and inversion of a $D$ by $D$ matrix. This changes however, when using iterative solvers working with mini-batches. We carry out such a time benchmark for iterative solvers in the following.

**Time benchmark for stochastic optimization.** We compare CtR-TensorSRHT against TensorSketch for a GP classification experiment using stochastic variational inference and mini-batching. The goal is to measure the impact of the feature construction time on the training time of the learning algorithm. The feature construction time plays a crucial role here since every iteration of the algo-

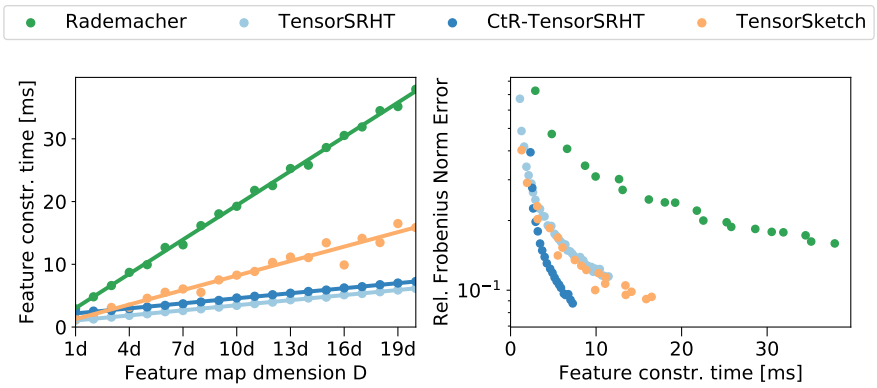



Figure 4.6: (Left) Feature construction time against feature map dimension $D$. (Right) Kernel approximation error (rel. Frobenius norm error) against feature construction time. We approximate the kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y} + 1)^6$ on 1000 random MNIST samples.

rithm carries out one forward and one backward pass through a two-layer neural network, where the first layer corresponds to the feature extraction layer, either TensorSRHT or TensorSketch in this case.

We choose $D = 10d$, where $d = 1024$ for the MNIST data set. We further set $p = 6$ as the polynomial degree implying the use of 6 random matrix multiplications. In such a scenario, the feature construction time has a big share on the total computation time of every iteration.

Fig. 4.7 shows the results of this stochastic variational inference experiment, where the goal is to obtain a factorized Gaussian posterior distribution over the weights of a GP classifier. We show the test error against the training time to reach this error. The feature maps approximate the kernel $(\gamma \boldsymbol{x}^\top \boldsymbol{y} + \nu)^6$, where $\gamma, \nu \geq 0$ are optimized through backpropagation. The mini-batch size is chosen to be 1000 and we use the Adam optimizer with learning rate $10^{-3}$. We further choose 50 Monte-Carlo samples for the posterior weights at each iteration.

When using CtR-TensorSRHT, the experiments were run for 150 epochs while the ones for TensorSketch were run for 50 epochs. We can thus see that TensorSRHT achieves a much faster convergence (in wall-clock time), which is due to a faster computation of the feature map as shown in Fig. 4.6. CtR-TensorSRHT outperforms TensorSRHT in terms of test error and is only slightly slower.

### 4.3.1 Systematic Evaluation on Different Data Sets

In this section, we carry out a systematic comparison of the CtR polynomial sketches discussed in this work against their real-valued analogs as well as TensorSketch (Pham and Pagh, 2013) on the same four data sets that we used for

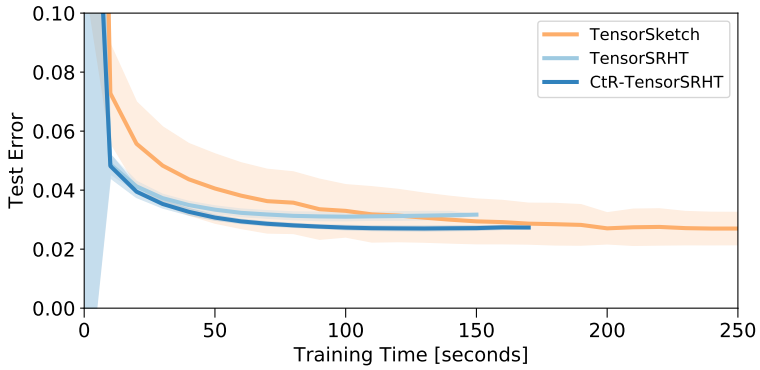



Figure 4.7: Stochastic variational inference on MNIST ($d = 1024$) for $D = 10d$ and $p = 6$. Results are averaged over 10 different runs. TensorSRHT finishes 150 training epochs before TensorSketch finishes 50 epochs and thus converges earlier.

the variance comparison experiments, namely Letter and Mocap (Dua and Graff, 2017), CIFAR-10 (Krizhevsky et al., 2009) and MNIST (Lecun et al., 1998), where we make sure that the data is positive-valued. This comparison includes the evaluation of kernel approximation errors as well as the downstream performance on the task of GP classification.

**Target kernel and its approximation.** The target polynomial kernel is always $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y} + 1)^p$[3] and we unit-normalize the data to make the kernel bounded and improve numerical stability. Since CtR polynomial sketches $\Phi_{\text{CtR}}$ are $2D$-dimensional when sampling $D$ random features, we compare them against real-valued polynomial sketches $\Phi_{\mathcal{R}}$ with $2D$ random features to obtain the *same* dimension of the feature map.

We measure kernel approximation quality through the relative Frobenius norm error, which is defined as $\|\hat{\boldsymbol{K}} - \boldsymbol{K}\|_F / \|\boldsymbol{K}\|_F$, where $\hat{\boldsymbol{K}}$ is the random feature approximation of the exact kernel matrix $\boldsymbol{K}$ with $(\boldsymbol{K})_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\boldsymbol{x}_i, \boldsymbol{x}_j \in X_{*,\text{sub}}$. $X_{*,\text{sub}}$ is a subset of the test data of size $N_{*,\text{sub}} = 1000$ of each dataset that is resampled for 100 different seeds used in these experiments. We use a 90/10 train/test split for Letter and Mocap that is recomputed for every seed, while it is provided for CIFAR-10 and MNIST.

**Downstream task.** We model GP classification as a multi-class GP regression problem with transformed labels (Milios et al., 2018) for which we obtain closed-form solutions. This allows us to measure the effects of the random feature approximations in isolation without needing to verify the convergence of iterative solvers. To measure the convergence towards a target GP when using random

[3]The kernel parameters are a standard choice used in, e.g., *scikit-learn* (Pedregosa et al., 2011).

features, we use the Kullback-Leibler (KL) divergence between the GP predictive distribution using the approximate kernel and the predictive distribution using the target kernel. These distributions are evaluated on the test data and the GPs are trained on a random subset of size 5000 of the training data as the reference GP cannot be scaled to a large number of datapoints. Classification errors instead, are obtained for the model trained on the whole training data, and evaluated on all test data.

**Results.** Fig. 4.8 and Fig. 4.9 show the results of this comparison for the polynomial degrees $p = 3$ and $p = 6$, respectively. We show kernel approximation errors (relative Frobenius norm errors) along with KL-Divergences to the ground-truth GP and test errors for each dataset. As shown in Fig. 4.4, CtR-TensorSRHT has a lower variance than TensorSketch for $p = 3$, and the advantage diminishes for larger $p$. At the same time, the advantage of CtR-sketches over real ones *increases* with $p$ as shown in Fig. 4.1. These two effects are reflected in the corresponding kernel approximation errors in Fig. 4.8 and Fig. 4.9, respectively.

For $p = 3$, the errors are the lowest for CtR-TensorSRHT and also fluctuate the least. TensorSRHT thus performs better than unstructured Gaussian and Rademacher sketches as discussed in Section 3.5 in Chapter 3. The CtR extension makes TensorSRHT perform similarly on the downstream task of GP classification as TensorSketch (similar KL-divergence and classification error). The results are worse for TensorSRHT without the CtR-extension.

For $p = 6$ (Fig. 4.9), CtR-TensorSRHT and TensorSketch perform similarly well, while the relative advantage of CtR-TensorSRHT over TensorSRHT increases compared to the case of $p = 3$ in Fig. 4.8.

We add a CtR-modification of TensorSketch[4] to both comparisons in Fig. 4.8 and Fig. 4.9. However, this modification does not yield any improvements in our experiments. CtR-extensions thus only seem to be beneficial for the Product-Sketches discussed in this work.

Lastly, we carry out the same comparison as before for zero-centered data to study the importance of the non-negativity condition of the data. Fig. 4.10 shows the results of this comparison. We can see that the relative advantage of CtR-sketches over their real analogs diminishes compared to the results shown in Fig. 4.9. However, CtR-sketches still achieve an advantage and never perform worse.

We will now conclude the empirical campaign carried out in this chapter with the application of our polynomial sketches to accelerate the computation of an evaluation metric used in *generative modelling*, namely the *Kernel Inception Distance*.

[4]We replace sign hash function $s : [d] \to \{1, -1\}$ with $s : [d] \to \{1, -1, \mathrm{i}, -\mathrm{i}\}$ in (Pham and Pagh, 2013, Def. 1) giving rise to a complex-valued TensorSketch.

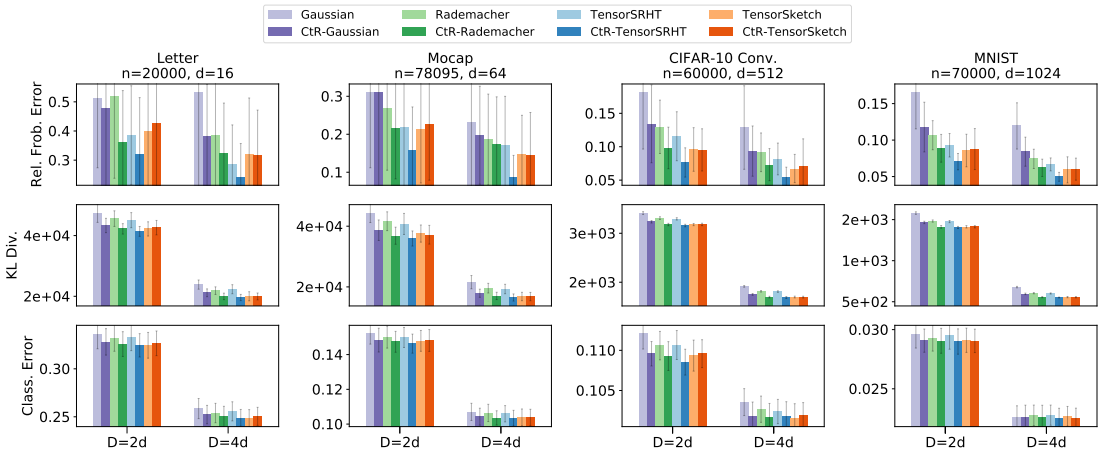



Figure 4.8: Gaussian Process classification experiments comparing different polynomial sketches along with their CtR extensions for $D \in \{2d, 4d\}$. The target polynomial kernel is $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^3$ and the data is unit-normalized.

### 4.3.2 Approximating the Kernel Inception Distance

The *Kernel Inception Distance (KID)* (Bińkowski et al., 2018) is used to measure the similarity between two image data sets (a lower distance refers to a higher similarity). It is especially used in generative modelling to evaluate the quality of a set of generated images by comparing them against a set of ground-truth images.

KID corresponds to the squared maximum mean discrepancy (MMD) (Gretton et al., 2012) using the polynomial kernel $(\frac{1}{d}\boldsymbol{x}^\top \boldsymbol{y} + 1)^p$, where this kernel is computed on top of the convolutional features obtained from each image through the Inception (Szegedy et al., 2016) neural network architecture.

As the computation of the MMD metric scales quadratically in the number of data points, approximation methods have to be used for large data sets such as CIFAR-10 that we use here. Therefore, the approximation of the KID is a great application for the polynomial sketches discussed in this work.

**Experimental setting.** We follow the experiment in the original work that introduced the KID (Bińkowski et al., 2018), which uses the average of blockwise MMD values (Block-MMD) as an approximation. This approximation will serve as a baseline to be compared against polynomial sketches. In this case, the exact squared MMD is computed on a subset of size $n$ of each of the two data sets being compared and this process is repeated for 100 independent subset samples. The final KID estimate is then given by the average of the squared MMDs computed on the subsets. As the KID estimate is unbiased, we compute the standard deviation of these samples to meausure the efficiency of the approximator. We use $n \in \{100, 1000, 5000, 10000, 15000, 20000\}$ for the experiments in this section.

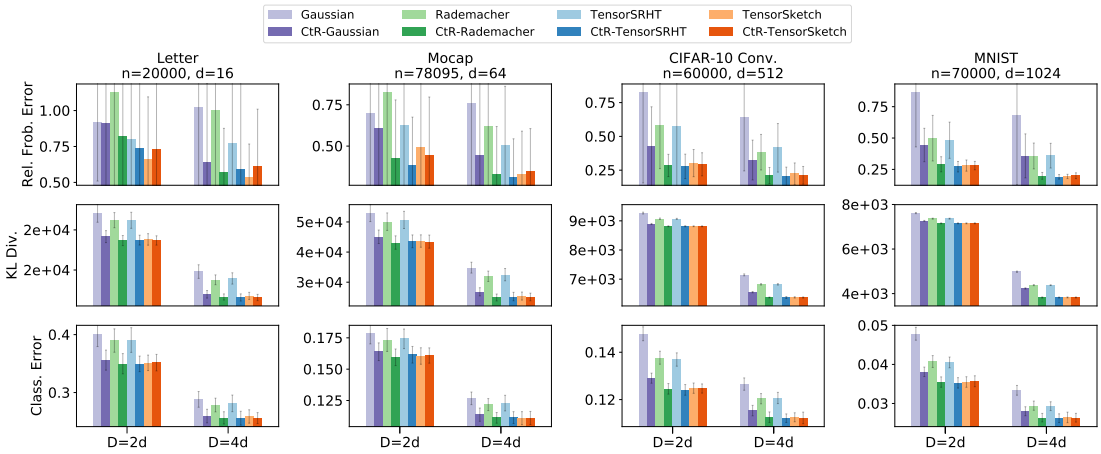



Figure 4.9: Gaussian Process classification experiments comparing different polynomial sketches along with their CtR extensions for $D \in \{2d, 4d\}$. The target polynomial kernel is $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^6$ and the data is unit-normalized.

We compare this baseline against squared MMD estimates obtained through polynomial sketches. We obtain unbiased squared MMD estimates using the method described in Sutherland and Schneider (2015, Section 3). We repeat this process 100 times as before and compute the standard deviation on the estimate. We use $D \in \{64, 128, 256, 512, 1024, d, 2d, 3d, 4d\}$ in our experiments, where $d = 2048$ is the dimension of the convolutional features obtained through the Inception neural network. The polynomial sketches being compared are CtR-TensorSRHT, TensorSRHT and TensorSketch.

**Results.** Fig. 4.11 shows the standard deviations of the squared MMD (KID) estimates computed as explained before. We see that all polynomial sketches achieve around one order of magnitude lower standard deviations than the Block-MMD baseline in the same computation time indicating that they estimate the KID more efficiently. TensorSRHT and CtR-TensorSRHT perform slightly better than TensorSketch for $p = 3$ and CtR-TensorSRHT performs better for $p = 6$.

Table 4.2 shows the same standard deviations as before but computed on 1000 instead of 100 resamples of the KID metric. The results are thus more accurate and confirm that CtR-TensorSRHT provides the best KID estimates for a given feature map dimension $D$. However, obtaining these accurate estimates is slower and therefore less samples are used in practice.

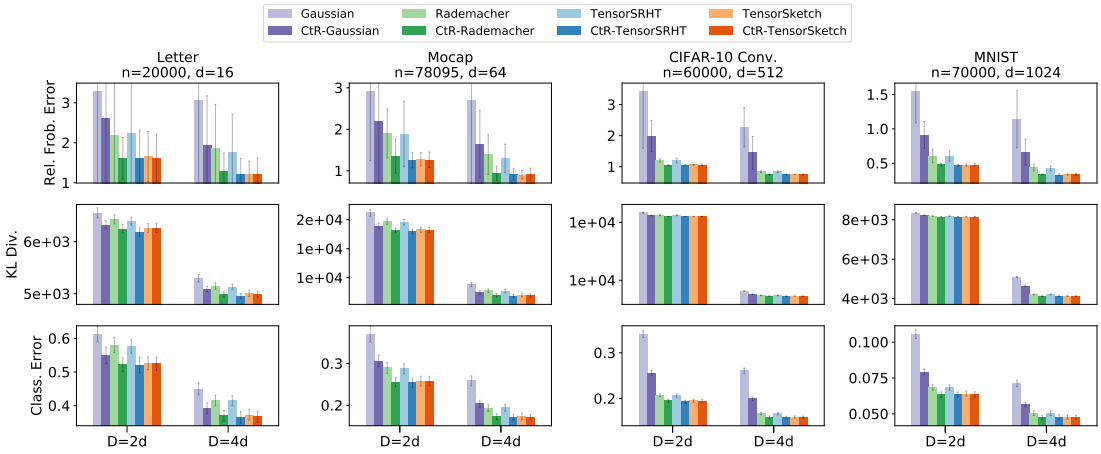
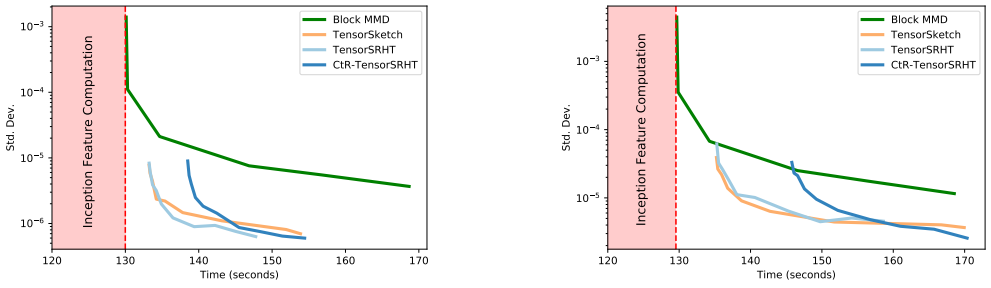



Figure 4.10: Gaussian Process classification experiments comparing different polynomial sketches along with their CtR-extensions for $D \in \{2d, 4d\}$. The target polynomial kernel is $(\boldsymbol{x}^\top \boldsymbol{y} + 1)^6$ and the data is unit-normalized as well as zero-centered.

Table 4.2: Standard deviation of the KID estimator between the CIFAR-10 train and test sets estimated on 1000 independent samples.

| | $p = 3$ | | | | $p = 6$ | | | |
|---|---|---|---|---|---|---|---|---|
| | $D = 1d$ | $D = 2d$ | $D = 3d$ | $D = 4d$ | $D = 1d$ | $D = 2d$ | $D = 3d$ | $D = 4d$ |
| TensorSketch | 1.363E-06 | 9.639E-07 | 7.837E-07 | 6.841E-07 | 6.974E-06 | 4.909E-06 | 4.015E-06 | 3.436E-06 |
| TensorSRHT | 1.435E-06 | 1.047E-06 | 7.869E-07 | 6.946E-07 | 9.476E-06 | 6.897E-06 | 5.331E-06 | 4.871E-06 |
| CtR-TensorSRHT | **1.312E-06** | **8.747E-07** | **7.420E-07** | **5.941E-07** | **6.697E-06** | **4.856E-06** | **3.762E-06** | **3.260E-06** |




(a) $p = 3$ (b) $p = 6$

Figure 4.11: Standard deviation of the KID estimator between the CIFAR-10 train and test sets against computation time. Estimates are obtained using 100 independent resamples of the approximate metric. Note that the first 130 seconds are spent on computing the Inception features (the plot starts at 120 to become better readable).

# Chapter 5

# Approximating Dot Product Kernels

We discuss in this chapter how the polynomial sketches described so far can be used for approximating more general *dot product kernels*, i.e., kernels whose values depend only on the inner product of the input vectors.

There exist alternative random feature approximations for dot product kernels using Gegenbauer polynomials (Han et al., 2022) that have been concurrently developed while we worked on the journal submission associated to the contents of this chapter (Wacker et al., 2022a). We therefore only focus on random feature approximations via polynomial sketches and leave a comparison against random Gegenbauer expansions for future work. We further do not make use of the Complex-to-Real Product-Sketches developed in the previous chapter here. This is because we developed CtR sketches *after* developing the contents of this chapter. Moreover, we will see that the Maclaurin expansions in this chapter are applied to zero-centred data, obstructing the non-negativity condition giving advantages to CtR-sketches in Chapter 4.

In Sections 5.1 and 5.2, we first review a key result on the Maclaurin expansion of dot product kernels and the resulting random sketching approach by Kar and Karnick (2012), and show how the polynomial sketches described so far can be used. In Section 5.3, we then introduce a data-driven optimization approach to improving the random sketches based on the Maclaurin expansion. In Section 5.3.1, we describe how this approach can also be applied to approximate the Gaussian kernel. We then carry out an empirical campaign comparing our optimized Maclaurin method against Spherical Random Features (Pennington et al., 2015) and random Fourier features (Rahimi and Recht, 2007) in Section 5.4. Moreover, we identify a pathology of vanishing approximate kernels for Maclaurin-based approximations of the Gaussian kernel and we propose a cure in Section 5.5.1 with an additional empirical evaluation.

## 5.1 Maclaurin Expansion of Dot Product Kernels

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a subset, and let $k : \mathcal{X} \times \mathcal{X} :\to \mathbb{R}$ be a positive definite kernel on $\mathcal{X}$. The kernel $k$ is called *dot product kernel*, if there exists a function $f : \mathbb{R} \to \mathbb{R}$ such that

$$k(\boldsymbol{x}, \boldsymbol{y}) = f(\boldsymbol{x}^\top \boldsymbol{y}) \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}. \tag{5.1}$$

Examples of dot product kernels include polynomial kernels $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y} + \nu)^p$ with $\nu \geq 0$ and $p \in \mathbb{N}$, which have been our focus in this thesis, and exponential kernels $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(\boldsymbol{x}^\top \boldsymbol{y}/l^2)$ with $l > 0$. Other examples of dot product kernels can be found in, e.g., Smola et al. (2000).

We focus on dot product kernels for which the function $f$ in Eq. (5.1) is an analytic function whose Maclaurin expansion has non-negative coefficients: $f(x) = \sum_{n=0}^{\infty} a_n x^n$ and $a_n \geq 0$ for $n \in \{0\} \cup \mathbb{N}$. In other words, we consider dot product kernels that can be expanded as

$$k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n=0}^{\infty} a_n (\boldsymbol{x}^\top \boldsymbol{y})^n \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}, \tag{5.2}$$

with $a_n \geq 0$ for all $n \in \{0\} \cup \mathbb{N}$.

Many dot product kernels can be expanded as Eq. (5.2). In fact, Kar and Karnick (2012, Theorem 1) show that, if $\mathcal{X}$ is the unit ball of $\mathbb{R}^d$, the function $k$ of the form of Eq. (5.1) is positive definite on $\mathcal{X}$ if and only if it can be written as Eq. (5.2).

We show here a few concrete examples. The polynomial kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y} + \nu)^p$ with $p \in \mathbb{N}$ and $\nu \geq 0$ can be expanded as

$$(\boldsymbol{x}^\top \boldsymbol{y} + \nu)^p = \sum_{n=0}^{p} \binom{p}{n} \nu^{p-n} (\boldsymbol{x}^\top \boldsymbol{y})^n, \tag{5.3}$$

and thus $a_n = \binom{p}{n} \nu^{p-n} \geq 0$ for $n \in \{0, \ldots, p\}$ and $a_n = 0$ for $n > p$ in Eq. (5.2). The exponential kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(\boldsymbol{x}^\top \boldsymbol{y}/l^2)$ can be expanded as

$$\exp\left(\frac{\boldsymbol{x}^\top \boldsymbol{y}}{l^2}\right) = \sum_{n=0}^{\infty} \frac{1}{n! l^{2n}} (\boldsymbol{x}^\top \boldsymbol{y})^n \tag{5.4}$$

and thus $a_n = 1/(n! l^{2n})$ for $n \in \mathbb{N}$ in Eq. (5.2).

**Gaussian kernel as a weighted dot product kernel** The Gaussian kernel defined as $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/(2l^2))$ with $l > 0$ can be written as a *weighted*

exponential kernel:

$$\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2l^2}\right) = \exp\left(-\frac{\|\boldsymbol{x}\|^2}{2l^2}\right)\exp\left(-\frac{\|\boldsymbol{y}\|^2}{2l^2}\right)\exp\left(\frac{\boldsymbol{x}^\top\boldsymbol{y}}{l^2}\right)$$
$$= \exp\left(-\frac{\|\boldsymbol{x}\|^2}{2l^2}\right)\exp\left(-\frac{\|\boldsymbol{y}\|^2}{2l^2}\right)\sum_{n=0}^{\infty}\frac{1}{n!l^{2n}}(\boldsymbol{x}^\top\boldsymbol{y})^n, \qquad (5.5)$$

where the second identity uses the Maclaurin expansion of the exponential kernel in Eq. (5.4). For approximating the Gaussian kernel, Cotter et al. (2011) proposed a finite dimensional feature map based on a truncation of this expansion.

## 5.2 Random Sketch based on the Maclaurin Expansion

We describe here the approach of Kar and Karnick (2012) on the unbiased approximation of dot product kernels based on the Maclaurin expansion in Eq. (5.2). We discuss this approach to provide a basis and motivation for our new approach for approximating dot product kernels.

First, we define a probability measure $\mu$ on $\{0\} \cup \mathbb{N}$. Kar and Karnick (2012) propose to define $\mu$ as

$$\mu(n) \propto c^{-(n+1)}, \quad n \in \{0\} \cup \mathbb{N}, \qquad (5.6)$$

for a constant $c > 1$ (e.g., $c = 2$).

Using this probability measure and the Rademacher sketch, Kar and Karnick (2012) propose a doubly stochastic approximation of the dot product kernel in Eq. (5.2).

This approach first generates an i.i.d. sample of size $D \in \mathbb{N}$ from this probability measure $\mu$

$$n_1, \ldots, n_D \overset{i.i.d.}{\sim} \mu \qquad (5.7)$$

and defines $D_n$ for $n \in \{0\} \cup \mathbb{N}$ as the number of times $n$ appears in $n_1, \ldots, n_D$; thus $\sum_{n=0}^{\infty} D_n = D$.

Then, for each $n \in \{0\} \cup \mathbb{N}$ with $D_n > 0$, construct a random feature map $\Phi_n : \mathcal{X} \to \mathbb{R}^{D_n}$ with $D_n$ features of the form in Eq. (3.1) in Chapter 3 that provide an unbiased approximation of the polynomial kernel $k_n(\boldsymbol{x}, \boldsymbol{y}) := (\boldsymbol{x}^\top\boldsymbol{y})^n$ of degree $n$:

$$\mathbb{E}[\Phi_n(\boldsymbol{x})^\top\Phi_n(\boldsymbol{y})] = (\boldsymbol{x}^\top\boldsymbol{y})^n. \qquad (5.8)$$

The original formulation of Kar and Karnick (2012) uses the Rademacher sketch as $\Phi_n$, but one can use other sketches introduced in Chapters 2, 3 and 4, such as the Gaussian sketch and TensorSRHT.

Finally, by defining a random variable $n^* \sim \mu$, the dot product kernel in Eq. (5.2) is rewritten and approximated as

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &= \sum_{n=0}^{\infty} a_n (\boldsymbol{x}^\top \boldsymbol{y})^n = \sum_{n=0}^{\infty} \frac{a_n}{\mu(n)} \mu(n) (\boldsymbol{x}^\top \boldsymbol{y})^n = \mathbb{E}_{n^* \sim \mu} \left[ \frac{a_{n^*}}{\mu(n^*)} (\boldsymbol{x}^\top \boldsymbol{y})^{n^*} \right] \\ &\approx \frac{1}{D} \sum_{n \in \{n_1, \ldots, n_D\}} D_n \frac{a_n}{\mu(n)} (\boldsymbol{x}^\top \boldsymbol{y})^n = \frac{1}{D} \sum_{n : D_n > 0} D_n \frac{a_n}{\mu(n)} (\boldsymbol{x}^\top \boldsymbol{y})^n \\ &\approx \frac{1}{D} \sum_{n : D_n > 0} D_n \frac{a_n}{\mu(n)} \Phi_n(\boldsymbol{x})^\top \Phi_n(\boldsymbol{y}), \end{aligned} \tag{5.9}$$

where the first approximation is the Monte Carlo approximation of the expectation $\mathbb{E}_{n^* \sim \mu}$ using the i.i.d. sample in Eq. (5.7) and the second approximation is using the random feature map in Eq. (5.8). The approximation in Eq. (5.9) is unbiased, since the two approximations are statistically independent and both are unbiased.

The first approximation for Eq. (5.9) can be interpreted as first selecting polynomial degrees $n \in \{0\} \cup \mathbb{N}$ and assigning the number of features $D_n$ to each selected degree, given a budget constraint $D = \sum_{n : D_n > 0} D_n$. While performing these assignments by random sampling as in Eq. (5.7) makes the approximation in Eq. (5.9) unbiased, the resulting variance of Eq. (5.9) can be large. In the next section, we introduce a data-driven optimization approach to this feature assignment problem, to achieve a good balance between the bias and variance.

## 5.3 Optimization for a Truncated Maclaurin Approximation

We develop here an optimization algorithm for selecting the polynomial degrees $n$ and assigning the number of random features to each selected polynomial degree in the Maclaurin sketch in Eq. (5.9) . The objective function is an estimate of the expected bias and variance of the resulting approximate kernel, and we define it using the variance formulas derived in Chapter 3.

We consider a biased approximation obtained by truncating the Maclaurin expansion in Eq. (5.2) up to the $p$-th degree polynomials, where $p$ is to be determined by optimization. Let $D_{\text{total}} \in \mathbb{N}$ be the total number of random features, which is specified by a user. For each $n = 1, \ldots, p$, let $D_n \in \{0\} \cup \mathbb{N}$ be the number of random features for approximating the $n$-th term $(\boldsymbol{x}^\top \boldsymbol{y})^n$ of the Maclaurin expansion in Eq. (5.2), such that $\sum_{n=1}^{p} D_n = D_{\text{total}}$. The numbers $D_n$ are to be determined by optimization. Let $\Phi_n : \mathbb{R}^d \to \mathbb{C}^{D_n}$ be a (possibly complex) random feature map defined in Chapter 3 such that $\mathbb{E}[\Phi_n(\boldsymbol{x})^\top \overline{\Phi_n(\boldsymbol{y})}] = (\boldsymbol{x}^\top \boldsymbol{y})^n$ for all $\boldsymbol{x}, \boldsymbol{y} \subset \mathcal{X} \subset \mathbb{R}^d$. Note that $\Phi_n$ can be a real-valued feature map, but we use the notation for the complex case since it subsumes the real case.

We then define an approximation to the dot product kernel in Eq. (5.2) as

$$\hat{k}(\boldsymbol{x}, \boldsymbol{y}) := a_0 + \sum_{n=1}^{p} a_n \Phi_n(\boldsymbol{x})^\top \overline{\Phi_n(\boldsymbol{y})}, \quad \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X} \tag{5.10}$$

This approximation is biased, since it ignores the polynomial terms whose degrees are higher than $p$ in the expansion of Eq. (5.2). One can reduce this bias by increasing $p$, but this may lead to a higher variance.

Therefore, there is a bias-variance trade-off in the choice of $p$. We describe below how to choose $p$ and the number of features $D_n$ of each random feature map $\Phi_n(\boldsymbol{x}), \Phi_n(\boldsymbol{y}) \in \mathbb{C}^{D_n}$ for $n = 1, \ldots, p$.

### Optimization Objective

For a given learning task, we are usually provided data points generated from an unknown probability distribution $P(\boldsymbol{x})$ on the input domain $\mathcal{X} \subseteq \mathbb{R}^d$. The approximate kernel $\hat{k}(\boldsymbol{x}, \boldsymbol{y})$ in Eq. (5.10) should be an accurate approximation of the target kernel $k(\boldsymbol{x}, \boldsymbol{y})$ for input vectors $\boldsymbol{x}, \boldsymbol{y}$ drawn from this unknown data distribution $P(\boldsymbol{x})$. Therefore, we consider the following *integrated mean squared error* as our objective function:

$$\int \int \mathbb{E}\left[\left(k(\boldsymbol{x}, \boldsymbol{y}) - \hat{k}(\boldsymbol{x}, \boldsymbol{y})\right)^2\right] dP(\boldsymbol{x}) dP(\boldsymbol{y}) \tag{5.11}$$

$$= \int \int \underbrace{\mathbb{V}[\hat{k}(\boldsymbol{x}, \boldsymbol{y})]}_{\text{variance}} dP(\boldsymbol{x}) dP(\boldsymbol{y}) + \int \int \underbrace{\left(k(\boldsymbol{x}, \boldsymbol{y}) - \mathbb{E}\left[\hat{k}(\boldsymbol{x}, \boldsymbol{y})\right]\right)^2}_{\text{bias}^2} dP(\boldsymbol{x}) dP(\boldsymbol{y}) \tag{5.12}$$

where the expectation $\mathbb{E}[\cdot]$ and variance $\mathbb{V}[\cdot]$ are taken with respect to the random feature maps in the approximate kernel in Eq. (5.10), and the identity follows from the standard bias-variance decomposition.

We study the variance and bias terms in Eq. (5.12). Let $\delta[D_n > 0]$ be an indicator such that $\delta[D_n > 0] = 1$ if $D_n > 0$ and $\delta[D_n > 0] = 0$ otherwise. Using this indicator, and since the $p$ random feature maps $\Phi_1, \ldots, \Phi_p$ in Eq. (5.10) are statistically independent, the variance term in Eq. (5.12) can be written as

$$\mathbb{V}\left[\hat{k}(\boldsymbol{x}, \boldsymbol{y})\right] = \sum_{n=1}^{p} \delta[D_n > 0] \; a_n^2 \mathbb{V}\left[\Phi_n(\boldsymbol{x})^\top \overline{\Phi_n(\boldsymbol{y})}\right]. \tag{5.13}$$

Each individual term $\mathbb{V}[\Phi_n(\boldsymbol{x})^\top \overline{\Phi_n(\boldsymbol{y})}]$ in Eq. (5.13) is the variance of the approximate kernel $\hat{k}_n(\boldsymbol{x}, \boldsymbol{y}) := \Phi_n(\boldsymbol{x})^\top \overline{\Phi_n(\boldsymbol{y})}$ for approximating the polynomial kernel $k_n(\boldsymbol{x}, \boldsymbol{y}) := (\boldsymbol{x}^\top \boldsymbol{y})^n$ of degree $n = 1, \ldots, p$. Therefore, one can explicitly compute $\mathbb{V}[\Phi_n(\boldsymbol{x})^\top \overline{\Phi_n(\boldsymbol{y})}]$ for any given $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ using the variance formulas derived Chapter 3. For the convenience of the reader, we summarize the variance formulas for

| Sketch | Variance |
|---|---|
| Real Gaussian | $D^{-1}\left[\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + 2(\boldsymbol{x}^\top\boldsymbol{y})^2\right)^n - (\boldsymbol{x}^\top\boldsymbol{y})^{2n}\right]$ |
| Complex Gaussian | $D^{-1}\left[\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top\boldsymbol{y})^2\right)^n - (\boldsymbol{x}^\top\boldsymbol{y})^{2n}\right]$ |
| Real Rademacher | $D^{-1}\left[\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + 2\left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{k=1}^d x_k^2 y_k^2\right)\right)^n - (\boldsymbol{x}^\top\boldsymbol{y})^{2n}\right]$ |
| Complex Rademacher | $D^{-1}\left[\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{k=1}^d x_k^2 y_k^2\right)^n - (\boldsymbol{x}^\top\boldsymbol{y})^{2n}\right]$ |
| Real stacked TensorSRHT | Real Rademacher Variance<br>$-\frac{c(D,d)}{D^2}\left[(\boldsymbol{x}^\top\boldsymbol{y})^{2n} - \left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{1}{d-1}\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top\boldsymbol{y})^2 - 2\sum_{k=1}^d x_k^2 y_k^2\right)\right)^n\right]$ |
| Comp. stacked TensorSRHT | Complex Rademacher Variance<br>$-\frac{c(D,d)}{D^2}\left[(\boldsymbol{x}^\top\boldsymbol{y})^{2n} - \left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{1}{d-1}\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 - \sum_{k=1}^d x_k^2 y_k^2\right)\right)^n\right]$ |
| Conv. Sur. TensorSRHT | $\begin{cases} D^{-1}\left(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}\right) & \text{if } \mathrm{Cov}_q^{(n)} > 0 \text{ or } D > d, \\ D^{-1}\left(V_q^{(n)} - \mathrm{Cov}_q^{(n)}\right) + \mathrm{Cov}_q^{(n)} & \text{otherwise.} \end{cases}$ |
| (Real case: $q = 1$) | $V_q^{(n)} = \left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + ((2q-1)^2+1)((\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{k=1}^d x_k^2 y_k^2)\right)^n - (\boldsymbol{x}^\top\boldsymbol{y})^{2n}$ |
| (Complex case: $q = 1/2$) | $\mathrm{Cov}_q^{(n)} = \left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1}\right)^n - (\boldsymbol{x}^\top\boldsymbol{y})^{2n}$ |

Table 5.1: Closed-form expressions for the variance $\mathbb{V}\left[\Phi_n(\boldsymbol{x})^\top\overline{\Phi_n(\boldsymbol{y})}\right]$ for different random feature maps $\Phi_n : \mathbb{R}^d \to \mathbb{C}^D$ to approximate polynomial kernel of order $n \in \mathbb{N}$. Here, $D \in \mathbb{N}$ is the number of random features and $c(D,d) := \lfloor D/d \rfloor d(d-1) + (D \mod d)(D \mod d-1)$. See Chapter 3 for details and more generic results. We also show convex surrogate functions in Eq. (D.4) and Eq. (D.5) for the variance of stacked TensorSRHT derived in Appendix D.1.

specific cases in Table 5.1[1]. Regarding the bias term in Eq. (5.12), the expectation of the approximate kernel (5.10) is given by

$$\mathbb{E}\left[\hat{k}(\boldsymbol{x},\boldsymbol{y})\right] = \sum_{n=0}^{p} \delta[D_n > 0]\; a_n(\boldsymbol{x}^\top\boldsymbol{y})^n, \tag{5.14}$$

since $\mathbb{E}\left[\Phi_n(\boldsymbol{x})^\top\overline{\Phi_n(\boldsymbol{y})}\right] = (\boldsymbol{x}^\top\boldsymbol{y})^n$ for $n = 1, \ldots, p$ with $D_n > 0$.

Note that the integrals in Eq. (5.12) with respect to $P$ are not available in practice, as $P$ is the unknown data distribution. We instead assume that an i.i.d. sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ of size $m \in \mathbb{N}$ from $P$ is available. This sample may be a subsample of a larger dataset from $P$. For example, in a supervised learning problem, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ may be a random subsample of training input points.

Using the i.i.d. sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$, the objective function in Eq. (5.12) can then

[1]We omit the variance formulas of CtR-sketches (Chapter 4), since the contents of Chapter 4 were developed after we developed the contents of this chapter. CtR-sketches are thus not included in the empirical evaluations that follow. We also only use stacked TensorSRHT here because we are able to derive a convex surrogate function for it.

be unbiasedly approximated in a U-statistics form as

$$\frac{1}{m(m-1)}\sum_{i\neq j}\mathbb{V}[\hat{k}(\boldsymbol{x}_i,\boldsymbol{x}_j)]+\frac{1}{m(m-1)}\sum_{i\neq j}\left(k(\boldsymbol{x}_i,\boldsymbol{x}_j)-\mathbb{E}[\hat{k}(\boldsymbol{x}_i,\boldsymbol{x}_j)]\right)^2$$

$$=\frac{1}{m(m-1)}\sum_{n=1}^{p}\delta[D_n>0]\;a_n^2\sum_{i\neq j}\mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top\overline{\Phi_n(\boldsymbol{x}_j)}\right] \tag{5.15}$$

$$+\frac{1}{m(m-1)}\sum_{i\neq j}\left(k(\boldsymbol{x}_i,\boldsymbol{x}_j)-\sum_{n=0}^{p}\delta[D_n>0]\;a_n(\boldsymbol{x}_i^\top\boldsymbol{x}_j)^n\right)^2, \tag{5.16}$$

$$=:g(p,(D_n)_{n=1}^p) \tag{5.17}$$

where we used Eq. (5.13) and Eq. (5.14).

Finally, we formulate our optimization problem. To make the problem tractable, we search for the degree $p$ of the approximate kernel in Eq. (5.10) from the range $\{p^*_{\min},p^*_{\min}+1,\ldots,p^*_{\max}\}$, where $p^*_{\min},p^*_{\max}\in\mathbb{N}$ with $p^*_{\min}<p^*_{\max}$ are lower and upper bounds of $p$ selected by the user. We then define our optimization problem as follows:

$$\begin{aligned}\min_{p,(D_n)_{n=1}^p}\quad & g(p,(D_n)_{n=1}^p), \qquad (5.18)\\ \text{subject to:}\quad & p\in\{p^*_{\min},p^*_{\min}+1,\ldots,p^*_{\max}\},\ D_n\in\{0,\ldots D_{\text{total}}\},\\ & \sum_{n=1}^{p}D_n=D_{\text{total}},\ \text{and } D_n\geq 1 \text{ if and only if } a_n>0\quad (n=1,\ldots,p),\end{aligned}$$

where $g(p,(D_n)_{n=1}^p)$ is defined in Eq. (5.17).

To present our approach to solving Eq. (5.18), we will first define a simplified optimization problem and describe an algorithm for solving it. We will then use this simplified problem and its solver to develop a solver for the full problem in Eq. (5.18).

### Solving a Simplified Problem

We consider a simplified problem of Eq. (5.18) in which the polynomial degree $p\in\mathbb{N}$ is fixed and given, and the number of random features $D_n$ is positive, $D_n\geq 1$, for every polynomial degree $n=1,\ldots,p$ with $a_n>0$. Note that the bias term of the objective function $g(p,(D_n)_{n=1}^n)$, i.e. Eq. (5.16), only depends on $(D_n)_{n=1}^n$ through the indicator function $\delta[D_n>0]$. Therefore, under the constraint that $D_n\geq 1$ for all $n=1,\ldots,p$ with $a_n>0$, Eq. (5.16) becomes constant with respect to $(D_n)_{n=1}^p$.

Thus, the optimization problem Eq. (5.18) under the additional constraint of $p$ being fixed and $D_n\geq 1$ for all $n=1,\ldots,p$ with $a_n>0$ is equivalent to the

following optimization problem:

$$\begin{aligned} \min_{(D_n)_{n=1}^p} \quad & \frac{1}{m(m-1)} \sum_{n=1}^{p} a_n^2 \sum_{i \neq j} \mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right] \\ \text{subject to} \quad & D_n \in \{0, \dots D_{\text{total}}\}, \quad \sum_{n=1}^{p} D_n = D_{\text{total}}, \\ & D_n \geq 1 \text{ if and only if } a_n > 0 \quad (n = 1, \dots, p). \end{aligned} \tag{5.19}$$

This is a discrete optimization problem with one equality constraint, and it is an instance of the so-called *Resource Allocation Problem* (Floudas and Pardalos, 2009).

We discuss properties of the objective function in Eq. (5.19) and describe a solver. To this end, we first consider the case where $\Phi_n : \mathbb{R}^d \to \mathbb{C}^{D_n}$ is one of the unstructured polynomial sketches in Chapter 3; we will later explain its extension to structured sketches from Section 3.5. In this case, we have $\mathbb{V}\left[\Phi_n(\boldsymbol{x})^\top \overline{\Phi_n(\boldsymbol{y})}\right] = C_{\boldsymbol{x},\boldsymbol{y}}^{(n)}/D_n$ for a constant $C_{\boldsymbol{x},\boldsymbol{y}}^{(n)}$ depending on $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and the polynomial degree $n \in \mathbb{N}$ but not on $D_n$, as summarized in Table 5.1. Therefore,

$$a_n^2 \sum_{i \neq j} \mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right] = \frac{a_n^2}{D_n} \sum_{i \neq j} C_{\boldsymbol{x}_i,\boldsymbol{x}_j}^{(n)} \tag{5.20}$$

is convex and monotonically decreasing with respect to $D_n$. From this property, one can use the *Incremental Algorithm* (Floudas and Pardalos, 2009, p. 384) to directly solve the optimization problem (5.19).

Algorithm 4 describes the Incremental Algorithm for solving the simplified problem in Eq. (5.19). At every iteration, the algorithm finds $n \in \{1, \dots, p\}$ such that adding one more feature to the feature map $\Phi_n$ (i.e., $D_n = D_n + 1$) decreases the objective function most, and sets $D_n = D_n + 1$. Note again that a closed form expression for $\mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right]$ is available from Table 5.1.

**Time and space complexities.** The time and space complexities of Algorithm 4 are $\mathcal{O}(pD_{\text{total}})$ and $\mathcal{O}(p)$, respectively. Note that from Eq. (5.20), the objective function can be written as

$$f(D_1, \dots, D_p) := \sum_{n=1}^{p} a_n^2 \sum_{i \neq j} \mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right] = \sum_{n=1}^{p} \frac{a_n^2}{D_n} \sum_{i \neq j} C_{\boldsymbol{x}_i,\boldsymbol{x}_j}^{(n)}$$

with $a_n$ and $C_{\boldsymbol{x}_i,\boldsymbol{x}_j}^{(n)}$ not depending on the optimizing variable $D_n$. Therefore, one can precompute the term $\sum_{i \neq j} C_{\boldsymbol{x}_i,\boldsymbol{x}_j}^{(n)}$ for each $n = 1, \dots, p$ before starting the iterations in Algorithm 4, and during the iterations one can use the precomputed values of $\sum_{i \neq j} C_{\boldsymbol{x}_i,\boldsymbol{x}_j}^{(n)}$. Thus, while the complexity of precomputing $\sum_{i \neq j} C_{\boldsymbol{x}_i,\boldsymbol{x}_j}^{(n)}$ is $\mathcal{O}(m^2)$, where $m$ is size of the dataset $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$ defining the objective function (5.19), the time and space complexities of Algorithm 4 do not depend on $m$.

**Algorithm 4:** Incremental Algorithm

**Result:** Optimal solution $D_1, \dots, D_p \geq 1$ to the optimization problem (5.19).

**Input:** Dot product kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n=0}^{\infty} a_n (\boldsymbol{x}^\top \boldsymbol{y})^n$ with $a_n \geq 0$, truncation order $p \in \mathbb{N}$, the total number of random features $D_{\text{total}} \in \mathbb{N}$ ;

Initialize $D_1 = \cdots = D_p = 1$ and $t = 0$ ;

Let $f(D_1, \dots, D_p) := \sum_{n=1}^{p} a_n^2 \sum_{i \neq j} \mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right]$.

**while** $t < D_{\text{total}}$ **do**

  Find $j^* = \arg\min_{j \in \{1,\dots,p\}} f(D_1, \dots, D_j + 1, \dots, D_p)$ ;

  $D_{j*} = D_{j*} + 1$ ;

  $t = t + 1$ ;

**end**

**Structured case.** We assumed here that $\Phi_n$ is one of the unstructured sketches studied in Chapter 3. This choice of $\Phi_n$ makes Eq. (5.20) convex and monotonically decreasing with respect to $D_n$, which enables the Incremental Algorithm to solve the optimization problem in Eq. (5.19).

However, if $\Phi_n$ is a structured sketch (i.e., either real or complex TensorSRHT) in Section 3.5, Eq. (5.20) is not convex with respect to $D_n$, and the Incremental Algorithm is not directly applicable. To overcome this problem, we propose to use *stacked TensorSRHT* along with the convex surrogate functions in Eq. (D.4) and Eq. (D.5) derived in Appendix D.1 to replace $\mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right]$ in the objective function (5.19), and then apply the Incremental Algorithm. We summarize the concrete form of the convex surrogate function in Table 5.1. For details, see Appendix D.1.

### Solving the Full Problem

We now address the full problem in Eq. (5.18) using Algorithm 4 developed for the simplified problem in Eq. (5.19). Recall that, by fixing $p \in \{p^*_{\min}, \dots, p^*_{\max}\}$ and constraining $D_n \geq 1$ for all $n = 1, \dots, p$, the full problem in Eq. (5.18) becomes equivalent to the simplified problem in Eq. (5.19), which can be solved by Algorithm 4. Thus, we propose to solve the full problem in Eq. (5.18) by i) first performing Algorithm 4 for each $p \in \{p_{\min}, \dots, p_{\max}\}$, ii) then evaluate each solution $(D_n)_{n=1}^p$ by computing the objective function $g(p, (D_n)_{n=1}^p)$ in Eq. (5.17), and finally pick up $p$ that gives the smallest objective function value.

Algorithm 5 summarizes the whole procedure for solving the full optimization problem in Eq. (5.18). Algorithm 5 returns the optimal truncation order $p^* \in \{p_{\min}, \dots, p_{\max}\}$ with the corresponding feature cardinalities $D^* = (D_1, \dots, D_{p^*})$. Given these values, one can construct a feature map as summarized in Algorithm 6. Note that the U-statistics in the empirical objective (5.17) can be precomputed for

**Algorithm 5:** Extended Incremental Algorithm

**Result:** Optimal polynomial degree $p^* \in \{p_{\min}, \dots, p_{\max}\}$ and feature cardinalities $D^* = (D_1, \dots, D_{p^*}) \in \mathbb{N}^{p^*}$ to the full optimization problem (5.18).

**Input:** Dot product kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n=0}^{\infty} a_n (\boldsymbol{x}^\top \boldsymbol{y})^n$ with $a_n \geq 0$, upper and lower bounds $p_{\min}, p_{\max} \in \mathbb{N}$, the total number of random features $D_{\text{total}} \in \mathbb{N}$ ;

Set $g^* = \infty$, $p^* = p_{\min}$ and $D^* = \{\}$ ;

**forall** $p \in \{p_{\min}, \dots, p_{\max}\}$ **do**

  Solve Algorithm 4 to obtain $D_1, \dots, D_p$ ;

  Compute $g(p, (D_n)_{n=1}^p)$ in Eq. (5.17) ;

  **if** $g(p, (D_n)_{n=1}^p) < g^*$ **then**

    $g^* = g(p, (D_n)_{n=1}^p)$ ;

    $D^* = (D_n)_{n=1}^p$ ;

    $p^* = p$ ;

**end**

**Algorithm 6:** Improved Random Maclaurin (RM) Features

**Result:** Feature map $\Phi(\boldsymbol{x}) \in \mathbb{C}^{D_{\text{total}}+1}$

**Input:** Dot product kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n=0}^{\infty} a_n (\boldsymbol{x}^\top \boldsymbol{y})^n$ with $a_n \geq 0$, polynomial degree $p^* \in \mathbb{N}$ and feature cardinalities $D_1, \dots, D_{p^*}$ from Algorithm 5 ;

Initialize $\Phi(\boldsymbol{x}) := [\sqrt{a_0}]$

**forall** $n \in \{1, \dots, p^*\}$ **do**

  Let $\Phi_n(\boldsymbol{x}) \in \mathbb{C}^{D_n}$ be an unbiased polynomial sketch of degree $n$ with $D_n$ features (see Chapter 3) ;

  Append $\sqrt{a_n}\, \Phi_n(\boldsymbol{x})$ to $\Phi(\boldsymbol{x})$ ;

**end**

all $p_{\min}, \dots, p_{\max}$ *before* running any optimization algorithm. They do not need to be re-evaluated for every execution of Algorithm 4.

### 5.3.1 Approximating a Gaussian Kernel

Here we describe how to adapt Algorithm 4 and Algorithm 5 for approximating a Gaussian kernel of the form $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/(2l^2))$ with $l > 0$. By Eq. (5.5), this Gaussian kernel can be written as

$$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x}\|^2}{2l^2}\right) \exp\left(-\frac{\|\boldsymbol{y}\|^2}{2l^2}\right) \sum_{n=0}^{\infty} a_n (\boldsymbol{x}^\top \boldsymbol{y})^n,$$

where $a_n := 1/(n!l^{2n})$ for $n \in \mathbb{N} \cup \{0\}$. Notice that $\left(-\frac{\|\boldsymbol{x}\|^2}{2l^2}\right)$ and $\left(-\frac{\|\boldsymbol{y}\|^2}{2l^2}\right)$ are scalar values and can be computed for any given input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$.

Thus, the objective function $g(p, (D_n)_{n=1}^p)$ in Eq. (5.17), which is an empirical approximation of the bias-variance decomposition of the mean squared error in Eq. (5.12) using an i.i.d. sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \overset{i.i.d.}{\sim} P$, can be adapted as

$$
\begin{aligned}
& g(p, (D_n)_{n=1}^p) \qquad (5.21)\\
=& \frac{1}{m(m-1)} \sum_{n=1}^{p} \delta[D_n > 0]\ a_n^2 \sum_{i \neq j} \exp\left(-\frac{\|\boldsymbol{x}_i\|^2}{l^2}\right) \exp\left(-\frac{\|\boldsymbol{x}_j\|^2}{l^2}\right) \mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right]\\
+& \frac{1}{m(m-1)} \sum_{i \neq j} \left(k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{n=0}^{p} \delta[D_n > 0]\ a_n \exp\left(-\frac{\|\boldsymbol{x}_i\|^2}{2l^2}\right) \exp\left(-\frac{\|\boldsymbol{x}_j\|^2}{2l^2}\right) (\boldsymbol{x}_i^\top \boldsymbol{x}_j)^n\right)^2.
\end{aligned}
$$

Accordingly, the objective function of the simplified problem in Eq. (5.19) is adapted as

$$
f(D_1, \ldots, D_p) := \frac{1}{m(m-1)} \sum_{n=1}^{p} a_n^2 \sum_{i \neq j} \exp\left(-\frac{\|\boldsymbol{x}_i\|^2}{l^2}\right) \exp\left(-\frac{\|\boldsymbol{x}_j\|^2}{l^2}\right) \mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right].
$$

By these modifications, Algorithm 4 and Algorithm 5 can be used to obtain the optimal truncation order $p^* \in \{p_{\min}, \ldots, p_{\max}\}$ and the corresponding feature cardinalities $D_1, \ldots, D_{p^*}$. Lastly, Algorithm 6 can be adapted by multiplying the scalar value $\exp\left(-\frac{\|\boldsymbol{x}\|^2}{2l^2}\right)$ to the feature map $\Phi(\boldsymbol{x})$ obtained from Algorithm 6: the new feature map is defined as $\Phi'(\boldsymbol{x}) := \exp\left(-\frac{\|\boldsymbol{x}\|^2}{2l^2}\right) \Phi(\boldsymbol{x})$.

### 5.3.2 Numerical Illustration of the Objective Function

To gain an insight about the behavior of Algorithm 5, we provide a numerical illustration of the bias and variance terms in the objective function $g(p, (D_n)_{n=1}^p)$ in Eq. (5.17) (or its version adapted for the Gaussian kernel in Eq. (5.21)). To this end, we used the Fashion MNIST dataset (Xiao et al., 2017) and randomly sampled data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ with $m = 500$ from the entire dataset of size $60,000$. As a target kernel to approximate, we consider (i) a polynomial kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y}/8 + 7/8)^{20}$ of degree $p = 20$; and (ii) the Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/(2l^2))$, where the length scale $l > 0$ is given by the median heuristic (Garreau et al., 2017), i.e., the median of the pairwise Euclidean distances of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$.

For the polynomial kernel (i), we computed (a) $\frac{a_n^2}{m(m-1)} \sum_{i \neq j} \mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right]$ for each $n = 1, \ldots, p\ (= 20)$, which is the variance component of the objective function in Eq. (5.17); and (b) $\frac{1}{m(m-1)} \sum_{i \neq j} \left(k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{\nu=0}^{n} a_\mu (\boldsymbol{x}_i^\top \boldsymbol{x}_j)^\nu\right)^2$ for each $n = 1, \ldots, p\ (= 20)$, which is the bias component in Eq. (5.17) computed up to

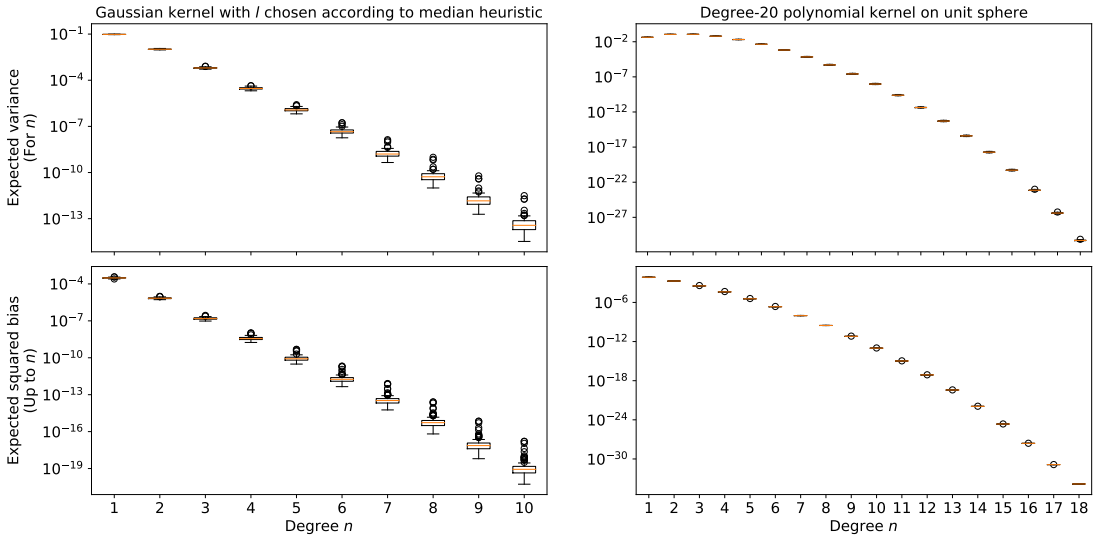



Figure 5.1: Numerical illustration of Section 5.3.2. The left two figures are box plots for the Gaussian kernel (i), and the right two figures are those for the polynomial kernel (ii). The top figures show the variance terms (a), and the bottom figures show the bias terms (b). See Section 5.3.2 for details.

$n$-th order. For the Gaussian kernel (ii), we computed corresponding quantities from the objective function in Eq. (5.21):
(a) $\frac{a_n^2}{m(m-1)} \sum_{i \neq j} \exp\left(-\frac{\|\boldsymbol{x}_i\|^2}{l^2}\right) \exp\left(-\frac{\|\boldsymbol{x}_j\|^2}{l^2}\right) \mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right]$ and
(b) $\frac{1}{m(m-1)} \sum_{i \neq j} \left(k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{\nu=0}^{n} a_\nu \exp\left(-\frac{\|\boldsymbol{x}_i\|^2}{2l^2}\right) \exp\left(-\frac{\|\boldsymbol{x}_j\|^2}{2l^2}\right) (\boldsymbol{x}_i^\top \boldsymbol{x}_j)^\nu\right)^2$
for $n = 1, \ldots, 10$.

We used the real Gaussian sketch for the feature map $\Phi_n$, for which Eq. (3.8) in Chapter 3 gives a closed form expression of the variance $\mathbb{V}\left[\Phi_n(\boldsymbol{x}_i)^\top \overline{\Phi_n(\boldsymbol{x}_j)}\right]$; see also Table 5.1. We set $D_n = 1$ for each $n$ to be evaluated (i.e., $\Phi_n(\boldsymbol{x}) \in \mathbb{R}$.)

To compute the means and standard deviations of the above quantities (a) and (b), we repeated this experiment 100 times by independently subsampling $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ with $m = 500$ from the entire dataset each time. Fig. 5.1 describes the results. First, we can see that the standard deviations of the quantities (a) and (b) are relatively small, and thus a subsample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ of size $m = 500$ is sufficient for providing accurate approximations of the respective population quantities of (a) and (b) (where the empirical average with respect to $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ is replaced by the corresponding expectation) in this setting.

Regarding the polynomial kernel (i), the variance terms (a) for polynomial degrees up to $n = 3$ have similar magnitudes, and they decay exponentially fast for polynomial degrees larger than $n = 3$ (notice that the vertical axis of the plot is in log scale). On the other hand, the bias term (b) decays exponentially fast as the polynomial degree $n$ increases. These trends suggest that Algorithm 5

would assign more features to lower order degrees $n$, in particular to the degree 3 or less. One explanation of these trends is that the parametrization of the kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\top \boldsymbol{y}/8 + 7/8)^{20}$ gives larger coefficients to lower polynomial degrees in the the Maclaurin expansion (see Eq. (5.3)), and that the distribution of pairwise inner products of the data points $\boldsymbol{x}_1, \ldots \boldsymbol{x}_m$ is centered around zero in this experiment.

Regarding the Gaussian kernel (ii), both the variance term (a) and the bias term (b) decay exponentially fast as the polynomial degree $n$ increases. This trend suggests that Algorithm 5 would assign more features to lower order polynomial degrees $n$.

To summarize, these observations suggest that, to minimize the mean squared error of the approximate kernel, it is more advantageous to assign more features to lower degree polynomial approximations. Algorithm 5 automatically achieves such feature assignments.

We will show later in this chapter for the Gaussian kernel that Algorithm 5 assigns more random features to *higher* polynomial degrees instead when the length scale $l > 0$ is small in comparison to the scaling of the data, i.e., when modelling high-frequency data and choosing $l$ *without* the median heuristic. This is because the variances of approximate polynomial kernels of high degrees increase in this case. In the following experiments we stick to the median heuristic for now.

### 5.3.3 Gaussian Process Regression Toy Example

We performed a toy experiment on one-dimensional Gaussian process (GP) regression, whose results are described in Fig. 5.2. The purpose is to gain a qualitative understanding of the optimized Maclaurin approximation in Section 5.3 (Algorithm 5). For comparison, we also used Random Fourier Features (RFF) of Rahimi and Recht (2007) described in Chapter 2 in this experiment. We use the real Rademacher sketch in the optimized Maclaurin approach.

We define the ground-truth function as a sinc function, $f(x) = \sin(ax)/x$, with $a > 0$ for which we consider three settings: $a \in \{5, 2, 0.5\}$. We generated training data by adding independent Gaussian noise of variance $\sigma^2_{\text{noise}} = 0.01$ to the ground-truth function $f(x)$. With this value of noise variance $\sigma^2_{\text{noise}}$, we then fit a GP regressor (see Chapter 1) using the Gaussian kernel $k(x, y) = \sigma^2 \exp(-(x - y)^2/(2l^2))$ to the training data, where we determined the hyperparameters $l, \sigma^2 > 0$ by maximizing the log marginal likelihood (e.g., Rasmussen and Williams, 2006, Chapter 2). We used the resulting posterior GP as a ground-truth and call it "full GP", treating it as a reference for assessing the quality of approximate GPs. As such, we used the same hyperparameters as the full GP in approximate GPs; this enables evaluating the effects of the approximation in the resulting GP predictive distributions.

We set the number of random features as $D = 10$. In this case, the optimized Maclaurin approach in Algorithm 5 selects the truncation degree $p^* = 9$ and simply

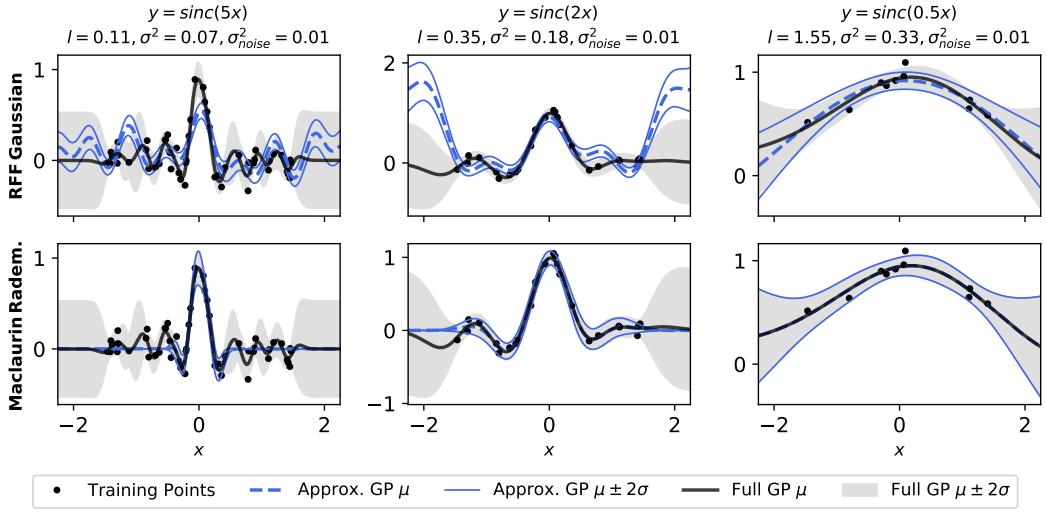



Figure 5.2: One-dimensional GP regression experiment in Section 5.3.3. The top row are the results of random Fourier features (Gaussian RFF) and the bottom row are those of the optimized Maclaurin approach. The left, middle, and right columns correspond to the ground-truth sinc functions with frequency 5, 2, and 0.5, respectively. The values of $l$ and $\sigma^2$ are the kernel hyperparameters obtained by maximizing the log likelihood of training data in the full GP (i.e., without approximation). Dashed blue curves represent approximate GP posterior mean functions; blue curves represent the posterior means plus and minus 2 times approximate posterior standard deviations; black curves represent the posterior mean functions of the full GP; and the shaded ares are the full GP posterior means plus and minus 2 times the full GP posterior deviations.

allocates the feature cardinalities as $D_1 = \cdots = D_9 = 1$. (Note that one feature is always allocated to the degree $n = 0$). This behavior is expected because the variance of the Rademacher sketch (see Table 5.1) is zero for all polynomial degrees $n$, as the input dimension is one ($d = 1$) in this experiment.[2]

We can make the following observations from Fig. 5.2. First, with the optimized Maclaurin approach, the approximate GP posterior mean function approximates the full GP posterior mean function around $x = 0$ more accurately than RFF. Moreover, the range of $x$ on which the Maclaurin approach is accurate becomes wider for a lower frequency $a$ of the ground-truth sinc function (for which the length scale $l$ is larger). This tendency suggests that the Maclaurin approach may be more advantageous than RFF in approximating around $x = 0$ and when the length scale $l$ is relatively large. Experiments in the next section, in particular those with high dimensional datasets, provide a further support for this observation.

One issue with the Maclaurin approximation is that, as can be seen from

[2]Thus, the error of the optimized Maclaurin approach stems solely from the finite truncation of the Maclaurin expansion in Eq. (5.10).

Fig. 5.2, the approximate GP posterior tends to collapse for an input location $x$ far from 0. We will study this issue at the end of this chapter, and show that the collapsing GP posterior approximation is due to the vanishing of the approximate kernel $\hat{k}(\boldsymbol{x}, \boldsymbol{y})$ when $\|\boldsymbol{x}\| / l + \|\boldsymbol{y}\| / l$ becomes large. We will also propose a cure for this pathology that is relevant in particular when modelling data of high frequency, i.e., when $l$ is short relative to the scaling of the data.

For now, we focus on a set of high-dimensional experiments, where we can make use of the median heuristic and approximations are well-behaved.

## 5.4 Empirical Evaluation of the Optimized Maclaurin Method

In this section, we perform systematic experiments to evaluate the optimized Maclaurin method proposed in this chapter. We use it in conjunction with the real and complex polynomial sketches introduced in Chapter 3, and we consider approximations of both polynomial kernels and Gaussian kernels. We do not make use of the Complex-to-Real sketches presented in Chapter 4 here, since we developed them independently from the methods discussed in this chapter, as mentioned at the beginning of this chapter.

We evaluate the performance of each approximation approach in terms of both i) the accuracy in kernel approximation and ii) the performance in downstream tasks. The downstream tasks we consider are Gaussian process regression and classification. For completeness, we explain how to use complex-valued random features in Gaussian process inference and discuss the resulting computational costs in Appendix B.3.

In Section 5.4.1, we first explain the setup of the experiments. In Section 5.4.2, we describe experiments on polynomial kernel approximation, comparing various approximation approaches. In Section 3.5.4, we report the results of wall-clock time comparison of real and complex random features, focusing on the downstream task performance of GP classification. In Section 5.4.3, we present detailed evaluations of the optimized Maclaurin approach for polynomial and Gaussian kernel approximations in GP classification and regression.

### 5.4.1 Experimental Setup

We explain here the common setup for the experiments in this section.

#### Datasets

Table 5.2 shows an overview of the datasets used in the experiments. All the datasets come from the UCI benchmark (Dua and Graff, 2017) except for Cod_rna (Uzilov et al., 2006), FashionMNIST (Xiao et al., 2017), and MNIST (Lecun et al.,

| Classification | Num. data points $N$ | Dimensionality $d$ | Regression | Num. data points $N$ | Dimensionality $d$ |
|---|---|---|---|---|---|
| Cod_rna | 331,152 | 8 | Boston | 506 | 16 |
| Covertype | 581,012 | 64 | Concrete | 1,030 | 8 |
| FashionMNIST | 70,000 | 1,024 | Energy | 768 | 8 |
| Magic | 19,020 | 16 | kin8nm | 8,192 | 8 |
| Miniboo | 130,064 | 64 | Naval | 11,934 | 16 |
| MNIST | 70,000 | 1,024 | Powerplant | 9,568 | 4 |
| Mocap | 78,095 | 64 | Protein | 45,730 | 16 |

Table 5.2: Datasets used in the experiments. The left and right columns are datasets for classification and regression, respectively.

1998). We pad input vectors with zeros so that the input dimensionality $d$ becomes a power of two to support Hadamard projections in TensorSRHT. The train/test split is 90/10 and is recomputed for every random seed for the UCI datasets; otherwise it is predefined.

For each dataset, we use its random subsets of size $m = \min(5000, N_{\text{train}})$ and $m_* = \min(5000, N_{\text{test}})$ to define training and test data in an experiment, respectively, where $N_{\text{train}}$ and $N_{\text{test}}$ are the sizes of the original training and test datasets. Denote by $X_{\text{sub}} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ and $X_{*,\text{sub}} = \{\boldsymbol{x}_{*,1}, \ldots, \boldsymbol{x}_{*,m_*}\}$ those subsets for training and test, respectively. We repeat each experiment 10 times independently using 10 different random seeds, and hence with 10 different subset partitions.

### Target Kernels to Approximate

We consider the approximation of (i) polynomial kernels and (ii) Gaussian kernels.

**(i) Polynomial kernel approximation.** We consider a polynomial kernel of the form

$$k(\boldsymbol{x}, \boldsymbol{y}) = \sigma^2 \left( \left(1 - \frac{2}{a^2}\right) + \frac{2}{a^2} \boldsymbol{x}^\top \boldsymbol{y} \right)^p = \sigma^2 \left(1 - \frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{a^2}\right)^p \tag{5.22}$$

with $p \in \mathbb{N}$, $a \geq 2$, $\sigma^2 > 0$, and $\|\boldsymbol{x}\| = \|\boldsymbol{y}\| = 1$. We choose this form of polynomial kernels because we use *Spherical Random Features (SRF)* of Pennington et al. (2015) as one of our baselines, and because SRF approximates the polynomial kernel in Eq. (5.22) defined on the unit sphere of $\mathbb{R}^d$. Following the experimental setup of Pennington et al. (2015), we set $a = 4$ and $p \in \{3, 7, 10, 20\}$ in Eq. (5.22). We set $\sigma^2$ as the variance of the labels of training subset $X_{\text{sub}}$.

To make SRF applicable, we unit-normalize the input vectors in each dataset so that they lie on the unit sphere in $\mathbb{R}^d$. In an experiment where we zero-centralize the input vectors, we unit-normalize after applying the zero-centering.

**(ii) Gaussian kernel approximation.** We consider the approximation of the Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \sigma^2 \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/(2l^2))$, where we choose the length

scale $l > 0$ by the median heuristic (Garreau et al., 2017), i.e., as the median of pairwise Euclidean distances of input vectors in the training subset $X_{\text{sub}}$; we set $\sigma^2 > 0$ as the variance of the labels of $X_{\text{sub}}$.

### Error Metrics

We define several error metrics for studying the quality of each approximation approach.

**Relative Frobenius norm error.** To define this error metric, we need to define some notation. Let $\Phi : \mathbb{R}^d \to \mathbb{C}^D$ be the (either real or complex) feature map of a given approximation method. For test input vectors $\boldsymbol{X}_{*,\text{sub}} = \{\boldsymbol{x}_{*,1}, \ldots, \boldsymbol{x}_{*,m_*}\}$, let $\hat{\boldsymbol{K}} \in \mathbb{C}^{m_* \times m_*}$ be the approximate kernel matrix such that $\hat{\boldsymbol{K}}_{i,j} = \Phi(\boldsymbol{x}_i)^\top \overline{\Phi(\boldsymbol{x}_j)}$. Similarly, let $\boldsymbol{K} \in \mathbb{R}^{m_* \times m_*}$ be the exact kernel matrix such that $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_{*,i}, \boldsymbol{x}_{*,j})$ with $k$ being the target kernel.

We then define the *relative Frobenius norm error* of $\hat{\boldsymbol{K}}$ against $\boldsymbol{K}$ as:

$$\|\boldsymbol{K} - \hat{\boldsymbol{K}}\|_F / \|\boldsymbol{K}\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m} |\boldsymbol{K}_{i,j} - \hat{\boldsymbol{K}}_{i,j}|^2} \Big/ \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m} \boldsymbol{K}_{i,j}^2}. \tag{5.23}$$

This error quantifies the quality of the feature map $\Phi$ in terms of the resulting approximation accuracy of the kernel matrix. As the target kernel matrix $\boldsymbol{K}$ is real-valued, we discard the imaginary part of $\hat{\boldsymbol{K}}$ if it is is complex-valued, unless otherwise specified.

We define other error metrics in terms of two downstream tasks: Gaussian process (GP) regression and classification (see Appendix B.3 for details of these GP tasks).

**Kullback-Leibler (KL) divergence.** We measure the *KL divergence* between two posterior predictive distributions at test input points: one is that of an approximate GP and the other is that of the exact GP without approximation; see Eq. (B.23) in Appendix B.3 for details. For GP classification, we measure the KL divergence between the corresponding latent GPs before transformation. Since there are as many GPs as the number of classes, we report the KL divergence averaged over those classes.

**Prediction performance.** For GP classification, we use the *test error rate* (i.e., the percentage of misclassified examples) for measuring the prediction performance. For GP regression, we report the *normalized mean squared error (norm. MSE)* between the posterior predictive outputs and true outputs, normalized by the variance of the test outputs. Here, we use the full training data of size $N_{\text{train}}$ for

computing the approximate GP posterior and the full test data of size $N_{\text{test}}$ for evaluating the prediction performance.[3]

**Mean negative log likelihood (MNLL).** We compute the *mean negative log likelihood (MNLL)* of the test data for the approximate GP predictive distribution. MNLL can capture the quality of prediction uncertainties of the approximate GP model (e.g. Rasmussen and Williams, 2006, p. 23). We use the full training and test data for computing the MNLL, as for the prediction performance.

#### Other Settings

**Optimized Maclaurin approach.** For the optimized Maclaurin approach in Algorithm 5, we set $p_{\min} = 2$ and $p_{\max} = 10$. We use the training subset $X_{\text{sub}} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}$ to precompute the U-statistics in Eq. (5.15) and Eq. (5.16).

**Regularization parameters.** We select the regularization parameter in GP classification and regression by a training-validation procedure. That is, we use the 90 % of training data for training and the remaining 10 % for validation, and select the regularization parameter that maximizes the MNLL on the validation set. For GP classification, we choose the regularization parameter from the range $\alpha \in \{10^{-5}, \dots, 10^{-0}\}$. For GP regression, we choose the noise variance from the range $\sigma^2_{\text{noise}} \in \{2^{-15}, \dots, 2^{15}\}$. See Appendix B.3 for the definition of these parameters.

Importantly, we perform this selection procedure using a baseline approach,[4] and after selecting the regularization parameter, we set the *same* regularization parameter for all the approaches (including our optimized Maclaurin approach) for computing error metrics. In this way, we make sure that the selected regularization parameter is not in favour of our approaches (and in this sense we give an advantage to the baseline).

### 5.4.2 Polynomial Kernel Approximation

We first study the approximation of the polynomial kernels in Eq. (5.22), comparing different polynomial sketches and the optimized Maclaurin method in terms of the relative Frobenius norm error in Eq. (5.23) on FashionMNIST. Fig. 5.3 describes the results. We consider the following polynomial sketches in this experiment:

[3] We did not use the full training and test datasets for evaluating the KL divergence, since it requires computing the exact GP posterior on the full training data of size $N_{\text{train}}$, which costs $\mathcal{O}(N^3_{\text{train}})$ and is not feasible for datasets with large $N_{\text{train}}$.

[4] More specifically, we use the Spherical Random Features (SRF) (Pennington et al., 2015) when the target kernel is a polynomial kernel, and Random Fourier Features (Rahimi and Recht, 2007) when the target kernel is Gaussian, for selecting the regularization parameter.

**(i) Rademacher Product-Sketch (Chapter 3).** We use the real Rademacher sketch, i.e., the unstructured polynomial sketch in Eq. (3.1) with Rademacher weights ("Radem." in Fig. 5.3).

**(ii) Stacked TensorSRHT (Section 3.5).** We consider the real stacked TensorSRHT in Eq. (3.27) with Rademacher weights ("TensorSRHT" in Fig. 5.3), and the complex TensorSRHT in Eq. (3.34) with complex Rademacher weights; see also Algorithm 2. We consider two versions of the complex TensorSRHT: one that keeps the imaginary part in the approximate kernel matrix ("TensorSRHT Comp. (Keep imag.)" in Fig. 5.3), and one that discards the imaginary part ("TensorSRHT Comp. (Disc. imag.)" in Fig. 5.3).

**(iii) Random Maclaurin (Section 5.2).** We use the Random Maclaurin approach by Kar and Karnick (2012) explained in Section 5.2. To improve its performance, we truncate the support of the importance sampling measure $\mu(n) = 2^{-(n+1)}$ in Eq. (5.6) to degrees $n \in \{1, \ldots, p\}$.[5] Note that the term $n = 0$ in Eq. (5.2) associated with coefficient $a_0$ does not need to be approximated, as we append $\sqrt{a_0}$ to the feature map. We consider two versions of the Random Maclaurin approach: one using the real Rademacher sketch ("Rand. Macl. Radem." in Fig. 5.3) and one using the real TensorSRHT ("Rand. Macl. TensorSRHT" in Fig. 5.3).

**(iv) Optimized Maclaurin (Section 5.3).** We consider our optimized Maclaurin approach introduced in this chapter using the real TensorSRHT ("Opt. Macl. TensorSRHT" in Fig. 5.3) and using the complex TensorSRHT. We consider two versions of the latter: one keeping the imaginary part of the approximate kernel matrix ("Opt. Macl. TensorSRHT Comp. (Keep imag. )" in Fig. 5.3) and one discarding the imaginary part ("Opt. Macl. TensorSRHT Comp. (Disc. imag. )" in Fig. 5.3).

**(v) TensorSketch** For completeness, we also include in this experiment *TensorSketch* of Pham and Pagh (2013), a state-of-the-art polynomial sketch ("TensorSketch" in Fig. 5.3).

**Setting.** We perform the experiments using FashionMNIST ("Non-centered data" in Fig. 5.3) and its centered version for which we subtract the mean of the input vectors from each input vector ("Centered data" in Fig. 5.3). For each approach, the number of random features is $D \in \{d, 2d\}$, where $d = 1,024$ for FashionMNIST.

[5] Without this restriction of the support, the randomized Maclaurin approach may sample polynomial degrees $n$ such that $n > p$ from $\mu(n)$, for which the associated coefficient in the Maclaurin expansion in Eq. (5.3) is zero. Therefore, the resulting feature maps may contain zeros, which are redundant and makes the kernel approximation inefficient.

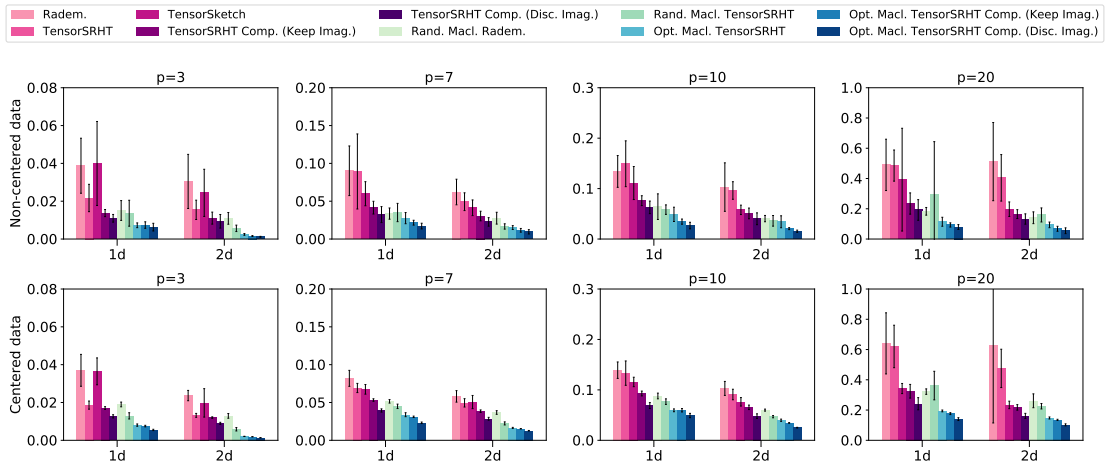



Figure 5.3: Results of the experiments in Section 5.4.2 using FashionMNIST. Each plot shows the relative Frobenius norm errors in Eq. (5.23) of different sketches for approximating the polynomial kernel in Eq. (5.22) with $p \in \{3, 7, 10, 20\}$ and $D \in \{1d, 2d\}$. The top and bottom rows show results with the data without and with zero-centring, respectively.

From the results in Fig. 5.3, we can make the following observations.

**Effectiveness of the optimization approach.** The optimized Maclaurin approach with the real TensorSRHT ("Opt. Mac. TensorSRHT") achieves smaller errors than the corresponding random Maclaurin approach ("Rand. Macl. TensorSRHT") for almost all cases, and with a large margin for $p = 3$ and $p = 20$. This improvement demonstrates the effectiveness of the proposed optimization approach that allocates more features to polynomial degrees with larger variance reduction.

**Variance reduction by complex features.** Complex TenorSRHT ("TensorSRHT Comp.") achieves significantly smaller errors than the real TensorSRHT ("TensorSRHT"), in particular for higher polynomial degrees $p$. Observe that even the complex TensorSRHT with $D = d$ achieves smaller errors than the real TensorSRHT with $D = 2d$ for the non-centered data. These improvements show the effectiveness of complex features in variance reduction, corroborating the preliminary results shown in Figures 3.2 and 3.3.

The optimized Maclaurin approach using complex features ("Opt. Macl. TensorSRHT Comp") also achieves smaller errors than the optimized Maclaurin approach using real features ("Opt. Macl. TensorSRHT"), but the improvements are relatively smaller than those for TensorSRHT. These milder improvements would be because the optimized Maclaurin approach tends to assign more features to lower degree polynomials, but the improvements for these lower degree polynomials by complex features are relatively smaller than for higher degree polynomials.

**Effectiveness of complex features on non-negative data.** The improvements by complex features are more significant for the non-centered data than those for the centered-data. The non-centered data here consist of *non-negative* input vectors, as FashionMNIST consists of such vectors. This observation agrees with the discussion in Section 3.3 in Chapter 3 suggesting that complex features yield an approximate kernel whose variance is smaller than that of real features, if the input vectors are non-negative.

**TensorSRHT v.s. TensorSketch.** While the real TensorSRHT produces larger errors than TensorSketch for all the cases except $p = 3$, the complex TensorSRHT outperforms TensorSketch for all the cases. This comparison shows the use of complex features can make TensorSRHT competitive to the state-of-the-art (and one can further improve its performance by using it in the optimized Maclaurin approach).

### 5.4.3 Systematic Evaluation of the Optimized Maclaurin Approach

Lastly, we systematically evaluate the performance of the optimized Maclaurin approach against Spherical Random Features (Pennington et al., 2015) and random Fourier features (Rahimi and Recht, 2007). We run experiments on approximate GP classification and regression on a variety of datasets, using a high-degree polynomial kernel and the Gaussian kernel.

**Optimized Maclaurin approach.** We consider the optimized Maclaurin approach in Section 5.3 using the real Rademacher sketch ("Opt. Macl. Radem."), one using the real TensorSRHT ("Opt. Macl. TensorSRHT") and its complex extension ("Opt. Macl. TensorSRHT Comp. ").

**Baselines.** We use here approximation approaches based on Random Fourier Features (RFF) (Rahimi and Recht, 2007) and their extensions such as *Spherical Random Features* (SRF) (Pennington et al., 2015) and *Structured Orthogonal Random Features* (SORF) (Yu et al., 2016) as baselines. The latter two approaches constitute the state-of-the-art and we refer the reader to Chapter 2 for an explanation.

These approaches generate a set of frequency samples $\omega_1, \ldots, \omega_{D/2} \in \mathbb{R}^d$ (suppose $D$ is even for simplicity) from a certain spectral density, and construct a

feature map[6] of dimension $D$ as, for any $\boldsymbol{x} \in \mathbb{R}^d$,

$$\Phi_{\mathcal{R}}(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \left[\cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \ldots, \cos(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}), \sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \ldots, \sin(\boldsymbol{w}_{D/2}^\top \boldsymbol{x})\right]^\top \in \mathbb{R}^D. \tag{5.24}$$

Each approach has its own way of generating the frequency samples $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_{D/2}$: the original RFF generates them in an i.i.d. manner from the spectral density of a kernel, SORF uses structured orthogonal matrices (thus we may call it "RFF Orth."), and SRF uses a certain optimized spectral density.

For a thorough comparison, we also consider a complex version of these RFF-based approaches. By generating frequency samples $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D \in \mathbb{R}^d$ in the specific way of each approach, one can define a corresponding complex feature map as, for any $\boldsymbol{x} \in \mathbb{R}^d$,

$$\Phi_{\mathcal{C}}(\boldsymbol{x}) := \sqrt{\frac{1}{D}} \left[\exp(\mathrm{i}\boldsymbol{\omega}_1^\top \boldsymbol{x}), \ldots, \exp(\mathrm{i}\boldsymbol{\omega}_D^\top \boldsymbol{x})\right]^\top \in \mathbb{C}^D. \tag{5.25}$$

One can see[7] that Eq. (5.25) is a complex version of Eq. (5.24) by defining an approximate kernel with $\Phi_{\mathcal{C}}(\boldsymbol{x})$ and taking its real part, which recovers Eq. (5.24) of dimension $2D$.

### Approximate GP Inference with a High-degree Polynomial Kernel

We first consider approximate GP classification and regression with a high-degree polynomial kernel.

**Setting.** We set the polynomial degree as $p = 20$, to make it challenging to approximate the polynomial kernel. We apply zero-centering to each dataset (i.e., we substract the mean of input vectors from each input vector), as it improves the MNLL values on most datasets (see Appendix D.2 for supplementary experiments). We evaluate all the four error metrics in Section 5.4.1, including the relative Frobenius norm error in Eq. (5.23). For each approach, the number of random features is $D \in \{d, 3d, 5d\}$ with $d$ being the dimensionality of input vectors.

[6] There is another popular version of the feature map in Eq. (5.24) defined as $\Phi_{\mathcal{R}}(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \left[\cos(\boldsymbol{w}_1^\top \boldsymbol{x} + b_1), \ldots, \cos(\boldsymbol{w}_D^\top \boldsymbol{x} + b_D)\right]^\top \in \mathbb{R}^D$ with $b_1, \ldots, b_D$ uniformly sampled on $[0, 2\pi]$. Following Sutherland and Schneider (2015) who suggested the superiority of Eq. (5.24), we use Eq. (5.24) here in all the methods using RFF, including SRF and SORF.

[7] Define an approximate kernel with Eq. (5.25) as $\hat{k}(\boldsymbol{x}, \boldsymbol{y}) := \Phi_{\mathcal{C}}(\boldsymbol{x})^\top \overline{\Phi_{\mathcal{C}}(\boldsymbol{y})} = \frac{1}{D} \sum_{i=1}^D \exp(\mathrm{i}\boldsymbol{\omega}_i^\top (\boldsymbol{x} - \boldsymbol{y})) = \frac{1}{D} \sum_{i=1}^D \exp(\mathrm{i}\boldsymbol{\omega}_i^\top \boldsymbol{x}) \overline{\exp(\mathrm{i}\boldsymbol{\omega}_i^\top \boldsymbol{y})}$. By taking its real part, we have $\mathrm{Re}\{\hat{k}(\boldsymbol{x}, \boldsymbol{y})\} = \frac{1}{D} \sum_{i=1}^D \cos(\boldsymbol{w}_i^\top (\boldsymbol{x} - \boldsymbol{y})) = \frac{1}{D} \sum_{i=1}^D \left(\cos(\boldsymbol{w}_i^\top \boldsymbol{x}) \cos(\boldsymbol{w}_i^\top \boldsymbol{y}) + \sin(\boldsymbol{w}_i^\top \boldsymbol{x}) \sin(\boldsymbol{w}_i^\top \boldsymbol{y})\right) =: \Phi_{\mathcal{R}}(\boldsymbol{x})^\top \Phi_{\mathcal{R}}(\boldsymbol{y})$, where $\Phi_{\mathcal{R}}(\boldsymbol{x}) := \sqrt{\frac{1}{D}} \left[\cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \ldots, \cos(\boldsymbol{w}_D^\top \boldsymbol{x}), \sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \ldots, \sin(\boldsymbol{w}_D^\top \boldsymbol{x})\right]^\top \in \mathbb{R}^{2D}$ is the $2D$-dim. version of Eq. (5.24).

**Baselines.** As a baseline, we use SRF (Pennington et al., 2015), a state-of-the-art approach to approximating polynomial kernels defined on the *unit sphere* in $\mathbb{R}^d$. Pennington et al. (2015) show that SRF works particularly well for approximating high degree polynomial kernels, and significantly outperforms the Random Maclaurin approach (Kar and Karnick, 2012) and TensorSketch (Pham and Pagh, 2013) for such kernels.

We also consider two other extensions of SRF for baselines. SRF generates the frequency samples $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_{D/2}$ in Eq. (5.24) from an optimized spectral density, by first drawing samples from the unit sphere in $\mathbb{R}^d$. Therefore, by replacing these samples on the unit sphere by structured orthogonal projections of SORF (Yu et al., 2016), one can construct a structured version of SRF. We use this structured SRF as another baseline ("SRF Orth." in Fig. 5.4). Moreover, we consider a complex extension of the structured SRF in the form of Eq. (5.25) ( "SRF Orth. Comp." in Fig. 5.4). While these extensions are themselves novel, we include them in the experiments, as they improve over the vanilla SRF and make the experiments more competitive.

Fig. 5.4 describes the results of approximate GP classification on four datasets from Table 5.2. We present the results on the other four datasets as well as the results of GP regression in Appendix D.2 to save the space. We can make the following observations from these results.

**Relative Frobenius norm error.** For most cases, the optimized Maclaurin approaches with TensorSRHT achieve lower relative Frobenius norm errors than the SRF approaches. Specifically, "Opt. Macl. TensorSRHT" outperforms "SRF" and "SRF Orth.", and "Opt. Macl. TensorSRHT Comp." outperforms "SRF Orth. Comp."

**KL divergence.** While the optimized Maclaurin approaches achieve lower KL divergences than the SRF approaches for most cases, the margins are smaller than those for the relative Frobenius norm errors. One possible reason is that the Maclaurin approaches in general (either random or optimized) can be inaccurate in approximating the GP posterior variances at test inputs far from $\boldsymbol{x} = \mathbf{0}$, as discussed in Section 5.3.3.

**Classification errors and mean negative log likelihood (MNLL).** The optimized Maclaurin approaches with TensorSRHT achieve equal or lower classification errors and MNLL than the SRF approaches. These results suggest that the optimized Maclaurin approaches are promising not only in kernel approximation accuracy but also in downstream task performance. Recall that we selected the regularization parameter in GP classification by maximizing the MNLL of *SRF* (on the validation set), and used the same regularization parameter in the other approaches (See Section 5.4.1). Therefore, the results of Fig. 5.4 are in favor of

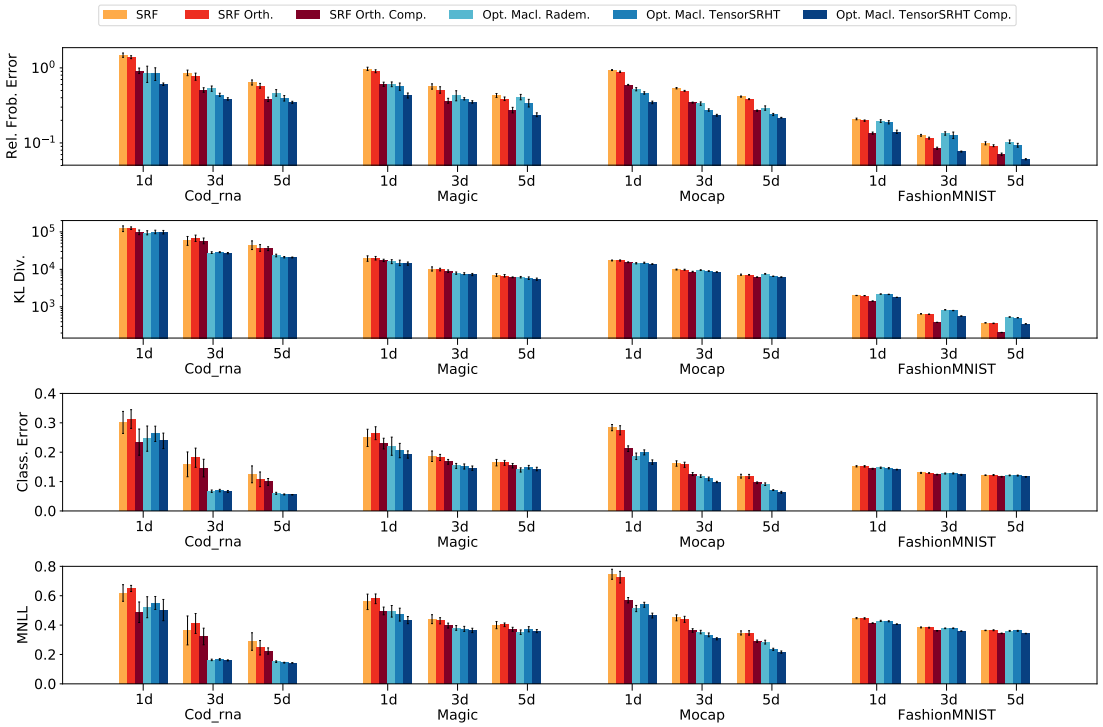



Figure 5.4: Results of the experiments in Section 5.4.3 on approximate GP classification with a high-degree polynomial kernel. Lower values are better for all the metrics. For each dataset, we show the number of random features $D \in \{1d, 3d, 5d\}$ used in each method on the horizontal axis, with $d$ being the input dimensionality of the dataset. We put the legend labels and the bars in the same order.

the SRF approaches, and the optimized Maclaurin approaches may perform even better if we choose the regularization parameter for them separately.

### Approximate GP Inference with a Gaussian kernel

We next consider GP classification using a Gaussian kernel. As in Section 5.4.3, we apply zero-centring to the input vectors of each dataset.

**Baselines.** We use RFF, SORF ("RFF Orth.") and a complex extension of SORF ("RFF Orth. Comp.") as baselines (see the beginning of Section 5.4.3 for details). SORF is a state-of-the-art approach to approximating a Gaussian kernel (e.g. Choromanski et al., 2018). As in Section 5.4.3, we consider its complex extension to make the experiments more competitive.

**Results.** Fig. 5.5 summarizes the results on four datasets from Table 5.2. We show the results on the rest of datasets as well as the results of GP regression in Appendix D.2. We can make similar observations for Fig. 5.5 as for the poly-

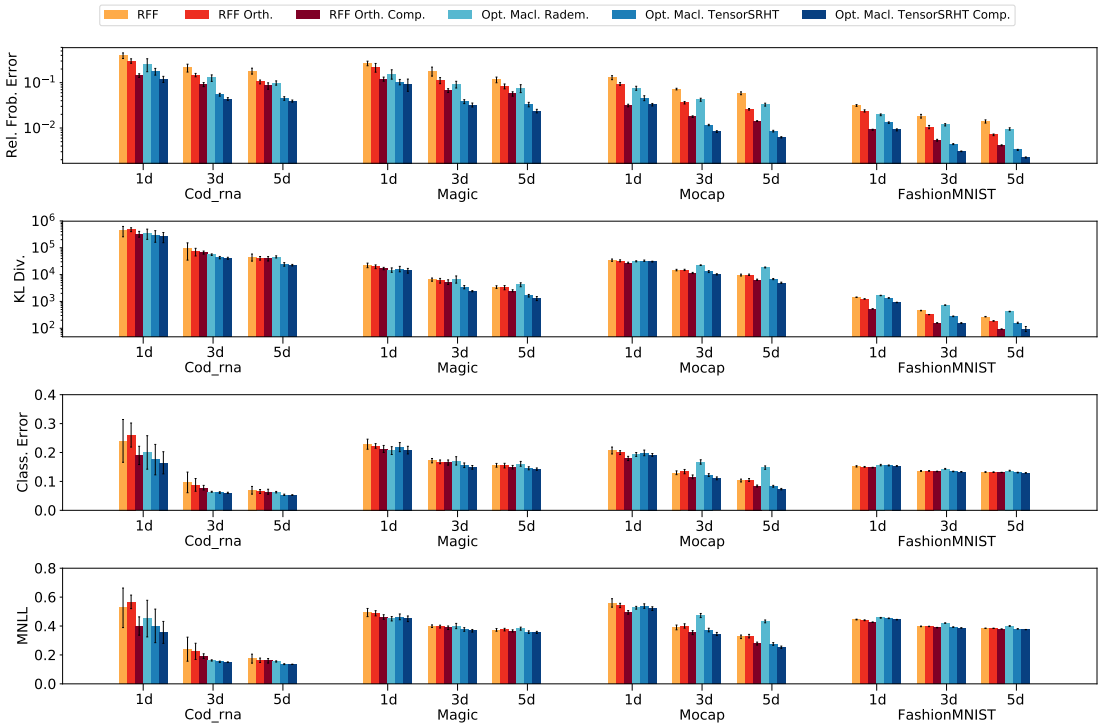



Figure 5.5: Results of the experiments in Section 5.4.3 on approximate GP classification with a Gaussian kernel. Lower values are better for all the metrics. For each dataset, we show the number of random features $D \in \{1d, 3d, 5d\}$ used in each method on the horizontal axis, with $d$ being the input dimensionality of the dataset. We put the legend labels and the bars in the same order.

nomial kernel experiments in Section 5.4.3 (and thus we omit explaining them). The results suggest the effectiveness of the optimized Maclaurin approach with TensorSRHT in approximating the Gaussian kernel.

### Influence of the Data Distribution on the Kernel Approximation

Lastly, we investigate a characterization of datasets for which the optimized Maclaurin approach performs well. We focus on polynomial kernel approximation, and make a comparison with SRF as in Section 5.4.3.

Fig. 5.6 describes a histogram of pairwise distances $\{\|\boldsymbol{x}_{*,i} - \boldsymbol{x}_{*,j}\|\}_{i \neq j}$ of the input vectors in a test subset $X_{*,\mathrm{sub}} = \{\boldsymbol{x}_{*,1}, \ldots, \boldsymbol{x}_{*,m_*}\}$, obtained after zero-centering and unit-normalization, of each of four representative datasets (kin8nm, Cod_rna, Naval, and Protein). For these datasets, the optimized Macluarin approach and SRF show stark contrasts in their performances; see Section 5.4.3 and Appendix D.2. Note that the polynomial kernel in Eq. (5.22) is a shift-invariant kernel on the unit sphere of $\mathbb{R}^d$, and thus its value depends only on the distance $\tau := \|\boldsymbol{x} - \boldsymbol{y}\|$ between the input vectors $\boldsymbol{x}, \boldsymbol{y}$ as long as $\|\boldsymbol{x}\| = \|\boldsymbol{y}\| = 1$. This

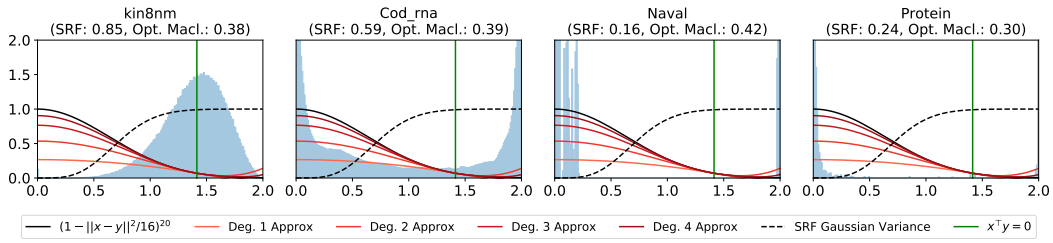



Figure 5.6: Histograms of pairwise Euclidean distances $\{\|\boldsymbol{x}_{*,i} - \boldsymbol{x}_{*,j}\|\}_{i \neq j}$ for test subsets of four datasets (Section 5.4.3). On the top of each figure, we show the relative Frobenius norm errors (5.23) of the optimized Maclaurin approach with real TensorSRHT and of SRF with structured orthogonal projections. The black curve represents the polynomial kernel in Eq. (5.22) with $p = 20$ as a function of $\tau := \|\boldsymbol{x} - \boldsymbol{y}\|$ (the horizontal axis); the orange curves describe its degree $n \in \{1, 2, 3, 4\}$ approximations (i.e., the truncation of the Maclaurin expansion (5.3) of the polynomial kernel up to the $n$-th degree terms.). The dashed curve represents the variance of the SRF approximation as a function of $\tau = \|\boldsymbol{x} - \boldsymbol{y}\|$. The green vertical line shows the value of $\tau = \|\boldsymbol{x} - \boldsymbol{y}\| = \sqrt{2}$ for which the input vectors $\boldsymbol{x}, \boldsymbol{y}$ are orthogonal, $\boldsymbol{x}^\top \boldsymbol{y} = 0$.

motivates us studying here the distribution of pairwise distances and its effects on approximating the polynomial kernel in Eq. (5.22).

In Fig. 5.6, the optimized Maclaurin approach yields lower relative Frobenius norm errors (5.23) than SRF for the left two plots, while the optimized Maclaurin approach is less accurate than SRF for the right two plots. For the datasets of the right two plots (Naval and Protein), the pairwise distances $\{\|\boldsymbol{x}_{*,i} - \boldsymbol{x}_{*,j}\|\}_{i \neq j}$ concentrate around $\tau = 0$ (and there is a smaller mass around $\tau = 2$). In comparison, for the datasets of the left two plots (kin8nm and Cod_rna), the pairwise distances are relatively more evenly distributed across the possible range $\tau \in [0, 2]$.

The above observation suggests that the optimized Maclaurin approach is more suitable for datasets in which the pairwise distances $\{\|\boldsymbol{x}_{*,i} - \boldsymbol{x}_{*,j}\|\}_{i \neq j}$ are not concentrating around 0, i.e., datasets in which there is a diversity in the input vectors $\{\boldsymbol{x}_{*,1}, \ldots, \boldsymbol{x}_{*,m_*}\}$. In fact, for approximating the polynomial kernel (black curve in Fig. 5.6), the finite-degree Maclaurin approximations (orange curves) tend to be less accurate for input vectors $\boldsymbol{x}, \boldsymbol{y}$ close to each other, $\tau = \|\boldsymbol{x} - \boldsymbol{y}\| \approx 0$, and become relatively more accurate as input vectors $\boldsymbol{x}, \boldsymbol{y}$ approach orthogonality, i.e. $\boldsymbol{x}^\top \boldsymbol{y} = 0$ (or $\tau = \|\boldsymbol{x} - \boldsymbol{y}\| = \sqrt{2}$; the vertical green line); see also the Maclaurin expansion (5.3) of the polynomial kernel. On the other hand, the variance of SRF is the lowest around $\tau = \|\boldsymbol{x} - \boldsymbol{y}\| = 0$ and increases as $\tau$ tends to 2. Therefore, the SRF performs well if the pairwise distances $\{\|\boldsymbol{x}_{*,i} - \boldsymbol{x}_{*,j}\|\}_{i \neq j}$ concentrate around 0, and may become inaccurate if they do not.

## 5.5 Pathology when Approximating the Gaussian Kernel

In this section, we uncover a pathology related to Maclaurin-based approximations of the Gaussian kernel that arises in particular when modelling high-frequency data characterized by a short length scale. This pathology causes the approximate GPR predictor in Section 5.3.3 to collapse away from $x = 0$ (see especially the left plot of Fig. 5.2). We will later on propose a cure for this pathology avoiding the collapsing GP predictions.

For ease of notation, we now absorb the length scale parameter $l > 0$ of the Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/(2l^2))$ in the input data, i.e., we define $\tilde{\boldsymbol{x}} := \boldsymbol{x}/l$ and $\tilde{\boldsymbol{y}} := \boldsymbol{y}/l$. In the following theorem, we show that Maclaurin-based kernel approximations vanish when the norms of the inputs become large. Since this effect is only related to the truncation of the Maclaurin series, it happens also without random feature approximations. The vanishing kernels in turn cause GP predictions to collapse, as we discuss later.

**Theorem 5.5.1** (Vanishing Maclaurin approximation of Gaussian kernels)**.** *The magnitude of the finite Maclaurin approximation* $k_p(\boldsymbol{x}, \boldsymbol{y}) := \sigma^2 \exp(-(\|\tilde{\boldsymbol{x}}\|^2 + \|\tilde{\boldsymbol{y}}\|^2)/2) \sum_{n=0}^{p} 1/n!(\tilde{\boldsymbol{x}}^\top \tilde{\boldsymbol{y}})^n$ *of the Gaussian kernel approaches zero as* $\|\tilde{\boldsymbol{x}}\|^2 + \|\tilde{\boldsymbol{y}}\|^2$ *increases. The error* $|k(\boldsymbol{x}, \boldsymbol{y}) - k_p(\boldsymbol{x}, \boldsymbol{y})|$ *between the exact kernel* $k$ *and its approximation* $k_p$ *is the largest for parallel* $\boldsymbol{x}, \boldsymbol{y}$ *and zero when they are orthogonal.*

*Proof.* We start by deriving an upper bound for $|k_p(\boldsymbol{x}, \boldsymbol{y})|$. We leave out the length scale here for ease of notation:

$$\begin{aligned} |k_p(\boldsymbol{x}, \boldsymbol{y})| &= \sigma^2 \exp(-(\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2)/2) \left| \sum_{n=0}^{p} \frac{1}{n!} (\boldsymbol{x}^\top \boldsymbol{y})^n \right| \\ &\leq \sigma^2 \exp(-(\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2)/2) \sum_{n=0}^{p} \frac{1}{n!} (\|\boldsymbol{x}\| \, \|\boldsymbol{y}\|)^n \end{aligned} \tag{5.26}$$

Next, we notice that $\|\boldsymbol{x} - \boldsymbol{y}\|^2 = \|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 - 2\boldsymbol{x}^\top \boldsymbol{y} \geq 0$ for any $\boldsymbol{x}, \boldsymbol{y}$. Thus, we can choose them to be parallel. So $\|\boldsymbol{x}\| \, \|\boldsymbol{y}\| \leq (\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2)/2$. From this inequality, it follows

$$\underbrace{\sum_{n=0}^{p} \frac{1}{n!} (\|\boldsymbol{x}\| \, \|\boldsymbol{y}\|)^n}_{\text{(A)}} \leq \underbrace{\sum_{n=0}^{p} \frac{1}{n!} ((\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2)/2)^n}_{\text{(B)}} \leq \underbrace{\exp((\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2)/2)}_{\text{(C)}}.$$

Now, the gap (C) − (B) $= \sum_{n=p+1}^{\infty} \frac{1}{n!} ((\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2)/2)^n$ increases as $\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2$ increases, which must decreases the ratio (A)/(C) and thus the upper bound of $|k_p(\boldsymbol{x}, \boldsymbol{y})|$ (5.26) goes to zero as $\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2$ increases.

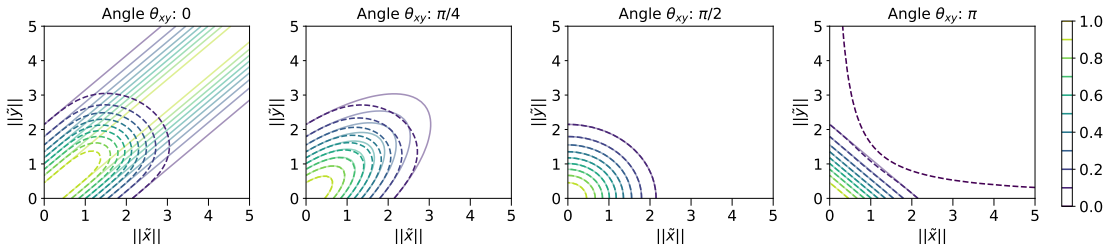



Figure 5.7: $k_p(\boldsymbol{x}, \boldsymbol{y})$ (dashed) vs. $k(\boldsymbol{x}, \boldsymbol{y})$ (transparent and solid) for fixed $p = 3$ and $\sigma^2 = l = 1$, where we use the relationship $\tilde{\boldsymbol{x}}^\top \tilde{\boldsymbol{y}} = \|\tilde{\boldsymbol{x}}\| \, \|\tilde{\boldsymbol{y}}\| \cos(\theta_{xy})$.

Next, we look at the approximation error related to the truncation of the Maclaurin series:

$$|k(\boldsymbol{x}, \boldsymbol{y}) - k_p(\boldsymbol{x}, \boldsymbol{y})| = \sigma^2 \exp(-(\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2)/2) \left| \sum_{n=p+1}^{\infty} \frac{1}{n!} (\boldsymbol{x}^\top \boldsymbol{y})^n \right|$$

If the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$ is zero such that $\boldsymbol{x}^\top \boldsymbol{y} = \|\boldsymbol{x}\| \, \|\boldsymbol{y}\|$, all addends in the infinite sum are maximized. The error thus becomes the largest. The error is zero when they are orthogonal. ■

We visualize the implications of Theorem 5.5.1 in Fig. 5.7, where we compare the approximation $k_p(\boldsymbol{x}, \boldsymbol{y})$ with the exact Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{y})$ for $p = 3$ over a range of values $\|\tilde{\boldsymbol{x}}\|, \|\tilde{\boldsymbol{y}}\|$ as well as the angle $\theta_{xy}$ between $\boldsymbol{x}$ and $\boldsymbol{y}$. One can see that $k_p(\boldsymbol{x}, \boldsymbol{y})$ approaches zero with increasing $\|\tilde{\boldsymbol{x}}\|^2 + \|\tilde{\boldsymbol{y}}\|^2$ regardless of $\theta_{xy}$. This deteriorates the approximation quality, in particular as $\theta_{xy}$ goes to zero. This development accelerates when choosing a shorter length scale $l$.

We now discuss the implications of Theorem 5.5.1 for the case of Gaussian process regression. Recall from Chapter 1 that the GPR predictor is given through

$$\mu_* := \boldsymbol{k}_{\mathbf{f}*}^\top (\boldsymbol{K}_{\mathbf{ff}} + \sigma^2_{\text{noise}} \boldsymbol{I})^{-1} \boldsymbol{y} \quad \text{and} \quad \sigma_*^2 := k_{**} - \boldsymbol{k}_{\mathbf{f}*}^\top (\boldsymbol{K}_{\mathbf{ff}} + \sigma^2_{\text{noise}} \boldsymbol{I})^{-1} \boldsymbol{k}_{\mathbf{f}*}, \quad (5.27)$$

where $\boldsymbol{K}_{\mathbf{ff}}$ is the kernel matrix evaluated on the training data $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\boldsymbol{k}_{\mathbf{f}*} = (k(\boldsymbol{x}_1, \boldsymbol{x}_*), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_*))^\top$, $k_{**} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$ and $\boldsymbol{I} \in \mathbb{R}^{N \times N}$ is the identity matrix.

When using $k_p$, the GPR predictive means $\mu_*$ and variances $\sigma_*^2$ in Eq. (5.27) collapse to zero for test points $\tilde{\boldsymbol{x}^*}$ with large $\|\tilde{\boldsymbol{x}^*}\|$ since $k_p(\boldsymbol{x}^*, \boldsymbol{x})$ goes to zero for any $\boldsymbol{x} \in \mathbb{R}^d$ in this case. This effect is shown in the left plot of Fig. 5.2. We will discuss this example in greater detail in Section 5.5.3.

A similar pathology was identified for the *Relevance Vector Machine* (Tipping, 1999) that was solved by adding new basis functions centered on the test points (Rasmussen and Quiñonero Candela, 2005). In the following, we present a similar strategy by centering our entire approximation around individual test points.

### 5.5.1 Curing the Pathology

We will now exploit a property of the Gaussian kernel that allows us to cure the aforementioned pathology.

The Gaussian kernel is shift-invariant, i.e., $k(\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}+\boldsymbol{\delta}) = k(\boldsymbol{x}, \boldsymbol{y})$ for any $\boldsymbol{\delta} \in \mathbb{R}^d$, because $\|(\boldsymbol{x}+\boldsymbol{\delta}) - (\boldsymbol{y}+\boldsymbol{\delta})\| = \|\boldsymbol{x}-\boldsymbol{y}\|$. Thus, when making a prediction at a test input $\boldsymbol{x}_*$, one can subtract $\boldsymbol{x}_*$ from all inputs used in Eq. (5.27) without changing the result of the prediction. More specifically, the values $\mu_*$ and $\sigma_*^2$ in Eq. (5.27) do not change if we substitute $k(\boldsymbol{x}, \boldsymbol{y})$ by $k(\boldsymbol{x}-\boldsymbol{x}_*, \boldsymbol{y}-\boldsymbol{x}_*)$ for the computation of $\boldsymbol{k}_{\mathbf{f}*}$, $\boldsymbol{K}_{\mathbf{ff}}$ and $k_{**}$.

Now we consider an approximate Gaussian kernel with random features $\hat{k}_p(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})$ with truncation degree $p$ of the Maclaurin series as defined in Section 5.3.1. Thus, for a set of polynomial sketches $\{\Phi_n(\boldsymbol{x})\}_{n=1}^p$ of dimensions $\{D_n\}_{n=1}^p$, the approximate kernel yields

$$\hat{k}_p(\boldsymbol{x}, \boldsymbol{y}) = \sigma^2 \exp\left(-\frac{\|\boldsymbol{x}\|^2}{2l^2}\right) \exp\left(-\frac{\|\boldsymbol{y}\|^2}{2l^2}\right) \left(1 + \sum_{n=1}^{p} \frac{1}{n! l^{2n}} \Phi_n(\boldsymbol{x})^\top \Phi_n(\boldsymbol{y})\right) \tag{5.28}$$

and we have $\mathbb{E}[\hat{k}_p(\boldsymbol{x}, \boldsymbol{y})] = k_p(\boldsymbol{x}, \boldsymbol{y})$. The dimension of the feature map $\Phi(\boldsymbol{x})$ here is thus $D = 1 + D_1 + \cdots + D_p$.

The approximate kernel $\hat{k}_p$ (5.28) is greatly affected by the subtraction of the test point $\boldsymbol{x}_*$. To see this, we define $\hat{k}_p^*(\boldsymbol{x}, \boldsymbol{y}) := \hat{k}_p(\boldsymbol{x}-\boldsymbol{x}_*, \boldsymbol{y}-\boldsymbol{x}_*)$ with:

$$\hat{k}_p^*(\boldsymbol{x}, \boldsymbol{y}) = \sigma^2 \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}^*\|^2 + \|\boldsymbol{y}-\boldsymbol{x}^*\|^2}{2l^2}\right) \sum_{n=1}^{p} \frac{1}{n!} \Phi_n\left(\frac{\boldsymbol{x}-\boldsymbol{x}^*}{l}\right)^\top \Phi_n\left(\frac{\boldsymbol{y}-\boldsymbol{x}^*}{l}\right) \tag{5.29}$$

where $\mathbb{E}[\hat{k}_p^*(\boldsymbol{x}, \boldsymbol{y})] = k_p(\boldsymbol{x}-\boldsymbol{x}^*, \boldsymbol{y}-\boldsymbol{x}^*) =: k_p^*(\boldsymbol{x}, \boldsymbol{y})$. As all $\{\Phi_n\}_{n=1}^p$ are sampled independently, the variance of $\hat{k}_p^*(\boldsymbol{x}, \boldsymbol{y})$ (5.29) can be written as

$$\mathbb{V}[\hat{k}_p^*(\boldsymbol{x}, \boldsymbol{y})] \propto \sum_{n=1}^{p} \left(\frac{1}{n!}\right)^2 \mathbb{V}\left[\Phi_n\left(\frac{\boldsymbol{x}-\boldsymbol{x}^*}{l}\right)^\top \Phi_n\left(\frac{\boldsymbol{y}-\boldsymbol{x}^*}{l}\right)\right], \tag{5.30}$$

When setting $\boldsymbol{x} = \boldsymbol{x}^*$ or $\boldsymbol{y} = \boldsymbol{x}^*$, the variance terms in Eq. (5.30) become *zero* for any of the polynomial sketches discussed in Chapter 3 as can be seen from Table 5.1. $\hat{k}_p^*(\boldsymbol{x}^*, \boldsymbol{y})$, $\hat{k}_p^*(\boldsymbol{x}, \boldsymbol{x}^*)$ and $\hat{k}_p^*(\boldsymbol{x}^*, \boldsymbol{x}^*)$ thus become *deterministic*.

We further have $\hat{k}_p^*(\boldsymbol{x}^*, \boldsymbol{y}) = \sigma^2 \exp(-\|\boldsymbol{x}^*-\boldsymbol{y}\|^2/(2l)^2)$, $\hat{k}_p^*(\boldsymbol{x}, \boldsymbol{x}^*) = \sigma^2 \exp(-\|\boldsymbol{x}-\boldsymbol{x}^*\|^2/(2l)^2)$ and $\hat{k}_p^*(\boldsymbol{x}^*, \boldsymbol{x}^*) = \sigma^2$, which are all equal to the exact kernel $k$ evaluated at these points. Therefore, $\boldsymbol{k}_{\mathbf{f}*}$ and $k_{**}$ in Eq. (5.27) become exact for our Maclaurin approximation. $\boldsymbol{K}_{\mathbf{ff}}$ unfortunately remains affected by the vanishing approximate kernels described by Theorem 5.5.1 and by non-zero random feature variances. A crucial advantage of using $\hat{k}_p^*$ instead of $\hat{k}_p$ is that the GP predictive distribution Eq. (5.27) does not collapse to zero anymore as $\|\tilde{\boldsymbol{x}}^*\|$ grows. We illustrate this effect on the same synthetic example that we used in Fig. 5.2.

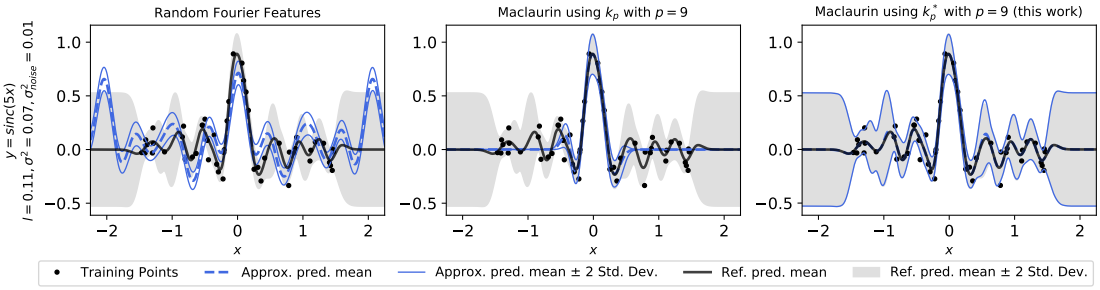



Figure 5.8: Approximating the predictive distribution of a reference GPR using different approximation methods and $D = 10$.

As before, we show the reference GPR along with three different approximation schemes in Fig. 5.8. The baseline Random Fourier features (RFF) Rahimi and Recht (2007) (left) struggles to recover the reference GPR for $D = 10$ random features. The dimension $D$ of the feature map here is equal to $p + 1$ for the optimized Maclaurin approximation, since the example is one-dimensional and a single random feature perfectly approximates a degree-$p$ polynomial kernel for the Rademacher sketch in this case. So $p = 9$ with $D_1 = \cdots = D_p = 1$ is chosen by the optimized Maclaurin method.

The Maclaurin approach using $\hat{k}_p^*$ (right) gives the best approximations while the predictive distribution of the one using $\hat{k}_p$ (middle) collapses to zero very quickly at points $x^* \in \mathbb{R}$ away from zero. This is because $\|\tilde{x^*}\|$ becomes large very quickly and $\hat{k}_p$ vanishes when being evaluated at these points. For values $x^*$ far away from the training data, the GP predictor (5.27) using $k_p^*$ even recovers the GPR prior distribution as desired, which can be explained as follows. When $x^*$ is far from the training data, $\boldsymbol{k}_{\mathbf{f}*}$ becomes zero. Then $\mu_* = 0$ and $\sigma_*^2 = k_{**} = \sigma^2$ in Eq. (5.27). Since $k_{**}$ and $\boldsymbol{k}_{\mathbf{f}*}$ in Eq. (5.27) are accurate when using $k_p^*$ as explained earlier, the convergence to the prior is kept for $k_p^*$.

### 5.5.2 Reducing Computational Costs through Clustering

A caveat of using the approximate kernel $\hat{k}_p^*(\boldsymbol{x}, \boldsymbol{y})$ (5.29) described in Section 5.5.1 is that we need to recompute the GPR predictor (5.27) for every test point $\boldsymbol{x}^*$ separately, which becomes expensive when many test points need to be predicted, even when using random features.

Recall from Chapter 1 that the GPR predictor using random features can be written as

$$\mu_* := \Phi(\boldsymbol{x}_*)^\top \boldsymbol{A}^{-1} \Phi(\boldsymbol{X})^\top \boldsymbol{y} / \sigma^2_{\text{noise}} \quad \text{and} \quad \sigma_*^2 := \Phi(\boldsymbol{x}_*)^\top \boldsymbol{A}^{-1} \Phi(\boldsymbol{x}_*), \tag{5.31}$$

with $\boldsymbol{A} := \Phi(\boldsymbol{X})^\top \Phi(\boldsymbol{X}) / \sigma^2_{\text{noise}} + \boldsymbol{I}$ and $\Phi(\boldsymbol{X}) = (\Phi(\boldsymbol{x}_1), \ldots, \Phi(\boldsymbol{x}_N))^\top \in \mathbb{R}^{N \times D}$.

We denote the number of test inputs by $N_*$. A direct computation of Eq. (5.31) now costs a total of $\mathcal{O}(N_* N D^2)$ for all test points.

**Algorithm 7:** Precomputing Localized Maclaurin Features for the Gaussian Kernel

**Input:** Hyperparameters $l, \sigma^2, \sigma^2_{\text{noise}} > 0$; training data $\{\boldsymbol{x}_i\}_{i=1}^N$; test data $\{\boldsymbol{x}_i^*\}_{i=1}^{N_*}$; number of features $D \geq 1$;
`// 1) Training`
**if** $d > 1$ **then** `// We use random features`
  Center the training data by subtracting the training mean ;
  Find $(p^*, D_1, \ldots, D_{p^*})$, the optimal feature allocation for the random Maclaurin method (see Algorithm 5) ;
**end**
$C := \{(\sum_{i=1}^N \boldsymbol{x}_i)/N\}$ `// Initialize centroids with training mean` ;
**while** *True* **do** `// Farthest point clustering`
  Let $\delta_i = \min\{\|\boldsymbol{x}_i - \boldsymbol{c}\| \,|\, \boldsymbol{c} \in C\}$ for $i = 1, \ldots, N$ ;
  `// Check if training Max.-Min.-Distance is below threshold`
  **if** $\max\{\delta_i\}_{i=1}^N < \theta$ **then**
    break;
  **end**
  Add $\boldsymbol{x_i}$ with $i = \arg\max_i \{\delta_i\}_{i=1}^N$ to $C$ ;
  Precompute $\boldsymbol{A}_c^{-1} := \left(\Phi(\boldsymbol{X} - \boldsymbol{c})^\top \Phi(\boldsymbol{X} - \boldsymbol{c})/\sigma^2_{\text{noise}} + \boldsymbol{I}\right)^{-1}$ with $\Phi$ defined as in Section 5.3.1 ;
**end**
**forall** $\{\boldsymbol{x}_i^*\}_{i=1}^{N_*}$ **do** `// 2) Inference`
  Assign $\boldsymbol{x}_i^*$ to closest centroid $\boldsymbol{c} \in C$ ;
  Compute $\mu_*$ and $\sigma_*^2$ in Eq. (5.31) by setting $\boldsymbol{A}^{-1} = \boldsymbol{A}_c^{-1}$, $\boldsymbol{x}^* = \boldsymbol{x}_i^* - \boldsymbol{c}$ and $\boldsymbol{X} = \boldsymbol{X} - \boldsymbol{c}$ ;
**end**

The problem is "embarrassingly parallel" and the computation of Eq. (5.31) for every $\boldsymbol{x}^*$ could be easily distributed on a cluster of compute nodes or parallelized using a single GPU, e.g., using JAX (Bradbury et al., 2018). However, we propose a different approach here that requires no separate training at test time while staying as close as possible to the training time of $\mathcal{O}(ND^2)$ as is generally desired for random feature approximations.

Our approach is to cluster the *training* data into $N_C$ clusters and to use the centroids of these clusters as *pseudo test inputs*. We choose a farthest point clustering for simplicity as it does not require convergence verification and determines the number of clusters using a threshold $\theta > 0$, but any clustering algorithm can be used instead (even a random selection of training points). As the centroids are known during training, we can pretrain a set of $N_C$ predictors using Eq. (5.31) and assign a new test point $\boldsymbol{x}^*$ to the closest centroid at prediction time. We summarize the complete procedure in Algorithm 7. The computational cost is now

$\mathcal{O}(N_C N D^2)$ and is thus much lower than $\mathcal{O}(N_* N D^2)$ if $N_C \ll N_*$.

### 5.5.3 Empirical Evaluation on Real-World Data

In this section, we evaluate our proposed method on real-world data of different dimensions for which we employ the real-valued polynomial sketches summarized in Table 5.1. As for the synthetic example in Fig. 5.8, we use the Gaussian kernel with hyperparameters $l$ and $\sigma^2 > 0$ that are found through gradient based optimization of the log marginal likelihood of a reference GPR, along with $\sigma^2_{\text{noise}}$.

We compare our method against a random Fourier features baseline Rahimi and Recht (2007) as well as its structured extension Yu et al. (2016) when the data is sufficiently high dimensional[8]. We also add the vanilla optimized Maclaurin method without recentering the data to this comparison. It is equivalent to Algorithm 7 using only a single cluster with its centroid being the training mean. We measure the approximation quality with respect to the reference GPR using the KL-divergence between the predictive means and variances in Eq. (5.27) and Eq. (5.31). We measure downstream regression performance using the root mean squared error (RMSE).

#### UK Apartment Price Data

We downloaded the monthly property sales data for England and Wales from the HM land registry[9]. We filtered for sold apartments for the month of January 2022 leading to a data set with 24 553 observations. Matching the post codes for each apartment sold with a database of latitudes and longitudes[10] allowed us to obtain a two-dimensional data set (latitude, longitude) that we could regress against the logarithm of the sales prices. We randomly split the data into 10 000 training points and kept the rest for testing.

In our first experiment, we aim to recover the reference GPR predictive distribution on a regular grid of latitudes (between $+50°$ and $+55°$) and longitudes (between $-6°$ and $+2°$) of size 100 by 100. Fig. 5.9 shows the results of this experiment. As for the sinc-example in Fig. 5.8, random Fourier features struggle to recover the predictive distribution using $D = 100$ random features and the vanilla Maclaurin method using $\hat{k}_p$ (5.28) suffers from vanishing kernels due to the short (compared to the scaling of the data) length scale of $l = 0.25$. Our proposed kernel $\hat{k}^*_p$ (5.29) improves predictions considerably leading to the lowest KL divergence with respect to the reference GP predictive distribution. It also converges to the prior for test points far from the training data.

In our second experiment, we evaluate the use of Algorithm 7 to precompute the matrix inversion in Eq. (5.31) on a set of pseudo test inputs. This time we

[8] Otherwise, the structured random Fourier features induce a large bias.

[9] https://www.gov.uk/government/statistical-data-sets/price-paid-data-downloads

[10] https://www.freemaptools.com/download-uk-postcode-lat-lng.htm

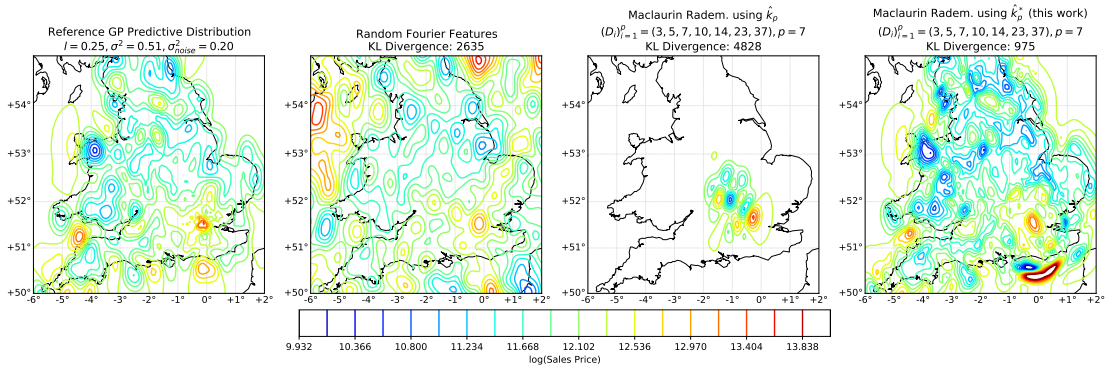



Figure 5.9: Approximating the predictive mean of a reference GP using different approximation methods and $D = 100$. KL Divergences also include predictive variances. The Maclaurin method using $\hat{k}_p$ makes predictions centered around the mean of the training data (close to London).

report results on the left-out test data instead of a regular grid. The left part of Fig. 5.10 shows these results. We can see that the KL divergence falls off considerably (top plot) as we add more clusters until reaching 57 clusters. From then on the KL divergence remains roughly the same indicating that 57 clusters give a good trade-off between efficiency and performance. In the bottom plot we show a comparison of RMSE values for these 57 clusters, where the Maclaurin method outperforms random Fourier features, in particular for small $D$.

### UCI Data Sets: Yacht and kin8nm

In the following, we repeat the evaluation of Algorithm 7 for two higher dimensional data sets that are taken from the UCI machine learning repository (Dua and Graff, 2017) in Fig. 5.10.

We obtain very similar results for the UCI Yacht data set as for the UK apartment price data set. Adding more clusters gives large gains initially but these diminish when setting $\theta > 2.0l$ in Algorithm 7 determining a good trade-off between performance and computational cost. This time we included structured orthogonal random Fourier features (Yu et al., 2016) that are also outperformed by the Maclaurin method.

For the UCI kin8nm data set, results look quite different. Adding more clusters *increases* the KL divergence towards the reference GP predictor, which is why only a single centroid, the mean of the training data, is chosen for the RMSE comparison in the plot below. This corresponds to the vanilla optimized Maclaurin method (Algorithm 5).

We explain this observation as follows. The length scales obtained for the sinc example, the UK apartment price data and for UCI Yacht are short. For the UCI Yacht data set it is 0.32, i.e., much less than 3.28, the median pairwise Euclidean

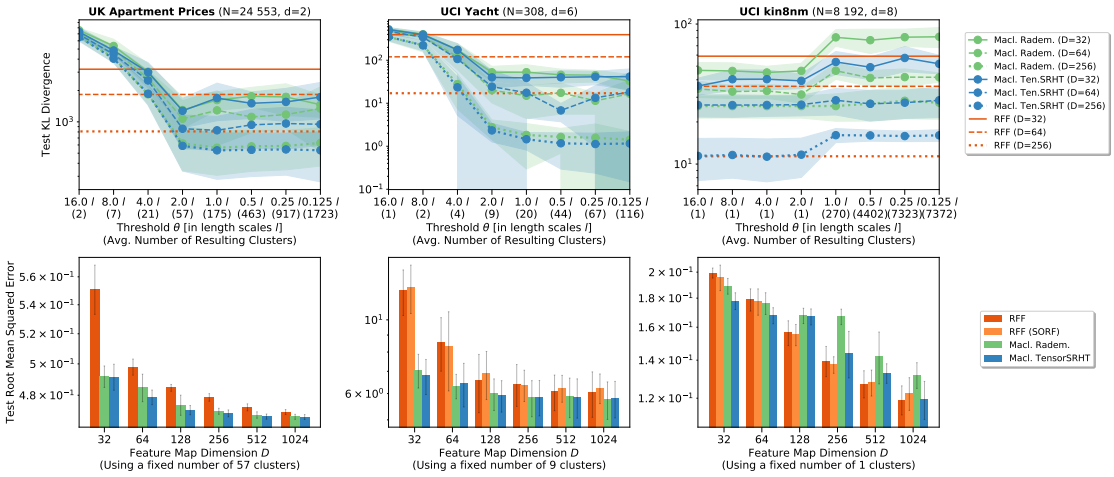



Figure 5.10: (Top) Test KL Divergence for a given number of clusters obtained from the training data using Algorithm 7 with threshold $\theta$. (Bottom) RMSE over feature map dimension $D$ for a fixed number of clusters.

distance of the training data, indicating that the data is fit by a reference GP of high frequency. For kin8nm the length scale is 2.15 compared to 3.93 (median heuristic) indicating a much smoother GP than the ones before.

In this case, the values of $\boldsymbol{k}_{\mathbf{f}*}$ and $k_{**}$ in Eq. (5.27) are less affected by the vanishing kernels Theorem 5.5.1 due to a longer length scale. However, the approximation of $\boldsymbol{K}_{\mathbf{ff}}$ in Eq. (5.27) is more accurate when using the vanilla Maclaurin approximation (5.28) because the data is centered around the training mean. This shows that the clusters need to be chosen depending on the smoothness of the target GP. In this work, we have provided a *generalization* of the vanilla Maclaurin method that (with an appropriate choice of clusters) can fit both, high and low-frequency data.

## 5.6 Discussion

Based on the derived variance formulas in Chapter 3, we developed a novel optimization algorithm for data-driven random feature approximations of dot product kernels in this chapter, which is also applicable to the Gaussian kernel. Our approach uses a truncated Maclaurin approximation of the kernel, which approximates the kernel as a finite sum of polynomial kernels of different degrees. Given a total number of random features, our optimization algorithm determines how many random features should be used for each polynomial degree in the Maclaurin approximation. We defined the objective function of this optimization algorithm as the mean squared error w.r.t. data distribution, and used the variance formulas derived in Chapter 3 to evaluate the objective in closed form. We empirically demonstrated that our optimized Maclaurin approach achieves state-of-the-

art performance on a variety of data sets both in terms of the kernel approximation accuracy and downstream task performance.

However, we also showed that Maclaurin approximations suffer from a pathology when approximating the Gaussian kernel. In this case, the GP predictive distribution collapses at locations with large input norms, which happens particularly fast if the length scale is set to a small value relative to the scaling of the data, leading to poor predictions. We proposed a cure for this pathology by recentering the data around the test point to predict, yielding once again state-of-the-art performance on a variety of data sets.

The methods presented in this chapter improve the empirical performance of Maclaurin approximations of dot product kernels and Gaussian kernels considerably (e.g. compared to the random Maclaurin approach (Kar and Karnick, 2012)). However, Algorithm 5 requires the evaluation of the exact kernel on a subset of the training data, where the size of the subset needs to be specified by the user. Moreover, the scaling of the data, or its rescaling via a length scale parameter, has a significant impact on the quality of the Maclaurin approximation. Future research could therefore investigate how the random feature allocation between polynomial degrees can be derived from the scale of the input data alone, without requiring pairwise kernel evaluations. Similarly, the number of clusters and their positions in Algorithm 7 could be further improved with such considerations, leading to a simpler implementation of the methods described in this chapter.

# Chapter 6

# The Optical Processing Unit

So far, the contents of this thesis have revolved around random feature approximations for polynomial kernels and more general dot product kernels. This research was originally inspired by working with an optical hardware, the so-called Optical Processing Unit (OPU).

The OPU (Saade et al., 2016; Ohana et al., 2020) computes random projections literally at the speed of light and without having to store the projection matrix in memory, and it does so at a low power consumption. Therefore, applying the OPU in machine learning applications could potentially contribute to a more ecologically responsible data science infrastructure by saving time and energy. The OPU has already been applied in applications like reservoir computing (Dong et al., 2018, 2019) and anomaly detection (Keriven et al., 2018) demonstrating its usefulness in practice.

Building on the milestone work of Saade et al. (2016), the goal of this chapter is threefold: a) we derive in full generality the kernel to which the random features computed by the OPU converge, generalizing the earlier computation of Saade et al. (2016) to a larger class of kernels; b) we present new examples and a benchmark of applications for the kernel of the OPU; and c) we give a detailed comparison of the running time and energy consumption between the OPU and a last generation GPU.

In fact, we show in this chapter that the kernel implicitly computed by the OPU has a close relation to the polynomial kernel, making its derivation closely related to the previous contents of this thesis.

## 6.1 The Optical Processing Unit (OPU)

We begin by describing the inner workings of the OPU giving rise to the random features it computes. Fig. 6.1 shows a schematic drawing of the physical apparatus.

The principle of the random projections performed by the OPU is based on the use of a semi-transparent "scattering" material that refracts the laser light passing through it. As discussed in Liutkus et al. (2014), light passing through such

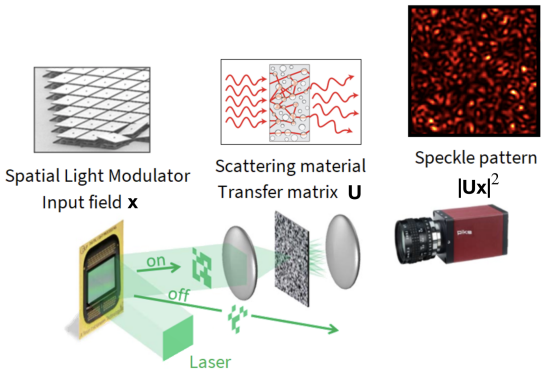



Figure 6.1: Experimental setup of the Optical Processing Unit (modified with permission from Saade et al. (2016)). The data vector is encoded in the coherent light from a laser using a DMD. Light then goes through a scattering medium and a speckle pattern is measured by a camera.

a scattering medium follows many complex paths that depend on the refractive index inhomogeneities at random positions. For a fixed scattering medium, the resulting process is still linear, deterministic, and reproducible.

We thus obtain linear projections of a numerical data vector $\boldsymbol{x} \in \mathbb{R}^d$ by filtering the light beam at different positions using a *digital micromirror device (DMD)*. This encoded light then passes through the heterogeneous medium, implicitly performing a random matrix multiplication, where the random matrix is inherently determined by the scattering medium. The scattered light results in a so-called *speckle pattern* that is recorded by a camera, where the light intensity at each measuring point is the result of a superposition of the light beams filtered by the DMD. This superposition is modelled mathematically by a sum of the components of $\boldsymbol{x}$ weighted by random coefficients. Measuring the intensity of light at an arbitrary pixel location recorded by the camera induces a non-linear transformation of this sum. $D$ arbitrary pixel values taken together then constitute the random feature map

$$\Phi(\boldsymbol{x}) = |\boldsymbol{U}\boldsymbol{x}|^2/\sqrt{D}, \quad \text{with} \quad \boldsymbol{U} \in \mathbb{C}^{D\times d} \quad \text{and} \quad (\boldsymbol{U})_{i,j} \overset{i.i.d.}{\sim} \mathcal{CN}(0,1), \tag{6.1}$$

where $|\cdot|^2$ is the squared absolute value taken element-wise.

## 6.2 Computing the Kernel

In this section, we compute the kernel associated with the random feature map $\Phi$ (6.1), which is our first contribution. The goal is to derive the limiting kernel that the inner product $\hat{k}_2(\boldsymbol{x}, \boldsymbol{y}) := \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})$ for two data points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ converges to as $D$ goes to infinity.

Let $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D)^\top$. We then write

$$\hat{k}_2(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{D} \sum_{\ell=1}^{D} |\boldsymbol{u}_\ell^\top \boldsymbol{x}|^2 |\boldsymbol{u}_\ell^\top \boldsymbol{y}|^2 \to \mathbb{E}[|\boldsymbol{u}^\top \boldsymbol{x}|^2 |\boldsymbol{u}^\top \boldsymbol{y}|^2] =: k_2(\boldsymbol{x}, \boldsymbol{y}) \quad (D \to \infty),$$

with $\boldsymbol{u} = (u_1, \ldots, u_d)^\top \in \mathbb{C}^d$ and $u_1, \ldots, u_d \overset{i.i.d.}{\sim} \mathcal{CN}(0, 1)$. The convergence here is due to the law of large numbers and defined as a convergence in probability, where $k_2(\boldsymbol{x}, \boldsymbol{y})$ is the limit of this convergence. Thus, we work out $k_2(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}[\hat{k}_2(\boldsymbol{x}, \boldsymbol{y})]$ and summarize the result in the following theorem.

**Theorem 6.2.1.** *The kernel $k_2$ approximated by the dot product of optical random features of Eq. (6.1) is given by*

$$k_2(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top \boldsymbol{y})^2 \tag{6.2}$$

*Proof.* Using the contents of this thesis that we developed so far, it is straightforward to solve $k_2(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}[|\boldsymbol{u}^\top \boldsymbol{x}|^2 |\boldsymbol{u}^\top \boldsymbol{y}|^2]$. In fact, it can be directly derived from the variance formula of complex Gaussian Product-Sketches in Eq. (3.19) in Chapter 3. Recall from Section 3.3 that the variance of a complex Product-Sketch

$$\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{p} (\boldsymbol{z}_i^\top \boldsymbol{x})(\overline{\boldsymbol{z}_i^\top \boldsymbol{y}})$$

with $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_p \sim \mathcal{CN}(\boldsymbol{0}, \boldsymbol{I}_d)$ is computed as $\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] = \mathbb{E}[|\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})|^2] - (\boldsymbol{x}^\top \boldsymbol{y})^{2p}$.

Hence, computing $\mathbb{E}[|\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})|^2] = \mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})] + (\boldsymbol{x}^\top \boldsymbol{y})^{2p}$ and setting $p = 1$ yields

$$\mathbb{E}[|\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})|^2] = \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top \boldsymbol{y})^2 = k_2(\boldsymbol{x}, \boldsymbol{y}),$$

where we used the variance formula for complex Gaussian Product-Sketches in Eq. (3.19). $\mathbb{E}[|\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})|^2]$ corresponds exactly to $k_2(\boldsymbol{x}, \boldsymbol{y})$ because we have

$$\mathbb{E}[|\hat{k}_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})|^2] = \mathbb{E}[|\boldsymbol{z}_1^\top \boldsymbol{x}|^2 |\boldsymbol{z}_1^\top \boldsymbol{y}|^2] = \mathbb{E}[|\boldsymbol{u}^\top \boldsymbol{x}|^2 |\boldsymbol{u}^\top \boldsymbol{y}|^2].$$

■

The result of Theorem 6.2.1 is remarkable because it shows that the optical random features computed by the OPU naturally give rise to a limiting kernel that is similar to a degree-2 polynomial kernel. In fact, when appending the norm of the inputs to this feature map, i.e., by setting $\Phi'(\boldsymbol{x}) := (\mathrm{i} \|\boldsymbol{x}\|, \Phi(\boldsymbol{x})^\top)^\top$ and $\Phi'(\boldsymbol{y}) := (\mathrm{i} \|\boldsymbol{y}\|, \Phi(\boldsymbol{y})^\top)^\top$ with $\mathrm{i} := \sqrt{-1}$, we obtain precisely $\mathbb{E}[\Phi'(\boldsymbol{x})^\top \Phi'(\boldsymbol{y})] = (\boldsymbol{x}^\top \boldsymbol{y})^2$. We will thus show in Section 6.3 that optical random features are useful for image classification tasks for which polynomial kernels are known to work well.

**Higher degree features.** In the following, we show that one can obtain higher degree kernels when numerically changing the exponent of the feature map in Eq. (6.1). We focus here on the case $m = 2s$ for some $s \in \mathbb{N}$, i.e., by setting:

$$\Phi_m(\boldsymbol{x}) := \frac{1}{\sqrt{D}} |\boldsymbol{U}\boldsymbol{x}|^m \tag{6.3}$$

In this case, we obtain the kernel $k_{2s}(\boldsymbol{x}, \boldsymbol{y})$ described through the following theorem.

**Theorem 6.2.2.** *When the exponent $m$ in Eq. (6.3) is even, i.e., $m = 2s$, for some $s \in \mathbb{N}$, the dot product of feature maps of Eq. (6.3) tends to the kernel $k_{2s}$ (for $D \to \infty$) with*

$$k_{2s}(\boldsymbol{x}, \boldsymbol{y}) := \|\boldsymbol{x}\|^m \|\boldsymbol{y}\|^m \sum_{i=0}^{s} (s!)^2 \binom{s}{i}^2 \frac{(\boldsymbol{x}^\top \boldsymbol{y})^{2i}}{\|\boldsymbol{x}\|^{2i} \|\boldsymbol{y}\|^{2i}}. \tag{6.4}$$

*The proof in this case is more involved and we move it to Appendix E.1.*

Eq. (6.4) is interesting for two reasons. On the one hand, it shows that numerically changing the exponent of the optical random features (6.1) leads to a positively weighted sum of normalized polynomial kernels, similar to the expansion of inhomogeneous polynomial kernels

$$(\nu + \boldsymbol{x}^\top \boldsymbol{y})^p = \sum_{i=0}^{p} \binom{p}{i} \nu^{p-i} (\boldsymbol{x}^\top \boldsymbol{y})^i \tag{6.5}$$

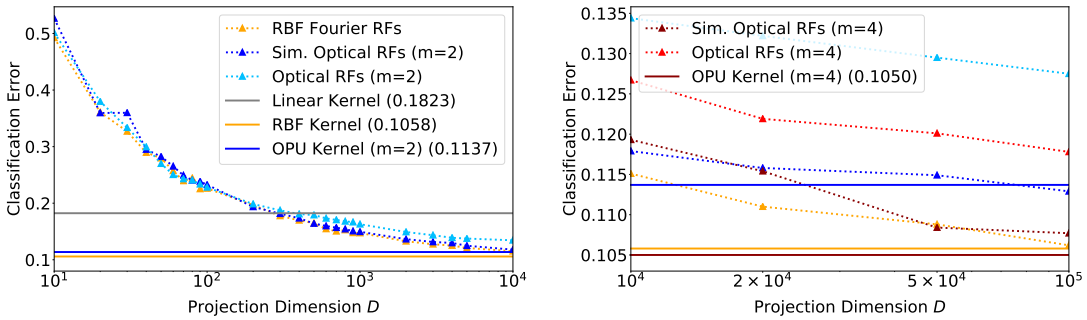
with $\nu \geq 0$ and $p \in \mathbb{N}$. The difference between Eq. (6.5) and Eq. (6.4) is that the polynomial kernels in the sum of Eq. (6.4) use inputs normalized to unit length. The entire kernel (6.4) is then rescaled by the norms $\|\boldsymbol{x}\|^m \|\boldsymbol{y}\|^m$ *after* computing this sum.

On the other hand, Eq. (6.4) allows us to compute the variance of the kernel estimate obtained through optical random features. Let $s \in \mathbb{N}$ and assume $D = 1$ for now. Then the following relationship holds:

$$\begin{aligned} \mathbb{V}[\hat{k}_s(\boldsymbol{x}, \boldsymbol{y})] &= \mathbb{E}[(\Phi_s(\boldsymbol{x})\Phi_s(\boldsymbol{y}))^2] - \mathbb{E}[\Phi_s(\boldsymbol{x})\Phi_s(\boldsymbol{y})] \\ &= \mathbb{E}[\Phi_{2s}(\boldsymbol{x})\Phi_{2s}(\boldsymbol{y})] - \mathbb{E}[\Phi_s(\boldsymbol{x})\Phi_s(\boldsymbol{y})] \\ &= k_{2s}(\boldsymbol{x}, \boldsymbol{y}) - k_s(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

For $D > 1$, we simply divide the variance $\mathbb{V}[\hat{k}_s(\boldsymbol{x}, \boldsymbol{y})]$ by $D$, as the random features are sampled i.i.d. The variance of $\hat{k}_s(\boldsymbol{x}, \boldsymbol{y})$ can thus be expressed in terms of $k_{2s}(\boldsymbol{x}, \boldsymbol{y})$.

The variance formula for optical random features cannot be directly used to derive a Bernstein-type exponential concentration bound as we did in Theorem 3.4.1 for Rademacher Product-Sketches. This is because both $\boldsymbol{U}$ *and* $\boldsymbol{x}, \boldsymbol{y}$ are




(a) $D \leq 10\,000$
(without optical RFs for $m = 4$)

(b) $D \geq 10\,000$
(with optical RFs for $m = 4$)

Figure 6.2: Ridge Regression test error on Fashion MNIST for different RFs and projection dimensions $D$. Horizontal lines show the test error using the limiting kernel. Standard deviations for different seeds are negligibly small and not shown in the plot. Plot (a) compares optical RFs of degree $m = 2$ to RFFs for the Gaussian kernel. Higher degree optical RFs are left out for better readability. The more slowly converging optical RFs for $m = 4$ are added for larger $D$ in plot (b).

unbounded here. Therefore, we used a Bernstein inequality for sub-exponential random variables (Vershynin, 2018, Theorem 2.8.2) in the associated work (Ohana et al., 2020) to show exponentially decaying errors of the kernel estimate under the simplifying assumption that the weights in Eq. (6.1) are real-valued, i.e., $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. In this case, we obtain the following theorem:

**Theorem 6.2.3.** *Let $\epsilon > 0$ and $\hat{k}_m(\boldsymbol{x}, \boldsymbol{y}) = \Phi_m(\boldsymbol{x})^\top \Phi_m(\boldsymbol{y})$ with $\Phi_m$ being defined through Eq. (6.3). Then we can provide the following exponential tail bound for the kernel approximation error:*

$$\Pr\left[\hat{k}_m(\boldsymbol{x}, \boldsymbol{y}) - k_m(\boldsymbol{x}, \boldsymbol{y}) \geq \epsilon\right] \leq D \exp(-c(D\epsilon)^{1/m}) \tag{6.6}$$

*with $c > 0$ being a constant.*

The bound given by Theorem 6.2.3 is not tight, but it shows that the kernel approximation error decays exponentially for a sufficiently large number $D$ of random features. Moreover, this decay is slowed down by an increasing exponent $m$ of the feature map. We skip a detailed derivation of this result here and refer the reader to the original work.

## 6.3 Classification Experiments

In this section, we assess the usefulness of optical random features (6.3) for $m = 2$ and $m = 4$ for different settings and datasets. The model of our choice in

each case is Ridge Regression. OPU experiments were performed remotely on the OPU prototype "Vulcain", running in the LightOn Cloud with library `LightOnOPU` v1.0.2. Since the current version only supports binary input data, we binarize inputs for all experiments using a threshold binarizer, where the threshold is chosen using cross-validation. The code of the experiments is publicly available[1].

### 6.3.1 Optical Random Features for Fashion MNIST

We compare optical random features (simulated as well as physical) to a random Fourier features (RFF) baseline (Rahimi and Recht, 2007) for different projection dimensions $D$ on Fashion MNIST (Xiao et al., 2017). We use individually optimized hyperparameters for all random feature methods that are found for $D = 10\,000$ using an extensive grid search on a held-out validation set. The same hyperparameters are also used for the precise kernel limit.

Fig. 6.2 shows how the overall classification error decreases as $D$ increases. Part (b) shows that simulated optical RFs for $m = 2$ and RFF (RBF Fourier RFs in the plot) reach the respective kernel test score at $D = 10^5$. Simulated optical RFs for $m = 4$ converge more slowly but outperform $m = 2$ features from $D = 2 \times 10^4$. They perform similarly well as RFF at $D = 10^5$. The performance gap between $m = 2$ and $m = 4$ also increases for the real optical RFs with increasing $D$. This gap is larger than for the simulated optical RFs due to an increase in regularization for the $m = 2$ features that was needed to add numerical stability when solving linear systems for large $D$.

The real OPU loses around 1.5% accuracy for $m = 2$ and 1.0% for $m = 4$ for $D = 100\,000$, which is due slightly suboptimal hyperparameters to improve numerical stability for large dimensions. Moreover, there is a small additional loss due to the quantization of the analog signal when the OPU camera records the visual projection.

### 6.3.2 Transfer Learning on CIFAR-10

An interesting use case for the OPU is transfer learning for image classification. For this purpose, we extract a diverse set of features from the CIFAR-10 (Krizhevsky et al., 2009) image classification dataset using three different convolutional neural networks (ResNet34 (He et al., 2015), AlexNet (Krizhevsky et al., 2009) and VGG16 (Chatfield et al., 2014)). The networks were pretrained on the well-known ImageNet classification benchmark (Russakovsky et al., 2015). For transfer learning, we can either fine-tune these networks and therefore the convolutional features to the data at hand, or we can directly apply a classifier on them assuming that they generalize well enough to the data. The latter case requires much less computational resources while still producing considerable performance gains over the use of the original features. This light-weight approach can be carried out on a

[1] https://github.com/joneswack/opu-kernel-experiments

| Architecture | ResNet34 | | | | AlexNet | | | VGG16 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Layer | L1 | L2 | L3 | Final | MP1 | MP2 | Final | MP2 | MP3 | MP4 | MP5 | Final |
| Dimension $d$ | 4 096 | 2 048 | 1 024 | 512 | 576 | 192 | 9 216 | 8 192 | 4 096 | 2 048 | 512 | 25 088 |
| Sim. Opt. RFs | 30.4 | **24.7** | **28.9** | **11.6** | **38.1** | **41.9** | 19.6 | 28.2 | **20.5** | **20.7** | **29.8** | 15.2 (**12.9**) |
| Optical RFs | 31.1 | 25.7 | 29.7 | 12.3 | 39.2 | 42.6 | 20.8 | 30.9 | 23.3 | 21.5 | 30.2 | 16.4 |
| RBF Four. RFs | **30.1** | 25.2 | 30.0 | 12.3 | 39.4 | **41.9** | **19.1** | 28.0 | 20.7 | **20.7** | 30.1 | 14.8 (13.0) |
| No RFs | 31.3 | 26.7 | 33.5 | 14.7 | 44.6 | 48.8 | 19.6 | **27.1** | 21.0 | 22.5 | 34.8 | **13.3** |

Table 6.1: Test errors (in %) on CIFAR-10 using $D = 10^4$ RFs for each kernel (except linear). Features were extracted from intermediate layers when using the original input size (32x32). Final convolutional layers were used with upscaled inputs (224x224). L(i) refers to the ith ResNet34 layer and MP(i) to the ith MaxPool layer of VGG16/AlexNet. Values for the kernel limit are shown in parenthesis (last column).

CPU in a short amount of time where the classification error can be improved with random features.

We compare optical random features and RFF (RBF Four. RFs) to a simple baseline that directly works with the provided convolutional features (no RFs). Table 6.1 shows the test errors achieved on CIFAR-10. Each column corresponds to convolutional features extracted from a specific layer of one of the three networks.

Since the projection dimension $D = 10^4$ was left constant throughout the experiments, it can be observed that all random features perform particularly well compared to a linear kernel when $D \gg d$, where $d$ is the input dimension. For the opposite case $D \ll d$, the lower dimensional projection leads to an increasing test error. This effect can be observed in particular in the last column where the test error of the random feature approximation is higher than without random features. The contrary can be achieved with large enough $D$ as indicated by the values for the true kernel in parenthesis.

A big drawback here is that the computation of sufficiently large dimensional random features may be very costly, especially when $d$ is large as well. This is a regime where the OPU outperforms CPU and GPU by a large margin (see Fig. 6.3) since its computation time is invariant to $d$ and $D$.

In general, the simulated as well as the physical optical random features yield similar performances as the RBF Fourier RFs on the provided convolutional data.

## 6.4 Projection Time and Energy Consumption

The main advantage of the OPU compared to a traditional CPU/GPU setup is that the OPU takes a constant time for computing the random features in Eq. (6.1) of arbitrary dimension $D$ (up to $D = 10^6$ on current hardware) for a single input. Moreover, its power consumption stays below 30 W independently of the workload. We therefore compare the computation time of the random feature map in Eq. (6.1)

taken by the OPU against the time it takes to carry out the same computation on a GPU. We simultaneously compare the energy consumption of both devices.

Fig. 6.3 shows the computation time and the energy consumption over time for GPU and OPU for different projection dimensions $D$. In both cases, the time and energy spending do not include matrix building and loading. For the GPU, only the calls to the PyTorch (Paszke et al., 2019) function `torch.matmul` are measured and energy consumption is the integration over time of power values given by the `nvidia-smi` command.

For the OPU, the energy consumption is constant w.r.t. $D$ and equal to 45 Joules (30 W multiplied by 1.5 seconds). The GPU computation time and energy consumption are monotonically increasing except for an irregular energy development between $D = 45\,000$ and $D = 56\,000$. This exact irregularity was observed throughout all simulations we performed and can most likely be attributed to an optimization routine that the GPU carries out internally. The GPU consumes more than 10 times as much energy as the OPU for $D = 58\,000$ (GPU memory limit). The GPU starts to use more energy than the OPU from $D = 18\,000$. The exact crossover points may change in future hardware versions. The relevant point we make here is that the OPU has a better scalability in $D$ with respect to computation time and energy consumption.

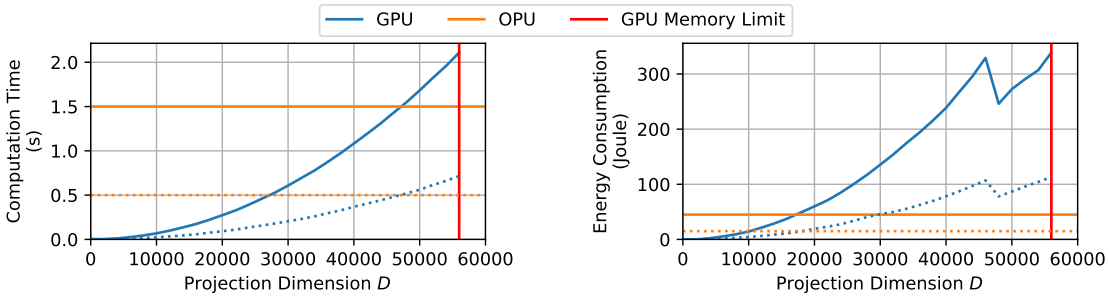



Figure 6.3: Time and energy spent for computing a matrix multiplication $(n, D) \times (D, D)$. The batchsize $n$ is 3000 (solid line) or 1000 (dotted). The curves cross each other at the same $D$ independent from $n$. We verified more precisely that time and energy are linear with $n$ for both OPU and GPU (experiments were run on an NVIDIA P100).

## 6.5 Discussion

In this chapter, we have derived the limiting kernel of the OPU and demonstrated its usefulness for practical machine learning applications. We further showed that the feature map can be numerically modified to yield higher order features, and carried out an energy consumption and projection time benchmark, where a modern GPU was outperformed when computing a large number of random features.

Although a large number of random features is needed to give the OPU a practical advantage over the GPU, these results show the *potential* of the OPU to be used for scalable machine learning applications. Future versions of the OPU could accelerate the device such that it becomes beneficial from a lower number of random features $D$.

Concurrently to the development of the results in this chapter, Gupta et al. (2019) have found a way to obtain an equivalent random projection as the one in Eq. (6.1) using the OPU, but *without* the non-linearity $|\cdot|^2$ that is inherently obtained when recording the light intensity. They thus managed to use the OPU for *linear* random projections. This result makes it possible to use the OPU for random feature approximations and numerical linear algebra methods using Gaussian random projections and thus widens the range of applications immensely. It will be interesting to see in the future how further applications can be accelerated by the OPU and how this can result in energy savings compared to using GPUs.

# Chapter 7

# Conclusions and Future Work

In this thesis, we carried out an in-depth analysis of random feature approximations for polynomial kernels and more general dot product kernels. In particular, we focused on Product-Sketches originally proposed by Kar and Karnick (2012) and Hamid et al. (2014), i.e., random feature approximations of the form

$$\hat{k}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{D} \sum_{\ell=1}^{D} \prod_{i=1}^{p} (\boldsymbol{w}_{i,\ell}^{\top} \boldsymbol{x})(\boldsymbol{w}_{i,\ell}^{\top} \boldsymbol{x}), \tag{7.1}$$

where $\hat{k}(\boldsymbol{x}, \boldsymbol{y})$ is an unbiased estimate of the polynomial kernel $(\boldsymbol{x}^{\top} \boldsymbol{y})^p$.

Using a detailed variance analysis (Chapter 3), we successively optimized these sketches in terms of the sampling distribution over the weights $\{\boldsymbol{w}_{i,\ell}\}_{\ell=1}^{D}$ that we assume to be independent over $i = 1, \ldots, p$. In particular, we showed that a variance lower bound can be attained when $\boldsymbol{w}_{i,\ell}$ are independent Rademacher vectors for $i = 1, \ldots, p$ and $\ell = 1, \ldots, D$. We further showed that structured polynomial sketches, i.e., sketches where the $\{\boldsymbol{w}_{i,\ell}\}_{\ell=1}^{D}$ are *dependent* over $\ell = 1, \ldots, D$ can further improve these variances. To the best of our knowledge, our work is the first one to report statistical advantages of structured polynomial sketches.

Moreover, we developed complex-valued generalizations for both structured and non-structured Product-Sketches and elucidated conditions under which such generalizations yield lower variances. As complex-valued random feature maps require the downstream application to handle complex data, we proposed a Complex-to-Real (CtR) transformation for them (Chapter 4) that makes it possible to use these sketches as a drop-in replacement for classical real-valued random features. We studied the impact of this transformation on the approximation variances and showed that CtR sketches keep the variance advantage of the complex sketches introduced before. We also derived a novel CtR-TensorSRHT sketch that outperforms the state-of-the-art in terms of computation speed and kernel approximation errors for low-degree polynomial kernels.

In Chapter 5, we proposed a technique to combine the aforementioned polynomial sketches in order to approximate more general dot product kernels. We

developed a novel feature distribution algorithm that exploits the previously derived variances in order to allocate random features among polynomial sketches of different degrees. We showed that these allocations can improve kernel approximations considerably, which confirms the importance of this subject that has been neglected in related works. At the same time, we uncovered a pathology associated with Maclaurin-based approximations of the Gaussian kernel and proposed a cure for this pathology, leading to state-of-the-art performance for the task of Gaussian kernel approximation across different length scale parameterizations.

Lastly, using the techniques developed in this thesis, we derived the limiting kernel for the random feature approximations obtained by an optical hardware, the Optical Processing Unit. We discussed relations of this kernel to the polynomial kernel and demonstrated its usefulness in image classification tasks.

With this thesis, we have set the foundation for a principled way to improve random feature approximations for dot product kernels. Yet, there is still space for improvements and we summarize some remaining challenges in the following.

**1) Exploration of alternatives to complex weights for variance reduction.** We have shown that Product-Sketches (7.1) using complex weights $\boldsymbol{w}_{i,\ell}$ can achieve considerable variance reductions, in particular for high-degree polynomial kernels. This is because complex random features approximate the linear kernel $(\boldsymbol{x}^\top \boldsymbol{y})$ with a lower variance than their real-valued analogs, and this effect is amplified through the product in Eq. (7.1), as we show in Chapter 4.

However, there is no guarantee that our proposed complex-valued estimate of the linear kernel is optimal. We have only proved the optimality of complex Rademacher vectors under the assumption that the elements of $\boldsymbol{w}_{i,\ell}$ are sampled i.i.d. and that the vectors are independent over $i = 1, \ldots, p$ and $\ell = 1, \ldots, D$. Future work should therefore explore whether there exist more efficient estimates of the linear kernel that would have an even larger effect on the variance reduction when approximating degree-p polynomial kernels using the estimator in Eq. (7.1).

It would also be interesting to see if an improvement can be achieved by removing the independence assumption over $i = 1, \ldots, p$ as it is done for some polynomial sketches in the literature such as TensorSketch (Pham and Pagh, 2013) and TensorSRHT by Ahle et al. (2020). In this case however, it becomes difficult to derive the associated approximation variances.

**2) Improving the optimized Maclaurin method.** Although we have shown that our proposed optimized Maclaurin approach optimally allocates the number of random features for a given *stochastic* optimization objective, it would be desirable to make this approach simpler to use in practice. The current algorithm relies on the evaluation of the variance formulas on pairs of data points from a given data set, where the user needs to specify the number of pairs being evaluated.

It would be desirable to achieve a feature allocation without the need for pairwise evaluations, simply by looking at the scale of the data at hand using a one-pass

algorithm with a linear time complexity. We showed in our empirical campaign that the scale of the data is the main driver for the distribution of random features between polynomial degrees. E.g., when the data is modelled using a large length scale, pairwise dot products are distributed closely around the origin and most random features are attributed to low polynomial degrees. The opposite is the case for high-frequency data modelled with a short length scale compared to the scale of the data. There may thus be ways to "shift" the random feature allocation between degrees according to the distribution of the norms of the data (taking the length scale being used into account).

Future work should further investigate the use of alternative basis functions for the approximation of general dot product kernels, e.g., Gegenbauer polynomials as they have lately been proposed by Han et al. (2022). In this case, the ideas developed in this thesis for Maclaurin expansions could be transferred to Gegenbauer expansions, potentially requiring only lower order expansions for the same approximation quality. These may eventually lead to lower approximation variances.

**3) Approximating extreme-frequency data.** Recently, a random feature approximation of the exponential kernel (Choromanski et al., 2021) has been proposed for Transformer architectures (Vaswani et al., 2017). Such a method has the potential of making Transformers scalable to long input sequences and to incur memory savings as explained in Chapter 1. This topic is closely related to the contents of this thesis because the exponential kernel $\exp(\boldsymbol{x}^\top \boldsymbol{y}/l^2)$ for $l > 0$ is a dot product kernel.

However, such random feature approximations require a significant amount of fine-tuning of entire Transformer architectures to work well in practice, and they still result in performance degradation compared to using the exact attention mechanism as shown in a recent benchmark (Tay et al., 2021). This is mainly because the attention patterns that typically arise in Transformer models are extremely sparse. This sparsity in turn stems from high-magnitude inputs being passed to the softmax function that is an integral part of attention layers.

We demonstrate this effect in Fig. 7.1, which shows unnormalized attention patterns, i.e., softmax evaluations without the softmax denominator, of the first layer of a pretrained Bert-based (Devlin et al., 2019) model when processing an IMDB movie review (Maas et al., 2011) as an example input. Every pair of rows in Fig. 7.1 corresponds to attention patterns obtained from a single attention head. The first column shows the first 50x50 out of 512x512 values contained in the matrix $\exp(\boldsymbol{Q}\boldsymbol{K}^\top)$ (see Chapter 1 for a definition), where the rows of $\boldsymbol{Q}$ and $\boldsymbol{K}$ are normalized to unit length in every second row of the figure.

Columns 2-4 show the random feature approximations of $\exp(\boldsymbol{Q}\boldsymbol{K}^\top)$, where approximation errors (relative Frobenius norm errors) are shown in parenthesis in the titles. Here we compare a modified random Fourier feature map (Rahimi and Recht, 2007) using structured projections (see Yu et al. (2016)) against FAVOR+

(Choromanski et al., 2021) and our optimized Maclaurin method (Chapter 5). It becomes clear that the approximation quality of all methods is quite poor, but improves considerably when approximating attention maps with normalized $\boldsymbol{Q}$ and $\boldsymbol{K}$.

This is because the distribution of the entries of $\boldsymbol{Q}\boldsymbol{K}^\top$ (right column) has a much larger spread in the unnormalized case producing extreme values for $\exp(\boldsymbol{Q}\boldsymbol{K}^\top)$ that ultimately lead to sparse attention patterns in practice. As shown in this thesis and in Choromanski et al. (2021), approximation variances typically scale with the norms of the inputs, $\boldsymbol{Q}$ and $\boldsymbol{K}$ in this case, and it is therefore very challenging to approximate the original sparse attention patterns. Although the optimized Maclaurin method achieves very good results when using normalized $\boldsymbol{Q}$ and $\boldsymbol{K}$, it still struggles when approximating the original attention matrix (the same holds for the RFF and FAVOR+ approaches proposed in the literature (Choromanski et al., 2021)).

We therefore conclude that the current state-of-the-art of random feature approximations for dot product kernels is not mature enough to approximate sparse attention patterns. We believe that more efficient approximations would be a significant step towards more efficient modern neural network architectures like Transformers and bilinear neural networks (Lin et al., 2018) (see Chapter 1). This thesis has laid the foundations for further developments in this direction.

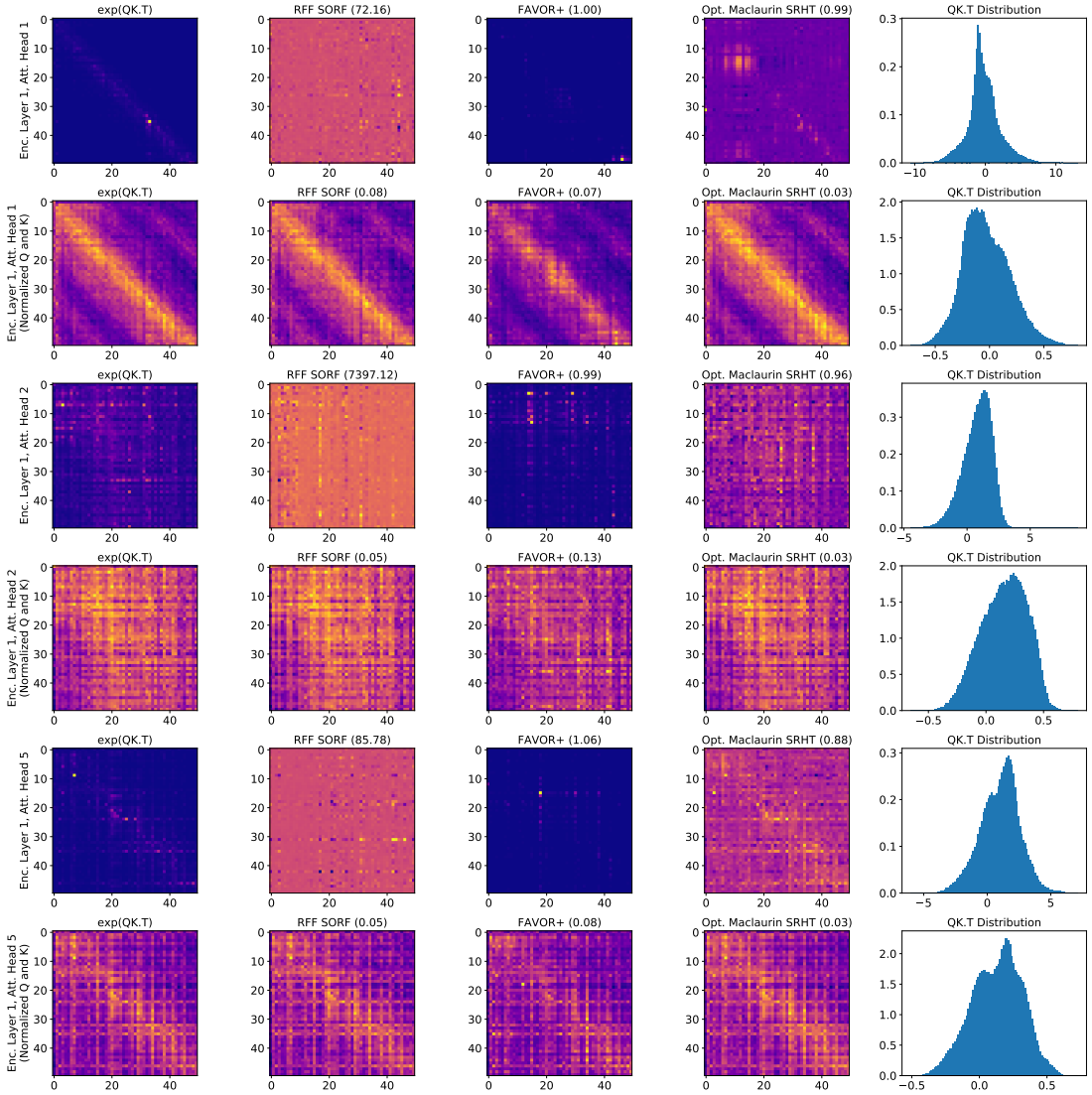



Figure 7.1: Unnormalized attention maps extracted from a single text fragment of length 512 from the IMDB movie reviews (Maas et al., 2011) using a pretrained Bert-base model (Devlin et al., 2019). We show the first 50x50 out of 512x512 pairwise attention evaluations of $\exp(\boldsymbol{QK}^\top)$ along with their approximations through different random feature methods (rel. Frob. error shown in parenthesis in the title). The rightmost column shows a histogram of the entries of $\boldsymbol{QK}^\top$. We show three different attention heads with the rows of $\boldsymbol{Q}$ and $\boldsymbol{K}$ being normalized to unit-length in every second row.

# Appendix A

# Appendix of Chapter 2

## A.1 Sum of Independent and Uniformly Distributed Variables

In Section 2.2.2, we use the following result. Let $A, B \sim U(\{0, \ldots, D-1\})$ be independent. Then $C = (A + B) \bmod D \sim U(\{0, \ldots, D-1\})$.

*Proof.* Without the modulus operation, we have:

$$\begin{aligned} P(A + B = c) &= \sum_{b=0}^{D-1} P(A = c - b | B = b) P(B = b) \\ &= \frac{1}{D} \sum_{b=0}^{D-1} P(A = c - b | B = b) = \frac{1}{D} \sum_{b=0}^{D-1} P(A = c - b) \end{aligned}$$

Now we use the relation $(A + B) \bmod D = c \iff A = (c - B) \bmod D$. So it follows that $((c-b) \bmod D) \in \{0, \ldots, D-1\}$ and hence $P(A = (c-b) \bmod D) = 1/D$. Therefore, $C \sim U(\{1, \ldots, D-1\})$.

■

# Appendix B

# Appendix of Chapter 3

## B.1 Proofs for Sections 3.1-3.4

### B.1.1 Proof of Theorem 3.3.1

We first show

$$\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})] = \Bigg(\sum_{k=1}^{d}\mathbb{E}[|z_k|^4]x_k^2y_k^2 + \|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 - 2\sum_{k=1}^{d}x_k^2y_k^2 + (\boldsymbol{x}^\top\boldsymbol{y})^2 + \sum_{i=1}^{d}\sum_{\substack{j=1\\j\neq i}}^{d}\mathbb{E}[z_i^2]\mathbb{E}[\overline{z_j}^2]x_ix_jy_iy_j\Bigg)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}. \tag{B.1}$$

where $z_i^2 := z_iz_i$ and $\overline{z_i}^2 := \overline{z_iz_i}$ are in general different from $|z_i|^2 = z_i\overline{z_i}$. We have

$$\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})] = \mathbb{E}[|\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})|^2] - |\mathbb{E}[\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})]|^2 = \mathbb{E}[|\prod_{i=1}^{p}\boldsymbol{z}_i^\top\boldsymbol{x}\overline{\boldsymbol{z}_i^\top\boldsymbol{y}}|^2] - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}$$
$$= \prod_{i=1}^{p}\mathbb{E}[|\boldsymbol{z}_i^\top\boldsymbol{x}\overline{\boldsymbol{z}_i^\top\boldsymbol{y}}|^2] - (\boldsymbol{x}^\top\boldsymbol{y})^{2p} = (\mathbb{E}[|\boldsymbol{z}^\top\boldsymbol{x}\overline{\boldsymbol{z}}^\top\boldsymbol{y}|^2])^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}. \tag{B.2}$$

Henceforth we focus on $\mathbb{E}[|\boldsymbol{z}^\top\boldsymbol{x}\overline{\boldsymbol{z}}^\top\boldsymbol{y}|^2]$ in the last expression (B.2). Write $\boldsymbol{z} = (z_1,\ldots,z_d)^\top$, $\boldsymbol{x} = (x_1,\ldots,x_d)^\top$, and $\boldsymbol{y} = (y_1,\ldots,y_d)^\top$. Since $\mathbb{E}[\boldsymbol{z}\overline{\boldsymbol{z}}^\top] = \boldsymbol{I}_d$, we have $\mathbb{E}[z_i\overline{z_j}] = 1$ if $i = j$ and $\mathbb{E}[z_i\overline{z_j}] = 0$ if $i \neq j$. Recall also that $z_1,\ldots,z_d \in \mathbb{C}$ are i.i.d, and $\mathbb{E}[z_i] = 0$ for $i = 1,\ldots,d$. Then

$$\mathbb{E}[|\boldsymbol{z}^\top\boldsymbol{x}\overline{\boldsymbol{z}}^\top\boldsymbol{y}|^2] = \mathbb{E}\left[(\sum_{i=1}^{d}z_ix_i)(\sum_{j=1}^{d}\overline{z_j}y_j)(\sum_{k=1}^{d}\overline{z_k}x_k)(\sum_{l=1}^{d}z_ly_l)\right]$$
$$= \sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{k=1}^{d}\sum_{l=1}^{d}\mathbb{E}[z_i\overline{z_jz_k}z_l]x_iy_jx_ky_l. \tag{B.3}$$

The expected value $\mathbb{E}\left[z_i\overline{z_jz_k}z_l\right]$ is different from 0, only if:

(a) $i = j = k = l$, for which there are $d$ terms and $\mathbb{E}\big[z_i\overline{z_j z_k} z_l\big] x_i y_j x_k y_l = \mathbb{E}[|z_i|^4] x_i^2 y_i^2$.

(b) $i = j \neq k = l$, for which there are $d(d-1)$ terms and $\mathbb{E}\big[z_i\overline{z_j z_k} z_l\big] x_i y_j x_k y_l = \mathbb{E}[|z_i|^2]\mathbb{E}[|z_k|^2] x_i x_k y_i y_k$.

(c) $i = k \neq j = l$, for which there are $d(d-1)$ terms and $\mathbb{E}\big[z_i\overline{z_j z_k} z_l\big] x_i y_j x_k y_l = \mathbb{E}[|z_i|^2]\mathbb{E}[|z_j|^2] x_i^2 y_j^2$.

(d) $i = l \neq j = k$, for which there are $d(d-1)$ terms and $\mathbb{E}\big[z_i\overline{z_j z_k} z_l\big] x_i y_j x_k y_l = \mathbb{E}[z_i^2]\mathbb{E}[\overline{z_j}^2] x_i x_j y_i y_j$.

Therefore,

$$
\begin{aligned}
\text{(B.3)} &= \underbrace{\sum_{i=1}^{d} \mathbb{E}[|z_i|^4] x_i^2 y_i^2}_{\text{case (a)}} + \underbrace{\sum_{i=1}^{d}\sum_{\substack{j=1\\ j\neq i}}^{d} \mathbb{E}[|z_i|^2]\mathbb{E}[|z_j|^2] x_i^2 y_j^2}_{\text{case (c)}} + \underbrace{\sum_{i=1}^{d}\sum_{\substack{j=1\\ j\neq i}}^{d} \mathbb{E}[|z_i|^2]\mathbb{E}[|z_j|^2] x_i x_j y_i y_j}_{\text{case (b)}} \\
&\quad + \underbrace{\sum_{i=1}^{d}\sum_{\substack{j=1\\ j\neq i}}^{d} \mathbb{E}[z_i^2]\mathbb{E}[\overline{z_j}^2] x_i x_j y_i y_j}_{\text{case (d)}} \\
&= \sum_{i=1}^{d} \mathbb{E}[|z_i|^4] x_i^2 y_i^2 + \sum_{i=1}^{d}\sum_{\substack{j=1\\ j\neq i}}^{d} x_i^2 y_j^2 + \sum_{i=1}^{d}\sum_{\substack{j=1\\ j\neq i}}^{d} x_i x_j y_i y_j + \sum_{i=1}^{d}\sum_{\substack{j=1\\ j\neq i}}^{d} \mathbb{E}[z_i^2]\mathbb{E}[\overline{z_j}^2] x_i x_j y_i y_j \\
&= \sum_{i=1}^{d} \mathbb{E}[|z_i|^4] x_i^2 y_i^2 + \left[\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 - \sum_{i=1}^{d} x_i^2 y_i^2\right] + \left[(\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^{d} x_i^2 y_i^2\right] \\
&\quad + \sum_{i=1}^{d}\sum_{\substack{j=1\\ j\neq i}}^{d} \mathbb{E}[z_i^2]\mathbb{E}[\overline{z_j}^2] x_i x_j y_i y_j
\end{aligned}
$$

The proof of Eq. (B.1) completes by using this expression of $\mathbb{E}[(\boldsymbol{z}^\top \boldsymbol{x}\overline{\boldsymbol{z}^\top \boldsymbol{y}})^2]$ in Eq. (B.2).

Eq. (3.16) follows from Eq. (B.1) and $\mathbb{E}[z_k^2] = \mathbb{E}[\overline{z_k}^2] = 2q - 1$, which uses Eq. (3.15). The lower bound Eq. (3.17) follows from Jensen's inequality $\mathbb{E}[|z_k|^4] \geq (\mathbb{E}[|z_k|^2])^2 = 1$.

### B.1.2 Proof of Theorem 3.4.1

We make use of Bernstein's inequality (e.g., Vershynin, 2018, Theorem 2.8.4): For independent random variables $X_1, \ldots, X_D \in \mathbb{R}$ such that $\mathbb{E}[X_i] = 0$ and $|X_i| \leq R$

almost surely for a constant $R > 0$, we have for any $t > 0$:

$$\Pr\left[\left|\sum_{i=1}^{D} X_i\right| \geq t\right] \leq 2\exp\left(\frac{-t^2/2}{\sum_{i=1}^{D}\mathbb{V}\left[X_i\right] + Rt/3}\right) \tag{B.4}$$

We define $X_i := \Phi_{\mathcal{C}}(\boldsymbol{x})_i\overline{\Phi_{\mathcal{C}}(\boldsymbol{y})_i} - (\boldsymbol{x}^\top\boldsymbol{y})^p/D \in \mathbb{R}$, where $\Phi_{\mathcal{C}}(\boldsymbol{x}) \in \mathbb{C}^D$ is defined in Eq. (3.10): $\Phi_{\mathcal{C}}(\boldsymbol{x}) = \frac{1}{\sqrt{D}}\left[(\prod_{i=1}^{p}\boldsymbol{z}_{i,1}^\top\boldsymbol{x}), \ldots, (\prod_{i=1}^{p}\boldsymbol{z}_{i,D}^\top\boldsymbol{x})\right]^\top$. Then we have $\mathbb{E}[X_i] = 0$. Moreover,

$$\begin{aligned}|X_i| &\leq |\Phi_{\mathcal{C}}(\boldsymbol{x})_i\overline{\Phi_{\mathcal{C}}(\boldsymbol{y})_i}| + |(\boldsymbol{x}^\top\boldsymbol{y})^p/D| = \frac{1}{D}\left(\prod_{j=1}^{p}|\boldsymbol{z}_j^\top\boldsymbol{x}||\overline{\boldsymbol{z}_j^\top\boldsymbol{y}}| + |(\boldsymbol{x}^\top\boldsymbol{y})^p|\right)\\ &\leq \frac{1}{D}(\|\boldsymbol{x}\|_1^p\|\boldsymbol{y}\|_1^p + \|\boldsymbol{x}\|_2^p\|\boldsymbol{y}\|_2^p) \leq \frac{2}{D}\|\boldsymbol{x}\|_1^p\|\boldsymbol{y}\|_1^p =: R\end{aligned}$$

where the first inequality is the triangle inequality. The second inequality uses Hölder's inequality (and that the absolute value of each element of $\boldsymbol{z}_j$ is 1) as well as the upper bound $\boldsymbol{x}^\top\boldsymbol{y} \leq \|\boldsymbol{x}\|_2\|\boldsymbol{y}\|_2$. Furthermore, by assumption we have

$$\mathbb{V}[X_i] = \frac{\sigma^2\|\boldsymbol{x}\|_2^{2p}\|\boldsymbol{y}\|_2^{2p}}{D^2} \leq \frac{\sigma^2\|\boldsymbol{x}\|_1^{2p}\|\boldsymbol{y}\|_1^{2p}}{D^2}$$

for some $\sigma^2 \geq 0$. Therefore, using Eq. (B.4) and setting $t := \|\boldsymbol{x}\|_1^p\|\boldsymbol{y}\|_1^p\epsilon$, we have

$$\Pr\left[\left|\sum_{i=1}^{D} X_i\right| \geq \epsilon\|\boldsymbol{x}\|_1^p\|\boldsymbol{y}\|_1^p\right] \leq 2\exp\left(\frac{-D\epsilon^2/2}{\frac{2}{3}\epsilon + \sigma^2}\right)$$

Setting $D \geq 2(\frac{2}{3\epsilon} + \frac{\sigma^2}{\epsilon^2})\log(\frac{2}{\delta})$ and taking the complementary probability gives the desired result.

## B.2 Proofs for Section 3.5

### B.2.1 Key Lemma

First, we state a key lemma that is needed for deriving the variance of real and complex TensorSRHT. This result is essentially given in Choromanski et al. (2017, Proof of Proposition 8.2). However, their proof contains a typo missing the negative sign, and they use a different definition of the Hadamard matrix from ours. Therefore, for completeness, we state the result formally and provide a proof. We also provide an extension to stacked Hadamard matrices that is used for our upsampled TensorSRHT algorithm.

**Lemma B.2.1.** *Let $d = 2^m$ for some $m \in \mathbb{N}$ and $\boldsymbol{H}_d = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_d) \in \{1, -1\}^{d\times d}$ be the unnormalized Hadamard matrix defined in Eq. (2.21), where $\boldsymbol{h}_\ell = (h_{\ell,1}, \ldots, h_{\ell,d})^\top \in$*

$\{1,-1\}^d$ *for* $\ell \in \{1,\dots,d\}$. *Let* $\pi : \{1,\dots,d\} \to \{1,\dots,d\}$ *be a uniformly random permutation. Then for any* $\ell, \ell' \in \{1,\dots,d\}$ *with* $\ell \neq \ell'$ *and* $t, u \in \{1,\dots,d\}$ *with* $t \neq u$, *we have*

$$\mathbb{E}[h_{\pi(\ell),t} h_{\pi(\ell'),t} h_{\pi(\ell),u} h_{\pi(\ell'),u}] = -\frac{1}{d-1},$$

*where the expectation is with respect to the random permutation* $\pi$.

*Let* $B := \lceil D/d \rceil$. *We can then extend this result to a concatenated matrix* $\boldsymbol{H}_d^B := (\underbrace{\boldsymbol{H}_d, \dots, \boldsymbol{H}_d}_{B \text{ times}}) \in \{1,-1\}^{d \times Bd}$ *whose columns we name as* $\boldsymbol{h}_1^B, \dots, \boldsymbol{h}_{Bd}^B$. *In this case, we use the uniform permutation* $\pi^B : \{1,\dots,Bd\} \to \{1,\dots,Bd\}$. *Then for any* $\ell, \ell' \in \{1,\dots,Bd\}$ *with* $\ell \neq \ell'$ *and* $t, u \in \{1,\dots,d\}$ *with* $t \neq u$, *we have*

$$\mathbb{E}[h_{\pi(\ell),t} h_{\pi(\ell'),t} h_{\pi(\ell),u} h_{\pi(\ell'),u}] = -\frac{1}{\lceil D/d \rceil d - 1}.$$

*Proof.* We first derive a few key identities needed for our proof and we focus on using $\boldsymbol{H}_d$ instead of $\boldsymbol{H}_d^B$ for simplicity for now. For ease of notation, define

$$\alpha_\ell := h_{\ell,t} h_{\ell,u}, \quad \ell \in \{1,\dots,d\}.$$

Since any two distinct rows (and any two distinct columns) of $\boldsymbol{H}_d$ are orthogonal, we have

$$\sum_{\ell=1}^{d} \alpha_\ell = \sum_{\ell=1}^{d} h_{\ell,t} h_{\ell,u} = 0.$$

Since $\alpha_\ell \in \{-1, 1\}$, this identity implies that exactly $d/2$ elements in $\{\alpha_1, \dots, \alpha_d\}$ are 1, and the rest are $-1$. Note that for each $\ell \in \{1,\dots,d\}$ the randomly permuted index $\pi(\ell)$ takes values in $\{1,\dots,d\}$ with equal probabilities. Therefore, the probability of $\alpha_{\pi(\ell)}$ being 1 and that of $\alpha_{\pi(\ell)}$ being $-1$ are equal:

$$\Pr(\alpha_{\pi(\ell)} = 1) = \Pr(\alpha_{\pi(\ell)} = -1) = 0.5.$$

Note that $\pi^b(\ell) \neq \pi^b(\ell')$ since $\ell \neq \ell'$ and $\pi$ is a (random) permutation. Therefore, we have the following conditional probabilities:

$$\Pr(\alpha_{\pi(\ell')} = a \mid \alpha_{\pi(\ell)} = b) = \begin{cases} \frac{d/2-1}{d-1} & \text{if } a = b = 1 \text{ or } a = b = -1 \\ \frac{d/2}{d-1} & \text{if } a = 1, b = -1 \text{ or } a = -1, b = -1 \end{cases}$$

Using the above identities, we now prove the assertion:

$$\begin{aligned}
&\mathbb{E}[h_{\pi(\ell),t} h_{\pi(\ell'),t} h_{\pi(\ell),u} h_{\pi(\ell'),u}] = \mathbb{E}[\alpha_{\pi(\ell)} \alpha_{\pi(\ell')}] \\
&= \Pr(\alpha_{\pi(\ell)} = 1) \mathbb{E}[\alpha_{\pi(\ell)} \alpha_{\pi(\ell')} \mid \alpha_{\pi(\ell)} = 1] + \Pr(\alpha_{\pi(\ell)} = -1) \mathbb{E}[\alpha_{\pi(\ell)} \alpha_{\pi(\ell')} \mid \alpha_{\pi(\ell)} = -1] \\
&= \frac{1}{2} \mathbb{E}[\alpha_{\pi(\ell')} \mid \alpha_{\pi(\ell)} = 1] - \frac{1}{2} \mathbb{E}[\alpha_{\pi(\ell')} \mid \alpha_{\pi(\ell)} = -1] \\
&= \frac{1}{2} \left( \frac{d/2-1}{d-1} - \frac{d/2}{d-1} \right) - \frac{1}{2} \left( \frac{d/2}{d-1} - \frac{d/2-1}{d-1} \right) = -\frac{1}{d-1}.
\end{aligned}$$

Extending this analysis to the case of using $\boldsymbol{H}_d^B$ and $\pi^B$ instead of $\boldsymbol{H}_d$ and $\pi$ is straightforward. In this case, the marginal probabilities $\Pr(\alpha_{\pi^B(\ell)} = 1)$ and $\Pr(\alpha_{\pi^B(\ell)} = -1)$ remain the same because for two fixed rows $t, u$ of $\boldsymbol{H}_d^B$ with $t \neq u$, we still have

$$\sum_{\ell=1}^{Bd} \alpha_\ell = \sum_{\ell=1}^{d} h_{\ell,t}^B h_{\ell,u}^B = 0$$

and thus $Bd/2$ of the elements in $\{\alpha_1, \dots, \alpha_d\}$ must be 1 and $-1$, respectively. The conditional probabilities however change, and become

$$\Pr(\alpha_{\pi^B(\ell')} = a \mid \alpha_{\pi(\ell)} = b) = \begin{cases} \frac{(\lceil D/d \rceil d)/2 - 1}{\lceil D/d \rceil d - 1} & \text{if } a = b = 1 \text{ or } a = b = -1 \\ \frac{(\lceil D/d \rceil d)/2}{\lceil D/d \rceil d - 1} & \text{if } a = 1, b = -1 \text{ or } a = -1, b = -1 \end{cases}$$

This leads to

$$\mathbb{E}[h_{\pi^B(\ell),t}^B h_{\pi^B(\ell'),t}^B h_{\pi^B(\ell),u}^B h_{\pi^B(\ell'),u}^B] = \mathbb{E}[\alpha_{\pi^B(\ell)} \alpha_{\pi^B(\ell')}] = -\frac{1}{\lceil D/d \rceil d - 1}$$

in this case. ∎

### B.2.2 Proof of Theorem 3.5.9

We first clarify the notation we use. Recall that our feature map $\Phi(\boldsymbol{x}) \in \mathbb{C}^D$ is given by

$$\Phi(\boldsymbol{x}) = \frac{1}{\sqrt{D}} \left[ (\prod_{i=1}^{p} \boldsymbol{s}_{i,1}^\top \boldsymbol{x}), \dots, (\prod_{i=1}^{p} \boldsymbol{s}_{i,D}^\top \boldsymbol{x}) \right]^\top \in \mathbb{C}^D.$$

The random vectors $\boldsymbol{s}_{i,\ell} \in \mathbb{C}^d$ are independently generated blockwise, and there are $B := \lceil D/d \rceil$ blocks in total (and note that $D = (B-1)d + \mathrm{mod}(D,d)$): For each $i = 1, \dots, p$,

$$\begin{aligned} &\underbrace{(\boldsymbol{s}_{i,1}, \dots, \boldsymbol{s}_{i,d})}_{\text{Block } 1}, \underbrace{(\boldsymbol{s}_{i,d+1}, \dots, \boldsymbol{s}_{i,2d})}_{\text{Block } 2}, \ \dots, \\ &\qquad \underbrace{(\boldsymbol{s}_{i,(B-2)d+1}, \dots, \boldsymbol{s}_{i,(B-1)d})}_{\text{Block } B-1}, \underbrace{(\boldsymbol{s}_{i,(B-1)d+1}, \dots, \boldsymbol{s}_{i,(B-1)d+\mathrm{mod}(D,d)})}_{\text{Block } B} \\ =: &\underbrace{(\boldsymbol{s}_{i,1}^1, \dots, \boldsymbol{s}_{i,d}^1)}_{\text{Block } 1}, \underbrace{(\boldsymbol{s}_{i,1}^2, \dots, \boldsymbol{s}_{i,d}^2)}_{\text{Block } 2}, \ \dots, \ \underbrace{(\boldsymbol{s}_{i,1}^{B-1}, \dots, \boldsymbol{s}_{i,d}^{B-1})}_{\text{Block } B-1}, \underbrace{(\boldsymbol{s}_{i,1}^B, \dots, \boldsymbol{s}_{i,\mathrm{mod}(D,d)}^B)}_{\text{Block } B}, \end{aligned}$$

where we introduced in the second line a new notation:

$$\boldsymbol{s}_{i,\ell}^b := \boldsymbol{s}_{i,(b-1)d+\ell} \quad (b = 1, \dots, B, \ \ell = 1, \dots, d).$$

Here $b$ serves as the indicator of the $b$-th block. Thus, using this notation,

$$\boldsymbol{s}_{i,\ell}^b = \boldsymbol{z}_i^b \circ \boldsymbol{h}_{\pi^b(\ell)} \in \mathbb{C}^d \quad (\ell = 1, \dots, d),$$

where $\boldsymbol{z}_i^b = (z_{i,1}^b, \dots, z_{i,d}^b)^\top \in \mathbb{C}^d$ is a random vector whose elements $z_{i,1}^b, \dots, z_{i,d}^b$ are i.i.d., and $\pi^b : \{1, \dots, d\} \to \{1, \dots, d\}$ is a random permutation of the indices. Note that $\boldsymbol{z}_i^b$ and $\pi^b$ are generated independently for each $b \in \{1, \dots, B\}$. Therefore, the random vectors $\boldsymbol{s}_{i,\ell}^b$ and $\boldsymbol{s}_{i,\ell'}^{b'}$ are statistically independent if they are from different blocks, i.e., if $b \neq b'$.

For each $b = 1, \dots, B$, define $\boldsymbol{z}^b = (z_1^b, \dots, z_d^b)^\top \in \mathbb{C}^d$ as a random vector independently and identically distributed as $\boldsymbol{z}_1^b, \dots, \boldsymbol{z}_p^b$. Define

$$\boldsymbol{s}_\ell^b := \boldsymbol{z}^b \circ \boldsymbol{h}_{\pi(\ell)}^b = (z_1^b h_{\pi^b(\ell),1}, \dots, z_d^b h_{\pi^b(\ell),d})^\top =: (s_{\ell,1}^b, \dots, s_{\ell,d}^b)^\top \in \mathbb{C}^d. \tag{B.5}$$

Then $\boldsymbol{s}_\ell^b$ is independently and identically distributed as $\boldsymbol{s}_{1,\ell}^b, \dots, \boldsymbol{s}_{p,\ell}^b$. Moreover, given the permutation $\pi^b$ fixed, $\boldsymbol{s}_\ell^b$ is identically distributed as $\boldsymbol{z}^b$. This is because 1) $z_1^b, \dots, z_d^b$ are i.i.d., 2) each $z_t^b$ is symmetrically distributed ($t = 1, \dots, d$), and 3) $h_{\pi^b(\ell),1}, \dots, h_{\pi^b(\ell),d} \in \{1, -1\}$.

Now let us start proving the assertion. We first have

$$\mathbb{V}[\hat{k}(\boldsymbol{x}, \boldsymbol{y})] = \mathbb{E}[|\hat{k}(\boldsymbol{x}, \boldsymbol{y})|^2] - |\mathbb{E}[\hat{k}(\boldsymbol{x}, \boldsymbol{y})]|^2 = \mathbb{E}[|\hat{k}(\boldsymbol{x}, \boldsymbol{y})|^2] - (\boldsymbol{x}^\top \boldsymbol{y})^{2p},$$

where the second identity follows from the approximate kernel being unbiased for both real and complex TensorSRHT. Thus, from now on we study the term $\mathbb{E}[|\hat{k}(\boldsymbol{x}, \boldsymbol{y})|^2]$.

For simplicity of notation, define $I_b := \{1, \dots, d\}$ for $b = 1, \dots, B-1$ and $I_b := \{1, \dots, \mathrm{mod}(D, d)\}$ for $b = B$. Since the approximate kernel can be written as

$$\hat{k}(\boldsymbol{x}, \boldsymbol{y}) := \Phi(\boldsymbol{x})^\top \overline{\Phi(\boldsymbol{y})} = \frac{1}{D} \sum_{\ell=1}^{D} \prod_{i=1}^{p} \boldsymbol{s}_{i,\ell}^\top \boldsymbol{x} \overline{\boldsymbol{s}_{i,\ell}^\top \boldsymbol{y}} = \frac{1}{D} \sum_{b=1}^{B} \sum_{\ell \in I_b} \prod_{i=1}^{p} (\boldsymbol{s}_{i,\ell}^{b\top} \boldsymbol{x}) \, \overline{(\boldsymbol{s}_{i,\ell}^{b\top} \boldsymbol{y})}$$

its second moment can be written as

$$\begin{aligned}
\mathbb{E}[|\hat{k}(\boldsymbol{x}, \boldsymbol{y})|^2] &= \frac{1}{D^2} \sum_{b,b'=1}^{B} \sum_{\ell \in I_b} \sum_{\ell \in I_{b'}} \mathbb{E}\left[ \prod_{i=1}^{p} (\boldsymbol{s}_{i,\ell}^{b\top} \boldsymbol{x}) \, \overline{(\boldsymbol{s}_{i,\ell}^{b\top} \boldsymbol{y})(\boldsymbol{s}_{i,\ell'}^{b'\top} \boldsymbol{x})} \, (\boldsymbol{s}_{i,\ell'}^{b'\top} \boldsymbol{y}) \right] \\
&= \frac{1}{D^2} \sum_{b,b'=1}^{B} \sum_{\ell \in I_b} \sum_{\ell \in I_{b'}} \prod_{i=1}^{p} \mathbb{E}\left[ (\boldsymbol{s}_{i,\ell}^{b\top} \boldsymbol{x}) \, \overline{(\boldsymbol{s}_{i,\ell}^{b\top} \boldsymbol{y})(\boldsymbol{s}_{i,\ell'}^{b'\top} \boldsymbol{x})} \, (\boldsymbol{s}_{i,\ell'}^{b'\top} \boldsymbol{y}) \right] \\
&= \frac{1}{D^2} \sum_{b,b'=1}^{B} \sum_{\ell \in I_b} \sum_{\ell \in I_{b'}} \left( \mathbb{E}\left[ (\boldsymbol{s}_{\ell}^{b\top} \boldsymbol{x}) \, \overline{(\boldsymbol{s}_{\ell}^{b\top} \boldsymbol{y})(\boldsymbol{s}_{\ell'}^{b'\top} \boldsymbol{x})} \, (\boldsymbol{s}_{\ell'}^{b'\top} \boldsymbol{y}) \right] \right)^p .
\end{aligned} \tag{B.6}$$

Now we study individual terms in (B.6), categorizing the indices $b, b' \in \{1, \dots, B\}$ and $\ell, \ell' \in \{1, \dots, d\}$ of indices into the following 3 cases:

1. $b = b'$ and $\ell = \ell'$ ($D$ terms): As mentioned earlier, conditioned on the permutation $\pi^b$, $\boldsymbol{s}_\ell^b$ is identically distributed as $\boldsymbol{z}^b$ (see the paragraph following Eq. (B.5)). Thus,

$$\mathbb{E}\left[\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{x}\right)^2\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{y}\right)^2\right] = \mathbb{E}_{\pi^b}\left[\mathbb{E}\left[\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{x}\right)^2\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{y}\right)^2 \mid \pi^b\right]\right]$$
$$= \mathbb{E}_{\pi^b}\left[\mathbb{E}\left[\left(\boldsymbol{z}^{b\top}\boldsymbol{x}\right)^2\left(\boldsymbol{z}^{b\top}\boldsymbol{y}\right)^2\right]\right] = \mathbb{E}\left[\left(\boldsymbol{z}^{b\top}\boldsymbol{x}\right)^2\left(\boldsymbol{z}^{b\top}\boldsymbol{y}\right)^2\right] = \mathbb{E}\left[\left(\boldsymbol{z}^{\top}\boldsymbol{x}\right)^2\left(\boldsymbol{z}^{\top}\boldsymbol{y}\right)^2\right],$$

where $\mathbb{E}_{\pi^b}$ denotes the expectation with respect to $\pi^b$ and $\boldsymbol{z} \in \mathbb{C}^d$ is a random vector identically distributed as $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^B$.

2. $b = b'$ and $\ell \neq \ell'$ ($c(D,d)$ terms, where $c(D,d)$ is defined in Eq. (3.29)): This case requires a detailed analysis, which we will do below.

3. $b \neq b'$ (The rest of terms $= D^2 - D - c(D,d)$ terms): Since $\boldsymbol{s}_\ell^b$ and $\boldsymbol{s}_{\ell'}^{b'}$ are independent in this case, we have

$$\mathbb{E}\left[\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{x}\right)\overline{\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{y}\right)\left(\boldsymbol{s}_{\ell'}^{b'\top}\boldsymbol{x}\right)}\left(\boldsymbol{s}_{\ell'}^{b'\top}\boldsymbol{y}\right)\right] = \mathbb{E}\left[\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{x}\right)\overline{\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{y}\right)}\right]\mathbb{E}\left[\overline{\left(\boldsymbol{s}_{\ell'}^{b'\top}\boldsymbol{x}\right)}\left(\boldsymbol{s}_{\ell'}^{b'\top}\boldsymbol{y}\right)\right]$$
$$= \mathbb{E}[\hat{k}(\boldsymbol{x},\boldsymbol{y})]\overline{\mathbb{E}[\hat{k}(\boldsymbol{x},\boldsymbol{y})]} = (\boldsymbol{x}^\top\boldsymbol{y})^2,$$

where the last equality follows from the approximate kernel being unbised.

We now analyze the case 2:

$$\mathbb{E}\left[\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{x}\right)\overline{\left(\boldsymbol{s}_\ell^{b\top}\boldsymbol{y}\right)\left(\boldsymbol{s}_{\ell'}^{b\top}\boldsymbol{x}\right)}\left(\boldsymbol{s}_{\ell'}^{b\top}\boldsymbol{y}\right)\right] = \sum_{t,u,w,v=1}^{d}\mathbb{E}[s_{\ell,t}^b\overline{s_{\ell,u}^b s_{\ell',v}^b}s_{\ell',w}^b]x_t y_u x_v y_w$$
$$= \sum_{t,u,w,v=1}^{d}\mathbb{E}[z_t^b\overline{z_u^b z_v^b}z_w^b]\underbrace{\mathbb{E}[h_{\pi^b(\ell),t}h_{\pi^b(\ell),u}h_{\pi^b(\ell'),v}h_{\pi^b(\ell'),w}]}_{=:E}x_t y_u x_v y_w$$

Note that we have $\mathbb{E}[z_t^b\overline{z_u^b z_v^b}z_w^b] = 0$ unless:

(a) $t = u = v = w$: $\mathbb{E}[z_t^b\overline{z_u^b z_v^b}z_w^b] = \mathbb{E}[|z_t^b|^4] = 1$ and $E = \mathbb{E}[h^2_{\pi^b(\ell),t}h^2_{\pi^b(\ell'),t}] = 1$.

(b) $t = u \neq v = w$: $\mathbb{E}[z_t^b\overline{z_u^b z_v^b}z_w^b] = \mathbb{E}[|z_t^b|^2|z_v^b|^2] = 1$ and $E = \mathbb{E}[h^2_{\pi^b(\ell),t}h^2_{\pi^b(\ell'),v}] = 1$.

(c) $t = v \neq u = w$: $\mathbb{E}[z_t^b\overline{z_u^b z_v^b}z_w^b] = \mathbb{E}[|z_t^b|^2|z_u^b|^2] = 1$ and $E = \mathbb{E}[h_{\pi^b(\ell),t}h_{\pi^b(\ell),u}h_{\pi^b(\ell'),t}h_{\pi^b(\ell'),u}]$.

(d) $t = w \neq u = v$: $\mathbb{E}[z_t^b\overline{z_u^b z_v^b}z_w^b] = \mathbb{E}[(z_t^b)^2(\overline{z_u^b})^2] = (2q-1)^2$ and $E = \mathbb{E}[h_{\pi^b(\ell),t}h_{\pi^b(\ell),u}h_{\pi^b(\ell'),u}h_{\pi^b(\ell'),t}]$.

Therefore, we have

$$\begin{aligned}
&\mathbb{E}\left[\boldsymbol{s}_\ell^\top \boldsymbol{x}\boldsymbol{s}_\ell^\top \boldsymbol{y}\boldsymbol{s}_{\ell'}^\top \boldsymbol{x}\boldsymbol{s}_{\ell'}^\top \boldsymbol{y}\right] \\
&= \sum_{t=1}^{d} x_t^2 y_t^2 + \sum_{t\neq v} x_t y_t x_v y_v + \sum_{t\neq u} \mathbb{E}[h_{\pi^b(\ell),t} h_{\pi^b(\ell'),t} h_{\pi^b(\ell),u} h_{\pi^b(\ell'),u}] \left(x_t^2 y_u^2 + (2q-1)^2 x_t y_t x_u y_u\right) \\
&= (\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{1}{d-1} \sum_{t\neq u} \left(x_t^2 y_u^2 + (2q-1)^2 x_t y_t x_u y_u\right) \quad (\because \text{Lemma B.2.1}) \\
&= (\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1},
\end{aligned}$$

where $V_q^{(1)} := \sum_{t\neq u} \left(x_t^2 y_u^2 + (2q-1)^2 x_t y_t x_u y_u\right)$ is Eq. (3.17) with $p = 1$, which is the variance of the unstructured polynomial sketch (3.10) with a single feature.

Now, using these identities in Eq. (B.6), the variance of the approximate kernel can be expanded as

$$\begin{aligned}
&\mathbb{V}[\hat{k}(\boldsymbol{x},\boldsymbol{y})] = \mathbb{E}[\hat{k}(\boldsymbol{x},\boldsymbol{y})^2] - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \\
=&\frac{1}{D}\left(\mathbb{E}\left[\left(\boldsymbol{z}^\top \boldsymbol{x}\right)^2 \left(\boldsymbol{z}^\top \boldsymbol{y}\right)^2\right]\right)^p + \frac{c(D,d)}{D^2}\left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1}\right)^p \\
&+ \frac{D^2 - D - c(D,d)}{D^2}(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - (\boldsymbol{x}^\top \boldsymbol{y})^{2p} \\
=&\frac{1}{D}\left[\left(\mathbb{E}\left[\left(\boldsymbol{z}^\top \boldsymbol{x}\right)^2 \left(\boldsymbol{z}^\top \boldsymbol{y}\right)^2\right]\right)^p - (\boldsymbol{x}^\top \boldsymbol{y})^{2p}\right] - \frac{c(D,d)}{D^2}\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1}\right)^p\right] \\
=&\frac{1}{D}V_q^{(p)} - \frac{c(D,d)}{D^2}\left[(\boldsymbol{x}^\top \boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top \boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1}\right)^p\right]
\end{aligned}$$

where $V_q^{(p)} \geq 0$ is Eq. (3.17) with the considered value of the polynomial degree $p$, which is the variance of the unstructured polynomial sketch (3.10) with a single feature. This completes the proof.

**Extending the proof to upsampled TensorSRHT**

For the case of upsampled TensorSRHT, we use a single block of size $\lceil D/d \rceil d$. Therefore, we only need to distinguish the first two cases in the analysis of Eq. (B.6), 1. $\ell = \ell'$ ($D$ terms) and 2. $\ell \neq \ell'$, for which there are now $D^2 - D$ terms.

Case 1. remains identical to stacked TensorSRHT. Case 2. is almost identical, the difference being that the number of non-zero terms for which $\ell \neq \ell'$ is now $D^2 - D$ instead of $c(D,d)$. Moreover, we do not permute the columns of a single matrix $\boldsymbol{H}_d$, but the ones of $\boldsymbol{H}_d^B$, which is why we need to make use of the second part of Lemma B.2.1. We thus substitute $\mathbb{E}[h_{\pi^b(\ell),t} h_{\pi^b(\ell'),t} h_{\pi^b(\ell),u} h_{\pi^b(\ell'),u}]$ by $-\frac{1}{\lceil D/d \rceil d - 1}$ instead of $-\frac{1}{d-1}$.

This finally leads to

$$\mathbb{V}[\hat{k}(\boldsymbol{x},\boldsymbol{y})] = \frac{1}{D}V_q^{(p)} - \left(1-\frac{1}{D}\right)\left[(\boldsymbol{x}^\top\boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{V_q^{(1)}}{\lceil D/d\rceil d-1}\right)^p\right].$$

## B.3 Gaussian Processes with Complex Random Features

We describe here how to use complex random features in Gaussian process (GP) regression and classification. Since real random features are special cases of complex random features, all derivations for the complex case also hold for the real case as well.

For GP classification, we employ the framework of Milios et al. (2018), which formulates GP classification using GP regression and provides a solution in closed form. Therefore, closed form solutions are available for both GP regression and classification, and this enables us to compare different random feature approximations directly.[1]

**Notation and definitions.** For a matrix $\boldsymbol{A} \in \mathbb{C}^{n\times m}$ with $n, m \in \mathbb{N}$, denote by $\boldsymbol{A}^H := \overline{\boldsymbol{A}}^\top \in \mathbb{C}^{m\times n}$ be its conjugate transpose. Note that if $\boldsymbol{A} \in \mathbb{R}^{n\times m}$, then $\boldsymbol{A}^H = \boldsymbol{A}^\top \in \mathbb{R}^{m\times n}$. For $n \in \mathbb{N}$, $\boldsymbol{I}_n \in \mathbb{R}^{n\times n}$ be the identity matrix.

For $\boldsymbol{\mu} \in \mathbb{C}^n$ and positive semi-definite[2] $\boldsymbol{\Sigma} \in \mathbb{C}^{n\times n}$ with $n \in \mathbb{N}$, we denote by $\mathcal{CN}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ the $n$-dimensional *proper* complex Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covaraince matrix $\boldsymbol{\Sigma}$, whose density function is given by (e.g., Neeser and Massey, 1993, Theorem 1)

$$\mathcal{CN}(\boldsymbol{v};\boldsymbol{\mu},\boldsymbol{\Sigma}) := \frac{1}{\pi^n\sqrt{|\boldsymbol{\Sigma}|}}\exp\left(-(\boldsymbol{v}-\boldsymbol{\mu})^H\boldsymbol{\Sigma}^{-1}(\boldsymbol{v}-\boldsymbol{\mu})\right), \quad \boldsymbol{v}\in\mathbb{C}^n,$$

where $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$. If a random vector $\boldsymbol{f} \in \mathbb{C}^n$ follows $\mathcal{CN}(\boldsymbol{v};\boldsymbol{\mu},\boldsymbol{\Sigma})$, we have $\mathbb{E}[\boldsymbol{f}] = \boldsymbol{\mu}$, $\mathbb{E}[(\boldsymbol{f}-\boldsymbol{\mu})(\boldsymbol{f}-\boldsymbol{\mu})^H] = \boldsymbol{\Sigma}$, and $\mathbb{E}[(\boldsymbol{f}-\boldsymbol{\mu})(\boldsymbol{f}-\boldsymbol{\mu})^\top] = 0$, where the last property is the definition of $\boldsymbol{f}$ being a proper complex random variable (Neeser and Massey, 1993, Definition 1).

### B.3.1 Complex GP Regression

We first describe the approach of *complex GP regression* (Boloix-Tortosa et al., 2018), a Bayesian nonparametric approach to complex-valued regression.

[1] If we use a formulation of GP classification that requires an optimization procedure, comparisons of random feature approximations become more involved, as we need to perform convergence verification for the optimization procedure.

[2] A Hermitian matrix $\boldsymbol{\Sigma} \in \mathbb{C}^{n\times n}$ is called positive semi-definite, if for all $\boldsymbol{v} \in \mathbb{C}^n$, we have $\boldsymbol{v}^H\boldsymbol{\Sigma}\boldsymbol{v} \geq 0$.

Suppose that there are training data $(\boldsymbol{x}_i, y_i)_{i=1}^N \subset \mathbb{R}^d \times \mathbb{C}$ for a complex-valued regression problem with $N \in \mathbb{N}$, and let $\boldsymbol{X} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^\top \in \mathbb{R}^{N \times d}$ and $\boldsymbol{y} := (y_1, \ldots, y_N)^\top \in \mathbb{C}^N$. We assume the following model for the training data:

$$y_i = f(x_i) + \varepsilon_i, \quad (i = 1, \ldots, N), \tag{B.7}$$

where $f : \mathbb{R}^d \to \mathbb{C}$ is an unknown complex-valued function, and $\varepsilon_i \sim \mathcal{CN}(0, \sigma_i^2)$ is an independent complex Gaussian noise with variance $\sigma_i^2 > 0$. Let $\boldsymbol{\sigma}^2 := (\sigma_1^2, \ldots, \sigma_N^2)^\top \in \mathbb{R}^N$.

The task of complex-valued function is to estimate the unknown complex-valued function $f$ in Eq. (B.7) from the training data $(\boldsymbol{x}_i, y_i)_{i=1}^N \subset \mathbb{R}^d \times \mathbb{C}$. In complex GP regression, one defines a *complex GP prior distribution* for the unknown function $f$, and derives a *complex GP posterior distribution* of $f$, given the data $(\boldsymbol{x}_i, y_i)_{i=1}^N \subset \mathbb{R}^d \times \mathbb{C}$ and the likelihood function given by Eq. (B.7). For the prior, we focus on a *proper* complex GP (Boloix-Tortosa et al., 2018, Section II-C), which we describe below.

**Proper complex Gaussian processes.** A complex-valued function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C}$ is called *positive definite kernel*, if 1) $k(\boldsymbol{x}, \boldsymbol{x}') = \overline{k(\boldsymbol{x}', \boldsymbol{x})}$ for all $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$; and ii) for all $n \in \mathbb{N}$ and all $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, the matrix $\boldsymbol{K} \in \mathbb{C}^{n \times n}$ with $\boldsymbol{K}_{i,j} = k(x_i, x_j)$ satisfies $\boldsymbol{v}^H \boldsymbol{K} \boldsymbol{v} \geq 0$.

Let $f : \mathbb{R}^d \to \mathbb{C}$ be a zero-mean complex-valued stochastic process, and $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C}$ be a positive definite kernel. We call $f$ a (zero-mean) *proper complex GP* with covariance kernel $k$, if for all $n \in \mathbb{N}$ and all $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, the random vector $\boldsymbol{f} := (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))^\top \in \mathbb{C}^n$ follows the proper complex Gaussian distribution $\mathcal{CN}(\boldsymbol{0}, \boldsymbol{K})$ with covariance matrix $\boldsymbol{K} \in \mathbb{C}^{n \times n}$ with $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. If $f$ is a zero-mean proper complex GP with covariance kernel $k$, we write $f \sim \mathcal{CGP}(0, k)$.

We now describe the approach of complex GP regression. For the unknown $f$ in Eq. (B.7), we define a proper complex GP prior with kernel $k$, assuming that

$$f \sim \mathcal{CGP}(0, k) \tag{B.8}$$

Then the observation model (B.7) and the prior (B.8) induce a joint distribution of the unknown function $f$ and the training observations $\boldsymbol{y} = (y_1, \ldots, y_N)^\top$. Conditioned on $\boldsymbol{y}$, we obtain the *posterior distribution* of $f$, which is also a proper complex GP (Boloix-Tortosa et al., 2018, Section II-C):

$$f \mid \boldsymbol{y} \sim \mathcal{CGP}(\mu_N, k_N), \tag{B.9}$$

where $\mu_N : \mathbb{R}^d \to \mathbb{C}$ is the *posterior mean function* and $k_N : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C}$ is the *posterior covariance function* given by

$$\mu_N(\boldsymbol{x}) := \boldsymbol{k}(\boldsymbol{x})^H (\boldsymbol{K} + \mathrm{diag}(\boldsymbol{\sigma}^2))^{-1} \boldsymbol{y}, \quad \boldsymbol{x} \in \mathbb{R}^d \tag{B.10}$$

$$k_N(\boldsymbol{x}, \boldsymbol{x}') := k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}(\boldsymbol{x})^H (\boldsymbol{K} + \mathrm{diag}(\boldsymbol{\sigma}^2))^{-1} \boldsymbol{k}(\boldsymbol{x}), \quad \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d, \tag{B.11}$$

where $\boldsymbol{k}(\boldsymbol{x}) := (k(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, k(\boldsymbol{x}, \boldsymbol{x}_N))^\top \in \mathbb{C}^N$, $\boldsymbol{K} \in \mathbb{C}^{N \times N}$ with $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $\mathrm{diag}(\boldsymbol{\sigma}^2) \in \mathbb{R}^{d \times d}$ is the diagonal matrix with diagonal elements $\boldsymbol{\sigma}^2 = (\sigma_1^2, \ldots, \sigma_N^2)^\top$. Notice that, if the kernel $k$ is real-valued and so are the observations $\boldsymbol{y}$, Eq. (B.10) and Eq. (B.11) reduce to the posterior mean and covariance functions of standard real-valued GP regression (e.g., Rasmussen and Williams, 2006, Chapter 2). In this sense, complex GP regression with a proper GP prior is a natural complex extension of standard GP regression.

### B.3.2 GP Regression with Complex Features

We next describe how to use complex features in GP regression. Let $\Phi : \mathbb{R}^d \to \mathbb{C}^D$ be a complex-valued (random) feature map,[3] and let $\hat{k}(\boldsymbol{x}, \boldsymbol{x}') := \Phi(\boldsymbol{x})^\top \overline{\Phi(\boldsymbol{x}')}$ be the approximate kernel. Define

$$\Phi(\boldsymbol{X}) := (\Phi(\boldsymbol{x}_1), \ldots, \Phi(\boldsymbol{x}_N))^\top \in \mathbb{C}^{N \times D}, \quad \hat{\boldsymbol{K}} := \Phi(\boldsymbol{X})\Phi(\boldsymbol{X})^H \in \mathbb{C}^{N \times N}, \tag{B.12}$$

where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$ are training inputs. Note that $\hat{\boldsymbol{K}}_{i,j} = \Phi(\boldsymbol{x}_i)^\top \overline{\Phi(\boldsymbol{x}_j)} = \hat{k}(\boldsymbol{x}_i, \boldsymbol{x}_j)$, i.e., $\hat{\boldsymbol{K}}$ is the kernel matrix with kernel $\hat{k}$.

The approximate kernel $\hat{k} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C}$ is complex-valued, and thus induces a proper complex GP, $f \sim \mathcal{CGP}(0, \hat{k})$. Using this GP as a prior for the unknown function $f$ in the observation model (B.7), and conditioning on the observations $\boldsymbol{y} = (y_1, \ldots, y_N)^\top$, we obtain the following approximate complex GP posterior:

$$f \mid \boldsymbol{y} \sim \mathcal{CGP}(\hat{\mu}_N, \hat{k}_N), \tag{B.13}$$

where $\hat{\mu}_N : \mathbb{R}^d \to \mathbb{C}$ is an approximate posterior mean function and $\hat{k}_N : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C}$ is an approximate posterior covariance function, defined as

$$\hat{\mu}_N(\boldsymbol{x}) := \hat{\boldsymbol{k}}(\boldsymbol{x})^H (\hat{\boldsymbol{K}} + \mathrm{diag}(\boldsymbol{\sigma}^2))^{-1} \boldsymbol{y}, \quad \boldsymbol{x} \in \mathbb{R}^d \tag{B.14}$$

$$\hat{k}_N(\boldsymbol{x}, \boldsymbol{x}') := \hat{k}(\boldsymbol{x}, \boldsymbol{x}') - \hat{\boldsymbol{k}}(\boldsymbol{x})^H (\hat{\boldsymbol{K}} + \mathrm{diag}(\boldsymbol{\sigma}^2))^{-1} \hat{\boldsymbol{k}}(\boldsymbol{x}), \quad \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d, \tag{B.15}$$

where $\hat{\boldsymbol{k}}(\boldsymbol{x}) := (\hat{k}(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \hat{k}(\boldsymbol{x}, \boldsymbol{x}_N))^\top \in \mathbb{C}^N$, and $\hat{\boldsymbol{K}} \in \mathbb{C}^{N \times N}$ with $\hat{\boldsymbol{K}}_{i,j} = \hat{k}(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Finally, we define a real-valued approximate GP posterior using the real parts of Eq. (B.14) and Eq. (B.15). That is, define $\hat{\mu}_{N,\mathbb{R}} : \mathbb{R}^d \to \mathbb{R}$ as the real part of the approximate posterior mean function in Eq. (B.14) , and $\hat{k}_{N,\mathbb{R}}$ as the real part of the approximate covariance function in Eq. (B.15):

$$\hat{\mu}_{N,\mathbb{R}}(\boldsymbol{x}) := \mathcal{R}\left\{\hat{\mu}_N(\boldsymbol{x})\right\}, \quad \boldsymbol{x} \in \mathbb{R}^d, \tag{B.16}$$

$$\hat{k}_{N,\mathbb{R}}(\boldsymbol{x}, \boldsymbol{x}') := \mathcal{R}\left\{\hat{k}_N(\boldsymbol{x}, \boldsymbol{x}')\right\}, \quad \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d. \tag{B.17}$$

[3] Again, this subsumes the case of real-valued feature maps.

Then, we define a real-valued GP with mean function $\hat{\mu}_{N,\mathbb{R}}$ and covariance function $\hat{k}_{N,\mathbb{R}}$:

$$f|\boldsymbol{y} \sim \mathcal{GP}(\hat{\mu}_{N,\mathbb{R}}, \hat{k}_{N,\mathbb{R}}).$$

We use this approximate GP for prediction tasks in our experiments.

Note that naive computations of Eq. (B.14) and Eq. (B.15) require $\mathcal{O}(N^3 + N^2D)$ complexity, and thus do not leverage the computational advantage of random features. We will show next how to reformulate Eq. (B.14) and Eq. (B.15) to compute them in $\mathcal{O}(D^3 + ND^2)$, which is linear in the number of training data points $N$.

### B.3.3 Computationally Efficient Implementation

We describe how to efficiently compute the approximate posterior mean and covariance functions in Eq. (B.14) and Eq. (B.15), respectively. To this end, recall the notation in Eq. (B.12). Let $\sigma^{-1} := (\sigma_1^{-1}, \dots, \sigma_N^{-1})^\top \in \mathbb{R}^N$ and $\sigma^{-2} := (\sigma_1^{-2}, \dots, \sigma_N^{-2})^\top \in \mathbb{R}^N$.

First we deal with Eq. (B.14). For a matrix $\boldsymbol{A} \in \mathbb{C}^{N\times D}$, we have $(\boldsymbol{A}^H\boldsymbol{A} + \boldsymbol{I}_N)\boldsymbol{A}^H = \boldsymbol{A}^H(\boldsymbol{A}\boldsymbol{A}^H + \boldsymbol{I}_D)$, and thus $\boldsymbol{A}^H(\boldsymbol{A}\boldsymbol{A}^H + \boldsymbol{I}_N)^{-1} = (\boldsymbol{A}^H\boldsymbol{A} + \boldsymbol{I}_D)^{-1}\boldsymbol{A}^H$. By using this last identity with $A = \operatorname{diag}(\boldsymbol{\sigma}^{-1})\Phi(X) \in \mathbb{C}^{N\times D}$, we can rewrite Eq. (B.14) as

$$\begin{aligned}
\hat{\mu}_N(\boldsymbol{x}) &= \hat{\boldsymbol{k}}(\boldsymbol{x})^H(\hat{\boldsymbol{K}} + \operatorname{diag}(\boldsymbol{\sigma}^2))^{-1}\boldsymbol{y}, \\
&= \Phi(\boldsymbol{x})^\top\Phi(\boldsymbol{X})^H\left(\Phi(\boldsymbol{X})\Phi(\boldsymbol{X})^H + \operatorname{diag}(\boldsymbol{\sigma}^2)\right)^{-1}\boldsymbol{y} \\
&= \Phi(\boldsymbol{x})^\top\Phi(\boldsymbol{X})^H\operatorname{diag}(\boldsymbol{\sigma}^{-1})\left(\operatorname{diag}(\boldsymbol{\sigma}^{-1})\Phi(\boldsymbol{X})\Phi(\boldsymbol{X})^H\operatorname{diag}(\boldsymbol{\sigma}^{-1}) + \boldsymbol{I}_N\right)^{-1}\operatorname{diag}(\boldsymbol{\sigma}^{-1})\boldsymbol{y} \\
&= \Phi(\boldsymbol{x})^\top\left(\Phi(\boldsymbol{X})^H\operatorname{diag}(\boldsymbol{\sigma}^{-2})\Phi(\boldsymbol{X}) + \boldsymbol{I}_D\right)^{-1}\Phi(\boldsymbol{X})^H\operatorname{diag}(\boldsymbol{\sigma}^{-2})\boldsymbol{y}. \qquad \text{(B.18)}
\end{aligned}$$

Next we deal with Eq. (B.15). For matrices $A, C, U, V$ of appropriate sizes with $A$ invertible, the Woodbury matrix identity states that $A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} = (A + UCV)^{-1}$. By using the Woodbury identity with $A = \boldsymbol{I}_D$, $C = \operatorname{diag}(\boldsymbol{\sigma}^{-2})$, $U = \Phi(X)^H$ and $V = \Phi(X)$, we can rewrite Eq. (B.15) as

$$\begin{aligned}
\hat{k}_N(\boldsymbol{x}, \boldsymbol{x}') &= \hat{k}(\boldsymbol{x}, \boldsymbol{x}') - \hat{\boldsymbol{k}}(\boldsymbol{x})^H(\hat{\boldsymbol{K}} + \operatorname{diag}(\boldsymbol{\sigma}^2))^{-1}\hat{\boldsymbol{k}}(\boldsymbol{x}) \\
&= \Phi(\boldsymbol{x})^\top\overline{\Phi(\boldsymbol{x})} - \Phi(\boldsymbol{x})^\top\Phi(\boldsymbol{X})^H\left(\Phi(\boldsymbol{X})\Phi(\boldsymbol{X})^H + \operatorname{diag}(\boldsymbol{\sigma}^2)\right)^{-1}\Phi(\boldsymbol{X})\overline{\Phi(\boldsymbol{x})} \\
&= \Phi(\boldsymbol{x})^\top\left(\boldsymbol{I}_D - \Phi(\boldsymbol{X})^H\left(\operatorname{diag}(\boldsymbol{\sigma}^2) + \Phi(\boldsymbol{X})\Phi(\boldsymbol{X})^H\right)^{-1}\Phi(\boldsymbol{X})\right)\overline{\Phi(\boldsymbol{x})} \\
&= \Phi(\boldsymbol{x})^\top\left(\boldsymbol{I}_D + \Phi(\boldsymbol{X})^H\operatorname{diag}(\boldsymbol{\sigma}^{-2})\Phi(\boldsymbol{X})\right)^{-1}\overline{\Phi(\boldsymbol{x})}. \qquad \text{(B.19)}
\end{aligned}$$

We now study the costs of computing Eq. (B.18) and Eq. (B.19). For both Eq. (B.18) and Eq. (B.19), the bottleneck is the computation of the inverse of the following matrix.

$$\boldsymbol{B} := \Phi(\boldsymbol{X})^H\operatorname{diag}(\boldsymbol{\sigma}^{-2})\Phi(\boldsymbol{X}) + \boldsymbol{I}_D \in \mathbb{C}^{D\times D}. \qquad \text{(B.20)}$$

The time complexity of computing $\boldsymbol{B}$ is $\mathcal{O}(ND^2)$, and that of the inverse $\boldsymbol{B}^{-1}$ is $\mathcal{O}(D^3)$, the latter being the complexity of computing the Cholesky decomposition $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^H$ with $\boldsymbol{L} \in \mathbb{C}^{D\times D}$ being a lower triangular matrix. Thus, the overall cost of computing $\boldsymbol{B}^{-1}$ is $\mathcal{O}(ND^2 + D^3)$.

We next conduct a more detailed analysis of the costs of $\boldsymbol{B}$ and its Cholesky decomposition, and compare them with the computational costs for the corresponding matrix inversion using real-valued features (i.e., when $\Phi(X) \in \mathbb{R}^{N\times D}$). Below we use the shorthand $\tilde{\Phi}(\boldsymbol{X}) := \mathrm{diag}(\boldsymbol{\sigma}^{-1})\Phi(\boldsymbol{X})$ so that $\boldsymbol{B} = \tilde{\Phi}(\boldsymbol{X})^H\tilde{\Phi}(\boldsymbol{X}) + \boldsymbol{I}_D$. Then the real and imaginary parts of $\boldsymbol{B}$ can be written as

$$\mathcal{R}\{\boldsymbol{B}\} = \mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})^H\tilde{\Phi}(\boldsymbol{X})\} + \boldsymbol{I}_D = \mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\} + \mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\} + \boldsymbol{I}_D$$
$$\mathcal{I}\{\boldsymbol{B}\} = \mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})^H\tilde{\Phi}(\boldsymbol{X})\} = \mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\} - \mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}.$$

Since $(\mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\})^\top = \mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}$, one can compute $\mathcal{I}\{\boldsymbol{B}\}$ by only computing $\mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\}$. Therefore, the computation of $\boldsymbol{B}$ requires the computations of the three real $D$-by-$D$ matrices (i.e., $\mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}$, $\mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\}$, and $\mathcal{R}\{\tilde{\Phi}(\boldsymbol{X})\}^\top\mathcal{I}\{\tilde{\Phi}(\boldsymbol{X})\}$). Thus, the total number of operations for computing $\boldsymbol{B}$ is $3 \cdot (ND^2) + 2 \cdot D^2$, where $3 \cdot (ND^2)$ is operations for the matrix products and $2 \cdot D^2$ for the addition and subtraction inside $\mathcal{R}\{\boldsymbol{B}\}$ and $\mathcal{I}\{\boldsymbol{B}\}$, respectively. Hence, assuming $N \gg D$, the computational cost for $\boldsymbol{B}$ is roughly 3 times more expensive than the corresponding cost for when $\Phi$ is real-valued.

Computing the Cholesky decomposition of a $D$ by $D$ matrix requires roughly $\frac{1}{6}D^3$ subtractions and $\frac{1}{6}D^3$ multiplications (e.g., Trefethen and Bau, 1997, p. 175). Therefore, when $\Phi$ is real-valued (and thus $\boldsymbol{B}$ is real-valued), the Cholesky decomposition of $\boldsymbol{B}$ requires $\frac{1}{6}D^3 + \frac{1}{6}D^3 = \frac{1}{3}D^3$ FLOPS. On the other hand, when $\Phi$ is complex-valued, the Cholesky decomposition of $\boldsymbol{B}$ require $\frac{4}{3}D^3$ FLOPS: one complex subtraction requires 2 real subtractions, and thus subtractions in total require $\frac{1}{6}D^3 \times 2 = \frac{1}{3}D^3$ FLOPS; one complex multiplication requires 4 real multiplications and 2 real subtractions, and thus multiplications in total require $\frac{1}{6}D^3 \times 6 = D^3$ FLOPS; thus $\frac{1}{3}D^3 + D^3 = \frac{4}{3}D^3$ FLOPS in total. Thus, the cost for computing the Cholesky decomposition of $\boldsymbol{B}$ when $\Phi$ is complex-valued is 4 times more expensive than the real-valued case.

The memory requirement for the complex case is 2 times as large as the real case, since the complex case requires storing both real and imaginary parts.

Note that, if one uses a $2D$-dimensional *real* feature map (i.e., $\Phi(\boldsymbol{X}) \in \mathbb{R}^{N\times 2D}$), then this requires 4 times as much memory, 4 times as many operations to compute the matrix $\boldsymbol{B}$, and 8 times as many operations for the Cholesky decomposition of $\boldsymbol{B}$ as those required for a $D$-dimensional real feature map. Therefore, using a $2D$-dimensional real feature map is computationally more expensive than using a $D$-dimensional complex feature map, since the latter only requires 2 times as much memory, 3 times as many operations for computing $\boldsymbol{B}$, and 4 times as many operations for computing the Cholesky decomposition of $\boldsymbol{B}$ as those required for a

$D$-dimensional real feature map, as shown above. Note also that the performance improvement from using a $D$-dimensional complex feature map is typically larger than using a $2D$-dimensional real feature map; see the experiments in Section 3.5.4 in Chapter 3.

### B.3.4 GP Classification as Closed-form Multi-output Regression

We now describe the GP classification approach of Milios et al. (2018), and how to use approximate posteriors for GP regression in this approach.

We assume that there are $C \in \mathbb{N}$ classes and that output labels are expressed by one-hot-encoding. Thus, for each class $c \in \{1, \dots, C\}$ and each training input $\boldsymbol{x}_i \in \mathbb{R}^d$ with $i = 1, \dots, N$, there exist an output $y_{c,i} \in \{0, 1\}$ such that $y_{c,i} = 1$ if $\boldsymbol{x}_i$ belongs to class $c$ and $y_{c,i} = 0$ otherwise.

**The approach of Milios et al. (2018).** Let $\alpha > 0$ be a constant. For each class $c \in \{1, \dots, C\}$, Milios et al. (2018) define transformed versions $\tilde{y}_{c,1}, \dots, \tilde{y}_{c,N} \in \mathbb{R}$ of the training outputs $y_{c,1}, \dots, y_{c,N}$ as

$$\tilde{y}_{c,i} := \log(y_{c,i} + \alpha) - \sigma_{c,i}^2/2, \quad \text{where} \quad \sigma_{c,i}^2 := \log((y_{c,i} + \alpha)^{-1} + 1), \quad i = 1, \dots, N.$$

Milios et al. (2018) then define an observation model of $\tilde{y}_{c,1}, \dots, \tilde{y}_{c,N}$ as

$$\tilde{y}_{c,i} = f_c(\boldsymbol{x}_i) + \varepsilon_{c,i}, \quad i = 1, \dots, N, \tag{B.21}$$

where $f_c : \mathbb{R}^d \to \mathbb{R}$ is a latent function and $\varepsilon_{c,i} \sim \mathcal{N}(0, \sigma_{c,i}^2)$ is an independent Gaussian noise with variance $\sigma_{c,i}^2$. Milios et al. (2018) propose to model $f_c$ for each $c \in \{1, \dots, C\}$ independently as a GP:

$$f_c \sim \mathcal{GP}(0, k), \tag{B.22}$$

where $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel. Eq. (B.21) and Eq. (B.22) define the joint distribution of the latent function $f_c$ and the transformed labels $\tilde{y}_{c,1}, \dots, \tilde{y}_{c,N}$. Thus, conditioning on $\tilde{y}_{c,1}, \dots, \tilde{y}_{c,N}$, one obtains a GP posterior of $f_c$. In other words, one can obtain a GP posterior of $f_c$ by performing GP regression for each class $c \in \{1, \dots, C\}$ using $(\boldsymbol{x}_i, \tilde{y}_{c,i})_{i=1}^N$ as training data.

The constant $\alpha$ is a hyperparameter, which Milios et al. (2018) propose to choose by cross validation, using the Mean Negative Log Likelihood (MNLL) (e.g., Rasmussen and Williams, 2006, p. 23) as an evaluation criterion.

**Using approximate GP posteriors.** We now explain how to use approximate posteriors for GP regression in the above approach: For each class $c \in \{1, \dots, C\}$, we perform approximate GP regression using $(\boldsymbol{x}_i, \tilde{y}_{c,i})_{i=1}^N$ as training data, to obtain an approximate GP posterior for the latent function $f_c$ in Eq. (B.21). For instance,

with our approach on approximate GP regression using complex random features in Appendix B.3.2, we obtain a GP posterior $f_c \sim \mathcal{GP}(\hat{\mu}_{N,\mathbb{R},c}, \hat{k}_{N,\mathbb{R},c})$ for each class $c \in \{1, \ldots, C\}$, where $\hat{\mu}_{N,\mathbb{R},c} : \mathbb{R}^d \to \mathbb{R}$ and $\hat{k}_{N,\mathbb{R},c} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ are the approximate GP posterior mean and covariance functions in Eq. (B.16) and Eq. (B.17), respectively, with $\boldsymbol{y} := (\tilde{y}_{c,1}, \ldots, \tilde{y}_{c,N})^\top$ and $\boldsymbol{\sigma}^2 := (\sigma^2_{c,1}, \ldots, \sigma^2_{c,N})^\top$.

For a given test input $\boldsymbol{x} \in \mathbb{R}^d$, one can obtain its posterior predictive probabilities over the $C$ classes in the following way. For each class $c \in \{1, \ldots, C\}$, we first generate a sample $z_c \in \mathbb{R}$ from the posterior distribution of the latent function value $f_c(\boldsymbol{x})$. We then apply the softmax transformation to $z_1, \ldots, z_C$ to obtain probabilities $p_1, \ldots, p_C \geq 0$ over the $C$ class labels: $p_c := \exp(z_c) / \sum_{j=1}^{C} \exp(z_j)$. Milios et al. (2018) show that these probabilities $p_1, \ldots, p_C$ are approximately a sample from a Dirichlet distribution, yielding well-calibrated predictions.

### B.3.5 Kullback-Leibler (KL) Divergence

In the experiments in Chapters 3 and 5, we use the Kullback-Leibler (KL) divergence between the exact and approximate GP posteriors, to evaluate the quality of each approximation approach. Let $\mu_{\text{exact}}(\boldsymbol{x})$ and $\sigma^2_{\text{exact}}(\boldsymbol{x})$ be the posterior mean and variance at $\boldsymbol{x} \in \mathbb{R}^d$ from the exact GP posterior, and let $\mu_{\text{appr}}(\boldsymbol{x})$ and $\sigma^2_{\text{appr}}(\boldsymbol{x})$ be those from an approximate GP posterior. Let $\boldsymbol{x}_{*,1}, \ldots, \boldsymbol{x}_{*,m_*} \in \mathbb{R}^d$ be test input points. Define $\boldsymbol{\mu}_{\text{exact}} := (\mu_{\text{exact}}(\boldsymbol{x}_{*,1}), \ldots, \mu_{\text{exact}}(\boldsymbol{x}_{*,m_*}))^\top$, $\boldsymbol{\sigma}^2_{\text{exact}} := (\sigma^2_{\text{exact}}(\boldsymbol{x}_{*,1}), \ldots, \sigma^2_{\text{exact}}(\boldsymbol{x}_{*,m_*}))^\top$, $\boldsymbol{\mu}_{\text{appr}} := (\mu_{\text{appr}}(\boldsymbol{x}_{*,1}), \ldots, \mu_{\text{appr}}(\boldsymbol{x}_{*,m_*}))^\top$, and $\boldsymbol{\sigma}^2_{\text{appr}} := (\sigma^2_{\text{appr}}(\boldsymbol{x}_{*,1}), \ldots, \sigma^2_{\text{appr}}(\boldsymbol{x}_{*,m_*}))^\top$.

We then measure the KL divergence between two diagonal Gaussian distributions, $\mathcal{N}(\boldsymbol{\mu}_{\text{appr}}, \text{diag}(\boldsymbol{\sigma}^2_{\text{appr}}))$ and $\mathcal{N}(\boldsymbol{\mu}_{\text{exact}}, \text{diag}(\boldsymbol{\sigma}^2_{\text{exact}}))$:

$$
\begin{aligned}
&KL\left[\mathcal{N}(\boldsymbol{\mu}_{\text{appr}}, \text{diag}(\boldsymbol{\sigma}^2_{\text{appr}})) \;||\; \mathcal{N}(\boldsymbol{\mu}_{\text{exact}}, \text{diag}(\boldsymbol{\sigma}^2_{\text{exact}}))\right] \\
&= \frac{1}{2} \sum_{i=1}^{m_*} \left( \frac{\sigma^2_{\text{exact}}(\boldsymbol{x}_{*,i})}{\sigma^2_{\text{appr}}(\boldsymbol{x}_{*,i})} + \log \frac{\sigma^2_{\text{exact}}(\boldsymbol{x}_{*,i})}{\sigma^2_{\text{appr}}(\boldsymbol{x}_{*,i})} - 1 + \frac{(\mu_{\text{exact}}(\boldsymbol{x}_{*,i}) - \mu_{\text{appr}}(\boldsymbol{x}_{*,i}))^2}{\sigma^2_{\text{appr}}(\boldsymbol{x}_{*,i})} \right),
\end{aligned}
\tag{B.23}
$$

We consider these diagonal Gaussian distributions, since the focus of our experiments in Chapters 3 and 5 is the prediction performance at test input points $\boldsymbol{x}_{*,1}, \ldots, \boldsymbol{x}_{*,m_*} \in \mathbb{R}^d$.

# Appendix C

# Appendix of Chapter 4

## C.1 Pseudo-Variances of Gaussian and Rademacher Polynomial Sketches

In this section, we work out the pseudo-variances for Gaussian and Rademacher Product-Sketches. Let

$$\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{D}\sum_{\ell=1}^{D}\prod_{i=1}^{p}(\boldsymbol{w}_{i,\ell}^{\top}\boldsymbol{x})(\boldsymbol{w}_{i,\ell}^{\top}\boldsymbol{y})$$

be the corresponding complex kernel estimate. As $\{\boldsymbol{w}_{i,\ell}\}_{\ell=1}^{D}$ are i.i.d. for Gaussian and Rademacher sketches, the variance of $\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y}) = \mathrm{Re}\{\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})\}$ is proportional to $1/D$ and we can assume $D = 1$ here for ease of presentation and drop the index $\ell$. We then rescale the variances by $1/D$ later.

Since the CtR-variance can be written as $\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] = \frac{1}{2}\mathrm{Re}\{\mathbb{V}[\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})] + \mathbb{PV}[\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})]\}$, the only thing left to work out is the pseudo-variance $\mathbb{PV}[\hat{k}_{\mathcal{C}}(\boldsymbol{x},\boldsymbol{y})])$.

As our estimator is unbiased, we further have $\mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] = k(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}^{\top}\boldsymbol{y})^{p}$. Thus, we only need to work out $\mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2]$.

$$\mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] = \mathbb{E}\left[\left(\prod_{i=1}^{p}\boldsymbol{w}_i^{\top}\boldsymbol{x}\overline{\boldsymbol{w}_i^{\top}\boldsymbol{y}}\right)^2\right] = \prod_{i=1}^{p}\mathbb{E}\left[(\boldsymbol{w}_i^{\top}\boldsymbol{x})^2(\overline{\boldsymbol{w}_i^{\top}\boldsymbol{y}})^2\right] \tag{C.1}$$

$$= \mathbb{E}\left[(\boldsymbol{w}^{\top}\boldsymbol{x})^2(\overline{\boldsymbol{w}^{\top}\boldsymbol{y}})^2\right]^p = \left(\sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{k=1}^{d}\sum_{l=1}^{d}\mathbb{E}[w_i w_j\overline{w_k w_l}]x_i x_j y_k y_l\right)^p \tag{C.2}$$

$\mathbb{E}_{ij\overline{kl}} := \mathbb{E}[w_i w_j\overline{w_k w_l}] \neq 0$, only if:

1. $i = j = k = l$: there are $d$ terms $(\mathbb{E}_{ij\overline{kl}})x_i x_j y_k y_l = \mathbb{E}[|w_i|^4]x_i^2 y_i^2$.
2. $i = k \neq j = l$: there are $d(d-1)$ terms $(\mathbb{E}_{ij\overline{kl}})x_i x_j y_k y_l = \mathbb{E}[|w_i|^2]x_i y_i\mathbb{E}[|w_j|^2]x_j y_j = x_i y_i x_j y_j$.

3. $i = l \neq j = k$: there are $d(d-1)$ terms $(\mathbb{E}_{ij\overline{kl}})x_i x_j y_k y_l = \mathbb{E}[|w_i|^2]x_i y_i \mathbb{E}[|w_j|^2]x_j y_j = x_i y_i x_j y_j$.

As for both the Gaussian and the Rademacher sketch, we have $\mathbb{E}[|w_i|^2] = 1$ for all $\{w_i\}_{i=1}^d$, we obtain:

$$\mathbb{E}\left[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2\right] = \left(\sum_{i=1}^{d}\mathbb{E}[|w_i|^4]x_i^2y_i^2 + 2\sum_{i=1}^{d}\sum_{j\neq i}^{d} x_i y_i x_j y_j\right)^p \tag{C.3}$$

We have $\mathbb{E}[|w_i|^4] = 2$ and $\mathbb{E}[|w_i|^4] = 1$ for the Gaussian and Rademacher case, respectively. So the pseudo-variances $\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] = \mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] - \mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]^2$ are given by the following real-valued expressions:

$$\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] = \frac{1}{D}\left(\left(2(\boldsymbol{x}^\top\boldsymbol{y})^2\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right) \quad \text{(Gaussian)} \tag{C.4}$$

$$\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] = \frac{1}{D}\left(\left(2(\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{i=1}^{d} x_i^2y_i^2\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right) \quad \text{(Rademacher)} \tag{C.5}$$

where we added the $1/D$ scaling that we left out before. Note that $\mathbb{E}[|w_i|^4] \geq (\mathbb{E}[|w_i|^2])^2 = 1$ by Jensen's inequality, which is why the Rademacher sketch yields the lowest possible pseudo-variance for the estimator studied in Eq. (3.11) in Section 3.2.

## C.2 Gaussian and Rademacher CtR Variance Advantage over their Real-Valued Analogs

In the following, we compare Gaussian and Rademacher CtR-sketches against their real-valued analogs assuming that the corresponding feature maps have equal dimensions. Thus, we assign $2D$ random features to the real feature map $\Phi_{\mathrm{R}} : \mathbb{R}^d \to \mathbb{R}^{2D}$ and only $D$ random features to the CtR feature map $\Phi_{\mathrm{CtR}} : \mathbb{R}^d \to \mathbb{R}^{2D}$ as it has twice as many dimensions for the same number of features.

We call the corresponding kernel estimates $\hat{k}_{\mathrm{R}}(\boldsymbol{x},\boldsymbol{y}) = \Phi_{\mathrm{R}}(\boldsymbol{x})^\top\Phi_{\mathrm{R}}(\boldsymbol{y})$ and $\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y}) = \Phi_{\mathrm{CtR}}(\boldsymbol{x})^\top\Phi_{\mathrm{CtR}}(\boldsymbol{y})$. $\mathbb{V}[\hat{k}_{\mathrm{R}}(\boldsymbol{x},\boldsymbol{y})]$ is given in Eq. (3.8) and Eq. (3.7).

We further have $\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] = \frac{1}{2}(\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] + \mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})])$ as shown in Section 4.2. $\mathbb{V}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$ is given in Eq. (3.19) and Eq. (3.18). $\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]$ is given in Eq. (C.4) and Eq. (C.5), respectively.

We start with the simpler Gaussian case and study the Rademacher case after.

**Gaussian case: Proof of Theorem (4.2.4).**

*Proof.* Taking into account the different number of random features for $\Phi_{\mathrm{R}}$ and $\Phi_{\mathrm{CtR}}$ to have equal dimensions, the variance difference of their kernel estimates

yields:

$$
\begin{aligned}
&\mathbb{V}[\hat{k}_{\mathrm{R}}(\boldsymbol{x},\boldsymbol{y})] - \mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] \\
&= \frac{1}{2D}\left(\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + 2(\boldsymbol{x}^\top\boldsymbol{y})^2\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right) \\
&\quad - \frac{1}{2D}\left(\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top\boldsymbol{y})^2\right)^p + \left(2(\boldsymbol{x}^\top\boldsymbol{y})^2\right)^p - 2(\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right) \\
&= \frac{1}{2D}\left(\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + 2(\boldsymbol{x}^\top\boldsymbol{y})^2\right)^p - \left(2(\boldsymbol{x}^\top\boldsymbol{y})^2\right)^p\right) - \frac{1}{2D}\left(\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top\boldsymbol{y})^2\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right) \\
&= \frac{1}{2D}\sum_{k=0}^{p-1}\binom{p}{k}\left(2(\boldsymbol{x}^\top\boldsymbol{y})^2\right)^k\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2\right)^{p-k} - \frac{1}{2D}\sum_{k=0}^{p-1}\binom{p}{k}(\boldsymbol{x}^\top\boldsymbol{y})^{2k}\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2\right)^{p-k} \\
&= \frac{1}{2D}\sum_{k=0}^{p-1}\binom{p}{k}(2^k-1)(\boldsymbol{x}^\top\boldsymbol{y})^{2k}\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2\right)^{p-k} \geq 0
\end{aligned}
$$

■

Thus, the Gaussian CtR-estimator is always better regardless of the choice of $\boldsymbol{x}, \boldsymbol{y}$ and $p$ and despite using only half the random feature samples. Note that the variance difference is zero if $p = 1$ and increases as $p$ increases. Moreover, the difference is maximized for parallel $\boldsymbol{x}$ and $\boldsymbol{y}$. In this case, we have $(\boldsymbol{x}^\top\boldsymbol{y}) = \|\boldsymbol{x}\|\,\|\boldsymbol{y}\|$ and the difference becomes

$$
\begin{aligned}
\mathbb{V}[\hat{k}_{\mathrm{R}}(\boldsymbol{x},\boldsymbol{y})] - \mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] &= \frac{1}{2D}\sum_{k=0}^{p-1}\binom{p}{k}(2^k-1)\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2\right)^p \\
&= \frac{1}{2D}\|\boldsymbol{x}\|^{2p}\|\boldsymbol{y}\|^{2p}\left(3^p - 2^{p+1} + 1\right)
\end{aligned}
$$

We analyze the more difficult Rademacher case next.

**Rademacher case: Proof of Theorem (4.2.3).**

*Proof.* Taking into account the different number of random features for $\Phi_{\mathrm{R}}$ and $\Phi_{\mathrm{CtR}}$ to have equal dimensions, the variance difference of their kernel estimates yields:

$$
\begin{aligned}
&\mathbb{V}[\hat{k}_{\mathrm{R}}(\boldsymbol{x},\boldsymbol{y})] - \mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] \\
&= \frac{1}{2D}\left(\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + 2\sum_{i=1}^{d}\sum_{j\neq i}x_i x_j y_i y_j\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right) \\
&\quad - \frac{1}{2D}\left\{\left(\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2 + \sum_{i=1}^{d}\sum_{j\neq i}x_i x_j y_i y_j\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p} + \left(2(\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{i=1}^{d}x_i^2 y_i^2\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right\}
\end{aligned}
$$

Next, we write $(\boldsymbol{x}^\top \boldsymbol{y})^{2p} = ((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2 + \sum_{i=1}^d x_i^2 y_i^2)^p$. In this way, we can factor out the term $a := (\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2 = \sum_{i=1}^d \sum_{j \neq i} x_i x_j y_i y_j$ and apply the binomial theorem to *all* addends. This gives:

$$\mathbb{V}[\hat{k}_{\mathrm{R}}(\boldsymbol{x}, \boldsymbol{y})] - \mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x}, \boldsymbol{y})] = \frac{1}{2D} \sum_{k=0}^{p} \binom{p}{k} a^{p-k}$$
$$\left( \left( \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2 \right)^k - \left( (\|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2)^k + (\boldsymbol{x}^\top \boldsymbol{y})^{2k} - (\sum_{i=1}^d x_i^2 y_i^2)^k \right) \right)$$

We now show that the following term is always non-negative:

$$B := \left( \left( \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2 \right)^k - \left( (\|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2)^k + (\boldsymbol{x}^\top \boldsymbol{y})^{2k} - (\sum_{i=1}^d x_i^2 y_i^2)^k \right) \right) \tag{C.6}$$

For $k = 0$ and $k = 1$, $B = 0$. For $k \geq 2$, we have:

$$\left( \|\boldsymbol{x}\|^2 \|\boldsymbol{y}\|^2 + (\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2 \right)^k = \sum_{j=0}^{k} \binom{k}{j} \|\boldsymbol{x}\|^{2j} \|\boldsymbol{y}\|^{2j} \left( (\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2 \right)^{k-j}$$

Plugging this expression into $B$ and cancelling out the addend for $j = k$ yields:

$$B = \sum_{j=0}^{k-1} \binom{k}{j} \|\boldsymbol{x}\|^{2j} \|\boldsymbol{y}\|^{2j} \left( (\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2 \right)^{k-j} - \left( (\boldsymbol{x}^\top \boldsymbol{y})^{2k} - (\sum_{i=1}^d x_i^2 y_i^2)^k \right)$$

Next, we refactor $(\boldsymbol{x}^\top \boldsymbol{y})^{2k} - (\sum_{i=1}^d x_i^2 y_i^2)^k$:

$$\begin{aligned}
(\boldsymbol{x}^\top \boldsymbol{y})^{2k} - (\sum_{i=1}^d x_i^2 y_i^2)^k &= ((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2 + \sum_{i=1}^d x_i^2 y_i^2)^k - (\sum_{i=1}^d x_i^2 y_i^2)^k \\
&= \sum_{j=0}^{k} \binom{k}{j} (\sum_{i=1}^d x_i^2 y_i^2)^j ((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2)^{k-j} - (\sum_{i=1}^d x_i^2 y_i^2)^k \\
&= \sum_{j=0}^{k-1} \binom{k}{j} (\sum_{i=1}^d x_i^2 y_i^2)^j ((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2)^{k-j}
\end{aligned}$$

Plugging this expression into $B$ yields:

$$B = \sum_{j=0}^{k-1} \binom{k}{j} \left( \|\boldsymbol{x}\|^{2j} \|\boldsymbol{y}\|^{2j} - (\sum_{i=1}^d x_i^2 y_i^2)^j \right) ((\boldsymbol{x}^\top \boldsymbol{y})^2 - \sum_{i=1}^d x_i^2 y_i^2)^{k-j}$$

Finally, we insert $B$ back into the original variance difference $\mathbb{V}[\hat{k}_{\mathrm{R}}(\boldsymbol{x},\boldsymbol{y})]-\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})]$ (remember that $B=0$ if $k<2$):

$$\begin{aligned}\mathbb{V}[\hat{k}_{\mathrm{R}}(\boldsymbol{x},\boldsymbol{y})]-\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})] &= \frac{1}{2D}\sum_{k=2}^{p}\binom{p}{k}a^{p-k}B \\ &= \frac{1}{2D}\sum_{k=2}^{p}\sum_{j=0}^{k-1}\binom{p}{k}\binom{k}{j}a^{p-j}\left(\|\boldsymbol{x}\|^{2j}\|\boldsymbol{y}\|^{2j}-(\sum_{i=1}^{d}x_i^2y_i^2)^j\right)\end{aligned}$$

Finally, we note that $b := \|\boldsymbol{x}\|^{2j}\|\boldsymbol{y}\|^{2j}-(\sum_{i=1}^{d}x_i^2y_i^2)^j = (\sum_{i=1}^{d}\sum_{\ell=1}^{d}x_i^2y_\ell^2)^j - (\sum_{i=1}^{d}x_i^2y_i^2)^j \geq 0$ and $\mathbb{V}[\hat{k}_{\mathrm{R}}(\boldsymbol{x},\boldsymbol{y})]-\mathbb{V}[\hat{k}_{\mathrm{CtR}}(\boldsymbol{x},\boldsymbol{y})]\geq 0$ if $a=\sum_{i=1}^{d}\sum_{j\neq i}x_ix_jy_iy_j^2 \geq 0$. ∎

## C.3 Pseudo-Variance of TensorSRHT

In the following, we work out the pseudo-variance of TensorSRHT. We focus on upsampled TensorSRHT here. The pseudo-variance of stacked TensorSRHT can be derived analogously by applying the same modifications as the ones used in Section B.2.2.

For the pseudo-variance $\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})] = \mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2]-\mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})]^2$, we need to work out $\mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2]$:

$$\begin{aligned}\mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] &= \frac{1}{D^2}\sum_{\ell=1}^{D}\sum_{\ell'=1}^{D}\prod_{i=1}^{p}\mathbb{E}\left[(\boldsymbol{w}_{i,\ell}^\top\boldsymbol{x})(\overline{\boldsymbol{w}_{i,\ell}^\top\boldsymbol{y}})(\boldsymbol{w}_{i,\ell'}^\top\boldsymbol{x})(\overline{\boldsymbol{w}_{i,\ell'}^\top\boldsymbol{y}})\right] \\ &= \frac{1}{D^2}\sum_{\ell=1}^{D}\sum_{\ell'=1}^{D}\underbrace{\mathbb{E}\left[(\boldsymbol{w}_{\ell}^\top\boldsymbol{x})(\overline{\boldsymbol{w}_{\ell}^\top\boldsymbol{y}})(\boldsymbol{w}_{\ell'}^\top\boldsymbol{x})(\overline{\boldsymbol{w}_{\ell'}^\top\boldsymbol{y}})\right]^p}_{e(\ell,\ell')^p}\end{aligned}$$

We dropped the index $i$ in the last equality for ease of notation, as all $\{\boldsymbol{w}_{i,\ell}\}_{i=1}^{p}$ are i.i.d. samples and the expectation is thus the same for any $i$. To work out the expectation $e(\ell,\ell')$, we need to distinguish different cases for $\ell$ and $\ell'$.

1. $\ell=\ell'$ ($D$ terms): $e(\ell,\ell')^p = \mathbb{E}\left[(\boldsymbol{w}_\ell^\top\boldsymbol{x})^2(\overline{\boldsymbol{w}_\ell^\top\boldsymbol{y}})^2\right]^p = \left(2(\boldsymbol{x}^\top\boldsymbol{y}^2)-\sum_{i=1}^{d}x_i^2y_i^2\right)^p$ (taken from Eq. (C.3) for the Rademacher case)
2. $\ell\neq\ell'$ ($D(D-1)$ terms):

$$\begin{aligned}e(\ell,\ell')^p &= \left(\sum_{q=1}^{d}\sum_{r=1}^{d}\sum_{s=1}^{d}\sum_{t=1}^{d}\mathbb{E}[w_{\ell,q}\overline{w_{\ell,r}}w_{\ell',s}\overline{w_{\ell',t}}]x_qy_rx_sy_t\right)^p \\ &= \left(\sum_{q=1}^{d}\sum_{r=1}^{d}\sum_{s=1}^{d}\sum_{t=1}^{d}\mathbb{E}[d_q\overline{d_r}d_s\overline{d_t}]\mathbb{E}[h_{p_\ell,q}h_{p_\ell,r}h_{p_{\ell'},s}h_{p_{\ell'},t}]x_qy_rx_sy_t\right)^p\end{aligned}$$

$d_q, d_r, d_s, d_t$ are uniform samples from $\{1, -1, \mathrm{i}, -\mathrm{i}\}$, i.e., complex Rademacher samples, that are independent from the index samples $p_{\ell,q}, p_{\ell,r}, p_{\ell',s}, p_{\ell',t}$, which is why we can factor out the two expectations. We will simplify the above sum by studying when $\mathbb{E}[d_q\overline{d_r}d_s\overline{d_t}] \neq 0$.

We have to distinguish three non-zero cases for $\mathbb{E}[d_q\overline{d_r}d_s\overline{d_t}]$:

1. $q = r = s = t$ ($d$ terms): $\mathbb{E}[d_q\overline{d_r}d_s\overline{d_t}] = \mathbb{E}[|d_q|^4] = 1$
2. $q = r \neq s = t$ ($d(d-1)$ terms): $\mathbb{E}[d_q\overline{d_r}d_s\overline{d_t}] = \mathbb{E}[|d_q|^2]\mathbb{E}[|d_s|^2] = 1$
3. $q = t \neq r = s$ ($d(d-1)$ terms): $\mathbb{E}[d_q\overline{d_r}d_s\overline{d_t}] = \mathbb{E}[|d_q|^2]\mathbb{E}[|d_r|^2] = 1$

because $\mathbb{E}[|d_q|^4] = \mathbb{E}[|d_q|^2] = 1$.

In the second part of Lemma B.2.1, we show that for $\ell \neq \ell'$ and $q \neq r$, $\mathbb{E}[h_{p_\ell,q}h_{p_\ell,r}h_{p_{\ell'},r}h_{p_{\ell'},q}] = -\frac{1}{\lceil D/d\rceil d-1}$ holds. Therefore, $e(\ell,\ell')^p$ for $\ell \neq \ell'$ yields:

$$
\begin{aligned}
e(\ell,\ell')^p &= \left(\sum_{i=1}^{d} x_i^2 y_i^2 + \sum_{i=1}^{d}\sum_{j\neq i}^{d} x_i y_i x_j y_j - \frac{1}{\lceil D/d\rceil d-1}\sum_{i=1}^{d}\sum_{j\neq i}^{d} x_i y_i x_j y_j\right)^p \\
&= \left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{1}{\lceil D/d\rceil d-1}\left[(\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{i=1}^{d} x_i^2 y_i^2\right]\right)^p
\end{aligned}
$$

In fact, $e(\ell,\ell')^p$ does not depend on $\ell$ and $\ell'$ anymore after working out the expectations involved. Plugging $e(\ell,\ell')^p$ back into $\mathbb{E}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2]$ yields the following pseudo-variance for TensorSRHT:

$$
\begin{aligned}
\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] &= \frac{1}{D}\left[\left(2(\boldsymbol{x}^\top\boldsymbol{y}^2) - \sum_{i=1}^{d} x_i^2 y_i^2\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right] \\
&\quad + \left(1-\frac{1}{D}\right)\left[\left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{1}{\lceil D/d\rceil d-1}\left[(\boldsymbol{x}^\top\boldsymbol{y})^2 - \sum_{i=1}^{d} x_i^2 y_i^2\right]\right)^p - (\boldsymbol{x}^\top\boldsymbol{y})^{2p}\right] \\
&= \frac{1}{D}\mathbb{PV}_{\text{Rad.}}^{(p)} - \left(1-\frac{1}{D}\right)\left[(\boldsymbol{x}^\top\boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{\mathbb{PV}_{\text{Rad.}}^{(1)}}{\lceil D/d\rceil d-1}\right)^p\right]
\end{aligned}
\tag{C.7}
$$

$\mathbb{PV}_{\text{Rad.}}^{(p)}$ and $\mathbb{PV}_{\text{Rad.}}^{(1)}$ are the Rademacher pseudo-variance (C.5) for a given degree $p$ and $p = 1$, respectively.

For stacked TensorSRHT, the pseudo-variance becomes

$$
\mathbb{PV}[\hat{k}_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})^2] = \frac{1}{D}\mathbb{PV}_{\text{Rad.}}^{(p)} - \left(\frac{c(D,d)}{D^2}\right)\left[(\boldsymbol{x}^\top\boldsymbol{y})^{2p} - \left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{\mathbb{PV}_{\text{Rad.}}^{(1)}}{d-1}\right)^p\right].
$$

with $c(D,d) = \lfloor D/d\rfloor d(d-1) + \mathrm{mod}(D,d)(\mathrm{mod}(D,d)-1)$. It can be derived using the same changes to the above derivation as presented in Section B.2.2.

# Appendix D

# Appendix of Chapter 5

## D.1 Convex Surrogate Functions for Stacked TensorSRHT Variances

To extend the applicability of the Incremental Algorithm in Algorithm 4 to TensorSRHT, we derive here convex surrogate functions for the variances of *stacked* TensorSRHT. To this end, we first analyze the variances of stacked TensorSRHT in Appendix D.1.1. We then derive convex surrogate functions in Appendix D.1.2.

### D.1.1 Analyzing the Variances of stacked TensorSRHT

We first derive another form of the variance of stacked TensorSRHT given in Eq. (3.35) of Theorem 3.5.9, which we will use in a later analysis. Let $\Phi_n : \mathbb{R}^d \to \mathbb{C}^D$ be a complex TensorSRHT sketch of degree $n \in \mathbb{N}$ satisfying the assumptions in Theorem 3.5.9 with $0 \leq q \leq 1$. For $q = 1$ we recover the real TensorSRHT and for $q = 1/2$ the complex one.

As shown in Appendix B.2.2, the approximate kernel of the complex TensorSRHT can be written as

$$\hat{k}(\boldsymbol{x}, \boldsymbol{y}) := \Phi_n(\boldsymbol{x})^\top \overline{\Phi_n(\boldsymbol{y})} = \frac{1}{D} \sum_{b=1}^{B} \sum_{\ell \in I_b} \prod_{i=1}^{n} \left(\boldsymbol{s}_{i,\ell}^{b\top} \boldsymbol{x}\right) \overline{\left(\boldsymbol{s}_{i,\ell}^{b\top} \boldsymbol{y}\right)},$$

where $B := \lceil D/d \rceil$, $I_b := \{1, \ldots, d\}$ for $b = 1, \ldots, B-1$ and $I_b := \{1, \ldots, \mathrm{mod}(D, d)\}$ for $b = B$, and $\boldsymbol{s}_{i,\ell}^b \in \mathbb{C}^d$ are the structured random weights defined in Eq. (B.5).

We can then write the variance of the approximate kernel as

$$\begin{aligned}\mathbb{V}[\hat{k}(\boldsymbol{x},\boldsymbol{y})] =& \frac{1}{D^2}\sum_{b=1}^{B}\mathbb{V}\left[\sum_{\ell\in I_b}\prod_{i=1}^{n}\left(\boldsymbol{s}_{i,\ell}^{b\top}\boldsymbol{x}\right)\overline{\left(\boldsymbol{s}_{i,\ell}^{b\top}\boldsymbol{y}\right)}\right] \\ =& \frac{1}{D^2}\sum_{b=1}^{B}\sum_{\ell\in I_b}\underbrace{\mathbb{V}\left[\prod_{i=1}^{n}\left(\boldsymbol{s}_{i,\ell}^{b\top}\boldsymbol{x}\right)\overline{\left(\boldsymbol{s}_{i,\ell}^{b\top}\boldsymbol{y}\right)}\right]}_{=\,V_q^{(n)}} \\ &+ \frac{1}{D^2}\sum_{b=1}^{B}\sum_{\substack{\ell,\ell'\in I_b,\\ \ell\neq\ell'}}\underbrace{\mathrm{Cov}\left(\prod_{i=1}^{n}\left(\boldsymbol{s}_{i,\ell}^{b\top}\boldsymbol{x}\right)\overline{\left(\boldsymbol{s}_{i,\ell}^{b\top}\boldsymbol{y}\right)},\prod_{i=1}^{n}\left(\boldsymbol{s}_{i,\ell'}^{b\top}\boldsymbol{x}\right)\overline{\left(\boldsymbol{s}_{i,\ell'}^{b\top}\boldsymbol{y}\right)}\right)}_{=:\,\mathrm{Cov}_q^{(n)}} \\ =& \frac{V_q^{(n)}}{D} + \frac{c(D,d)}{D^2}\mathrm{Cov}_q^{(n)} = \text{Eq. (3.35)}, \end{aligned} \tag{D.1}$$

where $c(D,d) = \lfloor D/d\rfloor d(d-1)+\mathrm{mod}(D,d)(\mathrm{mod}(D,d)-1)$ and the last line follows from that the values of $V_q^{(n)}$ and $\mathrm{Cov}_q^{(n)}$ do not depend on the choice of $\ell,\ell'$ and $b$ (which can be shown from the arguments in Appendix B.2.2). Here, $V_q^{(n)}$ is the variance of the unstructured Rademacher sketch with a single feature in Eq. (3.17) with $p=n$, and $\mathrm{Cov}_q^{(n)}$ is the covariance for distinct indices $\ell,\ell'$ inside each block $b$. By comparing Eq. (3.35) and Eq. (D.1), the concrete form of $\mathrm{Cov}_q^{(n)}$ is given by

$$\mathrm{Cov}_q^{(n)} = -\left[(\boldsymbol{x}^\top\boldsymbol{y})^{2n} - \left((\boldsymbol{x}^\top\boldsymbol{y})^2 - \frac{V_q^{(1)}}{d-1}\right)^n\right]$$

Eq. (D.1) is a useful representation of the variance of TensorSRHT in Eq. (3.35) for studying its (non-)convexity with respect to $D$. The following result shows a range of values of $D$ for which Eq. (3.35) is convex.

**Theorem D.1.1.** *The variance of the TensorSRHT sketch in Eq. (3.35) is convex and monotonically decreasing with respect to $D \in \{1,\dots,d\}$ and with respect to $D \in \{kd \mid k\in\mathbb{N}\}$.*

*Proof.* If $D\in\{1,\dots,d\}$, we have $c(D,d) = D(D-1)$ in Eq. (D.1). Therefore, Eq. (D.1) is equal to

$$\frac{1}{D}V_q^{(n)} + \left(1-\frac{1}{D}\right)\mathrm{Cov}_q^{(n)} = \frac{1}{D}\left(V_q^{(n)} - \mathrm{Cov}_q^{(n)}\right) + \mathrm{Cov}_q^{(n)}. \tag{D.2}$$

For two random variables $X,Y$ it generally holds that $|\mathrm{Cov}(X,Y)| \le \sqrt{\mathbb{V}[X]\mathbb{V}[Y]}$ by the Cauchy-Schwarz inequality. Hence, we have $|\mathrm{Cov}_q^{(n)}| \le V_q^{(n)}$ and thus $V_q^{(n)} - \mathrm{Cov}_q^{(n)} \ge 0$. Therefore, Eq. (D.2) is proportional to $1/D$ with a non-negative coefficient, and thus it is convex and monotonically decreasing for $D\in\{1,\dots,d\}$.

Next, suppose $D = kd$ for some $k \in \mathbb{N}$, in which case we have $c(D, d) = kd(d-1)$ in Eq. (D.1). Therefore Eq. (D.1) is equal to

$$\frac{1}{kd}\left(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}\right) = \frac{1}{D}\left(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}\right). \tag{D.3}$$

The term in the parenthesis is non-negative, because (D.1) is the variance of TensorSRHT and thus non-negative. Therefore, (D.1) is convex and monotonically decreasing with respect to $D \in \{kd \mid k \in \mathbb{N}\}$.

■

As we do next, Theorem D.1.1 is useful for designing a convex surrogate function for Eq. (3.35), as it shows the range of $D$ on which Eq. (3.35) is already convex and does not need to be modified.

### D.1.2 Convex Surrogate Functions

Based on Eq. (3.35), we now propose a convex surrogate function for the variance of TensorSRHT in Eq. (3.35). We consider the following two cases separately: i) $\mathrm{Cov}_q^{(n)} \leq 0$ and ii) $\mathrm{Cov}_q^{(n)} > 0$. For each case, we propose a convex surrogate function.

**i) Case $\mathrm{Cov}_q^{(n)} \leq 0$.** We define a surrogate function of Eq. (3.35) by concatenating the two expressions of Eq. (D.1) for $D \in \{1, \ldots, d\}$ and $D \in \{kd \mid k \in \mathbb{N}\}$ given in Eq. (D.2) and Eq. (D.3), respectively, and extend their ranges to the entire domain $D \in \mathbb{N}$:

$$V_{\mathrm{Surr.}}^{(n)}(D) := \begin{cases} \frac{1}{D}\left(V_q^{(n)} - \mathrm{Cov}_q^{(n)}\right) + \mathrm{Cov}_q^{(n)} & \text{if } D \leq d \\ \frac{1}{D}\left(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}\right) & \text{if } D > d. \end{cases} \tag{D.4}$$

**ii) Case $\mathrm{Cov}_q^{(n)} > 0$.** We use the expression (D.3) to define a surrogate function on $D \in \mathbb{N}$:

$$V_{\mathrm{Surr.}}^{(n)}(D) := \frac{1}{D}\left(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}\right) \tag{D.5}$$

The convexity of Eq. (D.5) immediately follows from $V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)} \geq 0$, which holds as we show in the proof of Theorem D.1.1. Note that $\mathrm{Cov}_q^{(n)} > 0$ can only occur when $n$ is even, as shown in Corollary 3.5.10 of Section 3.5.

We defined the surrogate function in Eq. (D.4) by interpolating the variances of TensorSRHT in Eq. (3.35) for $D \in \{1, \ldots, d\}$ and $D \in \{kd \mid k \in \mathbb{N}\}$ and extending the domain to $\mathbb{N}$. In fact, for $D \in \{1, \ldots, d\}$ and $D \in \{kd \mid k \in \mathbb{N}\}$, Eq. (D.4) is equal to Eq. (3.35), as shown in the proof of Theorem D.1.1. Fig. D.1 illustrates

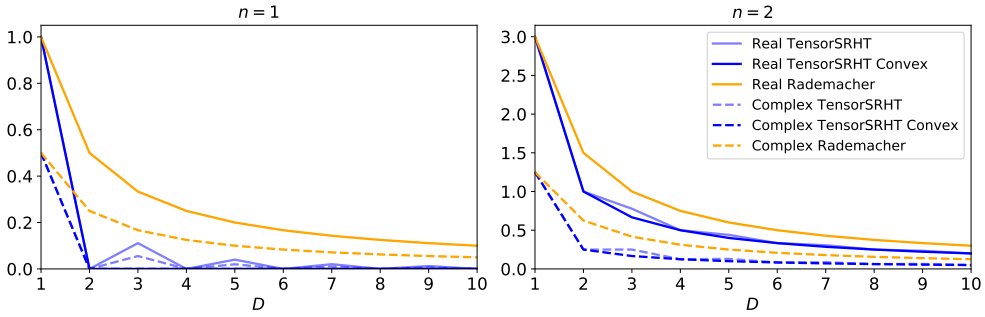



Figure D.1: Convex surrogate functions in Eq. (D.4) and the variances of TensorSRHT in (3.35) as a function of the number of random features $D$, with polynomial degrees $n = 1, 2$ and input vectors $\boldsymbol{x} = \boldsymbol{y} = [\sqrt{1/2}, \sqrt{1/2}]^\top$ $(d = 2)$. For a comparison, we also plot the variances of the real Rademacher sketch in Eq. (3.7) and the complex Rademacher sketch in Eq. (3.18).

the convex surrogate function in Eq. (D.4) and the variance of TensorSRHT in (3.35) when $\mathrm{Cov}_q^{(n)} \leq 0$ holds.

Note that, as mentioned later in Remark D.1.3, the surrogate function in Eq. (D.4) may not be convex over $D \in \mathbb{N}$ if the condition $\mathrm{Cov}_q^{(n)} \leq 0$ does not hold. This is why we defined another convex surrogate function as in Eq. (D.5) for the case $\mathrm{Cov}_q^{(n)} > 0$.

The following theorem shows that the surrogate function in Eq. (D.4) is convex in the considered case of i) $\mathrm{Cov}_q^{(n)} \leq 0$.

**Theorem D.1.2.** *If* $\mathrm{Cov}_q^{(n)} \leq 0$, *Eq.* (D.4) *is convex with respect to* $D \in \mathbb{N}$.

*Proof.* As shown in Theorem D.1.1, $V_{\mathrm{Surr.}}^{(n)}(D) = \frac{1}{D}(V_q^{(n)} - \mathrm{Cov}_q^{(n)}) + \mathrm{Cov}_q^{(n)}$ is convex over $D \in \{1, \ldots, d\}$. Likewise, $V_{\mathrm{Surr.}}^{(n)}(D) = \frac{1}{D}(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)})$ is convex over $D \in [d, \infty) \cap \mathbb{N}$, since $V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)} \geq 0$ holds as we show in the proof of Theorem D.1.1.

Therefore, the proof completes by showing that the concatenated function $V_{\mathrm{Surr.}}^{(n)}(D)$ in Eq. (D.4) is also convex over $D \in \{d-1, d, d+1\}$, i.e.,

$$\frac{1}{2}\left(V_{\mathrm{Surr.}}^{(n)}(d-1) + V_{\mathrm{Surr.}}^{(n)}(d+1)\right) \geq V_{\mathrm{Surr.}}^{(n)}(d). \tag{D.6}$$

By using the definition in Eq. (D.4), this inequality is equivalent to

$$\begin{aligned}&\frac{1}{2}\left(\frac{1}{d-1}\left(V_q^{(n)} + (d-2)\mathrm{Cov}_q^{(n)}\right) + \frac{1}{d+1}\left(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}\right)\right)\\ &\geq \frac{1}{d}\left(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}\right).\end{aligned} \tag{D.7}$$

Note that we have $V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)} \geq 0$ , as mentioned earlier. If $V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)} = 0$ holds, then we have $V_{\mathrm{Surr.}}^{(n)}(D) = 0$ for $D \geq d$ by the definition in

Eq. (D.4), and thus Eq. (D.6) holds (which concludes the proofs). Therefore, we assume that the inequality to be strict, i.e., $V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)} > 0$. Dividing the both sides of Eq. (D.7) by $(V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)})$, we obtain

$$\frac{1}{2}\left(\frac{1}{d-1}\frac{V_q^{(n)} + (d-2)\mathrm{Cov}_q^{(n)}}{V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}} + \frac{1}{d+1}\right) \geq \frac{1}{d},$$

which after some rearrangement gives

$$\frac{V_q^{(n)} + (d-2)\mathrm{Cov}_q^{(n)}}{V_q^{(n)} + (d-1)\mathrm{Cov}_q^{(n)}} \geq 1 - \frac{2}{d^2+d}. \tag{D.8}$$

This inequality holds because we have $(d-2)\mathrm{Cov}_q^{(n)} \geq (d-1)\mathrm{Cov}_q^{(n)}$, which follows from our assumption $\mathrm{Cov}_q^{(n)} \leq 0$. Therefore Eq. (D.7) holds.

■

**Remark D.1.3.** *Theorem D.1.2 shows the convexity of the surrogate function in Eq. (D.4), assuming* $\mathrm{Cov}_q^{(n)} \leq 0$. *If this condition does not hold, i.e., if* $\mathrm{Cov}_q^{(n)} > 0$, *then the surrogate function in Eq. (D.4) may not be convex. To see this, let* $d = 2, \boldsymbol{x} = (a, 0)^\top$ *with* $a > 0$, $\boldsymbol{y} = (0, b)^\top$ *with* $b > 0$, *and* $n$ *be even; then we have* $V_q^{(n)} = \mathrm{Cov}_q^{(n)} = a^{2n}b^{2n} > 0$, *and the inequality in Eq. (D.8) in the proof of Theorem D.1.2 does not hold, which implies that the surrogate function in Eq. (D.4) is not convex.*

As mentioned in Section 3.5, the variance of TensorSRHT in Eq. (3.35) becomes zero if $n = 1$ and $D \in \{kd \mid k \in \mathbb{N}\}$, i.e., $\mathbb{V}[\Phi_1(\boldsymbol{x})^\top \overline{\Phi_1(\boldsymbol{y})}] = 0$ holds. Therefore, because the convex surrogate functions in Eq. (D.4) and Eq. (D.5) are equal to the variance of TensorSRHT in Eq. (3.35) for $D \in \{kd \mid k \in \mathbb{N}\}$, these surrogate functions also become zero for $n = 1$ and $D \in \{kd \mid k \in \mathbb{N}\}$. Thus, the Incremental Algorithm (Algorithm 4), when used with the surrogate functions in Eq. (D.4) and Eq. (D.5), will not assign more than $D = d$ random features to the polynomial degree $n = 1$. Note that assigning $D = d$ random features is equivalent to appending the input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ to the approximate kernel (5.10), which is called *H0/1 heuristic* in Kar and Karnick (2012). Therefore, the Incremental Algorithm with the surrogate functions in Eq. (D.4) and Eq. (D.5) automatically achieve the H0/1 heuristic.

Finally, we describe briefly how to use the convex surrogate functions in Eq. (D.4) and Eq. (D.5) in the Incremental Algorithm in Algorithm 4. To this end, we rewrite Eq. (5.20) using the surrogate functions as follows: (Here, we make the dependence of $V_q^{(n)}$ and $\mathrm{Cov}_q^{(n)}$ on the input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ explicit and write

them as $V_q^{(n)}(\boldsymbol{x}, \boldsymbol{y})$ and $\mathrm{Cov}_q^{(n)}(\boldsymbol{x}, \boldsymbol{y})$, respectively.)

$$\text{Eq. (5.20)} = \begin{cases} \frac{a_n^2}{D_n}\left(\sum_{i\neq j} V_q^{(n)}(\boldsymbol{x}_i, \boldsymbol{x}_j) + (d-1)\sum_{i\neq j}\mathrm{Cov}_q^{(n)}(\boldsymbol{x}_i, \boldsymbol{x}_j)\right) \\ \qquad \text{if } \sum_{i\neq j}\mathrm{Cov}_q^{(n)}(\boldsymbol{x}_i, \boldsymbol{x}_j) > 0 \text{ or } D_n > d, \\ \frac{a_n^2}{D_n}\left(\sum_{i\neq j} V_q^{(n)}(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i\neq j}\mathrm{Cov}_q^{(n)}(\boldsymbol{x}_i, \boldsymbol{x}_j)\right) + a_n^2 \sum_{i\neq j}\mathrm{Cov}_q^{(n)}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \qquad \text{otherwise.} \end{cases}$$

After precomputing the constants $\sum_{i\neq j} V_q^{(n)}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\sum_{i\neq j}\mathrm{Cov}_q^{(n)}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for each $n \in \{1, \ldots, p\}$, which can be done in $\mathcal{O}(m^2)$ time, one can directly use the above modification of Eq. (5.20) in the objective function in Eq. (5.19). In this way, we adapt the objective function in (5.19) to be convex, so that the Incremental Algorithm in Algorithm 4 is directly applicable.

## D.2 Additional Experiments

We present here additional experimental results, supplementing those in Chapter 5. Table D.1 shows the effects of applying zero-centering to input vectors in the polynomial kernel approximation experiments. Fig. D.2 and Fig. D.3 show the results of additional experiments on GP regression. Fig. D.4 and Fig. D.5 show the results of additional experiments on GP classification.

| | MNLL | | | | Rel. Frob. Error | | | |
|---|---|---|---|---|---|---|---|---|
| | Non-centred | | Centred | | Non-centred | | Centred | |
| Dataset | SRF Gaus. | Opt. Macl. Rad. | SRF Gaus. | Opt. Macl. Rad. | SRF Gaus. | Opt. Macl. Rad. | SRF Gaus. | Opt. Macl. Rad. |
| Boston | 3.410±0.37 | 3.447±0.38 | 3.449±0.62 | **3.161**±0.28 | **0.044**±0.02 | 0.212±0.15 | 0.356±0.05 | 0.421±0.06 |
| Concrete | 3.779±0.07 | 3.811±0.04 | 3.660±0.12 | **3.542**±0.07 | **0.019**±0.01 | 0.276±0.17 | 0.610±0.07 | 0.482±0.03 |
| Energy | 6.090±0.12 | 6.090±0.12 | 5.116±0.20 | **5.012**±0.13 | **0.003**±0.00 | 0.222±0.14 | 0.507±0.08 | 0.484±0.05 |
| kin8nm | -0.203±0.07 | -0.310±0.03 | -0.203±0.07 | **-0.323**±0.03 | 0.946±0.04 | 0.525±0.04 | 0.947±0.03 | **0.521**±0.03 |
| Naval | -6.069±0.03 | -6.066±0.03 | **-8.083**±0.04 | -7.788±0.10 | **0.040**±0.03 | 0.183±0.06 | 0.112±0.04 | 0.384±0.11 |
| Powerplant | 3.064±0.03 | **3.061**±0.06 | 3.282±0.14 | 3.400±0.73 | **0.001**±0.00 | 0.062±0.04 | 0.609±0.09 | 0.527±0.10 |
| Protein | 3.233±0.01 | 3.233±0.01 | 3.072±0.02 | **3.060**±0.02 | **0.000**±0.00 | 0.002±0.00 | 0.277±0.05 | 0.429±0.14 |
| Yacht | 4.317±0.45 | 4.478±0.45 | **3.773**±0.21 | 3.844±0.28 | **0.028**±0.01 | 0.276±0.11 | 0.512±0.04 | 0.484±0.03 |
| Cod_rna | 0.307±0.00 | 0.308±0.00 | 0.288±0.06 | **0.151**±0.01 | **0.022**±0.01 | 0.087±0.05 | 0.641±0.05 | 0.467±0.05 |
| Covertype | 0.821±0.01 | - | 0.650±0.01 | **0.639**±0.01 | **0.024**±0.01 | - | 0.361±0.01 | 0.300±0.01 |
| Drive | 1.446±0.02 | 1.453±0.03 | 0.677±0.02 | **0.497**±0.01 | **0.068**±0.02 | 0.135±0.05 | 0.348±0.01 | 0.312±0.02 |
| FashionMNIST | **0.353**±0.00 | 0.364±0.00 | 0.364±0.00 | 0.361±0.00 | **0.029**±0.00 | 0.062±0.01 | 0.099±0.00 | 0.104±0.01 |
| Magic | 0.453±0.01 | 0.452±0.01 | 0.381±0.02 | **0.350**±0.01 | **0.068**±0.01 | 0.147±0.05 | 0.430±0.03 | 0.418±0.04 |
| Miniboo | 0.253±0.01 | - | 0.239±0.01 | **0.213**±0.01 | **0.027**±0.01 | - | 0.214±0.01 | 0.229±0.02 |
| MNIST | 0.076±0.00 | **0.074**±0.00 | 0.290±0.02 | 0.353±0.09 | **0.073**±0.00 | 0.082±0.00 | 0.085±0.00 | 0.089±0.01 |
| Mocap | 0.360±0.01 | 0.334±0.01 | 0.357±0.02 | **0.289**±0.01 | **0.115**±0.01 | 0.187±0.04 | 0.414±0.01 | 0.290±0.02 |

Table D.1: GP regression (top) and classification (bottom) for centred vs. non-centred data with $D = 5d$ features. Non-centred Miniboo and Covertype led to numerical issues for Maclaurin (no scores reported).

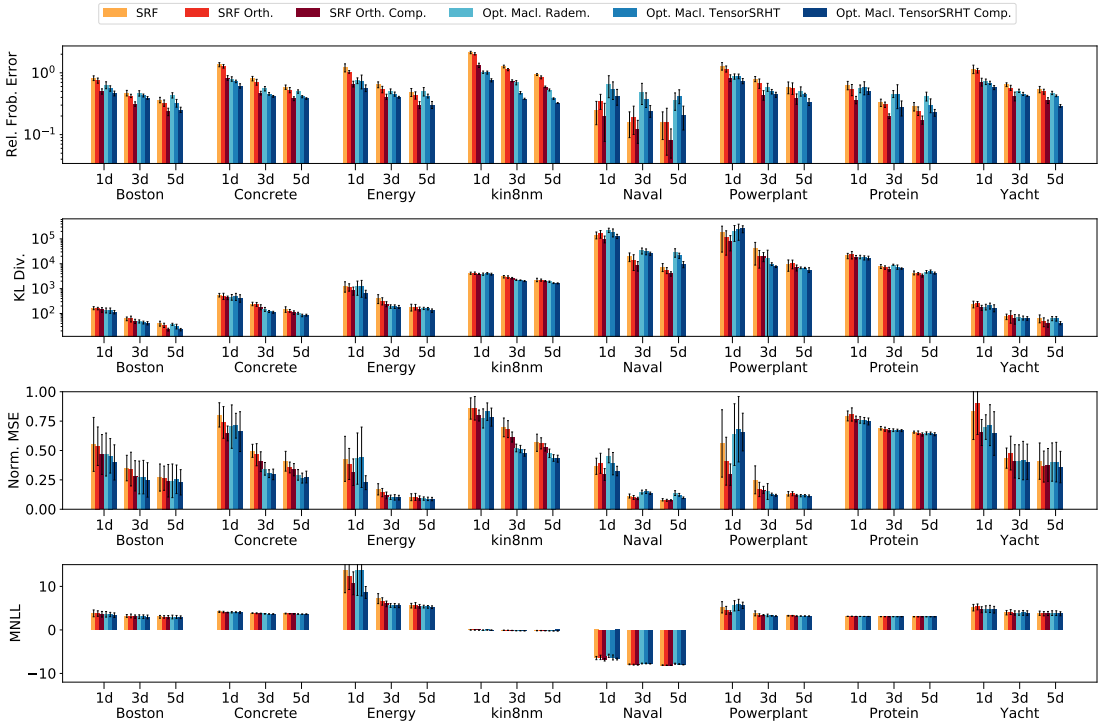



Figure D.2: Additional results of the experiments in Section 5.4.3 on approximate GP regression with a high-degree polynomial kernel. Lower values are better for all the metrics. For each dataset, we show the number of random features $D \in \{1d, 3d, 5d\}$ used in each method on the horizontal axis, with $d$ being the input dimensionality of the dataset. We put the legend labels and the bars in the same order.

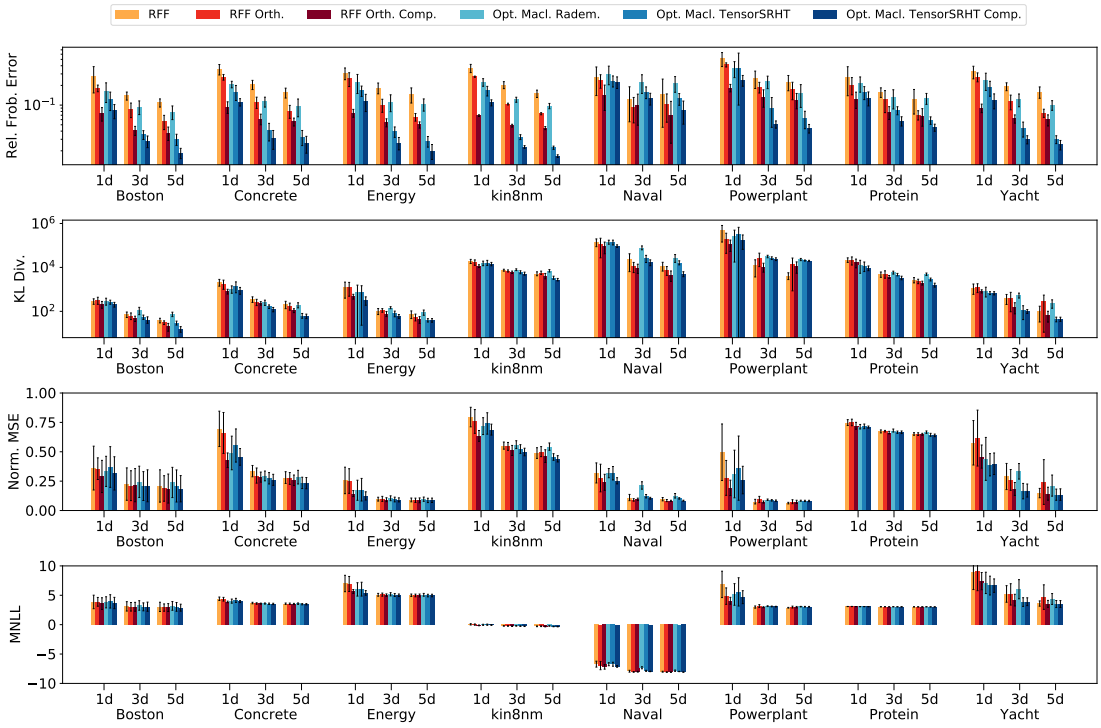



Figure D.3: Additional results of the experiments in Section 5.4.3 on approximate GP regression with a Gaussian kernel. Lower values are better for all the metrics. For each dataset, we show the number of random features $D \in \{1d, 3d, 5d\}$ used in each method on the horizontal axis, with $d$ being the input dimensionality of the dataset. We put the legend labels and the bars in the same order.

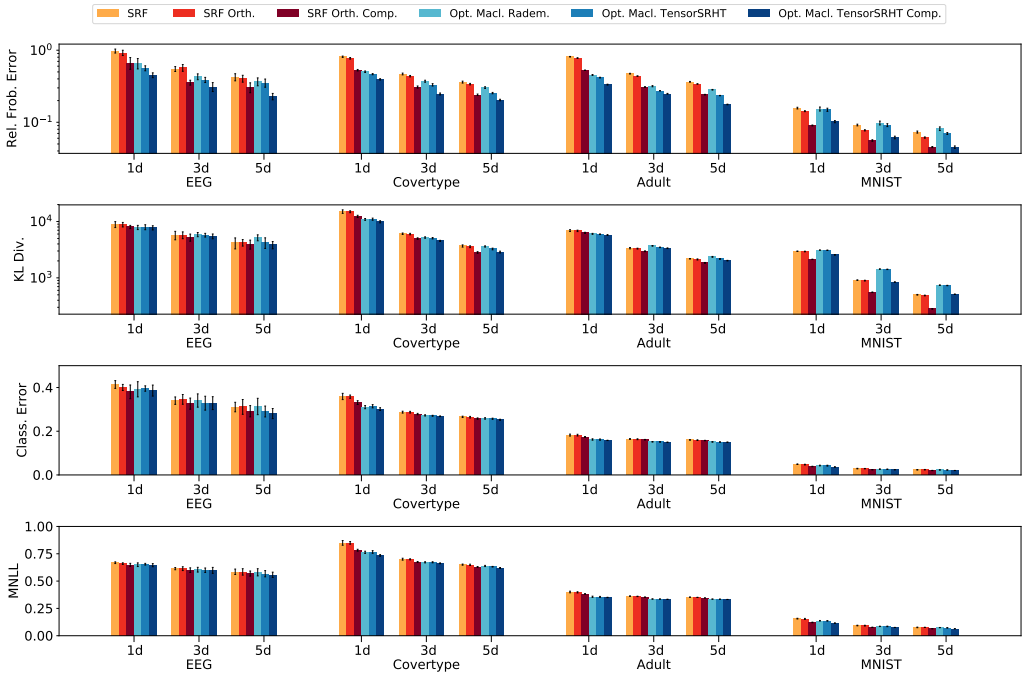



Figure D.4: Additional results of the experiments in Section 5.4.3 on approximate GP classification with a high-degree polynomial kernel. Lower values are better for all the metrics. For each dataset, we show the number of random features $D \in \{1d, 3d, 5d\}$ used in each method on the horizontal axis, with $d$ being the input dimensionality of the dataset. We put the legend labels and the bars in the same order.

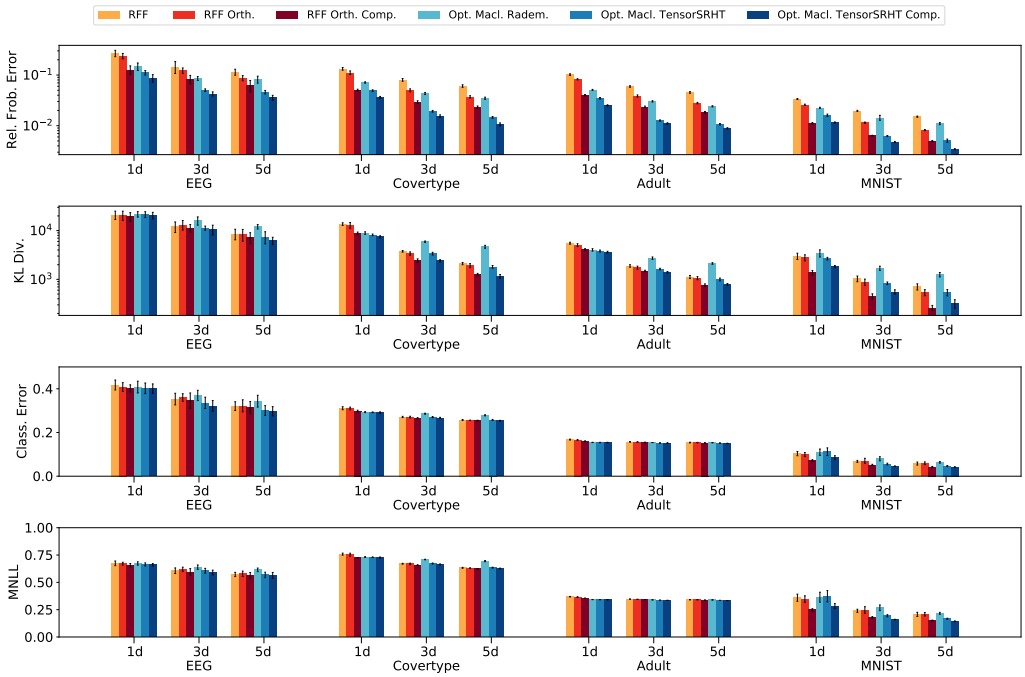



Figure D.5: Additional results of the experiments in Section 5.4.3 on approximate GP classification with a Gaussian kernel. Lower values are better for all the metrics. For each dataset, we show the number of random features $D \in \{1d, 3d, 5d\}$ used in each method on the horizontal axis, with $d$ being the input dimensionality of the dataset. We put the legend labels and the bars in the same order.

# Appendix E

# Appendix of Chapter 6

## E.1 Proof of Theorem 6.2.2

We now derive the limiting kernel when the optical random feature map is given by $\Phi_m(\boldsymbol{x}) = \frac{1}{\sqrt{D}}|\boldsymbol{U}\boldsymbol{x}|^m$ for some $m = 2s$ and $s \in \mathbb{N}$. Thus, we work out $k_m(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}[\Phi_m(\boldsymbol{x})^\top \Phi_m(\boldsymbol{y})]$ and obtain:

$$\begin{aligned} k_m(\boldsymbol{x}, \boldsymbol{y}) &= \int |\boldsymbol{x}^\top \boldsymbol{u}|^m |\boldsymbol{y}^\top \boldsymbol{u}|^m \; p(\boldsymbol{u}) d\boldsymbol{u} \\ &= \int \|\boldsymbol{x}\|^m \|\boldsymbol{y}\|^m |u_1|^m |u_1 \cos\theta + u_2 \sin\theta|^m \; p(\boldsymbol{u}) d\boldsymbol{u} \\ &= \|\boldsymbol{x}\|^m \|\boldsymbol{y}\|^m \int |u_1|^m |u_1 \cos\theta + u_2 \sin\theta|^m \; p(\boldsymbol{u}) d\boldsymbol{u} \end{aligned}$$

with $p(\boldsymbol{u}) = \mathcal{CN}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$.

Now we work out the term $A = |u_1 \cos\theta + u_2 \sin\theta|^{2s}$ (with $m = 2s$) as we focus on even powers:

$$\begin{aligned} A &= (\overline{u}_1 \cos\theta + \overline{u}_2 \sin\theta)^s (u_1 \cos\theta + u_2 \sin\theta)^s \\ &= \left( \sum_{j=0}^{s} \binom{s}{j} (\overline{u}_1 \cos\theta)^j (\overline{u}_2 \sin\theta)^{s-j} \right) \left( \sum_{i=0}^{s} \binom{s}{i} (u_1 \cos\theta)^i (u_2 \sin\theta)^{s-i} \right) \\ &= \sum_i \sum_j \binom{s}{j} \binom{s}{i} \overline{u}_1^j u_1^i \overline{u}_2^{s-j} u_2^{s-i} (\cos\theta)^{i+j} (\sin\theta)^{2s-i-j} \end{aligned}$$

One can notice that $\mathbb{E}[A] = \sum_i \sum_j \binom{s}{j}\binom{s}{i} \mathbb{E}[\overline{u}_1^j u_1^i \overline{u}_2^{s-j} u_2^{s-i}]$ has its cross terms equal to 0: when $i \neq j$, the expectation inside the sum is equal to zero because $u_1$ and $u_2$ are i.i.d (their correlation is then equal to 0) or we can see this by rotational invariance (any complex random variable with a phase uniform between 0 and $2\pi$

has a mean equal to zero). Therefore, we obtain:

$$\mathbb{E}[A] = \sum_{i=0}^{s} \binom{s}{i}^2 |u_1|^{2i} |\cos\theta|^{2i} |u_2|^{2(s-i)} |\sin\theta|^{2(s-i)}$$

Using the computation of $\mathbb{E}[A]$, we can therefore deduce the analytical formula for the kernel:

$$\begin{aligned}
\frac{k_{2s}(\boldsymbol{x},\boldsymbol{y})}{\|\boldsymbol{x}\|^m\|\boldsymbol{y}\|^m} &= \int |u_1|^{2s} \sum_{i=0}^{s} \binom{s}{i}^2 |u_1|^{2i} |\cos\theta|^{2i} |u_2|^{2(s-i)} |\sin\theta|^{2(s-i)}\ p(\boldsymbol{u}) d\boldsymbol{u} \\
&= \sum_{i=0}^{s} \binom{s}{i}^2 \mathbb{E}\left[|u_1|^{2(s+i)} |\cos\theta|^{2i}\right]\ \mathbb{E}\left[|u_2|^{2(s-i)} |\sin\theta|^{2(s-i)}\right] \\
&= \sum_{i=0}^{s} \binom{s}{i}^2 |\cos\theta|^{2i}\, 2^{s+i} \sigma^{*2(s+i)}\, \Gamma(s+i+1)\, |\sin\theta|^{2(s-i)}\, 2^{s-i} \sigma^{*2(s-i)}\, \Gamma(s-i+1) \\
&= \sum_{i=0}^{s} \binom{s}{i}^2 2^{2s} \sigma^{*4s}\, |\cos\theta|^{2i}\, |\sin\theta|^{2(s-i)} \Gamma(s+i+1)\, \Gamma(s-i+1),
\end{aligned}$$

where $\Gamma(\cdot)$ is the Gamma function and $\sigma^{*2} = \frac{1}{2}\sigma^2$ being the variance of the real and imaginary parts of $\boldsymbol{u}$. W.l.o.g. we will use $\sigma^2 = 1$ and thus $\sigma^{*2} = 0.5$ in the following.

We can simplify this formula, by noticing that $\sin^2\theta = 1 - \cos^2\theta$. The latter formula then becomes:

$$\frac{k_{2s}(\boldsymbol{x},\boldsymbol{y})}{\|\boldsymbol{x}\|^m\|\boldsymbol{y}\|^m} = \sum_{i=0}^{s} \binom{s}{i}^2 |\cos\theta|^{2i}\, (1-\cos^2\theta)^{(s-i)}\, 2^{2s} \sigma^{*4s}\, \Gamma(s+i+1)\, \Gamma(s-i+1)$$

The new term can be expanded using a binomial expansion: $(1-\cos^2\theta)^{(s-i)} = \sum_{t=0}^{s-i} (-\cos^2\theta)^t$, leading to

$$\frac{k_{2s}(\boldsymbol{x},\boldsymbol{y})}{\|\boldsymbol{x}\|^m\|\boldsymbol{y}\|^m} = \sum_{i=0}^{s} \sum_{t=0}^{s-i} \binom{s}{i}^2 (s+i)!(s-i)! \binom{s-i}{t} (-1)^t \cos^{2(i+t)}\theta$$

Using the change of variable $a = i + t$ and keeping $i = i$, we obtain:

$$\frac{k_{2s}(\boldsymbol{x},\boldsymbol{y})}{\|\boldsymbol{x}\|^m\|\boldsymbol{y}\|^m} = \sum_{a=0}^{s} \left[\sum_{i=0}^{a} \binom{s}{i}^2 (s+i)!(s-i)! \binom{s-i}{a-i} (-1)^{a-i}\right] \cos^{2a}\theta$$

We focus on the term between the parenthesis that we will call $T_a$:

$$\begin{aligned}
T_a &= \sum_{i=0}^{a} \frac{s!^2}{(s-i)!^2 i!^2} (s+i)!(s-i)! \frac{(s-i)!}{(a-i)!(s-a)!} (-1)^{a-i} \\
&= \sum_{i=0}^{a} (s!)^2 \frac{(s+i)!}{(i!)^2 (a-i)!(s-a)!} (-1)^{a-i} \\
&= (s!)^2 \binom{s}{a} (-1)^a \sum_{i=0}^{a} \binom{a}{i} \binom{s+i}{i} (-1)^i
\end{aligned}$$

Using the upper negation ($\binom{a}{b} = (-1)^b \binom{b-a-1}{b}$) so here $\binom{s+i}{i} = \binom{i-i-s-1}{i}(-1)^i = \binom{-s-1}{i}(-1)^i$ leads to

$$T_a = (s!)^2 \binom{s}{a} (-1)^a \sum_{i=0}^{a} \binom{a}{i} \binom{-s-1}{i} \tag{E.1}$$

$$= (s!)^2 \binom{s}{a} (-1)^a \sum_{i=0}^{a} \binom{a}{i} \binom{-s-1}{-s-1-i} \tag{E.2}$$

Now we can use the Vandermonde identity $\sum_{i=0}^{n} \binom{a}{i}\binom{b}{n-i} = \binom{a+b}{n}$, yielding:

$$T_a = (s!)^2 \binom{s}{a} (-1)^a \binom{a-s-1}{-s-1} = (s!)^2 \binom{s}{a} (-1)^a \binom{a-s-1}{a} \tag{E.3}$$

$$= (s!)^2 \binom{s}{a} \binom{s}{a} = (s!)^2 \binom{s}{a}^2 \tag{E.4}$$

where in the last line we used again the upper negation formula.
This leads to the desired result for $m = 2s$

$$k_{2s}(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x}\|^m \|\boldsymbol{y}\|^m \sum_{i=0}^{s} (s!)^2 \binom{s}{i}^2 \cos^{2i}\theta$$

# Bibliography


M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, Inc., USA, 1974.

T. D. Ahle, M. Kapralov, J. B. T. Knudsen, R. Pagh, A. Velingker, D. P. Woodruff, and A. Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, page 141–160. Society for Industrial and Applied Mathematics, 2020.

S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. VQA: Visual Question Answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433, 2015.

H. Aschard. A perspective on interaction effects in genetic association studies. *Genetic epidemiology*, 40(8):678–688, 2016.

H. Avron, H. L. Nguyen, and D. P. Woodruff. Subspace embeddings for the polynomial kernel. In *Advances in Neural Information Processing Systems 27*, pages 2258–2266. Curran Associates, Inc., 2014.

H. Avron, M. Kapralov, C. Musco, C. Musco, A. Velingker, and A. Zandieh. Random Fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 253–262. PMLR, 2017.

N. Banar, W. Daelemans, and M. Kestemont. Character-level transformer-based neural machine translation. In *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, page 149–156. Association for Computing Machinery, 2020.

M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton. Demystifying MMD GANs. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1lUOzWCW.

M. Blondel, M. Ishihata, A. Fujino, and N. Ueda. Polynomial networks and factorization machines: New insights and efficient training algorithms. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, volume 48, pages 850–858. PMLR, 2016.

R. Boloix-Tortosa, J. J. Murillo-Fuentes, F. J. Payán-Somet, and F. Pérez-Cruz. Complex gaussian processes for regression. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5499–5511, 2018.

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.

T. Brown et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11(4), 2010.

M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, page 693–703. Springer-Verlag, 2002.

K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*. BMVA Press, 2014.

K. Choromanski, M. Rowland, and A. Weller. The unreasonable effectiveness of structured random orthogonal embeddings. In *Advances in Neural Information Processing Systems 31*, pages 218–227. Curran Associates Inc., 2017.

K. Choromanski, M. Rowland, T. Sarlos, V. Sindhwani, R. Turner, and A. Weller. The geometry of random features. In A. Storkey and F. Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84, pages 1–9. PMLR, 2018.

K. Choromanski et al. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

A. Cotter, J. Keshet, and N. Srebro. Explicit approximations of the gaussian kernel. *CoRR*, abs/1109.4603, 2011.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893. IEEE Computer Society, 2005.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186. Association for Computational Linguistics, 2019.

J. Dong, S. Gigan, F. Krzakala, and G. Wainrib. Scaling up echo-state networks with multiple light scattering. In *2018 IEEE Statistical Signal Processing Workshop (SSP)*, pages 448–452. IEEE, 2018.

J. Dong, M. Rafayelyan, F. Krzakala, and S. Gigan. Optical reservoir computing using multiple light scattering for chaotic systems prediction. *IEEE Journal of Selected Topics in Quantum Electronics*, 26(1):1–12, 2019.

D. Dua and C. Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

B. J. Fino and V. R. Algazi. Unified matrix treatment of the fast walsh-hadamard transform. *IEEE Transactions on Computers*, 25(11):1142–1146, 1976.

C. A. Floudas and P. M. Pardalos, editors. *Encyclopedia of Optimization, Second Edition*. Springer, 2009.

A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 457–468. Association for Computational Linguistics, 2016.

Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact bilinear pooling. *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 317–326, 2016.

D. Garreau, W. Jitkrittum, and M. Kanagawa. Large sample analysis of the median heuristic. *arXiv preprint arXiv:1707.07269*, 2017.

M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2002.

A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

S. Gupta, R. Gribonval, L. Daudet, and I. Dokmanic. Don't take it lightly: Phasing optical random projections with unknown operators. In *Advances in Neural Information Processing Systems*, volume 32, pages 14826–14836. Curran Associates, Inc., 2019.

R. Hamid, Y. Xiao, A. Gittens, and D. DeCoste. Compact random feature maps. In *Proceedings of the 31th International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 19–27. PMLR, 2014.

I. Han, A. Zandieh, and H. Avron. Random gegenbauer features for scalable kernel methods. *arXiv preprint arXiv:2202.03474*, 2022.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

J. Hensman, N. Durrande, and A. Solin. Variational Fourier features for Gaussian processes. *Journal of Machine Learning Research*, 18(151):1–52, 2018.

P. Kar and H. Karnick. Random feature maps for dot product kernels. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *JMLR Proceedings*, pages 583–591. JMLR, 2012.

N. Keriven, D. Garreau, and I. Poli. Newma: a new method for scalable model-free online change-point detection. *arXiv preprint arXiv:1805.08061*, 2018.

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

T. Lin, A. RoyChowdhury, and S. Maji. Bilinear convolutional neural networks for fine-grained visual recognition. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 40(6):1309–1322, 2018.

F. Liu, X. Huang, Y. Chen, and J. A. K. Suykens. Random features for kernel approximation: A survey in algorithms, theory, and beyond. *CoRR*, abs/2004.11154, 2020.

Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

A. Liutkus, D. Martina, S. Popoff, G. Chardon, O. Katz, G. Lerosey, S. Gigan, L. Daudet, and I. Carron. Imaging with nature: Compressive imaging using a multiply scattering medium. *Scientific reports*, 4:5552, 2014.

D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, pages 1150–1157. IEEE Computer Society, 1999.

A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150. Association for Computational Linguistics, 2011.

D. Milios, R. Camoriano, P. Michiardi, L. Rosasco, and M. Filippone. Dirichlet-based gaussian processes for large-scale calibrated classification. In *Advances in Neural Information Processing Systems 31*, pages 6008–6018. Curran Associates, Inc., 2018.

F. D. Neeser and J. L. Massey. Proper complex random processes with applications to information theory. *IEEE transactions on information theory*, 39(4):1293–1302, 1993.

R. Ohana, J. Wacker, J. Dong, S. Marmin, F. Krzakala, M. Filippone, and L. Daudet. Kernel computations from large-scale random features obtained by optical processing units. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 9294–9298, 2020.

K. I. Park. *Fundamentals of Probability and Stochastic Processes with Applications to Communications*. Springer, 1st edition, 2018.

A. Paszke et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc., 2019.

F. Pedregosa, , et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

J. Pennington, F. X. X. Yu, and S. Kumar. Spherical random features for polynomial kernels. In *Advances in Neural Information Processing Systems 28*, pages 1846–1854. Curran Associates, Inc., 2015.

N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 239–247. Association for Computing Machinery, 2013.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*, pages 1177–1184. Curran Associates Inc., 2007.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

C. E. Rasmussen and J. Quiñonero Candela. Healing the relevance vector machine through augmentation. In *Proceedings of the 22nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, page 689–696. PMLR, 2005.

A. S. Rawat, J. Chen, F. Yu, A. T. Suresh, and S. Kumar. *Sampled Softmax with Random Fourier Features*, volume 32, pages 13834–13844. Curran Associates Inc., 2019.

S. Rendle. Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, pages 995–1000, 2010.

A. Rudi and L. Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

O. Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

A. Saade, F. Caltagirone, I. Carron, L. Daudet, A. Drémeau, S. Gigan, and F. Krzakala. Random projections through multiple optical scattering: Approximating kernels at the speed of light. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6215–6219, March 2016.

I. J. Schoenberg. Metric spaces and completely monotone functions. *Annals of Mathematics*, 39(4):811–841, 1938.

B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

A. Smola, Z. Óvári, and R. C. Williamson. Regularization with dot-product kernels. In *Advances in Neural Information Processing Systems 13*, pages 308–314. Curran Associates, Inc., 2000.

M. L. Stein. *Interpolation of spatial data*. Springer-Verlag, 1999.

I. Steinwart and A. Christmann. *Support Vector Machines*. Information science and statistics. Springer, 2008.

E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 3645–3650. Association for Computational Linguistics, 2019.

D. J. Sutherland and J. Schneider. On the error of random fourier features. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 862–871. AUAI Press, 2015.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

M. E. Tipping. The relevance vector machine. In *Advances in Neural Information Processing Systems 12*, page 652–658. Curran Associates, Inc., 1999.

M. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *JMLR Proceedings*, pages 567–574. JMLR, 2009.

L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

J. A. Tropp. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(1-2):115–126, 2011.

A. V. Uzilov, J. M. Keegan, and D. H. Mathews. Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. *BMC Bioinformatics*, 7:173, 2006.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 6000–6010. Curran Associates, Inc., 2017.

R. Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2018.

J. Wacker and M. Filippone. Local random feature approximations of the gaussian kernel. In *Proceedings of the 26th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, Procedia Computer Science. Elsevier, 2021.

J. Wacker, M. Ladeira, and J. E. V. Nascimento. Transfer learning for brain tumor segmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries - 6th International Workshop, BrainLes 2020, Held in Conjunction with MICCAI 2020.*, volume 12658 of *Lecture Notes in Computer Science*, pages 241–251. Springer, 2020.

J. Wacker, M. Kanagawa, and M. Filippone. Improved random features for dot product kernels. *arXiv preprint arXiv:2201.08712*, 2022a.

J. Wacker, R. Ohana, and M. Filippone. Complex-to-real random features for polynomial kernels. *arXiv preprint arXiv:2202.02031*, 2022b.

G. Wahba. *Spline Models for Observational Data*. SIAM, 1990.

K. Q. Weinberger, A. Dasgupta, J. Langford, A. J. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning*, volume 382, pages 1113–1120. ACM, 2009.

O. Weissbrod, D. Geiger, and S. Rosset. Multikernel linear mixed models for complex phenotype prediction. *Genome Research*, 26(7):969–979, 2016.

P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-ucsd birds 200. Technical report, Caltech, 2010. URL http://www.vision.caltech.edu/visipedia/CUB-200.html.

C. K. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688. Curran Associates, Inc., 2000.

D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

Y. Xiong, Z. Zeng, R. Chakraborty, M. Tan, G. Fung, Y. Li, and V. Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14138–14148, 2021.

H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 195–206, 2003.

Z. Yang, M. Moczulski, M. Denil, N. D. Freitas, A. Smola, L. Song, and Z. Wang. Deep fried convnets. *Proceedings of the 2015 IEEE International Conference on Computer Vision*, pages 1476–1483, 2015.

F. X. Yu, A. T. Suresh, K. Choromanski, D. Holtmann-Rice, and S. Kumar. Orthogonal random features. In *Advances in Neural Information Processing Systems*, volume 30, page 1983–1991. Curran Associates Inc., 2016.

M. Zaheer et al. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, pages 17283–17297. Curran Associates, Inc., 2020.