Skolkovo Institute of Science and Technology

FAST NUMERICAL LINEAR ALGEBRA METHODS FOR MACHINE LEARNING

Doctoral Thesis

by

MARINA MUNKHOEVA

DOCTORAL PROGRAM IN
COMPUTATIONAL AND DATA SCIENCE AND ENGINEERING

Supervisor
Professor Ivan Oseledets

Moscow — 2021

Declaration

I hereby declare that the work presented in this thesis was carried out by myself at Skolkovo Institute of Science and Technology, Moscow, except where due acknowledgement is made, and has not been submitted for any other degree.

Marina Munkhoeva
Professor Ivan Oseledets

Abstract

Machine learning has become ubiquitous. We can see extraordinary progress in many areas, including computer vision, natural language processing, and speech recognition. However, the high-dimensional and large-scale nature of the visual, language or audio data in modern applications requires a tremendous amount of compute and parameters. While it continues to succeed even further, there arises a need to tackle its ever-growing demand for computational resources.

One way of doing so is to introduce frugal computational principles directly into the design of data pipelines, parameter representation and transformations we use in machine learning models. To ease the computational load, one would often introduce approximations that trade performance off. For example, kernel methods, previously deemed unfit for large-scale datasets due to high computational complexity, have been extended to modern problem sizes via approximate feature maps facilitating the use of linear classical machine learning algorithms on nonlinear representations of data. In many cases, the quantities of interest are also simply intractable, leaving us no other option than to search for accurate estimation techniques based on Monte Carlo or quadrature rules.

In this thesis, we approach both these problems — approximation quality and computational complexity, simultaneously. Accordingly, we develop methods that leverage advanced numerical procedures to achieve state-of-the-art efficiency in both approximation and complexity aspects with derived theoretical guarantees in several settings. The contributions of this thesis include random sparse feature maps for kernel approximation, novel intrinsic measure for data distribution comparison, and practical means for graph summarisation.

Publications

The four papers that constitute the body of this thesis has been published in the following venues. * denotes equal contribution. **Key** contributions are in bold, marginal in light typeface.

Chapter 2 expands on:

- Marina Munkhoeva, Yermek Kapushev, Evgeny Burnaev, and Ivan Oseledets.
  "Quadrature-based Features for Kernel Approximation."
  Advances in Neural Information Processing Systems, pages 9147–9156, 2018.

  Author's contributions in the paper include:

  - **design**
  - **implementation**
  - analysis
  - **experiments**
  - **draft**
  - **revisions**

Chapter 3 expands on:

- Anton Tsitsulin*, Marina Munkhoeva*, Davide Mottin, Panagiotis Karras, Alex Bronstein, Ivan Oseledets, and Emmanuel Mueller.
  "The Shape of Data: Intrinsic Distance for Data Distributions."
  International Conference on Learning Representations. 2020.

  Author's contributions in the paper include:

  - **conception and design**
  - **implementation**
  - **theoretical analysis**
  - **experiments**
  - **draft**
  - **revisions**

Chapter 4 expands on:

- Anton Tsitsulin*, Marina Munkhoeva*, and Bryan Perozzi.
  "Just SLaQ When You Approximate: Accurate Spectral Distances for Web-Scale Graphs."
  Proceedings of The Web Conference, pages 2697–2703, 2020.

  Author's contributions in the paper include:

  - design
  - **implementation**
  - **theoretical analysis**
  - **experiments**
  - **draft**

Chapter 5 expands on:

- Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller.
  "FREDE: Linear-Space Anytime Graph Embeddings."
  accepted to *International Conference on Very Large Data Bases.* 2021.

  Author's contributions in the paper include:

  - design
  - **implementation**
  - **theoretical analysis**
  - **experiments**
  - draft
  - revisions

Table of Contents

List of Figures

List of Tables

Notation

Chapter 1

MVM	Matrix-Vector Multiplication
PCA	Principal Component Analysis
QR	QR decomposition
SVM	Support Vector Machine

Chapter 2

GQ	Gaussian Quadrature
MC	Monte Carlo
ORF	Orthogonal Random Features
PNG	Pointwise Nonlinear Gaussian (kernel)
QF	Quadrature-based Features
QMC	quasi-Monte Carlo
RBF	Radial Basis Function
RFF	Random Fourier Features
ROM	Random Orthogonal Matrices
SR	Spherical-radial rules

Chapter 3

FID	Fréchet Inception Distance

GAN	Generative Adversarial Network
GILBO	Generative Information Lower Bound
GS	Geometry Score
HKT	Heat Kernel Trace
IMD	Intrinsic Multiscale Distance
IS	Inception Score
KID	Kernel Inception Distance
kNN	k Nearest Neighbours
LBO	Laplace-Beltrami Operator
MMD	Maximum Mean Discrepancy
PIP	Pairwise Inner Product
SLQ	Stochastic Lanczos Quadrature
WGAN	Wasserstein Generative Adversarial Network
WGAN-GP	Wasserstein Generative Adversarial Network with Gradient Penalty

Chapter 4

FINGER	Fast Incremental von Neumann Graph Entropy
LAPACK	Linear Algebra PACKage
NetLSD	Network Laplacian Spectral Descriptor
VNGE	Von Neumann Graph Entropy

Chapter 5

FD	Frequent Directions
PPR	Personalized PageRank
SVD	Singular Value Decomposition

Introduction

Motivation

The extent to which machine learning solutions succeed in modern world problems is largely determined by the amount of compute one is ready to employ into training such systems. The progress in graphics processing units promoted a breakthrough in computer vision and supervised learning research [125, 195] and still advances other areas in deep learning [172]. Although throwing as much compute as possible at the task given might be a useful exploration and discovery strategy in some cases [121], in most applications it is not the preferred approach, especially considering the energy challenges humankind is facing nowadays [74].

If we hope to expedite training and deployment of machine learning algorithms (without harming the planet), it would be helpful to embed frugal computational principles into the design of data pipelines, parameter representation and transformations we use in ML models [159, 139, 118]. Fortunately, as we will see later in this thesis, numerical linear algebra methods not only permit such changes but also do come with additional advantages in the form of better approximation error bounds and guarantees.

One of the challenges we address in this work is the time and space complexity connected with an enormous amount of data, its high-dimensional nature, and computations. All of the above requires efficient work with matrices, be it data/parameter representation or (linear) transformations. Modern image generation models increasingly work with high dimensional pixel space [115, 120, 34]. While larger batch sizes no longer inhibit optimisation [196, 106], smart parameter design also proves beneficial in many settings, e.g. neural network compression[139], learning latent permutations [55].

Another challenge we will look into is the approximation of quantities unattainable in closed form; this as well should be accurate and fast. Very often these quantities involve matrix functions, e.g. matrix exponential, log determinant. These usually appear in ideas closely connected or coming from the fields like partial differential equations [40] or statistical physics [3], where for large-scale problems it is crucial to estimate some aggregate properties of the eigenstructure in corresponding matrices.

Numerical Linear Algebra provides many powerful tools, both deterministic and randomised, to tackle large-scale linear algebra problems, which are ubiquitous in machine learning [30, 183]. Matrix algorithms facilitate fast running times and low memory footprint along with comprehensive statistical properties, thus offering scalability and other desirable features like interpretability and robustness. They also offer new perspectives on problems which previously would be considered quixotic — Chapter 3 develops a novel algorithm to compute the distance between data manifolds based on their intrinsic information, where without a specialised numerical integration scheme it would be impractical to estimate such quantities.

In this thesis, we address problems where we simultaneously aim at high approximation quality and low time/space complexity and achieve both with quality guarantees which we derive as well. By interpreting some of the problems through the lens of numerical integration, we can leverage the most successful concepts offered by NLA, e.g. sparsity and low-rank structure, eigen and singular value decompositions.

Contributions, Novelty and Impact

The contributions of this thesis are detailed below.

Chapter 2 delves into the random features for kernel approximation as we develop a novel stochastic quadrature-based approach for obtaining structured unbiased feature maps that boast better empirical and analytical accuracy in both kernel approximation quality and downstream tasks. The highlight of the method is the generalisation of the previous work in random features for kernels, e.g. the celebrated Random Fourier Features is a special case of the proposed quadrature-based features with low order quadrature. We develop sparse structured feature maps that naturally lead to fast matrix-by-vector multiplication resulting in speed up and lower memory requirements. This work appeared as a spotlight at the Neural Information Processing Systems (NeurIPS) conference in 2018 [154].

Chapter 3 attempts to answer perhaps one of the most challenging questions in machine learning — how can we measure the distance between two sets of points coming from possibly distinct high-dimensional manifolds? We attack this problem by approximating these samples with graphs allowing us to draw on intrinsic information within data to quantify the difference between data distributions such as sets of word vectors in different languages with unaligned vocabularies, training and generated samples in generative modelling and intermediate representations in neural networks. This work was previously published as [215] at the International Conference on Learning Representations (ICLR) in 2020.

Chapter 4 studies the spectral distance approximation for Web-scale graphs. We leverage recent advances in numerical linear algebra to develop an algorithm for linear time approximation of spectral descriptors used for graph comparison and classification. The method provides state-of-the-art accuracy approximation for graphs with billions of nodes and edges on a single machine in less than an hour. This work appeared at the Web Conference (WWW) 2020 [217].

Chapter 5 proposes a linear-space algorithm for calculating graph embeddings. Our method boasts quality guarantees along with anytime nature, i.e. the quality of the embeddings grows with the number of processed rows of a specific graph similarity matrix but can be stopped anytime to obtain embeddings. To the best of our knowledge, this is the first anytime algorithm for node embeddings. The work has been accepted to the international conference on Very Large Data Bases (VLDB) 2021 [216].

Chapter 1

Preliminaries

In this chapter, we establish the necessary notions and definitions that are used in this thesis and review the essential background. We discuss relevant matrix types and properties as well as matrix decompositions and algorithms for computing either full matrix factorizations or their most essential parts only. We explain some basic graph analysis and kernel methods as we will encounter them later in the thesis. We also cover the basics of numerical integration as more advanced methods will follow in the next chapters.

1.1 Common Notation and Symbols

Throughout the thesis, we use bold capital letters ($\mathbf{A}, \mathbf{M}$) for matrices, lowercase bold letters($\mathbf{x}, \mathbf{y}$) for vectors, calligraphic letters ($\mathcal{X}, \mathcal{Y}$) for vector spaces and manifolds, lowercase letters (c, f, g) for scalars and functions. Some of the notable symbols and exceptions to these rules are presented in Table 1.1.

Notation	Description
$\mathbf{I}$	an identity matrix, $\mathbf{I}_{ii} = 1$ and $\mathbf{I}_{ij} = 0$ for $i \neq j$
$\mathbf{D}$	a diagonal matrix, $\mathbf{D}_{ij} = 0$ if $i \neq j$
$\mathrm{tr}(\mathbf{A})$	trace of a matrix $\mathbf{A}$, $\mathrm{tr}(\mathbf{A}) = \sum_i \mathbf{A}_{ii}$
$f(\mathbf{A})$	a function f of a matrix $\mathbf{A}$
$\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}$	eigenvalue decomposition of a square matrix $\mathbf{A}$, if $\mathbf{A}$ is normal
$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$	singular value decomposition of a matrix $\mathbf{A}$
$\mathbb{E}_{p(x)} f(x)$	expected value of a function $f(x)$ under a distribution density $p(x)$

Table 1.1 Common notation

1.2 Matrices

We begin with matrices — the most popular representation tool in most scientific fields, especially in data-intensive ones such as computer science, statistics, and machine learning. In this thesis, we will encounter matrices as a way to describe:

- data — we store information with rows and columns representing objects and their features, respectively.
- a graph — depending on an application, we may represent graphs with their adjacency, Laplacian, or node similarity matrices (e.g., PageRank).
- a linear map — we represent linear transformation $\mathbf{A} : \mathcal{X} \to \mathcal{Y}$ from one vector space to another with operator matrix $\mathbf{A}$ via matrix-vector multiplication (MVM) $\mathbf{u} = \mathbf{A}\mathbf{v}$.
- a bilinear map — we represent some function $\mathbf{A} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ such that $(\mathbf{v}, \mathbf{u}) = \mathbf{v}^\top \mathbf{A}\mathbf{u}$.

Thus, the universality of matrix representation is vital to employ the full potential of fast numerical linear algebra methods in machine learning. We will briefly remind a few concepts from matrix calculations that will be relevant in the next chapters.

One of the primary considerations of this thesis is efficient computation. When dealing with repetitive *matrix-vector multiplications* (MVMs), a relevant notion is that of *sparsity*. Sparse matrices are characterized by having very few nonzero elements. Scientific computing benefits whenever sparsity is present as sparse matrices facilitate fast MVMs, e.g., an $n \times n$ matrix with n nonzero elements admits matrix-vector multiplication costing a total of n operations. In contrast, a dense matrix of the same size requires n^2 operations.

Spectral analysis tools, used in this thesis, require some exposition of notions connected to eigenvalues and eigenvectors. The eigenvalues of a square $n \times n$ matrix $\mathbf{A}$ are the zeros of the characteristic polynomial $p(\mathbf{v}) = \det(\mathbf{A} - \mathbf{v}\mathbf{I})$. Every $n \times n$ matrix has n eigenvalues. If $\lambda \in \lambda(\mathbf{A})$, i.e. λ belongs to the set of $\mathbf{A}$'s eigenvalues, there exists a nonzero vector $\mathbf{v}$ so that $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$, and $\mathbf{v}$ is said to be an eigenvector associated with λ. If $\mathbf{A}$ has n independent eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ and $\mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}_i$ for $i = 1\,{:}\,n$, then A is called diagonalizable.

Diagonalizable matrices are an important class of matrices that can also be defined through their similarity with a diagonal matrix: $\mathbf{A}$ is diagonalizable when there exists a nonsingular matrix $\mathbf{U}$ such that $\mathbf{U}^{-1}\mathbf{A}\mathbf{U} = \mathbf{D}$, where $\mathbf{D}$ is diagonal matrix, i.e. matrix with nonzero elements only on its diagonal.

Normal matrix $\mathbf{A}$ is a square complex matrix that commutes with its conjugate transpose $\mathbf{A}^*$: $\mathbf{A}\mathbf{A}^* = \mathbf{A}^*\mathbf{A}$. Among normal matrices there are unitary and Hermitian matrices. In this thesis, we mostly deal with their real counterparts — orthogonal and symmetric matrices.

Orthogonal matrix $\mathbf{Q}$ is a square matrix with rows and columns being orthogonal unit vectors: $\mathbf{Q}\mathbf{Q}^\top = \mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$. Orthogonal matrices are norm preserving, and they determine unitary transformations, such as rotations and reflections. In this thesis, we consider orthogonal matrices acting as random rotators of the points on the surface of the unit d dimensional sphere. The set of $n \times n$ orthogonal matrices forms a group, $O(n)$, known as the orthogonal group.

Symmetric matrix $\mathbf{S}$ is a square matrix which is equal to its transpose: $\mathbf{S} = \mathbf{S}^\top$. Since symmetric matrices are Hermitian, all their eigenvalues are real. Symmetric property, found in many graph-related matrices, empowers us with spectral analysis in Chapters 3 and 4.

1.2.1 Matrix Decompositions and Algorithms

We now give a brief overview of matrix decompositions and algorithms used in this thesis. A thorough exposition of matrix computations algorithms is given in [87].

QR decomposition factorizes a rectangular $m \times n$ matrix $\mathbf{A}$ into a product $\mathbf{A} = \mathbf{Q}\mathbf{R}$ of an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$. This factorization is important in the least squares and eigenvalue computations. Its efficient computation relies on the two following matrix types: Householder reflectors and Givens rotators, as they facilitate the introduction of zeros to form upper triangular matrix $\mathbf{R}$.

Householder reflection is described by the matrix $\mathbf{H} = \mathbf{I} - \beta\mathbf{v}\mathbf{v}^\top$, where vector $\mathbf{v}$ is a normal vector to the reflection hyperplane and $\beta = \frac{2}{\mathbf{v}^\top\mathbf{v}}$. Givens rotation is determined by the chosen coordinate axes (k, l) and rotation angle θ. The Givens matrix has the following nonzero entries: $G_{ii} = 1$ if $i \neq k, l$, $\mathbf{G}_{ii} = \cos(\theta)$ if $i = k, l$ and $\mathbf{G}_{k,l} = -\mathbf{G}_{l,k} = \sin(\theta)$.

QR decomposition can also be obtained with the Gram-Schmidt process and can be used to obtain random orthogonal matrices. QR decomposition is also a basis for the QR algorithm used in eigenvalue problems [169].

Eigendecomposition factorizes a square matrix $\mathbf{A}$ into the product $\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}$, where columns of matrix $\mathbf{Q}$ contain eigenvectors $\mathbf{v}$ and diagonal matrix $\Lambda = \operatorname{diag}(\lambda_i)$ contains eigenvalues λ_i of $\mathbf{A}$, i.e. $\mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}_i$. Not all matrices can be diagonalized in this way, but, important in this thesis, a symmetric matrix $\mathbf{S}$ can be decomposed as $\mathbf{S} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}$, where $\mathbf{Q}$ is orthogonal, i.e. the eigenvectors of $\mathbf{S}$ are chosen to be orthonormal. Eigendecomposition is a central tool for studying spectral properties of various matrices, e.g. the eigenvalues and eigenvectors of Hessian matrix provide geometric information of a loss surface [90, 165, 185].

The matrix spectrum contains valuable information and finds a wide range of applications in many scientific areas, such as statistics, graph analysis, and optimization. As we will see later, whenever matrix admits an eigendecomposition, it facilitates the computations involving functions of matrices, defined on the spectrum of matrices (e.g., $\exp(\mathbf{A}), \log(\mathbf{A})$).

Singular Value Decomposition factorizes any real rectangular $n \times m$ matrix $\mathbf{A}$ into a product $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ of orthogonal matrices $\mathbf{U}, \mathbf{V}$ containing left and right singular vectors respectively, and a diagonal matrix $\mathbf{\Sigma} = \mathrm{diag}(\sigma_i)$ containing singular values σ_i of $\mathbf{A}$. The rank of $\mathbf{A}$ is then equal to the number of its nonzero singular values. SVD is another important decomposition extensively used in matrix approximation problems. The best rank-k approximation of some matrix $\mathbf{A}$ can be obtained with truncated SVD $\mathbf{A}_k = \sum_i^k \lambda_i \mathbf{u}_i \mathbf{v}_i^\top$ — a sum of rank-1 matrices of the first k eigenvectors and eigenvalues. Thanks to Eckart-Young-Mirsky theorem [64], this can be analytically derived and we can quantify the approximation error in terms of singular values σ_i:

$$\min_{rank(\mathbf{B})=k} \|\mathbf{A}-\mathbf{B}\|_2 = \sigma_{k+1}, \quad \min_{rank(\mathbf{B})=k} \|\mathbf{A}-\mathbf{B}\|_F = \sqrt{\sum_{i>k}^{\min(m,n)} \sigma_i^2}. \tag{1.1}$$

Lanczos Algorithm is often used to compute a few most important eigenvalues and eigenvalues of sparse symmetric matrices [87]. The algorithm performs tridiagonalization and outputs an $n \times m$ matrix $\mathbf{V}$ with orthonormal columns and a tridiagonal real symmetric matrix $\mathbf{T} = \mathbf{V}^\top\mathbf{A}\mathbf{V}$ of size $m \times m$. Lanczos algorithm is primarily used to reduce the dimensionality of the problem under the assumption of the low-rank structure of the input matrix $\mathbf{A}$ as $\mathbf{T}$ is much easier to work with when m is small. When matrix $\mathbf{A}$ is large and sparse, the Lanczos process is especially advantageous since it relies on matrix-vector multiplications and does not even require access to explicit matrix. The algorithm has an iterative nature and can be seen as an improved power method that computes the orthogonal basis in the Krylov subspace $K(\mathbf{A}, \mathbf{q}, k) = \mathrm{span}\{\mathbf{q}, \mathbf{A}\mathbf{q}, \dots, \mathbf{A}^{k-1}\mathbf{q}\}$. Lanczos algorithm is sensitive to numerical instability resulting in a loss of orthogonality among the Lanczos vectors $\mathbf{v}_k$ — columns of $\mathbf{V}$, which is typically overcome with reorthogonalization techniques.

1.2.2 Matrices in Kernel Methods

We refer to [188, 153] for a thorough introduction to the kernel methods. Below, we cover some of the basic notions used in the later chapters.

Many classical algorithms, such as Support Vector Machine (SVM) [53], principal component analysis (PCA) [166, 108], employ training points $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ only through their inner product $\langle \mathbf{x}, \mathbf{y} \rangle$, which can be thought of a similarity measure or distance between these points. However, the linear functions induced by the inner product are often not sufficiently expressive for many realistic learning tasks. Kernel methods solve this problem by replacing the inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ in $\mathcal{X}$ by some other nonlinear similarity measure, an inner product in some other feature space $\mathcal{H}$.

Extending the inner product in the original space into high-dimensional Hilbert space $\mathcal{H}$ can be done by means of a *feature map* ψ:

$$\begin{aligned} \psi : \mathcal{X} &\to \mathcal{H}, \\ \mathbf{x} &\to \psi(\mathbf{x}). \end{aligned}$$

The inner product in $\mathcal{H}$ can now be written as:

$$k(\mathbf{x}, \mathbf{y}) = \langle \psi(\mathbf{x}), \psi(\mathbf{y}) \rangle_{\mathcal{H}}, \tag{1.2}$$

we refer to function k as *kernel function*. For example, consider a polynomial feature map $\psi(\mathbf{x}) = (\mathbf{x}_1^2, \mathbf{x}_2^2, \sqrt{2}\mathbf{x}_1\mathbf{x}_2)$ for $\mathbf{x} \in \mathbb{R}^2$. The corresponding kernel

$$\langle \psi(\mathbf{x}), \psi(\mathbf{y}) \rangle_{\mathcal{H}} = \mathbf{x}_1^2\mathbf{y}_1^2 + \mathbf{x}_2^2\mathbf{y}_2^2 + 2\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 = \langle \mathbf{x}, \mathbf{y} \rangle^2,$$

yields the square of the inner product in the original space.

Thus, by substituting $\langle \mathbf{x}, \mathbf{y} \rangle$ with $\langle \psi(\mathbf{x}), \psi(\mathbf{y}) \rangle_{\mathcal{H}}$, we nonlinearly extend classical algorithms, relying only on the inner product between examples. However, the direct application of the map is often computationally restrictive and in some cases impossible due to infinite dimensionality of the target feature space. We thus rely on the *kernel trick* — there are alternative ways to evaluate $\langle \psi(\mathbf{x}), \psi(\mathbf{y}) \rangle_{\mathcal{H}}$ whenever the kernel is positive definite. The kernel is said to be positive definite when it is symmetric $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$ and its Gram matrix is positive definite

$$\sum_{i,j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0,$$

for any $n \in \mathbb{N}$, any $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathcal{X}$ and any coefficients $c_i, c_j \in \mathbb{R}$. The Gram matrix of all data points in the new feature space is called the *kernel matrix* and is defined as $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The kernel trick has been an efficient technique when dataset sizes did not exceed several thousand points, today, they are generally considered limited for large-scale learning due to demanding time and space requirements, e.g., it is often required to invert the kernel matrix, which is cubic in time.

The most celebrated technique addressing scalability of kernel methods by *reverting* the kernel trick is called Random Fourier Features (RFF) [174]. In short, it can be described as a feature map construction $\hat{\psi} : \mathbb{R}^D \to \mathbb{R}^d$. This randomised map should approximate the inner product in (1.2):

$$k(\mathbf{x}, \mathbf{y}) = \langle \psi(\mathbf{x}), \psi(\mathbf{y}) \rangle_{\mathcal{H}} \approx \hat{\psi}(\mathbf{x})^\top \hat{\psi}(\mathbf{x}),$$

This technique is possible for shift-invariant kernels thanks to Bochner's theorem, which we will encounter later in the thesis. Kernel function is called shift-invariant if $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} - \mathbf{y})$ holds. Many popular kernels are shift-invariant, e.g., Gaussian, Laplace, Cauchy. The widely used Gaussian (or radial basis function) kernel is:

$$k(\mathbf{x}, \mathbf{y}) = \exp\Big(- \frac{||\mathbf{x} - \mathbf{y}||^2}{2\sigma^2} \Big),$$

where σ is a parameter. The RFF mapping most often consists of some nonlinear function (e.g., sine and cosine for the Gaussian kernel) applied to a linear transformation defined by some rectangular matrix $\mathbf{W}$, i.e., $\hat{\psi}(\mathbf{x}) = [\cos(\mathbf{W}\mathbf{x}), \sin(\mathbf{W}\mathbf{x})]$. After applying such nonlinear approximate map to the data, we can use this new representation in linear algorithms, such as ridge regression. In this thesis, we will encounter several RFF extensions for scaling up kernel methods and their generalisation through the use of numerical integration.

1.2.3 Matrix Representations of Graphs

Graphs are an essential representation tool for various data. In turn, a graph can be represented with associated matrices such as adjacency matrix or graph Laplacian. We will briefly review basic graph notation and facts relevant to the problems in this thesis.

Let $G = (V, E)$ be an undirected graph, represented with a set of vertices $V = (v_1, \dots, v_n)$, $|V| = n$ and a set of edges $E \subseteq V \times V$, $|E| = m$. The adjacency matrix $\mathbf{A}$ is an $n \times n$ matrix having a positive weight (1 if the graph is unweighted) $\mathbf{A}_{ij} > 0$ associated with each edge (v_i, v_j) and 0 otherwise.

In this thesis, we work a lot with *graph Laplacian*, an analogue of divergence of the gradient of a function in vector spaces when considering scalar-valued functions on vertices and edges of the graph. We refer to [101] for a detailed exposition of the topic. There are two formulations of graph Laplacian we use in this work. The first is an *unnormalized* Laplacian, defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. However, we focus more on the *normalized* Laplacian matrix $\mathscr{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{D}$ is the diagonal matrix with node degrees on the diagonal $\mathbf{D} = \text{diag}(\deg(v_i))$, i.e. $\mathbf{D}_{ii} = \sum_{j=1}^n \mathbf{A}_{ij}$ and $\mathbf{I}$ is the identity matrix. Where applicable, we use the notation for graphs and corresponding matrices as defined in Table 1.2.

Notation	Description
V	a set of nodes, $V = (v_1, \dots, v_n)$, $\|V\| = n$
E	a set of edges, $E \subseteq (V \times V)$, $\|E\| = m$
G	a graph $G = (V, E)$
$N(v)$	set of neighbours of node v
$\deg(v)$	degree of node v
$\mathbf{A}$	an $n \times n$ adjacency matrix, $\mathbf{A}_{ij} \in \mathbb{R}$
$\mathbf{D}$	an $n \times n$ degree matrix, $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$
$\mathbf{L}$	an $n \times n$ Laplacian matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$
$\mathscr{L}$	an $n \times n$ normalized Laplacian matrix, $\mathscr{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$

Table 1.2 Graph-related notation

1.2.4 Matrix Sketching

In large-scale settings, we are often more restricted in time and space so that it is no longer feasible to use most matrix approximation and decomposition methods. With heavier restrictions on computational complexity, there are also relaxed precision requirements. Matrix sketching allows us to efficiently approximate input matrix $\mathbf{M}$ by a low-dimensional sketch matrix $\mathbf{S}$ retaining most of the information in $\mathbf{M}$ with some theoretical guarantees on approximation error. These algorithms work in a streaming setting, i.e., rows/columns arrive one after another, thus alleviating the need to store and manipulate the full matrix when dealing with matrix factorization algorithms. We refer to [232] for a detailed exposition of the topic. We will cover relevant sketching algorithms and their minimization objectives later in Chapter 5.

1.2.5 Matrix Functions

As we will consider estimating aggregate quantities of matrix functions in Chapters 3 and 4, we also need to cover some background on functions of matrices. We refer to [104] for an extensive introduction to the topic.

In this thesis, we encounter several types of matrix functions — elementwise operations on matrices such as $\sin(\mathbf{A}) = (\sin a_{ij})$, functions with scalar output such as $\det(\mathbf{A})$, $\operatorname{tr}(\mathbf{A})$, matrix factorizations and mappings like $A^\top$. However, in this section we are interested in matrix function viewed as a generalisation of scalar function $f(x)$, where $x \in \mathbb{R}$. Two examples of such matrix functions are matrix logarithm and exponential, whose scalar counterparts admit power series representations:

$$\log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots,$$

with $|x| < 1$, and

$$\exp(x) = 1 + x + \frac{x^2}{2} + \dots .$$

We can substitute x with $\mathbf{A}$ to obtain matrix function

$$\log(\mathbf{I} + \mathbf{A}) = \mathbf{A} - \frac{\mathbf{A}^2}{2} + \frac{\mathbf{A}^3}{3} - \dots , \qquad \exp(\mathbf{A}) = \mathbf{I} + \mathbf{A} + \frac{\mathbf{A}^2}{2} + \dots ,$$

with spectral radius $\rho(\mathbf{A}) < 1$ to ensure convergence of the series in the logarithm case.

There are several equivalent ways to formally define matrix functions to ensure consistency and applicability to arbitrary functions f. Perhaps the most simple is to use Jordan Canonical Form $\mathbf{A} = \mathbf{ZJZ}^{-1}$ with nonsingular $\mathbf{Z}$, where $\mathbf{J}$ consists of bidiagonal blocks on the diagonal corresponding to each eigenvalue of $\mathbf{A}$, each block has 1s on the superdiagonal and corresponding λ_i on the main diagonal. The function is then applied to the Jordan block-diagonal matrix only. In this thesis, we treat diagonalizable matrices that permit even simpler form as JCF reduces to eigendecomposition, and thus $f(\mathbf{A}) = \mathbf{Q}f(\mathbf{\Lambda})\mathbf{Q}^{-1}$.

We will treat two functions in the next chapters — exponential $\exp(x)$ and $x\log(x)$ that can be defined on the spectrum of $\mathbf{A}$. Although we will not explicitly compute their values as matrix functions, we will need this exposition for the theoretical derivation of the aggregate quantities of interest that depend on these functions.

1.3 Numerical Integration

Integrals are ubiquitous in machine learning. Most problems are defined as an expected value of some function $f(x)$ over data distribution $p(x)$, $\mathbb{E}_{p(x)}f(x)$. Usually, such integrals in realistic high-dimensional settings are rarely analytically tractable (with some exceptions requiring restrictive assumptions on the model). Thus, it is common to use empirical estimates for such quantities with a finite data sample.

In this thesis, we adopt a different perspective and find integrals in offbeat settings. We rewrite kernels (Chapter 2) and bilinear forms (Chapters 3 and 4) as integrals and apply quadrature rules to obtain their accurate estimates. To this end, we will briefly review here the basics of numerical integration — a family of methods to estimate a numerical value of a definite integral.

1.3.1 Gaussian Quadrature

Historically the term *quadrature* stands for calculating area. Today, a quadrature rule is an approximation of the definite integral of a function, commonly represented as a weighted sum of integrand values at given points inside the domain of integration. A particular instantiation of integration points and their weights depend on the method used and the accuracy required from the estimator.

Gauss quadrature rule is a n-point quadrature devised to yield exact value for polynomials of degree $d \leq 2n - 1$. The most common example is the Gauss-Legendre rule for integration region $[-1, 1]$:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i), \tag{1.3}$$

where the choice of points x_i and weights w_i is dictated by the roots of the orthogonal polynomial associated with the quadrature. For different integration regions and integrand end-point behaviour, many other quadrature rules with the corresponding orthogonal polynomials (Gauss-Hermite, Chebyshev-Gauss, etc.) exist. For an extensive introduction to the topic, we refer to [200].

In the following chapters, we will see quadrature rules when the integration region is an infinite range or a surface of d-dimensional sphere. We will also appreciate the connection between the Lanczos process and orthogonal polynomials associated with a bilinear form $\mathbf{v}^\top \mathbf{A}\mathbf{v}$ in Chapter 3, where we will estimate the trace of matrix exponential $\mathrm{tr}(\exp(\mathbf{A}))$.

Chapter 2

Structured Feature Maps for Kernel Approximation

In this chapter, we apply numerical integration to kernel approximation. We start by describing how scalability of kernel methods is addressed through random feature maps by interpreting the kernel as an integral and discuss relevant methods. We then define spherical and radial quadratures and show how they generalise existing methods. For space and time efficiency we also use sparsity in the factors in the proposed quadrature-based feature maps.

2.1 Introduction

Kernel methods proved to be an efficient technique in numerous real-world problems. The core idea of kernel methods is the kernel trick – compute an inner product in a high-dimensional (or even infinite-dimensional) feature space by means of a kernel function k:

$$k(\mathbf{x}, \mathbf{y}) = \langle \psi(\mathbf{x}), \psi(\mathbf{y}) \rangle, \tag{2.1}$$

where $\psi : \mathcal{X} \to \mathcal{H}$ is a non-linear feature map transporting elements of input space $\mathcal{X}$ into a feature space $\mathcal{H}$. It means that kernel methods implicitly perform the high-dimensional non-linear transformation ψ of input variables. The kernel function provides a distance measure in the feature space $\mathcal{H}$ of this nonlinear transform without the actual transformation, thus allowing nonlinear nonparametric learning, e.g., through Gaussian Processes or kernel-based Support Vector Machines. Despite enjoying excellent theoretical guarantees, the kernel methods are often considered to be limited in large-scale learning applications due to their demanding computational complexity.

Today, it is a common knowledge that, at least in their basic form, kernel methods incur space and time complexity infeasible to be used with large-scale datasets directly. For example, kernel regression has $\mathcal{O}(N^3 + Nd^2)$ training time, $\mathcal{O}(N^2)$ memory, $\mathcal{O}(Nd)$ prediction time complexity for N data points in original d-dimensional space $\mathcal{X}$.

One of the most successful techniques to handle this problem, known as Random Fourier Features (RFF) proposed by [174], introduces a low-dimensional randomized approximation to feature maps:

$$k(\mathbf{x}, \mathbf{y}) \approx \hat{\Psi}(\mathbf{x})^\top \hat{\Psi}(\mathbf{y}). \tag{2.2}$$

This is essentially carried out by using Monte-Carlo sampling to approximate scalar product in Equation (2.1). A randomized D-dimensional mapping $\hat{\Psi}(\cdot)$ applied to the original data input allows employing standard linear methods, i.e. reverting the kernel trick. In doing so, one reduces the complexity to that of linear methods, e.g. D-dimensional approximation admits $\mathcal{O}(ND^2)$ training time, $\mathcal{O}(ND)$ memory and $\mathcal{O}(N)$ prediction time.

It is well known that as $D \to \infty$, the inner product in Equation (2.2) converges to the exact kernel $k(\mathbf{x}, \mathbf{y})$. Since smaller dimension size D leads to an increased approximation error and a subsequent downstream performance decrease, recent research [236, 71, 43] aims to improve the convergence of approximation, which pursue distinct improvement techniques such as quasi-Monte Carlo points, orthogonality and structure in mapping matrices of $\hat{\Psi}$. We propose a unifying perspective towards all of the existing methods in regard to kernel approximation through *quadrature* rules.

Most attention in the field has been drawn to the kernels that allow the following integral representation:

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= \mathbb{E}_{p(\mathbf{w})} f_{\mathbf{xy}}(\mathbf{w}) = I(f_{\mathbf{xy}}), \\ p(\mathbf{w}) &= \frac{1}{(2\pi)^{d/2}} e^{-\frac{\|\mathbf{w}\|^2}{2}}, \\ f_{\mathbf{xy}}(\mathbf{w}) &= \phi(\mathbf{w}^\top \mathbf{x})^\top \phi(\mathbf{w}^\top \mathbf{y}). \end{aligned} \tag{2.3}$$

For example, the Gaussian kernel admits such representation with $\phi(\cdot) = \begin{bmatrix} \cos(\cdot) & \sin(\cdot) \end{bmatrix}^\top$. Generally, the class of kernels admitting the form in Equation (2.3) covers shift-invariant kernels (e.g. radial basis function (RBF) kernels) and Pointwise Nonlinear Gaussian (PNG) kernels. They are widely used in practice and have interesting connections with neural networks [42, 229].

The main challenge for the construction of low-dimensional feature maps is the approximation of the expectation in Equation (2.3) which is d-dimensional integral with Gaussian weight. Unlike other research studies we refrain from using simple Monte Carlo estimate of the integral, instead, we propose to use specific quadrature rules.

We summarize the contributions of this Chapter as follows:

1. We propose to use spherical-radial quadrature rules to improve kernel approximation accuracy and generalise the RFF-based techniques.
2. We use structured orthogonal matrices (so-called *butterfly matrices*) when designing quadrature rule that allow fast matrix by vector multiplications. As a result, we speed up the approximation of the kernel function and reduce memory requirements.
3. We carry out an extensive empirical study comparing our methods with the state-of-the-art ones on a set of different kernels in terms of both kernel approximation error and downstream tasks performance. The study supports our hypothesis on the exceeding accuracy of the method.

2.2 Related Work

The most popular methods for scaling up kernel methods are based on a low-rank approximation of the kernel using either data-dependent or independent basis functions. For example, in Random Fourier Features, the basis functions (i.e., sine and cosine functions) are sampled from a distribution independent of the training data, meanwhile, Nyström method samples basis functions from training examples, rendering the method data-dependent.

Data-dependent approach

The first one includes Nyström method [63], greedy basis selection techniques [197], incomplete Cholesky decomposition [72], a combination of stochastic subsampling, iterative solvers and preconditioning for approximating kernel ridge regression [183]. The construction of basis functions in these techniques utilizes the given training set making them more attractive for some problems compared to Random Fourier Features approach. In general, data-dependent approaches perform better than data-independent approaches when there is a gap in the eigen-spectrum of the kernel matrix. The rigorous study of generalisation performance of both approaches can be found in [238].

Data-independent approach

In data-independent techniques, the kernel function is approximated directly. Most of the methods (including the proposed approach) that follow this idea are based on Random Fourier Features [174]. They require so-called weight matrix that can be generated in a number of ways.

[126] form the weight matrix as a product of structured matrices. It enables fast computation of matrix-vector products and speeds up generation of random features.

Another work [71] orthogonalizes the features by means of orthogonal weight matrix. This leads to less correlated and more informative features increasing the quality of approximation. They support this result both analytically and empirically. The authors also introduce matrices with some special structure for fast computations. [44] propose a generalisation of the ideas from [126] and [71], delivering an analytical estimate for the mean squared error (MSE) of approximation.

All these works use simple Monte Carlo sampling. However, the convergence can be improved by changing Monte Carlo sampling to quasi-Monte Carlo sampling. Following this idea [236] apply quasi-Monte Carlo to Random Fourier Features. In [242] the authors make attempt to improve quality of the approximation of Random Fourier Features by optimizing sequences conditioning on a given dataset.

Among the recent papers there are works that, similar to our approach, use the numerical integration methods to approximate kernels. While [18] carefully inspects the connection between random features and quadratures, they did not provide any practically useful explicit mappings for kernels. Leveraging the connection [54] propose several methods with Gaussian quadratures. Among them three schemes are data-independent and one is data-dependent. The authors do not compare them with the approaches for random feature generation other than random Fourier features. The data-dependent scheme optimizes the weights for the quadrature points to yield better performance. A closely related work [138] constructs features for kernel approximation by approximating spherical-radial integral and designs QMC points to speed up approximation and reduce memory.

2.3 Quadrature Rules and Random Features

We start with rewriting the expectation in Equation (2.3) as integral of $f_{\mathbf{xy}}$ with respect to $p(\mathbf{W})$:

$$I(f_{\mathbf{xy}}) = (2\pi)^{-\frac{d}{2}} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{-\frac{\mathbf{w}^\top \mathbf{w}}{2}} f_{\mathbf{xy}}(\mathbf{W}) d\mathbf{W}.$$

Integration can be performed by means of quadrature rules. The rules usually take a form of interpolating function that is easy to integrate. Given such a rule, one may sample points from the domain of integration and calculate the value of the rule at these points. Then, the sample average of the rule values would yield the approximation of the integral.

The connection between integral approximation and mapping ψ is straightforward. In what follows we show a brief derivation of the quadrature rules that allow for an explicit mapping of the form:

$$\psi(\mathbf{x}) = [\ a_0\phi(0)\ \ a_1\phi(\mathbf{W}_1^\top \mathbf{x})\ \ \ldots\ \ a_D\phi(\mathbf{W}_D^\top \mathbf{x})\ ], \tag{2.4}$$

where the choice of the weights a_i and the points $\mathbf{W}_i$ is dictated by the quadrature.

We use the average of sampled quadrature rules developed by [77] to yield unbiased estimates of $I(f_{\mathbf{xy}})$. A change of coordinates is the first step to facilitate stochastic spherical-radial rules. Now, let $\mathbf{W} = r\mathbf{z}$, with $\mathbf{z}^\top \mathbf{z} = 1$, so that $\mathbf{W}^\top \mathbf{W} = r^2$ for $r \in [0, \infty]$, leaving us with (to ease the notation we substitute $f_{\mathbf{xy}}$ with f)

$$I(f) = (2\pi)^{-\frac{d}{2}} \int_{U_d} \int_0^\infty e^{-\frac{r^2}{2}} r^{d-1} f(r\mathbf{z}) dr d\mathbf{z} = \frac{(2\pi)^{-\frac{d}{2}}}{2} \int_{U_d} \int_{-\infty}^\infty e^{-\frac{r^2}{2}} |r|^{d-1} f(r\mathbf{z}) dr d\mathbf{z}, \tag{2.5}$$

$I(f)$ is now a double integral over the unit d-sphere $U_d = \{\mathbf{z} : \mathbf{z}^\top \mathbf{z} = 1, \mathbf{z} \in \mathbb{R}^d\}$ and over the radius. To account for both integration regions we apply a combination of spherical (S) and radial (R) rules known as spherical-radial (SR) rules. To provide an intuition how the rules work, here we briefly state and discuss their form[1].

Stochastic radial rules of degree $2l+1$ have the form of the weighted symmetric sums $R(h) = \sum_{i=0}^{l} \hat{w}_i \frac{h(\rho_i)+h(-\rho_i)}{2}$ and approximate the infinite range integral

$$T(h) = \int_{-\infty}^\infty e^{-\frac{r^2}{2}} |r|^{d-1} h(r) dr. \tag{2.6}$$

Note that when h is set to the function f of interest, $T(f)$ corresponds to the inner integral in Equation (2.5). To get an unbiased estimate for $T(h)$, points ρ_i are sampled from specific distributions. The weights $\hat{w}_i$ are derived so that the rule is exact for polynomials of degree $2l+1$ and give unbiased estimate for other functions.

Stochastic spherical rules have a form of a weighted sum $S_{\mathbf{Q}}(s) = \sum_{j=1}^{p} \widetilde{w}_j s(\mathbf{Q}\mathbf{z}_j)$, where $\mathbf{Q}$ is a random orthogonal matrix, approximate an integral of a function $s(\mathbf{z})$ over the surface of unit d-sphere U_d and $\mathbf{z}_j$ are points on U_d, i.e. $\mathbf{z}_j^\top \mathbf{z}_j = 1$. Remember that the outer integral in Equation (2.5) has U_d as its integration region. The weights $\widetilde{w}_j$ are stochastic with distribution such that the rule is exact for polynomials of degree p and gives unbiased estimate for other functions.

[1]Please see [77] for detailed derivation of the stochastic radial (section 2), spherical (section 3) and spherical radial rules (section 4)

Stochastic spherical-radial rules SR of degree $(2l+1, p)$ are given by the following expression

$$SR^{(2l+2,p)}_{\mathbf{Q},\rho} = \sum_{j=1}^{p} \widetilde{w}_j \sum_{i=1}^{l} \hat{w}_i \frac{f(\rho\mathbf{Q}\mathbf{z}_i) + f(-\rho\mathbf{Q}\mathbf{z}_i)}{2},$$

where the distributions of weights are such that if degrees of radial rules and spherical rules coincide, i.e. $2l + 1 = p$, then the rule is exact for polynomials of degree $2l + 1$ and gives unbiased estimate of the integral for other functions.

2.3.1 Spherical-Radial Rules of Degree (1,1) is RFF

If we take radial rule of degree 1 and spherical rule of degree 1, we arrive at the following rule

$$SR^{(1,1)}_{\mathbf{Q},\rho} = \frac{f(\rho\mathbf{Q}\mathbf{z}) + f(-\rho\mathbf{Q}\mathbf{z})}{2},$$

where $\rho \sim \chi(d)$. It is easy to see that $\rho\mathbf{Q}\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$. For shift invariant kernel $f(\mathbf{W}) = f(-\mathbf{W})$, thus, the rule reduces to $SR^{(1,1)}_{\mathbf{Q},\rho} = f(\mathbf{W})$, where $\mathbf{W} \sim \mathcal{N}(0, \mathbf{I})$.

Now, RFF [174] makes approximation of the RBF kernel in exactly the same way: it generates random vector from Gaussian distribution and calculates the corresponding feature map.

Proposition 2.3.1. *Random Fourier Features for RBF kernel are SR rules of degree* $(1, 1)$.

2.3.2 Spherical-Radial Rules of Degree (1,3) is ORF

Now, let's take radial rule of degree 1 and spherical rule of degree 3. In this case we get the following spherical-radial rule

$$SR^{1,3}_{\mathbf{Q},\rho} = \sum_{i=1}^{d} \frac{f(\rho\mathbf{Q}\mathbf{e}_i) + f(-\rho\mathbf{Q}\mathbf{e}_i)}{2},$$

where $\rho \sim \chi(d)$, $\mathbf{Q}$ is a random orthogonal matrix, $\mathbf{e}_i = (0, \dots, 0, 1, 0, \dots, 0)^\top$ is an i-th column of the identity matrix.

Let us compare SR^{1,3} rules with Orthogonal Random Features [71] for the RBF kernel. In the ORF approach, the weight matrix $\mathbf{W} = \mathbf{S}\mathbf{Q}$ is generated, where $\mathbf{S}$ is a diagonal matrix with the entries drawn independently from $\chi(d)$ distribution and $\mathbf{Q}$ is a random orthogonal matrix. The approximation of the kernel is then given by $k_{\text{ORF}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} f(\mathbf{W}_i)$, where $\mathbf{W}_i$ is the i-th row of the matrix $\mathbf{W}$. As the rows of $\mathbf{Q}$ are orthonormal, they can be represented as $\mathbf{Q}\mathbf{e}_i$.

Proposition 2.3.2. *Orthogonal Random Features for RBF kernel are SR rules of degree* $(1, 3)$.

2.3.3 Spherical-Radial Rules of Degree (3,3)

We go further and take both spherical and radial rules of degree 3, where we use original and reflected vertices $\mathbf{v}_j$ of randomly rotated unit vertex regular d-simplex $\mathbf{v}$ as the points on the unit sphere

$$SR^{3,3}_{\mathbf{Q},\rho}(f) = \left(1 - \frac{d}{\rho^2}\right) f(\mathbf{0}) + \frac{d}{d+1} \sum_{j=1}^{d+1} \left[\frac{f(-\rho\mathbf{Q}\mathbf{v}_j) + f(\rho\mathbf{Q}\mathbf{v}_j)}{2\rho^2}\right], \tag{2.7}$$

where $\rho \sim \chi(d+2)$. We apply quadrature in Equation (2.7) to the approximation of the integral in Equation (2.5) by averaging the samples of $SR^{3,3}_{\mathbf{Q},\rho}$:

$$I(f) = \mathbb{E}_{\mathbf{Q},\rho}[SR^{3,3}_{\mathbf{Q},\rho}(f)] \approx \hat{I}(f) = \frac{1}{n}\sum_{i=1}^{n} SR^{3,3}_{\mathbf{Q}_i,\rho_i}(f), \tag{2.8}$$

where n is the number of sampled SR rules. Speaking in terms of the approximate feature maps, the new feature dimension D in case of the quadrature based approximation equals $2n(d+1)+1$ as we sample n rules and evaluate each of them at $2(d+1)$ random points and 1 zero point.

We propose to modify the quadrature by generating $\rho_j \sim \chi(d+2)$ for each $\mathbf{v}_j$:

$$SR^{3,3}_{\mathbf{Q},\rho}(f) = \left(1 - \sum_{j=1}^{d+1} \frac{d}{(d+1)\rho_j^2}\right) f(\mathbf{0}) + \frac{d}{d+1} \sum_{j=1}^{d+1} \left[\frac{f(-\rho_j\mathbf{Q}\mathbf{v}_j) + f(\rho_j\mathbf{Q}\mathbf{v}_j)}{2\rho_j^2}\right]. \tag{2.9}$$

While the formulation in Equation (2.9) does not affect the quality of approximation, it does simplify an analysis of the quadrature-based random features.

Explicit Mapping We finally arrive at the map

$$\psi(\mathbf{x}) = [\ a_0\phi(0)\ \ a_1\phi(\mathbf{W}_1^\top\mathbf{x})\ \ \dots\ \ a_D\phi(\mathbf{W}_D^\top\mathbf{x})\ ], \tag{2.10}$$

where

$$a_0 = \sqrt{1 - \sum_{d+1}^{j=1} \frac{d}{\rho^2}}^{[2]}, \quad a_j = \frac{1}{\rho_j}\sqrt{\frac{d}{2(d+1)}},$$

[2]To get $a_0^2 \geq 0$, you need to sample ρ_j two times on average (see Supplementary Materials for details).

and $\mathbf{W}_j$ is the j-th row in the matrix $\mathbf{W} = \rho \otimes \begin{bmatrix} (\mathbf{Q}\mathbf{v})^\top \\ -(\mathbf{Q}\mathbf{v})^\top \end{bmatrix}$, $\rho = [\rho_1 \dots \rho_D]^\top$. To get D features one simply stacks $n = {}^{D}/_{2(d+1)+1}$ matrices

$$\mathbf{W}^k = \rho^k \begin{bmatrix} (\mathbf{Q}^k\mathbf{v})^\top \\ -\ (\mathbf{Q}^k\mathbf{v})^\top \end{bmatrix}, \tag{2.11}$$

to get $\mathbf{W} \in \mathbb{R}^{D\times d}$, where only $\mathbf{Q}^k \in \mathbb{R}^{d\times d}$ and ρ^k are generated randomly ($k = 1, \dots, n$). For Gaussian kernel, $\phi(\cdot) = \begin{bmatrix}\cos(\cdot) & \sin(\cdot)\end{bmatrix}^\top$. For the 0-order arc-cosine kernel, $\phi(\cdot) = \Theta(\cdot)$, where $\Theta(\cdot)$ is the Heaviside function. For the 1-order arc-cosine kernel, $\phi(\cdot) = \max(0, \cdot)$. We call this novel feature map method Quadrature-based Features (QF). Below is the pseudo code for explicit mapping with QF.

Algorithm 1 Quadrature-based Features Explicit Mapping Algorithm.

```
function QF($\mathbf{X}, d, n$)
    $\mathbf{V} \leftarrow$ simplex($d$)
    $\rho$s $\leftarrow$ sample($\chi(d+2), n$)
    $\mathbf{Q}$s $\leftarrow$ random_orthogonal_matrix($d, n$)    ▷ see Section 2.3.4
    $\mathbf{W} \leftarrow$ weight_matrix($\mathbf{V}, \rho$s$, \mathbf{Q}$s)        ▷ from Equation (2.11)
    return $\psi(\mathbf{X}, \rho$s$, \mathbf{W})$                 ▷ from Equation (2.10)
```

2.3.4 Generating Uniformly Random Orthogonal Matrices

The SR rules require a random orthogonal matrix $\mathbf{Q}$. If $\mathbf{Q}$ follows Haar distribution, the averaged samples of $SR^{3,3}_{\mathbf{Q},\rho}$ rules provide an unbiased estimate for the integral in Equation (2.5). Essentially, Haar distribution means that all orthogonal matrices in the group are equiprobable, i.e. uniformly random. Methods for sampling such matrices vary in their complexity of generation and multiplication. Below, we review them in generality.

Gram-Schmidt process. QR decomposition with a little tweak [144] on the random matrix with entries $\{x_{ij}\} \sim \mathcal{N}(0, 1)$ is the most straight forward way to obtain the required matrix, however its complexity is cubic in d and the resulting matrix has no special structure to allow fast matrix by vector multiplication.

Sequence of reflectors. A sequence of random reflectors [199] or rotations [10] is a way to get uniformly random $\mathbf{Q}$ from the orthogonal group $\mathrm{O}(d)$. The method has a better generation complexity — quadratic in d — in the form of a product of transformations, e.g. reflections:

$\mathbf{Q} = \mathbf{H}_1 \dots \mathbf{H}_{n-1}\mathbf{D}$, where $\mathbf{H}_i$ is a random Householder matrix $I - 2\frac{\mathbf{x}_i\mathbf{x}_i^\top}{\|\mathbf{x}_i\|^2}$ with Householder vectors $\mathbf{x}_i \sim \mathcal{N}(0, \mathbf{I})$ sampled from standard Gaussian distribution, and diagonal matrix $\mathbf{D}$ with entries d_{ii} sampled from Rademacher distribution, i.e. $\mathbb{P}(d_{ii} = \pm 1) = 1/2$, but it implicates no fast matrix multiplication. We test the method with Householder reflections, denoted $\mathbf{H}$ in Section 2.6.

Sparse and diagonal blocks product. Random orthogonal matrices (ROM) [44] generalise the idea of using a product of diagonal and structured orthogonal matrices to reduce computational complexity and approximation error: $\mathbf{Q} = \sqrt{d}\prod_{i=1}^{3} \mathbf{S}\mathbf{D}_i$, where matrix $\mathbf{D}$ is a diagonal matrix with i.i.d. Rademacher random variables on the diagonal (same as above). An instantiation of structured orthogonal matrix $\mathbf{S}$ is a normalized Hadamard matrix $\mathbf{H}_d$, defined recursively with the base case $\mathbf{H}_1 = 1$:

$$\mathbf{H}_i = \frac{1}{\sqrt{2}}\begin{bmatrix} \mathbf{H}_{i-1} & \mathbf{H}_{i-1} \\ \mathbf{H}_{i-1} & -\mathbf{H}_{i-1} \end{bmatrix}.$$

Thanks to the structured nature of their factors, ROM enable fast matrix by vector products. Unfortunately, they are not guaranteed to follow the Haar distribution.

Orthogonal matrices in Quadrature-based Features.

For QF, we test two algorithms for obtaining $\mathbf{Q}$. The first uses a QR decomposition of a random matrix to obtain a product of a sequence of reflectors/rotators $\mathbf{Q} = \mathbf{H}_1 \dots \mathbf{H}_{n-1}\mathbf{D}$, where $\mathbf{H}_i$ is a random Householder/Givens matrix and a diagonal matrix $\mathbf{D}$ has entries such that $\mathbb{P}(d_{ii} = \pm 1) = 1/2$. It implicates no fast matrix multiplication. We test both methods for random orthogonal matrix generation and, since their performance coincides, we leave this one out for cleaner figures in the Section 2.6.

The other choice for $\mathbf{Q}$ are so-called *butterfly matrices* [76]. An example for $d = 4$:

$$\mathbf{B}^{(4)} = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & c_3 & -s_3 \\ 0 & 0 & s_3 & c_3 \end{bmatrix}\begin{bmatrix} c_2 & 0 & -s_2 & 0 \\ 0 & c_2 & 0 & -s_2 \\ s_2 & 0 & c_2 & 0 \\ 0 & s_2 & 0 & c_2 \end{bmatrix} = \begin{bmatrix} c_1c_2 & -s_1c_2 & -c_1s_2 & s_1s_2 \\ s_1c_2 & c_1c_2 & -s_1s_2 & -c_1s_2 \\ c_3s_2 & -s_3s_2 & c_3c_2 & -s_3c_2 \\ s_3s_2 & c_3s_2 & s_3c_2 & c_3c_2 \end{bmatrix},$$

where s_i, c_i is sine and cosine, respectively, of some angle θ_i, $i = 1, \dots, d-1$. A thorough definition is given below.

Definition 1. *Let $c_i = \cos\theta_i$, $s_i = \sin\theta_i$ for $i = 1, \dots, d-1$ be given. Assume $d = 2^k$ with $k > 0$. Then an orthogonal matrix $\mathbf{B}^{(d)} \in \mathbb{R}^{d\times d}$ is defined recursively as follows*

$$\mathbf{B}^{(2d)} = \begin{bmatrix} \mathbf{B}^{(d)}c_d & -\mathbf{B}^{(d)}s_d \\ \hat{\mathbf{B}}^{(d)}s_d & \hat{\mathbf{B}}^{(d)}c_d \end{bmatrix}, \quad \mathbf{B}^{(1)} = 1,$$

where $\hat{\mathbf{B}}^{(d)}$ is the same as $\mathbf{B}^{(d)}$ with indexes i shifted by d, e.g.

$$\mathbf{B}^{(2)} = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix}, \quad \hat{\mathbf{B}}^{(2)} = \begin{bmatrix} c_3 & -s_3 \\ s_3 & c_3 \end{bmatrix}.$$

The factors of $\mathbf{B}^{(d)}$ are structured and allow fast matrix multiplication. Specifically, matrix by vector products with $\mathbf{B}^{(d)}$ have computational complexity $O(d\log d)$ since $\mathbf{B}^{(d)}$ has $\lceil \log d \rceil$ factors and each factor requires $O(d)$ operations. Another advantage is space complexity: $\mathbf{B}^{(d)}$ is fully determined by $d-1$ angles θ_i, yielding $O(d)$ memory requirement.

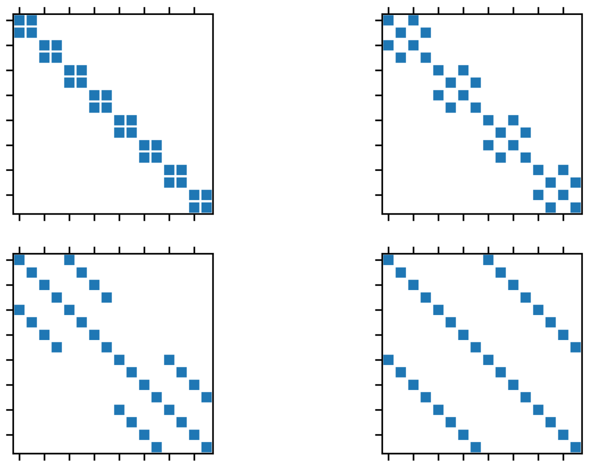


Fig. 2.1 Visual representaion of butterfly orthogonal matrix factors for $d = 16$

The randomization is based on the sampling of angles θ. We follow the generation algorithm in [70] that first computes a uniform random point $\mathbf{u}$ from U_d. It then calculates the angles by taking the ratios of the appropriate $\mathbf{u}$ coordinates $\theta_i = \frac{u_i}{u_{i+1}}$, followed by computing cosines and sines of the θ's. One can easily define butterfly matrix $\mathbf{B}^{(d)}$ for the cases when d is not a power of 2.

Let the number of butterfly factors $k = \lceil \log d \rceil$. Then $\mathbf{B}^{(d)}$ is constructed as a product of k factor matrices of size $d \times d$ obtained from k matrices used for generating $\mathbf{B}^{(2^k)}$. For each matrix in the product for $\mathbf{B}^{(2^k)}$, we delete the last $2^k - d$ rows and columns. We then replace with 1 every c_i in the remaining $d \times d$ matrix that is in the same column as deleted s_i.

When d is not a power of two, the resulting $\mathbf{B}$ has deficient columns with zeros (Figure 2.2, right), which introduces a bias to the integral estimate. To correct for this bias one may apply additional randomization by using a product $\mathbf{BP}$, where $\mathbf{P} \in \{0,1\}^{d\times d}$ is a permutation matrix. Even better, use a product of several $\mathbf{BP}$'s: $\widetilde{\mathbf{B}} = (\mathbf{BP})_1(\mathbf{BP})_2 \dots (\mathbf{BP})_t$. We set $t = 3$ in the experiments. The method using butterfly matrices is denoted by $\mathbf{B}$ in the Section 2.6.

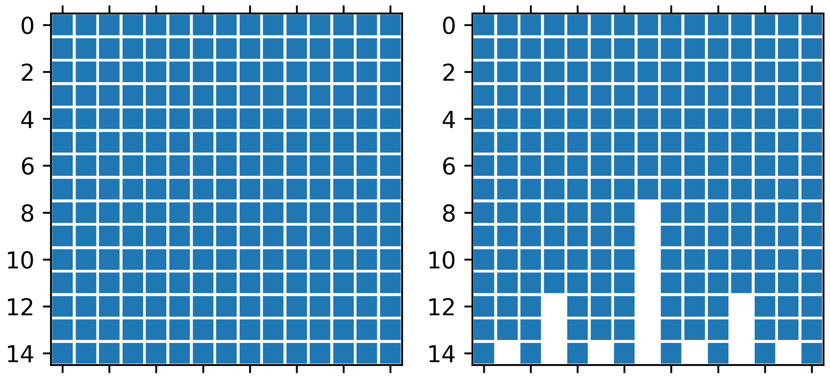


Fig. 2.2 Sparsity pattern for **BPBPBP** (left) and **B** (right), $d = 15$

2.4 Error Bounds

For the completeness of the exposition of the method, below we discuss the error bounds on the kernel approximation error with Quadrature-based Features followed by a bound on a downstream error in kernel ridge regression.

Proposition 2.4.1. *Let l be a diameter of the compact set $\mathcal{X}$ and $p(\mathbf{w}) = \mathcal{N}(0, \sigma_p^2\mathbf{I})$ be the probability density corresponding to the kernel. Let us suppose that $|\phi(\mathbf{w}^\top\mathbf{x})| \le \kappa$, $|\phi'(\mathbf{w}^\top\mathbf{x})| \le \mu$ for all $\mathbf{w} \in \Omega$, $\mathbf{x} \in \mathcal{X}$ and $\left|\frac{1-f_{\mathbf{xy}}(\rho\mathbf{z})}{\rho^2}\right| \le M$ for all $\rho \in [0,\infty)$, where $\mathbf{z}^\top\mathbf{z} = 1$. Then for Quadrature-based Features approximation $\hat{k}(\mathbf{x},\mathbf{y})$ of the kernel function $k(\mathbf{x},\mathbf{y})$ and any $\varepsilon > 0$ it holds*

$$\mathbb{P}\left(\sup_{\mathbf{x},\mathbf{y}\in\mathcal{X}} |\hat{k}(\mathbf{x},\mathbf{y}) - k(\mathbf{x},\mathbf{y})| \ge \varepsilon\right) \le \beta_d \left(\frac{\sigma_p l \kappa \mu}{\varepsilon}\right)^{\frac{2d}{d+1}} \exp\left(-\frac{D\varepsilon^2}{8M^2(d+1)}\right),$$

where $\beta_d = \left(d^{\frac{-d}{d+1}} + d^{\frac{1}{d+1}}\right) 2^{\frac{6d+1}{d+1}} \left(\frac{d}{d+1}\right)^{\frac{d}{d+1}}$. *Thus we can construct approximation with error no more than* ε *with probability at least* $1 - \delta$ *as long as the number of features*

$$D \geq \frac{8M^2(d+1)}{\varepsilon^2}\left[\frac{2}{1+\frac{1}{d}} \log \frac{\sigma_p l \kappa \mu}{\varepsilon} + \log \frac{\beta_d}{\delta}\right].$$

The proof of this proposition closely follows [203], we defer the derivations to the Appendix A.1.

Term β_d depends on dimension d, its maximum is $\beta_{86} \approx 64.7 < 65$, and $\lim_{d\to\infty} \beta_d = 64$, though it is lower for small d. Let us compare this probability bound with the similar result for RFF in [203]. Under the same conditions the required number of samples to achieve error no more than ε with probability at least $1 - \delta$ for RFF is the following

$$D \geq \frac{8(d+1)}{\varepsilon^2}\left[\frac{2}{1+\frac{1}{d}} \log \frac{\sigma_p l}{\varepsilon} + \log \frac{\beta_d}{\delta} + \frac{d}{d+1} \log \frac{3d+3}{2d}\right].$$

QF for RBF kernel $M = \frac{1}{2}, \kappa = \mu = 1$, therefore, we obtain

$$D \geq \frac{2(d+1)}{\varepsilon^2}\left[\frac{2}{1+\frac{1}{d}} \log \frac{\sigma_p l}{\varepsilon} + \log \frac{\beta_d}{\delta}\right].$$

The asymptotics are the same, however, the constants are smaller for our approach. See Section 2.6 for empirical justification of the obtained result.

We now look at the error in terms of a downstream task of kernel ridge regression leveraging a known result from [203].

Proposition 2.4.2 ([203]). *Given a training set* $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, *with* $\mathbf{x}_i \in \mathbb{R}^d$ *and* $y_i \in \mathbb{R}$, *let* $h(\mathbf{x})$ *denote the result of kernel ridge regression using the positive semi-definite training kernel matrix* $\mathbf{K}$, *test kernel values* $\mathbf{k}_\mathbf{x}$ *and regularization parameter* λ. *Let* $\hat{h}(\mathbf{x})$ *be the same using a PSD approximation to the training kernel matrix* $\widehat{\mathbf{K}}$ *and test kernel values* $\hat{\mathbf{k}}_\mathbf{x}$. *Further, assume that the training labels are centered,* $\sum_{i=1}^n y_i = 0$, *and let* $\sigma_y^2 = \frac{1}{n}\sum_{i=1}^n y_i^2$. *Also suppose* $||\mathbf{k}_\mathbf{x}||_\infty \leq \kappa$. *Then*

$$|\hat{h}(\mathbf{x}) - h(\mathbf{x})| \leq \frac{\sigma_y\sqrt{n}}{\lambda}||\hat{\mathbf{k}}_\mathbf{x} - \mathbf{k}_\mathbf{x}||_2 + \frac{\kappa\sigma_y n}{\lambda^2}||\widehat{\mathbf{K}} - \mathbf{K}||_2.$$

Suppose that $\sup |k(\mathbf{x},\mathbf{x}') - \hat{k}(\mathbf{x},\mathbf{x}')| \le \varepsilon$ for all $\mathbf{x},\mathbf{x}' \in \mathbb{R}^d$. Then $||\hat{\mathbf{k}}_\mathbf{x} - \mathbf{k}_\mathbf{x}||_2 \le \sqrt{n}\varepsilon$ and $||\widehat{\mathbf{K}} - \mathbf{K}||_2 \le ||\widehat{\mathbf{K}} - \mathbf{K}||_F \le n\varepsilon$. By denoting $\lambda = n\lambda_0$ we obtain $|\hat{h}(\mathbf{x}) - h(\mathbf{x})| \le \frac{\lambda_0+1}{\lambda_0^2}\sigma_y\varepsilon$. Therefore,

$$\mathbb{P}\left(\ |\hat{h}(\mathbf{x}) - h(\mathbf{x})| \ge \varepsilon\right) \le \mathbb{P}\left(||\hat{k}(\mathbf{x},\mathbf{x}') - k(\mathbf{x},\mathbf{x}')||_\infty \ge \frac{\lambda_0^2\varepsilon}{\sigma_y(\lambda_0+1)}\right).$$

So, for the quadrature rules we can guarantee $|\hat{h}(\mathbf{x}) - h(\mathbf{x})| \le \varepsilon$ with probability at least $1-\delta$ as long as the feature dimension

$$D \ge 8M^2(d+1)\sigma_y^2\left(\frac{\lambda_0+1}{\lambda_0^2\varepsilon}\right)^2\left[\frac{2}{1+\frac{1}{d}}\log\frac{\sigma_y\sigma_p l\kappa\mu(\lambda_0+1)}{\lambda_0^2\varepsilon} + \log\frac{\beta_d}{\delta}\right].$$

2.5 Arc-cosine Kernels

In this section, we extend our method to another class of kernels which have a particularly simple approximation form via Spherical-Radial rules. Arc-cosine kernels were originally introduced by [42] upon studying the connections between deep learning and kernel methods. The integral representation of the b^{th}-order arc-cosine kernel is

$$k_b(\mathbf{x},\mathbf{y}) = 2\int_{\mathbb{R}^n} \Theta(\mathbf{W}^\top\mathbf{x})\Theta(\mathbf{W}^\top\mathbf{y})(\mathbf{W}^\top\mathbf{x})^b(\mathbf{W}^\top\mathbf{y})^b p(\mathbf{W})d\mathbf{W},$$

$$k_b(\mathbf{x},\mathbf{y}) = 2\int_{\mathbb{R}^d} \phi_b(\mathbf{W}^\top\mathbf{x})\phi_b(\mathbf{W}^\top\mathbf{y})p(\mathbf{W})d\mathbf{W},$$

where $\phi_b(\mathbf{W}^\top\mathbf{x}) = \Theta(\mathbf{W}^\top\mathbf{x})(\mathbf{W}^\top\mathbf{x})^b$, $\Theta(\cdot)$ is the Heaviside function and p is the density of the standard Gaussian distribution. Such kernels can be seen as an inner product between the representation produced by infinitely wide single layer neural network with random Gaussian weights. They have closed form expressions in terms of the angle $\theta = \cos^{-1}\left(\frac{\mathbf{x}^\top\mathbf{y}}{||\mathbf{x}||||\mathbf{y}||}\right)$ between $\mathbf{x}$ and $\mathbf{y}$.

Arc-cosine kernel of 0^{th}-order shares the property of mapping the input on the unit hypersphere with RBF kernels, while order 1 arc-cosine kernel preserves the norm as linear kernel (Gram matrix on original features):

These expressions for 0^{th}-order and 1^{st}-order arc-cosine kernels are given by

$$k_0(\mathbf{x},\mathbf{y}) = 1 - \frac{\theta}{\pi}, \qquad k_1(\mathbf{x},\mathbf{y}) = \frac{||\mathbf{x}||||\mathbf{y}||}{\pi}(\sin\theta + (\pi-\theta)\cos\theta).$$

The 0-order arc-cosine kernel is given by $k_0(\mathbf{x}, \mathbf{y}) = 1 - \frac{\theta}{\pi}$, the 1-order kernel is given by $k_1(\mathbf{x}, \mathbf{y}) = \frac{\|\mathbf{x}\|\|\mathbf{y}\|}{\pi}(\sin\theta + (\pi - \theta)\cos\theta)$.

Let $\phi_0(\mathbf{W}^\top\mathbf{x}) = \Theta(\mathbf{W}^\top\mathbf{x})$ and $\phi_1(\mathbf{W}^\top\mathbf{x}) = \max(0, \mathbf{W}^\top\mathbf{x})$. We now can rewrite the integral representation as follows:

$$k_b(\mathbf{x}, \mathbf{y}) = 2\int\limits_{\mathbb{R}^d} \phi_b(\mathbf{W}^\top\mathbf{x})\phi_b(\mathbf{W}^\top\mathbf{y})p(\mathbf{W})d\mathbf{W} \approx \frac{2}{n}\sum_{i=1}^{n} SR^{3,3}_{\mathbf{Q}_i,\rho_i}.$$

For arc-cosine kernel of order 0 the value of the function $\phi_0(0) = \Theta(0) = 0.5$ results in

$$SR^{3,3}_{\mathbf{Q},\rho}(f) = 0.25\left(1 - \sum_{j=1}^{d+1}\frac{d}{(d+1)\rho_j^2}\right) + \frac{d}{d+1}\sum_{j=1}^{d+1}\frac{f(\rho_j\mathbf{Q}\mathbf{v}_j) + f(-\rho_j\mathbf{Q}\mathbf{v}_j)}{2\rho^2}.$$

In the case of arc-cosine kernel of order 1, the value of $\phi_1(0)$ is 0 and the $SR^{3,3}$ rule reduces to

$$SR^{3,3}_{\mathbf{Q},\rho}(f) = \frac{d}{d+1}\sum_{j=1}^{d+1}\frac{f(|\rho\mathbf{Q}\mathbf{v}_j|)}{2\rho_j^2}.$$

2.6 Experiments

We extensively study the proposed method on several established benchmarking datasets: Powerplant, LETTER, USPS, MNIST, CIFAR100 [124], LEUKEMIA [88]. In Section 2.6.2 we show kernel approximation error across different kernels and number of features. We also report the quality of SVM models with approximate kernels on the same data sets in Section 2.6.3.

2.6.1 Methods

We present a comparison of QF (denoted **B** in experiments) with estimators based on a simple Monte Carlo, quasi-Monte Carlo [236] and Gaussian quadratures [54]. The Monte Carlo approach has a variety of ways to generate samples: unstructured Gaussian [174], structured Gaussian [71], random orthogonal matrices (ROM) [44].

Monte Carlo integration (G, Gort, ROM)
The kernel is estimated as $\hat{k}(\mathbf{x}, \mathbf{y}) = \frac{1}{D}\phi(\mathbf{M}\mathbf{x})\phi(\mathbf{M}\mathbf{y})$, where $\mathbf{M} \in \mathbb{R}^{D\times d}$ is a random weight matrix. For unstructured Gaussian based approximation $\mathbf{M} = \mathbf{G}$, where $\mathbf{G}_{ij} \sim \mathcal{N}(0, 1)$. Structured Gaussian has $\mathbf{M} = \mathbf{G}_{\text{ort}}$, where $\mathbf{G}_{\text{ort}} = \mathbf{D}\mathbf{Q}$, $\mathbf{Q}$ is obtained from RQ decomposition of $\mathbf{G}$, $\mathbf{D}$ is a diagonal matrix with diagonal elements sampled from the $\chi(d)$ distribution.

Dataset	N	d	**#samples**	**#runs**
Powerplant	9568	4	550	500
LETTER	20000	16	550	500
USPS	9298	256	550	500
MNIST	70000	784	550	100
CIFAR100	60000	3072	50	50
LEUKEMIA	72	7129	10	10

Table 2.1 Experimental settings (# of objects N, dimensionality d, # of samples and runs).

In compliance with the previous work on ROM we use **S**-Rademacher with three blocks: $\mathbf{M} = \sqrt{d}\prod_{i=1}^{3}\mathbf{SD}_i$, where $\mathbf{S}$ is a normalized Hadamard matrix and $\mathbb{P}(\mathbf{D}_{ii} = \pm 1) = 1/2$.

Quasi-Monte Carlo integration (QMC)

Quasi-Monte Carlo integration boasts improved rate of convergence $1/D$ compared to $1/\sqrt{D}$ of Monte Carlo, however, as empirical results illustrate its performance is poorer than that of orthogonal random features [71]. It has larger constant factor hidden under $\mathcal{O}$ notation in computational complexity. For QMC the weight matrix $\mathbf{M}$ is generated as a transformation of quasi-random sequences. We run our experiments with Halton sequences in compliance with the previous work.

Gaussian quadratures (GQ)

We included subsampled dense grid method from [54] into our comparison as it is the only data-independent approach from the paper that is shown to work well. We reimplemented code for the paper to the best of our knowledge as it is not open sourced.

Method	**Space**	**Time**
ORF	$\mathcal{O}(Dd)$	$\mathcal{O}(Dd)$
QMC	$\mathcal{O}(Dd)$	$\mathcal{O}(Dd)$
ROM	$\mathcal{O}(d)$	$\mathcal{O}(d\log d)$
QF	$\mathcal{O}(d)$	$\mathcal{O}(d\log d)$

Table 2.2 Computational complexity of different kernel approximation algorithms.

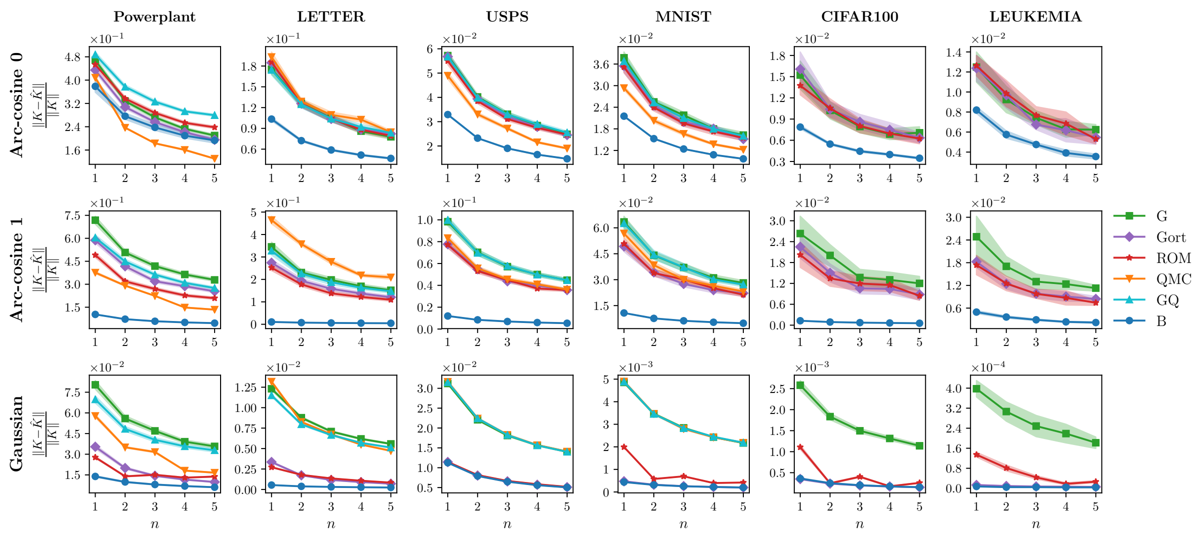


Fig. 2.3 Kernel approximation error across three kernels and 6 datasets. Lower is better. The x-axis represents the factor to which we extend the original feature space, $n = \frac{D}{2(d+1)+1}$, where d is the dimensionality of the original feature space, D is the dimensionality of the new feature space.

2.6.2 Kernel Approximation

To measure kernel approximation quality we use relative error in Frobenius norm $\frac{\|\mathbf{K}-\hat{\mathbf{K}}\|_F}{\|\mathbf{K}\|_F}$, where $\mathbf{K}$ and $\hat{\mathbf{K}}$ denote exact kernel matrix and its approximation. In line with previous work we run experiments for the kernel approximation on a random subset of a dataset. Table 2.1 displays the settings for the experiments across the datasets.

Approximation was constructed for different number of SR samples $n = \frac{D}{2(d+1)+1}$, where d is an original feature space dimensionality and D is the new one. For the Gaussian kernel we set hyperparameter $\gamma = \frac{1}{2\sigma^2}$ to the default value of $\frac{1}{d}$ for all the approximants, while the arc-cosine kernels have no hyperparameters.

We run experiments for each [kernel, dataset, n] tuple and plot 95% confidence interval around the mean value line. Figure 2.3 shows the results for kernel approximation error on LETTER, MNIST, CIFAR100 and LEUKEMIA datasets.

QMC method almost always coincides with RFF except for arc-cosine 0 kernel. It particularly enjoys Powerplant dataset with $d = 4$, i.e. small number of features. Possible explanation for such behaviour can be due to the connection with QMC quadratures. The worst case error for QMC quadratures scales with $n^{-1}(\log n)^d$, where d is the dimensionality and n is the number of sample points [162]. It is worth mentioning that for large d it is also a problem to construct a proper QMC point set. Thus, in higher dimensions QMC may bring little practical advantage over MC. While recent randomized QMC techniques indeed in some cases have no dependence

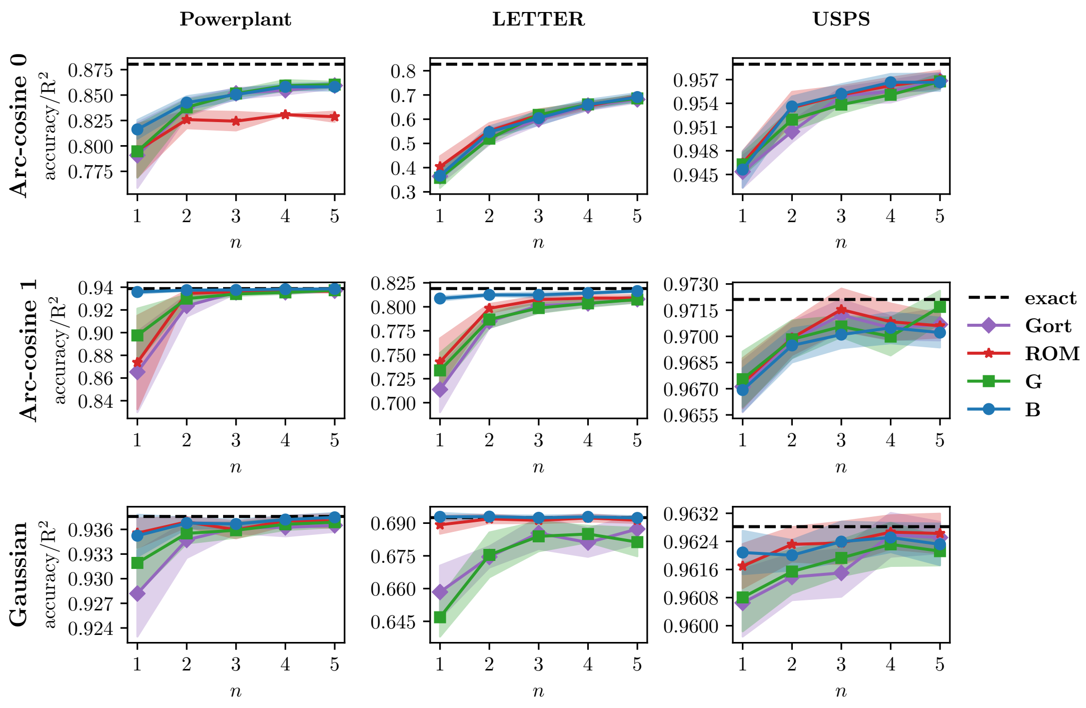


Fig. 2.4 Accuracy/R^2 score using embeddings with three kernels on 3 datasets. Higher is better. The x-axis represents the factor to which we extend the original feature space, $n = \frac{D}{2(d+1)+1}$.

on d, our approach is still computationally more efficient thanks to the structured matrices. GQ method as well matches the performance of RFF. We omit both QMC and GQ from experiments on datasets with large $d = [3072, 7129]$ (CIFAR100, LEUKEMIA).

The empirical results in Figure 2.3 support our hypothesis about the advantages of **SR** quadratures applied to kernel approximation compared to SOTA methods. With an exception of a couple of cases: (arc-cosine 0, Powerplant) and (Gaussian, USPS), our method displays clear exceeding performance.

2.6.3 Classification/Regression with New Features

We report accuracy and R^2 scores for the classification/regression tasks on some of the datasets (Figure 2.4). We examine the performance with the same setting as in experiments for kernel approximation error, except now we map the whole dataset. We use Support Vector Machines to obtain predictions.

Kernel approximation error does not fully define the final prediction accuracy — the best performing kernel matrix approximant not necessarily yields the best accuracy or R^2 score. However, the empirical results illustrate that our method delivers comparable and often superior quality on the downstream tasks.

2.6.4 Walltime Experiment

We measure time spent on explicit mapping of features by running each experiment 50 times and averaging the measurements. Indeed, Figure 2.5 demonstrates that the method scales as theoretically predicted with larger dimensions thanks to the structured nature of the mapping.

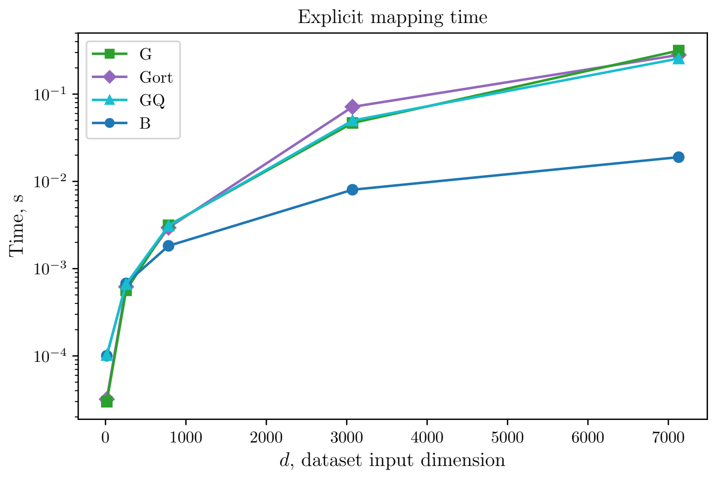


Fig. 2.5 Time spent on explicit mapping. The x-axis represents the 5 datasets with increasing input number of features: LETTER, USPS, MNIST, CIFAR100 and LEUKEMIA.

2.7 Summary and Future Work

We propose an approach for the random features methods for kernel approximation, Quadrature-based Features, revealing a new interpretation of RFF and ORF. The latter are special cases of the spherical-radial quadrature rules with degrees (1,1) and (1,3) respectively. We take this further and develop a more accurate technique for the random features preserving the time and space complexity of the random orthogonal embeddings.

Our experimental study confirms that for many kernels on the most datasets the proposed Quadrature-based Features deliver the best kernel approximation.

Additionally, the results showed that the quality of the downstream task (classification and regression) is also superior or comparable to the state-of-the-art baselines. Although there is not yet a clear established dependency between kernel approximation error and the downstream performance, QF shows favourable performance for a range of problems. This connection is left for a future work as it is not clear whether optimal methods in terms of approximation may lead to noise filtering for each particular task.

Since this work has been published there has been several papers leveraging their advantageous properties in other settings, e.g., in neural network design [168] and in data augmentation [55].

Chapter 3

Intrinsic Geometry for Sample Comparison

In the previous chapter, we demonstrated the utility of quadrature rules for the unifying perspective on kernel approximation methods, based on Random Fourier Features, and developed a practical algorithm for the sparse and fast feature mapping. In this chapter, we demonstrate how the other type of quadrature facilitates calculation of a lower bound on the distance between two data manifolds, based on intrinsic geometry of the data provided.

We start by noting some drawbacks of the existing methods for sample comparison, such as Fréchet Inception Distance [103], and discussing some of its applications, e.g., comparison of generative models, representations, tracking model evolution. We then establish theoretical background and develop a practical algorithm for Intrinsic Multi-scale Distance, which is based on the algorithms for symmetric eigenvalue problems and stochastic estimation.

3.1 Introduction

Machine learning models deal with various types of data, and generate even more data for their subsequent analysis. For example, when learning generative models, we might use their samples to choose the best model or decide when to stop training. Another example is representation comparison — we may need ways to compare representations that should be performant and generic enough for the downstream tasks. We might also study the changes of the models in training, to understand if there are any similarities or interesting behaviour. Oftentimes, we wish to compare data lying in entirely different spaces, for example to track model evolution or compare models having different representation space.

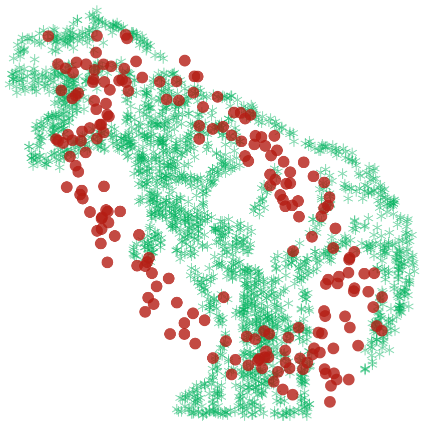


Fig. 3.1 Two distributions having the same first 3 moments, meaning Fréchet Distance and Kernel Distance (degree 3 polynomial kernel) scores are close to 0.

To enable the comparison of various learning artifacts (e.g., samples from generative models), we need an appropriate measure to quantify their distance or similarity. This problem has gained a lot of attention recently, especially in generative modelling. As generative models aim to reproduce the true data distribution $\mathbb{P}_d$ by means of the model distribution $\mathbb{P}_g(\mathbf{z};\Theta)$, delicate evaluation procedures are required.

In order to evaluate the performance of generative models, past research has proposed several *extrinsic* evaluation measures, most notably the Fréchet [103] and Kernel [27] Inception Distances (FID and KID). These measures only reflect the first two or three moments of distributions, meaning they can be insensitive to global structural problems. We showcase this inadvertence in Figure 3.1: here FID and KID are insensitive to the global structure of the data distribution. Besides, as FID and KID are based only on *extrinsic* properties they are unable to compare *unaligned* data manifolds.

In this chapter, we aim to approach the problem from the geometric point of view, leveraging the intrinsic geometry of the data. The geometric properties of neural networks already provide insights about their internals [151, 225] and help researchers in the design of more robust models [12, 27]. We start out from the observation that models capturing the *multi-scale* nature of the data manifold by utilizing higher distribution moment matching, such as MMD-GAN [132] and Sphere-GAN [164], perform consistently better than their single-scale counterparts. On the other hand, using *extrinsic* information can be misleading, as it is dependent on factors external to the data, such as representation.

We summarize the contributions of this Chapter as follows:

1. To address the above-mentioned drawback, we propose IMD, an Intrinsic Multi-scale Distance, that is able to compare distributions using only *intrinsic* information about the data.

2. We provide an efficient approximation thereof that renders computational complexity nearly linear, including error bound analysis and variance reduction scheme.

3. We demonstrate that IMD effectively quantifies difference in data distributions in three distinct application scenarios: comparing word vectors in languages with *unaligned* vocabularies, tracking dynamics of intermediate neural network representations, and evaluating generative models.

3.2 Related Work

The geometric perspective on data is ubiquitous in machine learning. Geometric techniques enhance unsupervised and semi-supervised learning, generative and discriminative models [24, 12, 142]. We outline the applications of the proposed manifold comparison technique and highlight the geometric intuition along the way.

3.2.1 Generative Model Evaluation

Past research has explored many different directions for the evaluation of generative models. Setting aside models that ignore the true data distribution, such as the Inception Score [186] and GILBO [8], we discuss most relevant geometric ideas below; we refer the reader to Borji [29] for a comprehensive survey.

Critic model-based metrics. Classifier two-sample tests (C2ST) [136] aim to assess whether two samples came from the same distribution by means of an auxiliary classifier. This idea is reminiscent of the GAN discriminator network [89]: if it is possible to train a model that distinguishes between samples from the model and the data distributions, it follows that these distributions are not entirely similar. The convergence process of the GAN-like discriminator [12, 27] lends itself to creating a family of metrics based on training a discriminative classifier [113]. Still, training a separate critic model is often computationally prohibitive and requires careful specification. Besides, if the critic model is a neural network, the resulting metric falls short of interpretability and training stability.

Many advanced GAN models such as Wasserstein, MMD, Sobolev and Spherical GANs impose different constraints on the function class so as to stabilize training [12, 27, 152, 164]. Higher-order moment matching [27, 164] enhances GAN performance, enabling GANs to capture multi-scale data properties, while multi-scale noise ameliorates GAN convergence problems [114]. Still, no feasible multi-scale GAN evaluation metric has been proposed to date.

Positional distribution comparison. In certain settings, it is acceptable to assign zero probability mass to the real data points [160]. In effect, metrics that estimate a distribution's location and dispersion provide useful input for generative model evaluations. For instance, the Fréchet Inception Distance (FID) [103] computes the Wasserstein-2 (i.e., Fréchet) distance between distributions approximated with Gaussians, using only the estimated mean and covariance matrices; the Kernel Inception Distance (KID) [27] computes a polynomial kernel $\mathrm{k}(x, y) = (\frac{1}{d}x^\top y + 1)^3$ and measures the associated Kernel Maximum Mean Discrepancy (kernel MMD). Unlike FID, KID has an unbiased estimator [91, 27]. However, even while such methods, based on a limited number of moments, may be computationally inexpensive, they only provide a rudimentary characterization of distributions from a geometric viewpoint.

Intrinsic geometric measures. The Geometry Score [117] characterizes distributions in terms of their estimated persistent homology, which roughly corresponds to the number of holes in a manifold. Still, the Geometry Score assesses distributions merely in terms of their *global* geometry. In this work, we aim to provide a *multi-scale geometric assessment*.

3.2.2 Similarities of Neural Network Representations

Learning how representations evolve during training or across initializations provides a pathway to the interpretability of neural networks [173]. Still, state-of-the-art methods for comparing representations of neural networks [122, 151, 225] consider only linear projections. The intrinsic nature of IMD renders it appropriate for the task of comparing neural network representations, which can only rely on intrinsic information.

Yin and Shen [240] introduced the Pairwise Inner Product (PIP) loss, an unnormalized covariance error between sets, as a dissimilarity metric between word2vec embedding spaces with common vocabulary. We show in Section 3.4.4 how IMD is applicable to this comparison task too.

3.3 Multi-Scale Intrinsic Distance

At the core of deep learning lies the *manifold hypothesis*, which states that high-dimensional data, such as images or text, lie on a low-dimensional manifold [156, 24, 25]. We aim to provide a theoretically motivated comparison of data manifolds based on rich intrinsic information. Our target measure should have the following properties:

intrinsic – it is invariant to isometric transformations of the manifold, e.g. translations or rotations.

multi-scale – it captures both local and global information.

We expose our method starting out with heat kernels, which admit a notion of manifold metric and can be used to lower-bound the distance between manifolds.

3.3.1 Heat Kernels on Manifolds and Graphs

Based on the heat equation, the heat kernel captures *all* the information about a manifold's intrinsic geometry [202]. Given the Laplace-Beltrami operator (LBO) $\Delta_{\mathcal{X}}$ on a manifold $\mathcal{X}$, the *heat equation* is $\frac{\partial u}{\partial t} = \Delta_{\mathcal{X}} u$ for $u : \mathbb{R}^{+} \times \mathcal{X} \to \mathbb{R}^{+}$. A smooth function u is a *fundamental solution* of the heat equation at point $x \in \mathcal{X}$ if u satisfies both the heat equation and the Dirac condition $u(t, x') \to \delta(x' - x)$ as $t \to 0^{+}$. We assume the Dirichlet boundary condition $u(t, x) = 0$ for all t and $x \in \partial\mathcal{X}$. The heat kernel $\mathrm{k}_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \times \mathbb{R}^{+} \to \mathbb{R}_0^{+}$ is the unique solution of the heat equation; while heat kernels can be defined on hyperbolic spaces and other exotic geometries, we restrict our exposition to Euclidean spaces $\mathcal{X} = \mathbb{R}^d$, on which the heat kernel is defined as:

$$\mathrm{k}_{\mathbb{R}^d}(x, x', t) = \frac{1}{(4\pi t)^{d/2}} \exp\left(-\frac{\|x - x'\|^2}{4t}\right). \tag{3.1}$$

For a compact $\mathcal{X}$ including submanifolds of $\mathbb{R}^d$, the heat kernel admits the expansion

$$\mathrm{k}_{\mathcal{X}}(x, x', t) = \sum_{i=0}^{\infty} e^{-\lambda_i t} \phi_i(x) \phi_i(x'), \tag{3.2}$$

where λ_i and ϕ_i are the i-th eigenvalue and eigenvector of $\Delta_{\mathcal{X}}$. For $t \simeq 0^{+}$, according to Varadhan's lemma, the heat kernel approximates geodesic distances. Importantly for our purposes, the Heat kernel is *multi-scale: for a local domain $\mathcal{D}$ with Dirichlet condition, the localized heat kernel $\mathrm{k}_{\mathcal{D}}(x, x', t)$ is a good approximation of $\mathrm{k}_{\mathcal{X}}(x, x', t)$ if either (i) $\mathcal{D}$ is arbitrarily small and t is small enough, or (ii) t is for arbitrarily large and $\mathcal{D}$ is big enough. Formally,*

Definition 2. Multi-scale property *[92, 202] (i) For any smooth and relatively compact domain $\mathcal{D} \subseteq \mathcal{X}$, $\lim_{t \to 0} \mathrm{k}_{\mathcal{D}}(x, x', t) = \lim_{t \to 0} \mathrm{k}_{\mathcal{X}}(x, x', t)$ (ii) For any $t \in \mathbb{R}^{+}$ and any $x, x' \in \mathcal{D}_1$ localized heat kernel $\mathrm{k}_{\mathcal{D}_1}(x, x', t) \leq \mathrm{k}_{\mathcal{D}_2}(x, x', t)$ if $\mathcal{D}_1 \subseteq \mathcal{D}_2$. Moreover, if $\{\mathcal{D}_n\}$ is an expanding and exhausting sequence $\bigcup_{i=1}^{\infty} \mathcal{D}_i = \mathcal{X}$ and $\mathcal{D}_{i-1} \subseteq \mathcal{D}_i$, then $\lim_{i \to \infty} \mathrm{k}_{\mathcal{D}_i}(x, x', t) = \mathrm{k}_{\mathcal{X}}(x, x', t)$ for any t.*

Heat kernels are also defined for graphs in terms of their Laplacian matrices. Since the Laplacian matrix is symmetric, its eigenvectors $\phi_1, \ldots, \phi_n$, are real and orthonormal. Thus, it is factorized as $\mathscr{L} = \Phi \boldsymbol{\Lambda} \Phi^{\top}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix with the sorted eigenvalues $\lambda_1 \leq \ldots \leq \lambda_n$, and Φ is the orthonormal matrix $\Phi = (\phi_1, \ldots, \phi_n)$ having the eigenvectors of $\mathscr{L}$ as its columns.

The heat kernel on a graph is also given by the solution to the heat equation on a graph, which requires an eigendecomposition of its Laplacian: $\mathbf{H}_t = e^{-t\mathcal{L}} = \Phi e^{-t\mathbf{\Lambda}}\Phi^\top = \sum_i e^{-t\lambda_i}\phi_i\phi_i^\top$.

A useful invariant of the heat kernel is the *heat kernel trace* $\mathrm{hkt}_{\mathcal{X}} : \mathcal{X}\times\mathbb{R}_0^+ \to \mathbb{R}_0^+$, defined by a diagonal restriction as

$$\mathrm{hkt}_{\mathcal{X}}(t) = \int_{\mathcal{X}} \mathrm{k}_{\mathcal{X}}(x,x,t)dx = \sum_{i=0}^{\infty} e^{-\lambda_i t}, \tag{3.3}$$

or, in the discrete case,

$$\mathrm{hkt}_{\mathcal{L}}(t) = \mathrm{tr}(\mathbf{H}_t) = \sum_i e^{-t\lambda_i}. \tag{3.4}$$

Heat kernels traces (HKTs) have been successfully applied to the analysis of 3D shapes [202] and graphs [210]. The HKT contains *all* the information in the graph's spectrum, both local and global, as the eigenvalues λ_i can be inferred therefrom [142, Remark 4.8]. For example, if there are c connected components in the graph, then $\lim_{t\to\infty} \mathrm{hkt}_{\mathcal{L}}(t) = c$.

3.3.2 Convergence to the Laplace-Beltrami Operator

An important property of graph Laplacians is that it is possible to construct a graph among points sampled from a manifold $\mathcal{X}$ such that the spectral properties of its Laplacian resemble those of the Laplace-Beltrami operator on $\mathcal{X}$. Belkin and Niyogi [24] proposed such a construction, the point cloud Laplacian, which is used for dimensionality reduction in a technique called Laplacian eigenmaps. Convergence to the LBO has been proven for various definitions of the graph Laplacian, including the one we use [25, 101, 51, 209]. We recite the convergence results for the point cloud Laplacian from Belkin and Niyogi [25]:

Theorem 1. *Let $\lambda_{n,i}^{t_n}$ and $\phi_{n,i}^{t_n}$ be the i^{th} eigenvalue and eigenvector, respectively, of the point cloud Laplacian $\mathcal{L}^{t_n}$; let λ_i and ϕ_i be the i^{th} eigenvalue and eigenvector of the LBO Δ. Then, there exists $t_n \to 0$ such that*

$$\lim_{n\to\infty} \lambda_{n,i}^{t_n} = \lambda_i,$$
$$\lim_{n\to\infty} \left\|\phi_{n,i}^{t_n} - \phi_i\right\|_2 = 0.$$

Still, the point cloud Laplacian involves the creation of an $\mathcal{O}(n^2)$ matrix; for the sake of scalability, we use the k-nearest-neighbours (kNN) graph by OR-construction (i.e., based on bidirectional kNN relationships among points), whose Laplacian converges to the LBO for data with sufficiently high intrinsic dimension [209]. As for the choice of k, a random geometric

kNN graph is connected when $k \geq {}^{\log n}/_{\log 7} \approx 0.5139 \log n$ [21]; $k = 5$ yields connected graphs for all sample sizes we tested.

3.3.3 Spectral Gromov-Wasserstein Distance

Even while it is a multi-scale metric *on* manifolds, the heat kernel can be spectrally approximated by finite graphs constructed from points sampled from these manifolds. In order to construct a metric *between* manifolds, Mémoli [142] suggests an optimal-transport-based “meta-distance”: a spectral definition of the Gromov-Wasserstein distance between Riemannian manifolds based on matching the heat kernels at all scales. The cost of matching a pair of points (x, x') on manifold $\mathcal{M}$ to a pair of points (y, y') on manifold $\mathcal{N}$ at scale t is given by their heat kernels $\mathrm{k}_{\mathcal{M}}, \mathrm{k}_{\mathcal{N}}$:

$$\Gamma(x, y, x', y', t) = \left|\mathrm{k}_{\mathcal{M}}(x, x', t) - \mathrm{k}_{\mathcal{N}}(y, y', t)\right| .$$

The distance between the manifolds is then defined in terms of the infimal measure coupling

$$d_{\mathrm{GW}}(\mathcal{M}, \mathcal{N}) = \inf_{\mu} \sup_{t>0} e^{-2(t+t^{-1})} \, \|\Gamma\|_{L^2(\mu\times\mu)},$$

where the infimum is sought over all measures μ on $\mathcal{M} \times \mathcal{N}$ marginalizing to the standard measures on $\mathcal{M}$ and $\mathcal{N}$. For finite spaces, μ is a doubly-stochastic matrix. This distance is lower-bounded [142] in terms of the respective heat kernel traces as:

$$d_{\mathrm{GW}}(\mathcal{M}, \mathcal{N}) \geq \sup_{t>0} e^{-2(t+t^{-1})} \left|\mathrm{hkt}_{\mathcal{M}}(t) - \mathrm{hkt}_{\mathcal{N}}(t)\right| . \tag{3.5}$$

This lower bound is the scaled L_∞ distance between the *heat trace signatures* $\mathrm{hkt}_{\mathcal{M}}$ and $\mathrm{hkt}_{\mathcal{N}}$. The scaling factor $e^{-2(t+t^{-1})}$ favors medium-scale differences, meaning that this lower bound is not sensitive to local perturbations. The maximum of the scaling factor occurs at $t = 1$, and more than $1 - 10^{-8}$; of the function mass lies between $t = 0.1$ and $t = 10$.

3.3.4 Heat Trace Estimation

Calculating the heat trace signature efficiently and accurately is a challenge on a large graph as it involves computing a trace of a large matrix exponential, i.e. $\mathrm{tr}(e^{-t\mathscr{L}})$. A naive approach would be to use an eigendecomposition $\exp(-t\mathscr{L}) = \Phi \exp(-t\Lambda)\Phi^\top$, which is infeasible for large n. Recent work [210] suggested using either truncated Taylor expansion or linear interpolation of the interloping eigenvalues, however, both techniques are quite coarse. To combine accuracy and speed, we use the Stochastic Lanczos Quadrature (SLQ) [219, 83]. This method combines the Hutchinson trace estimator [112, 3] and the Lanczos algorithm for eigenvalues. We aim to estimate the trace of a matrix function with a Hutchinson estimator:

$$\operatorname{tr}(f(\mathscr{L})) = \mathbb{E}_{p(\mathbf{v})}(\mathbf{v}^\top f(\mathscr{L})\mathbf{v}) \approx \frac{n}{n_v}\sum_{i=1}^{n_v} \mathbf{v}_i^\top f(\mathscr{L})\mathbf{v}_i, \tag{3.6}$$

where the function of interest $f(\cdot) = \exp(\cdot)$ and $\mathbf{v}_i$ are n_v random vectors drawn from a distribution $p(\mathbf{v})$ with zero mean and unit variance. A typical choice for $p(\mathbf{v})$ is Rademacher or a standard normal distribution. In practice, there is little difference, although in theory Rademacher has less variance, but Gaussian requires less random vectors [16].

To estimate the quadratic form $\mathbf{v}_i^\top f(\mathscr{L})\mathbf{v}_i$ in Equation (3.6) with a symmetric real-valued matrix $\mathscr{L}$ and a smooth function f, we plug in the eigendecomposition $\mathscr{L} = \Phi\Lambda\Phi^\top$, rewrite the outcome as a Riemann-Stieltjes integral and apply the m-point Gauss quadrature rule [85]:

$$\mathbf{v}_i^\top f(\mathscr{L})\mathbf{v}_i = \mathbf{v}_i^\top \Phi f(\Lambda)\Phi^\top \mathbf{v}_i = \sum_{j=1}^{n} f(\lambda_j)\mu_j^2 = \int_a^b f(t)d\mu(t) \approx \sum_{k=1}^{m} \omega_k f(\theta_k), \tag{3.7}$$

where $\mu_j = [\Phi^\top \mathbf{v}_i]_j$ and $\mu(t)$ is a piecewise constant function defined as follows

$$\mu(t) = \begin{cases} 0, & \text{if } t < a = \lambda_n, \\ \sum_{j=1}^{i} \mu_j^2, & \text{if } \lambda_i \le t < \lambda_{i-1}, \\ \sum_{j=1}^{n} \mu_j^2, & \text{if } b = \lambda_1 \le t, \end{cases}$$

and θ_k are the quadrature's nodes and ω_k are the corresponding weights. We obtain ω_k and θ_k with the m-step Lanczos algorithm [83]. The Lanczos process is connected to the orthogonal polynomials associated with the quadrature. Below, we describe this link succinctly.

Given the symmetric matrix $\mathscr{L}$ and an arbitrary *starting unit-vector* $\mathbf{q}_0$, the m-step Lanczos algorithm computes an $n \times m$ matrix $\mathbf{Q} = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_{m-1}]$ with orthogonal columns and an $m \times m$ tridiagonal symmetric matrix $\mathbf{T}$, such that $\mathbf{Q}^\top \mathscr{L} \mathbf{Q} = \mathbf{T}$. The columns of $\mathbf{Q}$ constitute an orthonormal basis for the Krylov subspace $\mathcal{K}$ that spans vectors $\{\mathbf{q}_0, \mathscr{L}\mathbf{q}_0, \dots, \mathscr{L}^{m-1}\mathbf{q}_0\}$; each $\mathbf{q}_i$ vector is given as a polynomial in $\mathscr{L}$ applied to the initial vector $\mathbf{q}_0$: $\mathbf{q}_i = p_i(\mathscr{L})\mathbf{q}_0$. These Lanczos polynomials are orthogonal with respect to the integral measure $\mu(t)$. As orthogonal polynomials satisfy the three term recurrence relation, we obtain p_{k+1} as a combination of p_k and p_{k-1}. The tridiagonal matrix storing the coefficients of such combinations, called the Jacobi matrix $\mathbf{J}$, is exactly the tridiagonal symmetric matrix $\mathbf{T}$. A classic result tells us that the nodes θ_k and the weights ω_k of the Gauss quadrature are the eigenvalues of $\mathbf{T}$, λ_k, and the squared first components of its normalized eigenvectors, τ_k^2, respectively (see Golub and Welsch [85], Wilf [228], Golub and Meurant [83]). Thereby, setting $\mathbf{q}_0 = \mathbf{v}_i$, the estimate for the quadratic form becomes:

$$\mathbf{v}_i^\top f(\mathscr{L})\mathbf{v}_i \approx \sum_{k=1}^{m} \tau_k^2 f(\lambda_k), \quad \tau_k = \mathbf{U}_{0,k} = \mathbf{e}_1^\top \mathbf{u}_k, \quad \lambda_k = \Lambda_{k,k} \quad \mathbf{T} = \mathbf{U}\Lambda\mathbf{U}^\top, \tag{3.8}$$

Applying (3.8) over n_v random vectors in the Hutchinson trace estimator (3.6) yields the SLQ estimate:

$$\operatorname{tr}(f(\mathscr{L})) \approx \frac{n}{n_v} \sum_{i=1}^{n_v} \left(\sum_{k=0}^{m} \left(\tau_k^i\right)^2 f\left(\lambda_k^i\right) \right) = \Gamma. \tag{3.9}$$

We derive error bounds for the estimator based on the Lanczos approximation of the matrix exponential, and show that even a few Lanczos steps, i.e., $m = 10$, are sufficient for an accurate approximation of the quadratic form. However, the trace estimation error is theoretically dominated by the error of the Hutchinson estimator, e.g. for Gaussian $p(\mathbf{v})$ the bound on the number of samples to guarantee that the probability of the relative error exceeding ϵ is at most δ is $8\epsilon^{-2}\ln(2/\delta)$ [179]. Although, in practice, we observe performance much better than the bound suggests. Hutchinson error implies nearing accuracy roughly 10^{-2} with $n_v \geq 10$k random vectors, however, with as much as $n_v = 100$ the error is already 10^{-3}. Thus, we use default values of $m = 10$ and $n_v = 100$ in all experiments in Section 3.4.

3.3.5 Trace Estimation Error Bounds

We will use the error of the Lanczos approximation of the action of the matrix exponential $f(\mathscr{L})\mathbf{v} = \exp(-t\mathscr{L})\mathbf{v}$ to estimate the error of the trace. We first rewrite quadratic form under summation in the trace approximation to a convenient form,

$$\mathbf{v}^\top f(\mathscr{L})\mathbf{v} \approx \sum_{k=0}^{m} \tau_k^2 f(\lambda_k) = \sum_{k=0}^{m} [\mathbf{e}_1^\top \mathbf{u}_k]^2 f(\lambda_k) = \mathbf{e}_1^\top \mathbf{U} f(\Lambda) \mathbf{U}^\top \mathbf{e}_1 = \mathbf{e}_1^\top f(\mathbf{T})\mathbf{e}_1. \tag{3.10}$$

Because the Krylov subspace $\mathcal{K}_m(\mathscr{L}, \mathbf{v})$ is built on top of vector $\mathbf{v}$ with $\mathbf{Q}$ as an orthogonal basis of $\mathcal{K}_m(\mathscr{L}, \mathbf{v})$, i.e. $\mathbf{q}_0 = \mathbf{v}$ and $\mathbf{v} \perp \mathbf{q}_i$ for $i \in (1, \ldots, m-1)$, the following holds

$$\mathbf{v}^\top f(\mathscr{L})\mathbf{v} \approx \mathbf{v}^\top \mathbf{Q} f(\mathbf{T})\mathbf{e}_1 = \mathbf{e}_1^\top f(\mathbf{T})\mathbf{e}_1. \tag{3.11}$$

Thus, the error in quadratic form estimate $\mathbf{v}^\top f(\mathscr{L})\mathbf{v}$ is exactly the error of Lanczos approximation $f(\mathscr{L})\mathbf{v} \approx \mathbf{Q} f(\mathbf{T})\mathbf{e}_1$. To obtain the error bounds, we use the Theorem 2 in Hochbruck and Lubich [105], which we recite below.

Theorem 2. *Let $\mathscr{L}$ be a real symmetric positive semi-definite matrix with eigenvalues in the interval* $[0, 4\rho]$. *Then the error in the m-step Lanczos approximation of* $\exp(-t\mathscr{L})\mathbf{v}$:

$$\epsilon_m = \| \exp(-t\mathscr{L})\mathbf{v} - \mathbf{Q}_m \exp(-t\mathbf{T}_m)\mathbf{e}_1\|, \tag{3.12}$$

is bounded in the following ways:

$$\epsilon_m \leq \begin{cases} 10e^{-m^2/(5\rho t)}, & \text{if } \sqrt{4\rho t} \leq m \leq 2\rho t, \\ 10(\rho t)^{-1}e^{-\rho t}\left(\frac{e\rho t}{m}\right)^m, & \text{if } m \geq 2\rho t. \end{cases} \tag{3.13}$$

Since $\mathbf{v}$ is a unit vector, thanks to Cauchy-Bunyakovsky-Schwarz inequality, we can upper bound the error of the quadratic form approximation by the error of the $\exp(-t\mathscr{L})\mathbf{v}$ approximation, i.e. $|\mathbf{v}^\top f(\mathscr{L})\mathbf{v} - \mathbf{e}_1^\top \mathbf{U} f(\Lambda)\mathbf{U}^\top \mathbf{e}_1| \leq \| \exp(-t\mathscr{L})\mathbf{v} - \mathbf{Q}_m \exp(-t\mathbf{T}_m)\mathbf{e}_1\| = \epsilon_m$.

Following the argumentation in Ubaru et al. [219], we obtain a condition on the number of Lanczos steps m by setting $\epsilon_m \leq {}^{\epsilon}/_{2} f_{min}(\lambda)$, where $f_{min}(\lambda)$ is the minimum value of f on $[\lambda_{min}, \lambda_{max}]$. We now derive the absolute error between the Hutchinson estimate of Equation (3) and the SLQ of Equation (6):

$$\begin{aligned} \left| \mathrm{tr}_{n_v}(f(\mathscr{L})) - \Gamma \right| &= \frac{n}{n_v} \left| \sum_{i=1}^{n_v} \mathbf{v}_i^\top f(\mathscr{L})\mathbf{v}_i - \sum_{i=1}^{n_v} \mathbf{e}_1^\top f(\mathbf{T}^{(i)})\mathbf{e}_1 \right| \\ &\leq \frac{n}{n_v} \sum_{i=1}^{n_v} \left| \mathbf{v}_i^\top f(\mathscr{L})\mathbf{v}_i - \mathbf{e}_1^\top f(\mathbf{T}^{(i)})\mathbf{e}_1 \right| \\ &\leq \frac{n}{n_v} \sum_{i=1}^{n_v} \epsilon_m = n\epsilon_m, \end{aligned}$$

where $\mathbf{T}^{(i)}$ is the tridiagonal matrix obtained with Lanczos algorithm with starting vector $\mathbf{v}_i$.

$$\left| \mathrm{tr}_{n_v} f(\mathscr{L}) - \Gamma \right| \leq n\epsilon_m \leq \frac{n\epsilon}{2} f_{min}(\lambda) \leq \frac{\epsilon}{2} \mathrm{tr}(f(\mathscr{L})), \tag{3.14}$$

Finally, we formulate SLQ as an (ϵ, δ) estimator,

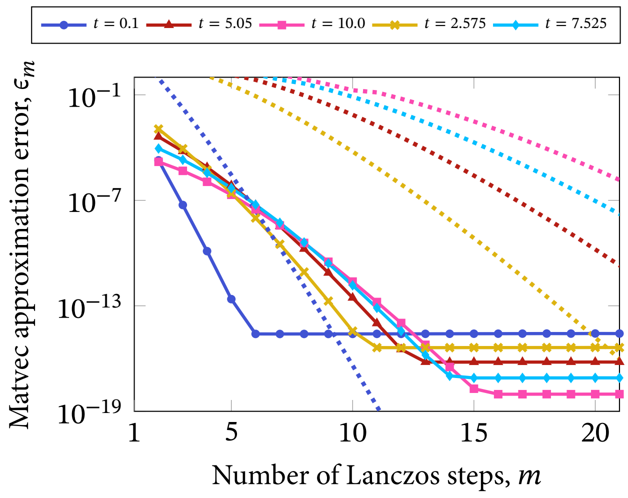


Fig. 3.2 Errors (solid) and error bounds (dotted) for the approximation of matrix exponential action with varying temperature t.

$$\begin{aligned}1-\delta &\le \Pr\left[\left|\operatorname{tr}(f(\mathscr{L}))-\operatorname{tr}_{n_v}(f(\mathscr{L}))\right| \le \frac{\epsilon}{2}\left|\operatorname{tr}(f(\mathscr{L}))\right|\right]\\ &\le \Pr\left[\left|\operatorname{tr}(f(\mathscr{L}))-\operatorname{tr}_{n_v}(f(\mathscr{L}))\right| + \left|\operatorname{tr}_{n_v}(f(\mathscr{L}))-\Gamma\right| \le \frac{\epsilon}{2}\left|\operatorname{tr}(f(\mathscr{L}))\right| + \frac{\epsilon}{2}\left|\operatorname{tr}(f(\mathscr{L}))\right|\right]\\ &\le \Pr\left[\left|\operatorname{tr}(f(\mathscr{L}))-\Gamma\right| \le \epsilon\left|\operatorname{tr}(f(\mathscr{L}))\right|\right],\end{aligned}$$

For the normalized Laplacian $\mathscr{L}$, the minimum eigenvalue is 0 and $f_{\min}(0) = \exp(0) = 1$, hence $\epsilon_m \le \frac{\epsilon}{2}$, and the eigenvalue interval has $\rho = 0.5$. We can thus derive the appropriate number of Lanczos steps m to achieve error ϵ,

$$\epsilon \le \begin{cases} 20e^{-m^2/(2.5t)}, & \text{if } \sqrt{2t} \le m \le t, \\ 40t^{-1}e^{-0.5t}\left(\frac{0.5et}{m}\right)^m, & \text{if } m \ge t. \end{cases} \tag{3.15}$$

Figure 3.2 shows the tightness of the bound for the approximation of the matrix exponential action on vector $\mathbf{v}$, $\epsilon_m = ||e^{-t\mathscr{L}} - \mathbf{Q}_m e^{-t\mathbf{T}_m}\mathbf{e}_1||$. We can see that for most of the temperatures t, very few Lanczos steps m are sufficient, i.e. we can set $m = 10$. However, the error from the Hutchinson estimator dominates the overall error. Figure 3.3 shows the error of trace estimation does not change with m and for $t = 0.1$ is around 10^{-3}. In case of a Rademacher $p(\mathbf{v})$, the bound on the number of random samples is $n_v \ge \frac{6}{\epsilon^2}\log(2/\delta)$ [179]. Employing 10k vectors results in the error bound of roughly 10^{-2}. In practice, we observe the performance much better than given by the bound, see Figure 3.3.

One particular benefit of small m value is that we do not have to worry about the orthogonality loss in the Lanczos algorithm which often undermines its convergence. Since we do only a few Lanczos iterations, the rounding errors hardly accumulate causing little burden in terms of orthogonality loss between the basis vectors of the Krylov subspace.

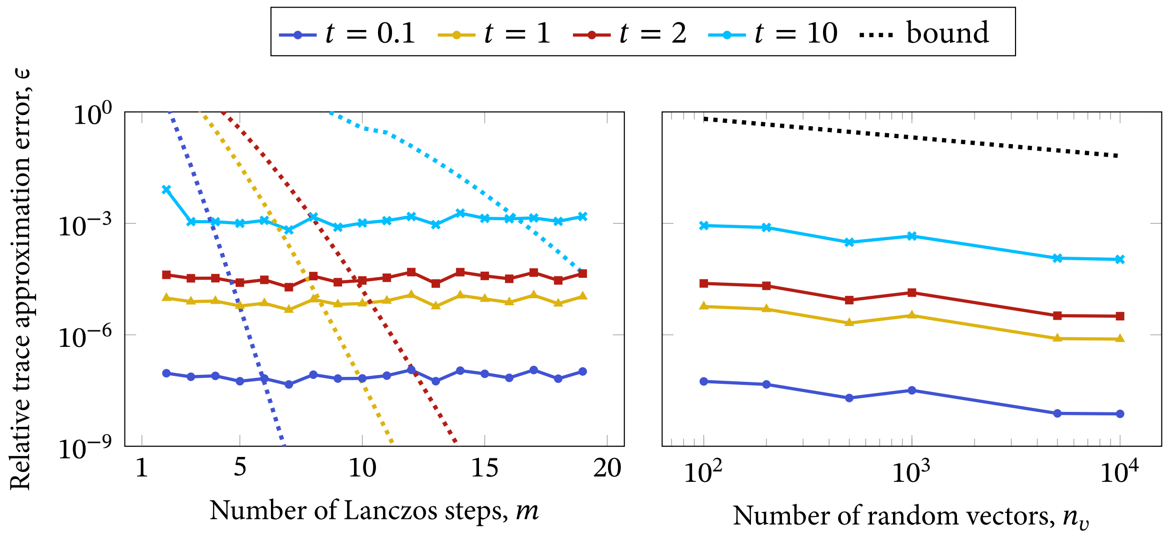


Fig. 3.3 Trace estimation errors (solid) and error bounds (dotted) for: *(left)* the number of Lanczos steps m with fixed number of random vectors $n_v = 100$; *(right)* the number of random vectors n_v in Hutchinson estimator with fixed number of Lanczos steps $m = 10$. Lines correspond to varying temperatures t.

3.3.6 Variance Reduction

We reduce variance of the randomized estimator through control variates. The idea is to use Taylor expansion to substitute a part of the trace estimate with its easily computed precise value,

$$\begin{aligned}\operatorname{tr}(e^{-t\mathscr{L}}) &= \mathtt{slq}\Big[e^{-t\mathscr{L}} - (\mathbf{I} - t\mathscr{L} + \frac{t^2\mathscr{L}^2}{2})\Big] + \operatorname{tr}(\mathbf{I} - t\mathscr{L} + \frac{t^2\mathscr{L}^2}{2}) && (3.16)\\ &= \mathtt{slq}\Big[e^{-t\mathscr{L}} - (\mathbf{I} - t\mathscr{L} + \frac{t^2\mathscr{L}^2}{2})\Big] + n + \operatorname{tr}(-t\mathscr{L}) + \frac{t^2\|\mathscr{L}\|_F^2}{2} && (3.17)\\ &= \mathtt{slq}\Big[e^{-t\mathscr{L}}\Big] + \mathtt{slq}\Big[t\mathscr{L}\Big] - \mathtt{slq}\Big[\frac{t^2\mathscr{L}^2}{2}\Big] - tn + \frac{t^2\|\mathscr{L}\|_F^2}{2}, && (3.18)\end{aligned}$$

where $\mathtt{slq}\left[\cdot\right]$ denotes an SLQ estimate of the trace of $\cdot$, we use the fact that $\|\mathscr{L}\|_F = \sqrt{\operatorname{tr}(\mathscr{L}^\top\mathscr{L})}$ and that the trace of normalized Laplacian is equal to n. It does reduce the variance of the trace estimate for smaller temperatures $t \leq 1$.

To obtain this advantage over the whole range of t, we utilize the following variance reduction form:

$$\operatorname{tr}(e^{-t\mathscr{L}}) = \mathtt{slq}\left[e^{-t\mathscr{L}} - (\mathbf{I} - \alpha t\mathscr{L})\right] + n(1 - \alpha t), \tag{3.19}$$

where there exists an alpha that is optimal for every t, namely setting $\alpha = 1/\exp(t)$. We can see the variance reduction that comes from this procedure in the Figure 3.4.

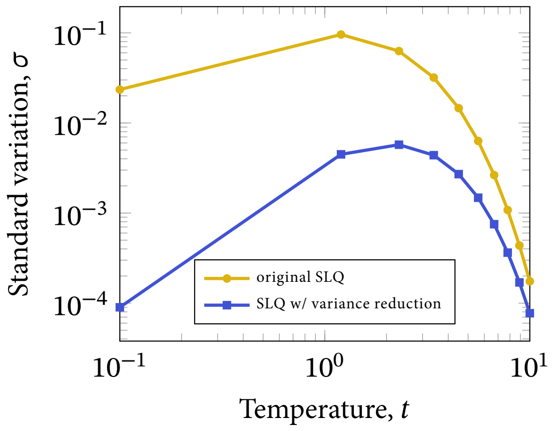


Fig. 3.4 Variance of the trace estimate.

3.3.7 Putting IMD Together

We employ the heretofore described advances in differential geometry and numerical linear algebra to create IMD (*Multi-Scale Intrinsic Distance*), a fast, intrinsic method to lower-bound the spectral Gromov-Wasserstein distance between manifolds.

We describe the overall computation of IMD in Algorithm 4. Given data samples in $\mathbb{R}^d$, we build a kNN graph G by OR-construction such that its Laplacian spectrum approximates the one of the Laplace-Beltrami operator of the underlying manifold [209], and then compute $\operatorname{hkt}_G(t) = \sum_i e^{-\lambda_i t} \approx \Gamma$. We compare heat traces in the spirit of Equation (3.5), i.e., we compute the distance $\left|\operatorname{hkt}_{G_1}(t) - \operatorname{hkt}_{G_2}(t)\right|$ between heat kernel traces for $t \in (0.1, 10)$ sampled from a logarithmically spaced grid.

Constructing exact kNN graphs is an $\mathcal{O}(dn^2)$ operation; however, approximation algorithms take near-linear time $\mathcal{O}(dn^{1+\omega})$ [61, 14]. In practice, approximate kNN graph construction [61] yields low computational time while still preserving variance similar to the exact case. The m-step

Algorithm 2 IMD algorithm.

```
function IMDESC($X$)
    $G \leftarrow \texttt{kNN}(X)$
    $\mathscr{L} \leftarrow \texttt{Laplacian}(G)$
    return $\Gamma = \texttt{slq}(\mathscr{L}, s, n_v)$

function IMDIST($X, Y$)
    $\texttt{hkt}_X \leftarrow \texttt{IMDesc}(X)$
    $\texttt{hkt}_Y \leftarrow \texttt{IMDesc}(Y)$
    return $\sup e^{-2(t+t^{-1})} |\texttt{hkt}_X - \texttt{hkt}_Y|$
```

Lanczos algorithm on a sparse $n \times n$ kNN Laplacian $\mathscr{L}$ with one starting vector has $\mathcal{O}(knm)$ complexity, where kn is the number of nonzero elements in $\mathscr{L}$. The eigendecomposition of the symmetric tridiagonal matrix $\mathbf{T}$ incurs an additional $\mathcal{O}(m \log m)$ [49]. We apply this algorithm over n_v starting vectors, yielding a complexity of $\mathcal{O}(n_v(m \log m + kmn))$, with constant $k = 5$ and $m = 10$ by default. In effect, IMD's time complexity stands between those of two common GAN evaluation methods: KID, which is $\mathcal{O}(dn^2)$ and FID, which is $\mathcal{O}(d^3 + dn)$. The time complexity of Geometry Score is unspecified in [117], yet in Section 3.4.8 we show that its runtime grows exponentially in sample size.

3.4 Experiments

We evaluate IMD on the ability to compare intermediate representations of machine learning models. For instance, in a recommender system we could detect whether a problem is related to the representation or the the classifier in the end of a pipeline. In this section, we show the effectiveness of our intrinsic measure on multiple tasks and show how our intrinsic distance can provide insights beyond previously proposed extrinsic measures.

Summary of experiments. We examine the ability of IMD[1] to measure several aspects of difference among data manifolds. We first consider a task from unsupervised machine translation with unaligned word embeddings and show that IMD captures correlations among language kinship (affinity or genealogical relationships). Second, we showcase how IMD handles data coming from data sources of unequal dimensionalities. Third, we study how IMD highlights differences among image data representations across initializations and through training process of neural networks.

[1]Our code is available open-source: https://github.com/xgfs/imd.

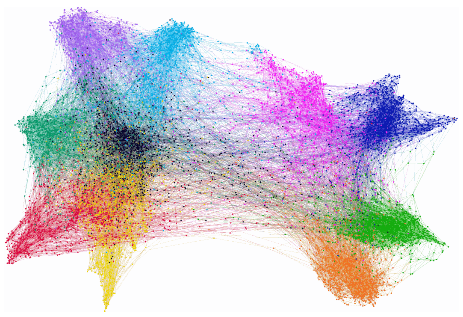


Fig. 3.5 CIFAR10 graph colored with true class labels.

3.4.1 Graph Example

Figure 3.5 provides visual evidence that the 5NN graph reflects the underlying manifold structure of the CIFAR10 dataset. Clusters in the graph exactly correspond to CIFAR10 classes.

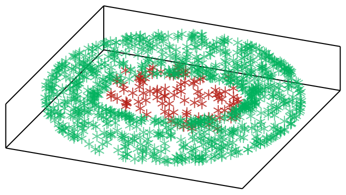
3.4.2 Experimental Settings

We train all our models on a single server with NVIDIA V100 GPU with 16Gb memory and 2×20 core Intel E5-2698 v4 CPU. For the experiment summarized in Table 3.1 in the Section 3.4.6 we train WGAN and WGAN-GP models on 4 datasets: MNIST, FashionMNIST, CIFAR10 and CelebA and sample 10k samples, $\mathbf{Y}$, from each of the GANs. We uniformly subsample 10k images from the original datasets, $\mathbf{X}$, and compute the IMD, KID and FID scores between $\mathbf{X}$ and $\mathbf{Y}$. FID and KID are substituted with Fréchet Distance and Kernel Distance (via a degree 3 polynomial kernel) where representations are not obtained through Inception network, e.g. word embeddings. We report the mean as well as the 99% confidence interval across 100 runs.

We report the architectures, hyperparameters in Appendix B.2.1. We train each of the GANs for 200 epochs on MNIST, FMNIST and CIFAR10, and for 50 epochs on CelebA dataset. For WGAN we use RMSprop optimizer with learning rate of 5×10^{-5}. For WGAN-GP we use Adam optimizer with learning rate of 10^{-4}, $\beta_1 = 0.9, \beta_2 = 0.999$.

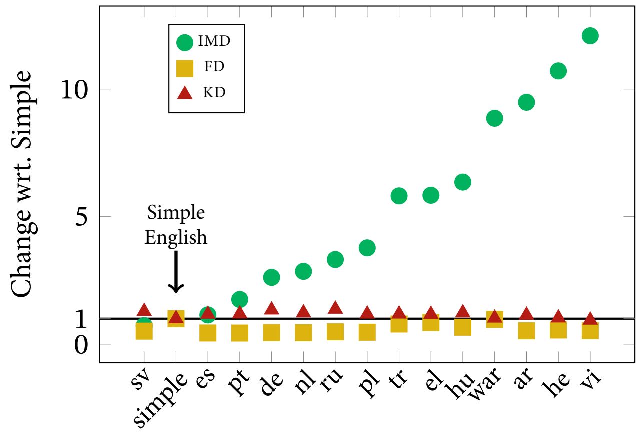
Vanilla GAN on Torus

We first test IMD via a very simple experiment clearly showing the case where IMD is superior to its main competitors, Fréchet Distance (FD) and Kernel Distance (KD). We train two vanilla GANs on the points of a 3D torus. The bad GAN fails to learn the topology of the dataset it tries to mimic generating points in the hole, yet previous metrics cannot detect this fact. IMD, on the contrary, can tell the difference. Figure 3.6 shows the points sampled from the GAN with some of the points inside the hole. KID and FID confidence intervals overlap for good and bad GANs, meanwhile IMD scores are clearly distinct from each other.



metric	good GAN	bad GAN
FD$\times10^3$	5.29 ± 0.70	6.27 ± 0.76
KD$\times10^3$	1.72 ± 0.73	2.59 ± 0.77
IMD	9.02 ± 1.52	**14.07** $\pm$ 2.17

Fig. 3.6 Bad GAN produces samples inside the torus hole (red). FD and KD cannot detect such behaviour as their confidence intervals overlap, while IMD does not.



Arabic	ar	Dutch	nl
English	en	German	de
Greek	el	Hebrew	he
Hungarian	hu	Polish	pl
Portugese	pt	Russian	ru
Simple English	simple	Spanish	sp
Swedish	sv	Turkish	tr
Vietnamese	vi	Waray-Waray	war

Fig. 3.7 Distances from the simple English Wikipedia visualized for IMD, FD and KD.

3.4.3 Comparing Unaligned Language Manifolds

The problem of unaligned representations is particularly severe in the domain of natural language processing as the vocabulary is rarely comparable across different languages or even different documents.

We employ IMD to measure the relative closeness of pairs of languages based on the word embeddings with different vocabularies. Figure 3.8 shows a heatmap of pairwise Fréchet Distance (FD), Kernel Distance (KD) and IMD scores. IMD detects similar languages (Slavic, Semitic, Romanic, etc.) despite the lack of ground truth vocabulary alignment. FD and KD can not find meaningful structure in the data in the same way as IMD as they rely on extrinsic data properties.

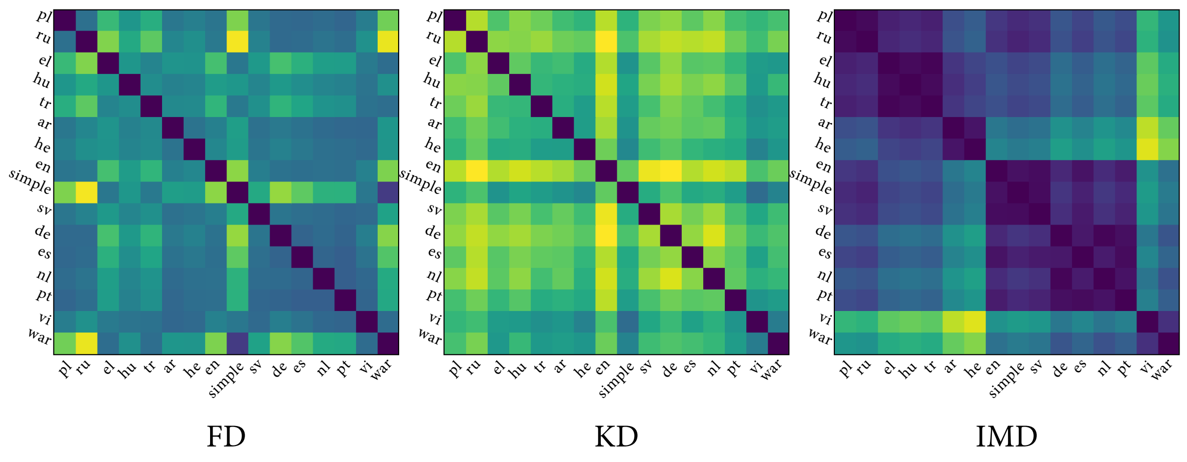


Fig. 3.8 FD and KD are not able to capture language affinity from unaligned word2vec embeddings. Darker color indicates language closeness according to word embedding sets from Wikipedia.

Word Embedding Experiment Details.

We use gensim [175] to learn word vectors on the latest Wikipedia corpus snapshot on 16 languages: Polish, Russian, Greek, Hungarian, Turkish, Arabic, Hebrew, English, Simple English, Swedish, German, Spanish, Dutch, Portugese, Vietnamese, and Waray-Waray. We then compute FD, KD and IMD scores on all the pairs, we average 100 runs for the heatmap Figure 3.9.

For the different dimensionality experiment, we learn vectors on the English Wikipedia of sizes equal to the powers of 2 from 4 to 512. After that we compute IMD and covariance error, i.e. normalized PIP loss, between the pairs of sizes to generate the heatmap Figure 3.9.

3.4.4 Optimizing Dimensionality of Word Embeddings

Comparing data having different dimensionality is cumbersome, even when representations *are* aligned. We juxtapose IMD by PIP loss [240] which allows the comparison of aligned representations for word embeddings.

To this end, we measure IMD distance between English embeddings of varying dimensions. Figure 3.9 shows the heatmap of the IMD and PIP scores between sets of word vectors of different dimensionalities. Closer dimensionalities have lower distance scores for both metrics. However, IMD better highlights gradual change of the size of word vectors, e.g. word vectors of size 4 and 8 are clearly closer to each other than embeddings of size 4 and 16 in terms of IMD, which is not true for PIP.

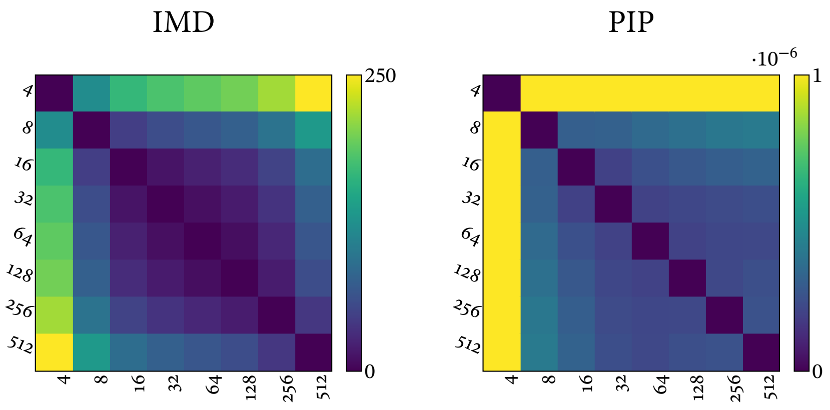


Fig. 3.9 Comparison of IMD and PIP loss on word embeddings of different dimension. IMD detects subtle changes in the dimensionality.

3.4.5 Tracking the Evolution of Image Manifolds

Next, we employ IMD to inspect the internal dynamics of neural networks. We investigate the stability of output layer manifolds across random initializations. We train 10 instances of the VGG-16 [195] network using different weight initializations on the CIFAR10 and CIFAR100 datasets. We compare the average IMD scores across representations in each network layer relative to the last layer. As Figure 3.10 (left) shows, for both CIFAR10 and CIFAR100, the convolutional layers exhibit similar behavior; According to IMD, consequent layers do not monotonically contribute to the separation of image representations, but start to do so after initial feature extraction stage comprised of 4 convolutional blocks. A low variance across the 10 networks trained from different random initializations indicates stability in the network structure.

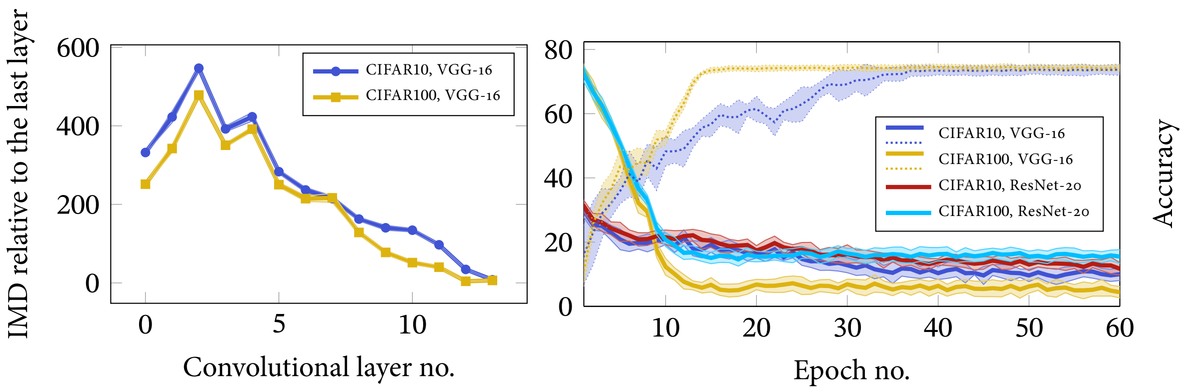


Fig. 3.10 *(left)* IMD score across convolutional layers of the VGG-16 network on CIFAR10 and CIFAR100 datasets; *(right)* training progression in terms of accuracy (dotted) and IMD (solid) on CIFAR10 and CIFAR100 datasets for VGG-16 and ResNet-20, with respect to VGG-16.

We now examine the last network layers during training from different initializations. Figure 3.10 (right) plots the VGG-16 validation errors and IMD scores relative to the final layer representations of *two* pretrained networks, VGG-16 itself with last layer dimension $d = 512$ and ResNet-20 with $d = 64$ and ~50 times less parameters. We observe that even in such unaligned spaces, IMD correctly identifies the convergence point of the networks. Surprisingly, we find that, in terms of IMD, VGG-16 representations progress towards not only the VGG-16 final layer, but the ResNet-20 final layer representation as well; this result suggests that these networks of distinct architectures share similar final structures.

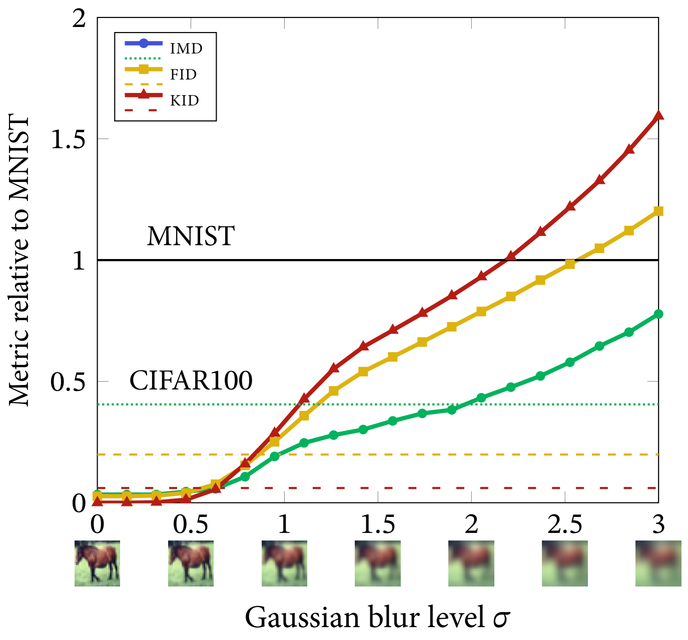


Fig. 3.11 FID, KID and IMD on the CIFAR10 dataset with Gaussian blur.

3.4.6 Evaluating Generative Models

We now move on to apply IMD to evaluation of generative models. First, we evaluate the sensitivity of IMD, FID, and KID to simple image transformations as a proxy to more intricate artifacts of modern generative models. We progressively blur images from the CIFAR10 training set, and measure the distance to the original data manifold, averaging outcomes over 100 subsamples of 10k images each. To enable comparison across methods, we normalize each distance measure such that the distance between CIFAR10 and MNIST is 1. Figure 3.11 reports the results at different levels σ of Gaussian blur. We additionally report the normalized distance to the CIFAR100 training set (dashed lines).

	MNIST		FashionMNIST		CIFAR10		CelebA	
Metric	WGAN	WGAN-GP	WGAN	WGAN-GP	WGAN	WGAN-GP	WGAN	WGAN-GP
IMD	**57.74** $\pm$ 0.47	**10.77** $\pm$ 0.42	**118.14** $\pm$ 0.52	**13.45** $\pm$ 0.54	**18.10** $\pm$ 0.36	**10.84** $\pm$ 0.42	**10.11** $\pm$ 0.33	**2.84** $\pm$ 0.31
KID$\times 10^3$	**47.26** $\pm$ 0.07	**5.53** $\pm$ 0.03	**119.93** $\pm$ 0.14	**25.49** $\pm$ 0.07	**93.89** $\pm$ 0.09	**59.59** $\pm$ 0.09	**217.28** $\pm$ 0.14	**92.71** $\pm$ 0.08
FID	**31.75** $\pm$ 0.07	**8.95** $\pm$ 0.03	**152.44** $\pm$ 0.12	**35.31** $\pm$ 0.07	**101.43** $\pm$ 0.09	**80.65** $\pm$ 0.09	**205.63** $\pm$ 0.09	**85.55** $\pm$ 0.08

Table 3.1 IMD agrees with KID and FID across varying datasets for GAN evaluation.

FID and KID quickly drift away from the original distribution and match MNIST, a dataset of a completely different nature. Contrariwise, IMD is more robust to noise and follows the datasets structure, as the relationships between objects remain mostly unaffected on low blur levels. Moreover, with both FID and KID, low noise ($\sigma = 1$) applied to CIFAR10 suffices to exceed the distance of CIFAR100, which is similar to CIFAR10. IMD is much more robust, exceeding that distance only with $\sigma = 2$.

Next, we turn our attention to the sample-based evaluation of generative models. We then train the WGAN [12] and WGAN-GP [94] models on four datasets: MNIST, FashionMNIST, CIFAR10 and CelebA. Below we also show generated samples of the models used for the experiments in Figures 3.12–3.15. We sample 10k samples, $\mathbf{Y}$, from each GAN. We then uniformly subsample $10k$ images from the corresponding original dataset, $\mathbf{X}$, and compute the IMD, KID and FID scores between $\mathbf{X}$ and $\mathbf{Y}$. Table 3.1 reports the average measure and its 99% confidence interval across 100 runs. IMD, as well as both FID and KID, reflect the fact that WGAN-GP is a more expressive model.

3.4.7 Interpreting IMD

To understand how IMD operates, we investigate the behavior of heat kernel traces of different datasets that are normalized by a null model. Tsitsulin et al. [210] proposed a normalization by the heat kernel trace of an empty graph, which amounts to taking the average, rather than the sum, of the original heat kernel diagonal. However, this normalization is not an appropriate null model as it ignores graph connectivity. We propose a heat kernel normalization by the *expected* heat kernel of an Erdős-Rényi graph.

For the purpose of normalizing IMD, we need to approximate that graph's eigenvalues. Coja-Oghlan [52] proved that $\lambda_1 \leq 1 - c\bar{d}^{-1/2} \leq \lambda_2 \leq \lambda_n \leq 1 + c\bar{d}^{-1/2}$ for the core of the graph for some constant c. We have empirically found that $c = 2$ provides a tight approximation for random graphs. That coincides with the analysis of Chung et al. [45], who proved that

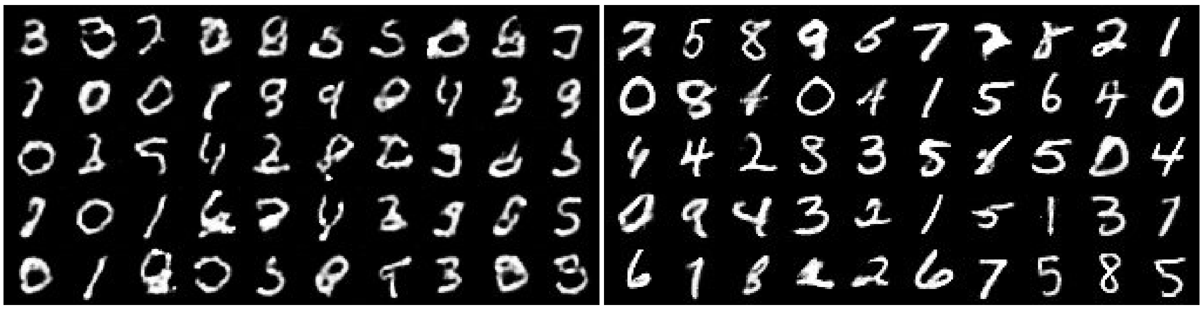


Fig. 3.12 MNIST samples (left: WGAN, right: WGAN-GP)

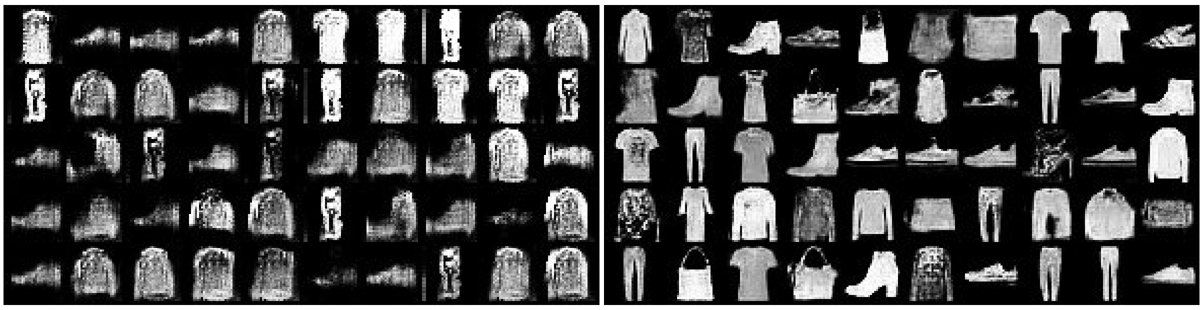


Fig. 3.13 FashionMNIST samples (left: WGAN, right: WGAN-GP)

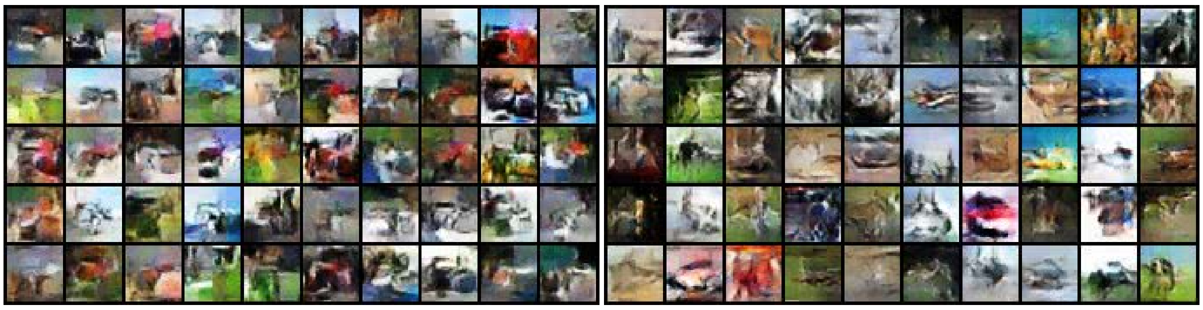


Fig. 3.14 CIFAR10 samples (left: WGAN, right: WGAN-GP)

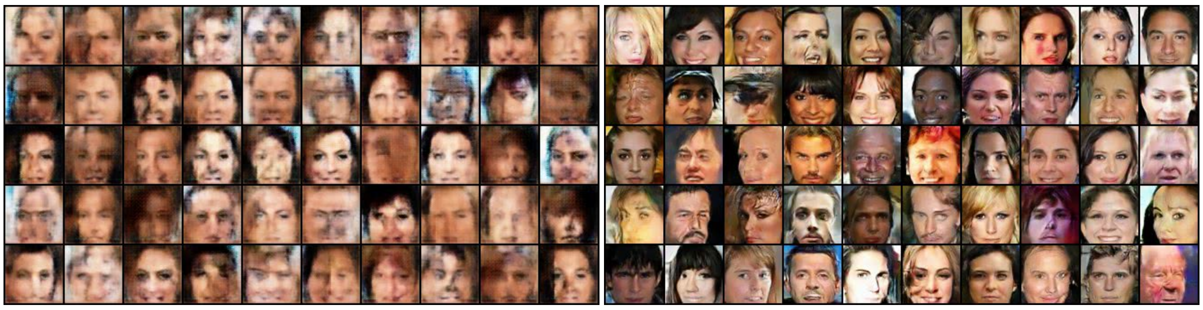


Fig. 3.15 CelebA samples (left: WGAN, right: WGAN-GP)

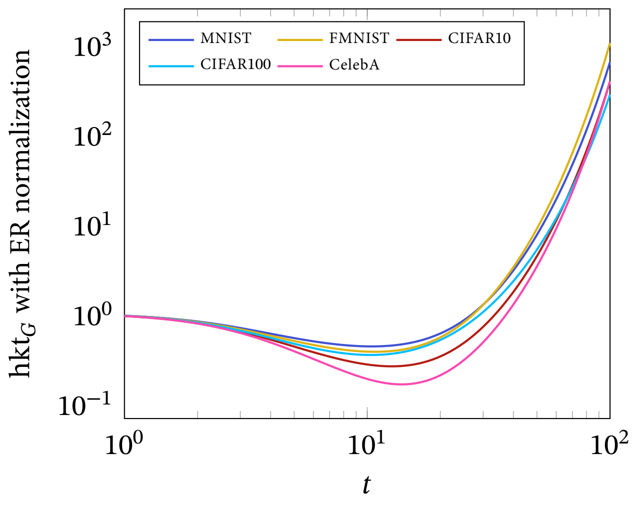


Fig. 3.16 Plotting the normalized heat trace allows interpretation of medium- and global-scale structure of datasets. Best viewed in color.

$\lambda_n = (1 + o(1))2\bar{d}^{-1/2}$ if $d_{\min} \gg \sqrt{\bar{d}} \log^3 n$ even though in our case $d_{\min} = \bar{d} = k$. We thus estimate the spectrum of a random Erdős-Rényi graph as growing linearly between $\lambda_1 = 1 - 2\bar{d}^{-1/2}$ and $\lambda_n = 1 + 2\bar{d}^{-1/2}$, which corresponds to the underlying manifold being two-dimensional [210].

Figure 3.16 depicts the obtained normalized hkt_g for all datasets we work with. We average results over 100 subsamples of $10k$ images each. For $t = 10$, i.e., at a medium scale, CelebA is most different from the random graph, while for large-scale t values, which capture global community structure, $\frac{\mathrm{dhkt}_g(t)}{\mathrm{d}t}$ reflects the approximate number of clusters in the data. Surprisingly, CIFAR100 comes close to CIFAR10 for large t values; we have found that this is due to the fact that the pre-trained Inception network does not separate the CIFAR100 data classes well enough. We conclude that the heat kernel trace is interpretable if we normalize it with an appropriate null model.

3.4.8 Verifying Stability and Scalability of IMD

In terms of scalability, Figure 3.17 (right) shows that the theoretical complexity is supported in practice. Using approximate kNN, we break the $\mathcal{O}(n^2)$ performance of KID. FID's time complexity appears constant, as its runtime is dominated by the $\mathcal{O}(d^3)$ matrix square root operation. Geometry score (GS) fails to perform scalably, as its runtime grows exponentially. Due to this prohibitive computational cost, we eschew other comparison with GS. Furthermore, as IMD distance is computed through a low-dimensional heat trace representation of the manifold, we can store HKT for future comparisons, thereby enhancing performance in the case of many-to-many comparisons.

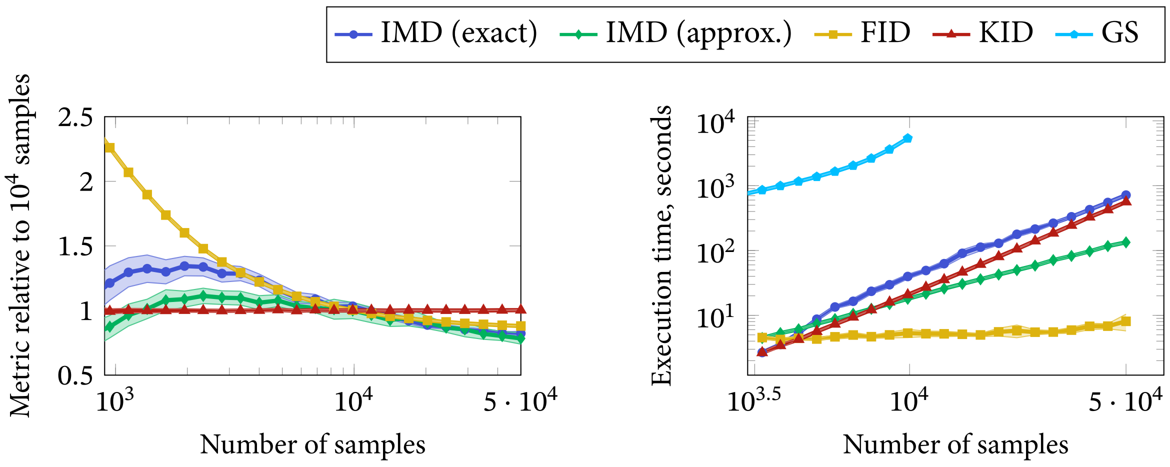


Fig. 3.17 Stability and scalability experiment: *(left)* stability of FID, KID and IMD wrt. sample size on CIFAR10 and CIFAR100 dataset; *(right)* scalability of FID, KID and IMD wrt. sample size on synthetic datasets.

3.5 Summary

We introduced IMD, a geometry-grounded, first-of-its-kind intrinsic multi-scale method for comparing unaligned manifolds, which we approximate efficiently with guarantees, utilizing the Stochastic Lanczos Quadrature. We have shown the expressiveness of IMD in quantifying the change of data representations in NLP and image processing, evaluating generative models, and in the study of neural network representations. Since IMD allows comparing diverse manifolds, its applicability is not limited to the tasks we have evaluated, while it paves the way to the development of even more expressive techniques founded on geometric insights.

Chapter 4

Approximating Spectral Distances for Web-Scale Graphs

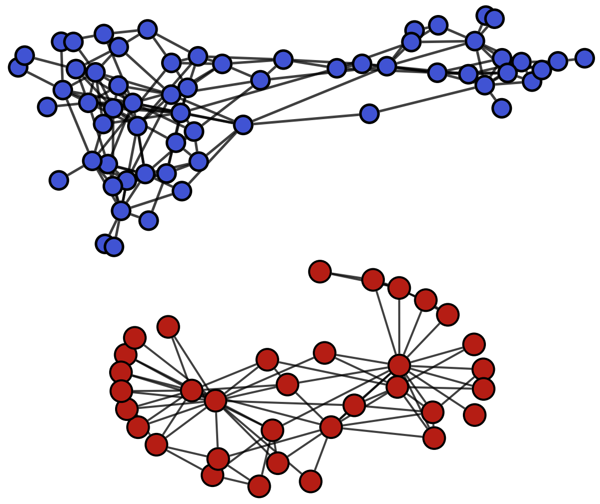
In the previous chapter, we applied Stochastic Lanczos Quadrature for calculating the proposed distance for sample comparison. In this chapter, we develop a practical algorithm for fast computation of spectral graph descriptors, based on the efficient computation provided by SLQ.

We start by describing spectral analysis tools applied in graph comparison. As their exact computation is infeasible in most large-scale settings, we discuss their approximate computation techniques, such as spectrum interpolation and Taylor expansion of the matrix functions. We then present a SLaQ algorithm and verify its efficiency for huge graphs.

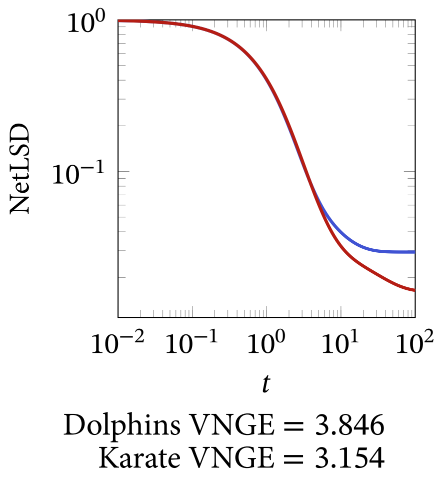
4.1 Introduction

Many complex systems, including social and biological, and interactions on the Web can be concisely modeled as graphs. Solving data mining tasks such as classification or anomaly detection on graphs requires sophisticated techniques. However, if one has a meaningful notion of similarity between two graphs, classic data mining techniques can be effortlessly applied to graphs. As many real-world graphs are huge (millions of nodes and edges), recent research [123, 211, 39] has focused on providing *scalable* graph distances.

Spectral analysis provides powerful tools for graph clustering [223, 35, 157], alignment [158, 99], comparison [211, 39], and characterization of graphs [67, 97, 96, 213, 39]. In practice, however, the applicability of these methods is often limited by the scalability of eigendecomposition itself: it takes cubic time to compute all eigenvalues and eigenvectors of a given graph. Several graph comparison methods such as Von Neumann Graph Entropy (VNGE) [33, 39] and Network Laplacian Spectral Descriptor (NetLSD) [211] require only information derived from a graph's eigenvalues – not the full decomposition. Although they depend on less information, naïve computation of such metrics is just as expensive as a full eigendecomposition. To scale up for large graphs, these methods resort to low-order (two terms) Taylor expansion having loose bounds and poor empirical approximation performance. Few works discuss approximation accuracy or experiment on how it affects performance on downstream tasks.



(a) Dolphin and karate graphs



(b) Spectral descriptors

Fig. 4.1 Spectral analysis provides valuable insights in the structure of graphs. Can we scale it to billions of nodes?

In this chapter, we propose SLaQ, an approximation algorithm for computing spectral distances for very large graphs. By leveraging recent advances in numerical linear algebra [219, 3, 50, 60], we achieve state-of-the-art approximation accuracy in time linear in the number of graphs' edges.

We summarize the contributions of this Chapter as follows:

1. We introduce SLaQ, an efficient approximation technique for two spectral graph distances, VNGE and NetLSD.

2. We derive corresponding approximation error bounds and experimentally observe an average reduction in the approximation error of **30×–200×** over a diverse set of real-world graphs.

3. We demonstrate that faithful approximation is *necessary* for accurate graph comparison and current approximation techniques are unfit for accurate yet fast approximation.

4. We show that accurate computation of VNGE and NetLSD is possible for a graph with billions of nodes and edges on a single machine *in less than an hour*.

4.2 Preliminaries

Below, we review two techniques, NetLSD [211] and VNGE [33, 39], as spectral descriptors operating on two graph Laplacians, namely the *Laplacian* matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$ and the *normalized Laplacian* matrix $\mathscr{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{D}$ is the diagonal matrix with the degree of the node i as entry $\mathbf{D}_{ii}$. Both graph descriptors rely on the spectrum $\{\lambda_1, \dots, \lambda_n\}$ of the Laplacians, thus we will need their spectral factorization $\mathscr{L} = \Phi\mathbf{\Lambda}\Phi^\top$, where $\Phi = [\phi_1\phi_2 \dots \phi_n]$ contains eigenvectors ϕ_i and $\mathbf{\Lambda}$ is diagonal with the sorted eigenvalues $\lambda_1 \leq \dots \leq \lambda_n$. We note that although NetLSD and VNGE operate on different Laplacian matrices, for convenience's sake we will refer to the spectrum of both matrices $(\mathscr{L}, \mathbf{L})$ as λ_i.

Spectrum of the Normalized Laplacian $\mathscr{L}$ is bounded to $[0; 2]$, for the unnormalized counterpart $\mathbf{L}$ it is $[0; 2\max_i \mathbf{D}_{ii}]$ as per the Gershgorin circle theorem [79, 83]. The two Laplacians each reflect different properties of a graph. For example, the normalized Laplacian can not distinguish number of edges [46] (as it operates on local densities), however its second eigenvector can be used to optimize normalized cut size [192].

4.2.1 Von Neumann Graph Entropy

In the Standard Quantum Mechanics model, the state of a quantum mechanical system associated with the n-dimensional Hilbert space is identified with a $n \times n$ positive semidefinite, trace-one, Hermitian *density matrix*. Von Neumann entropy [224] is a quantitative measure of mixedness of this density matrix, and is defined as follows:

Definition 3. ***Von Neumann Graph Entropy*** *$\mathcal{H}$ is defined as $\mathcal{H} = -\sum_i \lambda_i \ln \lambda_i$. VNGE is completely determined by the spectrum.*

By convention, $0 \log 0 = 0$. Braunstein et al. [33] reinterprets the graph Laplacian matrix $\mathscr{L}$ as a quantum mechanical system and introduces Von Neumann Graph Entropy (VNGE) by scaling graph Laplacian $\mathscr{L}$ by its trace to get the density matrix $\mathbf{P} = \frac{1}{\mathrm{tr}(\mathscr{L})}\mathscr{L}$. Scaling the Laplacian does not affect the shape of its spectrum, as each eigenvalue is simply multiplied by $\frac{1}{\mathrm{tr}(\mathscr{L})}$.

VNGE is related to the centralization of graphs [194], yet, general structural interpretation of this measure is unknown mainly due to the lack of accurate scalable approximation [97, 146].

4.2.2 Network Laplacian Spectral Descriptors

Tsitsulin et al. [211] introduced NetLSD as a spectral distance between graphs grounded in differential geometry. Instead of directly operating on the Laplacian matrix, NetLSD is defined in terms of the *heat kernel* of a graph. The heat equation associated with the Laplacian is:

$$\frac{\partial \mathbf{u}_t}{\partial t} = -\mathscr{L}\mathbf{u}_t, \tag{4.1}$$

where $\mathbf{u}_t \in \mathbb{R}^n$ is a vector representing the heat of each vertex at time t. The solution to the heat equation provides the heat at each vertex at time t, when the initial heat $\mathbf{u}_0$ is initialized with the same value on all vertices. Its closed-form solution is given by the *heat kernel* matrix $\mathbf{H}_t \in \mathbb{R}^{n\times n}$ that can be computed by directly exponentiating the Laplacian in the spectral domain [46]:

$$\mathbf{H}_t = e^{-t\mathscr{L}} = \Phi e^{-t\mathbf{\Lambda}}\Phi^\top = \sum_{i=1}^{n} e^{-t\lambda_i}\phi_i\phi_i^\top, \tag{4.2}$$

where $\mathbf{H}_{ij}$ represents the amount of heat transferred from vertex v_i to vertex v_j at time t.

The *trace* of the heat kernel matrix provides a useful lower bound on the Gromov-Wasserstein distance between the underlying manifolds [142]:

$$\mathbf{h}_t = \operatorname{tr}(\mathbf{H}) = \sum_i e^{-t\lambda_i}. \tag{4.3}$$

Then the NetLSD representation is a **heat trace signature** of graph G, i.e., a collection of heat traces at different *time scales*, $\mathbf{h}(G) = \{\mathbf{h}_t\}_{t>0}$. In practice, t is sampled on a logarithmically spaced grid, so $\mathbf{h}(G) \in \mathbb{R}^d$ is a vector of some small fixed dimensionality d.

4.2.3 Approximation Methods

Both VNGE and NetLSD can be represented as a function $\operatorname{tr} f(\mathbf{\Lambda})$ of Laplacian eigenvalues. A naïve approach would be to compute the exact eigenvalues and compute that function as $\sum_i f(\lambda_i)$, however, as we mentioned before, the computational complexity of full eigendecomposition is $\mathcal{O}(n^3)$, which is infeasible for large n.

Below we review approximation techniques which have been proposed in the literature [97, 39, 211]. We empirically evaluate their approximation performance in Section 4.4.1.

4.2.4 Taylor Expansion

A natural impulse for dealing with complex matrix functions is to approximate the function with first few terms of its Taylor expansion. Even though it is known that Taylor expansion provides an unreliable approximation of matrix functions [149], both NetLSD and VNGE rely on this approximation [211, 39, 97, 146], as the first two Taylor terms can be computed in $\mathcal{O}(m)$.

NetLSD's expansion depends on the parameter t, and its approximation is reasonable for only small t values [211]:

$$h_t = \sum_i e^{-t\lambda_i} = \sum_{k=0}^{\infty} \frac{\operatorname{tr}((-t\mathscr{L})^k)}{k!} \approx n - t \operatorname{tr}(\mathscr{L}) + \frac{t^2}{2} \operatorname{tr} \mathscr{L}^2. \tag{4.4}$$

The expansion used in VNGE is slightly different [39, 97, 146]:

$$\mathcal{H} = \sum_i \lambda_i \ln \lambda_i \approx 1 - \frac{1}{\operatorname{tr}(\mathscr{L})^2}(\operatorname{tr}(\mathscr{L}) + 2\operatorname{tr}(\mathscr{L}^2)). \tag{4.5}$$

These first two terms are easily computed, even for very large graphs, as $\operatorname{tr}(\mathscr{L}) = n$ and $\operatorname{tr}\left(\mathscr{L}^2\right) = \sum_{ij} \mathscr{L}_{ij}{}^2$ since $\mathscr{L}$ is self-adjoint, and the error rate of the Taylor expansion of the matrix exponential depends on the largest eigenvalue of the matrix [149].

Chen et al. [39] introduce two approximation algorithms for VNGE based on a two-term Taylor expansion, FINGER-$\overline{\mathcal{H}}$ and FINGER-$\widehat{\mathcal{H}}$:

$$\begin{aligned} &\mathcal{Q} = 1 - \frac{1}{\operatorname{tr}(\mathscr{L})^2}(\operatorname{tr}(\mathbf{D})^2 + 2\operatorname{tr}(\mathscr{L}^2)),\\ &\text{FINGER}-\overline{\mathcal{H}} = -\mathcal{Q}\ln\left(\frac{2\max \mathbf{D}}{\operatorname{tr}(\mathscr{L})^2}\right), \qquad \text{FINGER}-\widehat{\mathcal{H}} = -\mathcal{Q}\ln\left(\lambda_{\max}\right). \end{aligned}$$

4.2.5 Spectral Interpolation

We conclude by noting that the Taylor expansion is useful on very large graphs, on which computing any part of the spectrum is prohibitive. For manageable graph sizes, NetLSD adopts a more accurate strategy based on approximating the eigenvalue growth rate, adapted from [221]. It takes $\mathcal{O}(km + k^2n)$ to compute k extremal eigenvalues of a graph [83], thus it is possible to compute k eigenvalues on both ends of the spectrum, and interpolate a *linear growth* of the interior eigenvalues.

While Tsitsulin et al. [211] do not provide approximation guarantees of their method, it is easy to see that the worst-case scenario is the graph with exactly k isolated nodes and a fully connected component having $n-k$ nodes, meaning $\lambda_{:k} = 0$ and $\lambda_{k:} = 2$. Then, absolute error in the approximation of h_t becomes $\|n - 2k - \sum_{i=0}^{n-2k} \frac{2(i-k)}{n-2k}\|$. This bound is very loose; we further verify that the approximation accuracy of the linear interpolation strategy is poor in the Section 4.4.1 and that it does not scale to very large graphs in the Section 4.4.5.

4.3 SLaQ

As noted above, the main approximation techniques that have been proposed for VNGE and NetLSD have limited guarantees on their approximation quality, and these weak guarantees have not been fully explored in the literature. In this section, we address these deficiencies and propose our method for improved approximation of spectral distances between graphs.

Setting aside computational infeasibility of the naïve eigenvalues calculation, loose Taylor expansion error bounds and linear interpolation heuristics, we attain theoretically guaranteed accuracy and speed by means of Stochastic Lanczos Quadrature (SLQ) [219] described in detail in Section 3.3.4. As both spectral distances in question are essentially functions of matrices, we use f to denote the trace of matrix exponential $f(\mathscr{L}) = \exp(-t\mathscr{L})$ for NetLSD or the matrix logarithm $f(\mathscr{L}) = -\mathscr{L}\log\mathscr{L}$ for VNGE. We now give a brief reminder of SLQ computation, which falls into two parts. First goes a randomized trace estimation:

$$\operatorname{tr}(f(\mathscr{L})) = \mathbb{E}_{p(\mathbf{v})}(\mathbf{v}^\top f(\mathscr{L})\mathbf{v}) \approx \frac{n}{n_v}\sum_{i=1}^{n_v}\mathbf{v}_i^\top f(\mathscr{L})\mathbf{v}_i,$$

Second — an analogue of Gaussian Quadrature with nodes ω_k and weights θ_k computed by Lanczos algorithm:

$$\mathbf{v}_i^\top f(\mathscr{L})\mathbf{v}_i = \mathbf{v}_i^\top \Phi f(\mathbf{\Lambda})\Phi^\top \mathbf{v}_i = \sum_{j=1}^{n} f(\lambda_j)\mu_j^2 = \int_a^b f(t)d\mu(t) \approx \sum_{k=1}^{m}\omega_k f(\theta_k),$$

For the matrix exponential used in NetLSD, bounds in (3.15) suggest that we do not need many Lanczos steps s to achieve error ϵ. Another source of error lies in the Monte Carlo estimation of the trace as a mean of the quadratic forms $\mathbf{v}^\top f(\mathscr{L})\mathbf{v}$. To reduce the variance of the estimate, we apply the 2-term polynomial variance reduction technique (see Section 3.3.6). Although matrix-vector product approximation bounds for the matrix exponential have been well studied [105], analogous error estimates for the von Neumann entropy remain an open problem in numerical linear algebra. We summarize the overall SLaQ method in Algorithm 4 for both LSD and VNGE.

4.4 Experiments

We evaluate SLaQ against all approximation methods proposed in [39, 211], in addition to the exact computation of the spectrum (where allowed by the graph size). We perform our experiments on the Google Cloud's `c2-standard-60` virtual machine with 60 virtual cores and

Algorithm 3 SLaQ algorithm

```
1:  function SLaQ _LSD($G, s, n_v$)
2:      $\mathscr{L} \leftarrow$ Laplacian($G$)
3:      descriptor $\leftarrow$ slq($\mathscr{L}, s, n_v, \exp(x)$)
4:      return descriptor
5:  function SLaQ _VNGE($G, s, n_v$)
6:      $\mathbf{P} \leftarrow$ DensityMatrix($G$)
7:      descriptor $\leftarrow$ slq($\mathbf{P}, s, n_v, x \ln(x)$)
8:      return descriptor
9:  function slq($\mathscr{L}, s, n_v$, fun)
10:     $\mathbf{T}$ = lanczos($\mathscr{L}, s, n_v$)                ▷ $\mathbf{T} \in \mathrm{R}^{n_v \times m \times m}$
11:     $\mathbf{\Lambda}, \mathbf{U} \leftarrow$ eigh($\mathbf{T}$)                ▷ eigendecomposition($\mathbf{T}$)
12:     return $\frac{1}{n_v} \sum_i^{n_v} \left( \sum_k^{s} (\mathrm{fun}(\lambda_k^i)[\mathbf{u}_{k,0}^i]^2) \right)$
```

240GB RAM, averaging 10 times for all experiments unless stated otherwise. We use LAPACK [9] as the linear algebra library of choice. We open-source the implementation[1].

Parameter settings. Unless otherwise mentioned, we evaluate SLaQ using $n_v = 100$ starting vectors and $s = 10$ Lanczos iterations. We provide an additional experimental investigation into parameter settings of SLaQ in Section 4.4.4. For the linear approximation of [211], we use the default ($k = 300$) eigenvalues from each end of the spectrum, following the notation of the original paper. Taylor series-based approximation techniques do not depend on any additional parameters.

Datasets. We use four types of graph collections to measure efficiency and effectiveness of SLaQ. First, we consider the accuracy of the method compared to other approximation techniques on the two subsets of graphs: synthetically generated Erdos-Renyi graphs and 73 graphs from the Network Repository[2] [180] with a number of nodes from 2500 up to 25000. In total, we use 27 biological, 12 interaction, 10 technological networks, 5 small web graphs, and 19 uncategorized networks (mostly, optimization problem graphs).

We follow up with large graphs to test ability of SLaQ to efficiently compute descriptors of Web-scale graphs. For that, we use five datasets[3]:

- DBLP [235] is a co-authorship network constructed from DBLP, a major online computer science bibliography resource.
- SNAP-Orkut [235] was an online social network.

[1]github.com/google-research/google-research/tree/master/graph_embedding/slaq
[2]networkrepository.com
[3]All but ClueWeb09 are from SNAP network collection, available at snap.stanford.edu

- LiveJournal [235] is an online blogging community where users can form friendships with each other.
- Friendster [235] was an online social network.
- ClueWeb09 [47, 180] is a web crawl from 2009.

	Size		**Statistics**	
dataset	$\lvert V\rvert$	$\lvert E\rvert$	Avg. deg.	Density
DBLP	317k	1.05M	6.62	2.08×10^{-5}
SNAP-Orkut	3.07M	117.2M	76.28	2.48×10^{-5}
LiveJournal	4M	34.7M	17.35	4.34×10^{-6}
Friendster	65.6M	1.8B	55.06	8.39×10^{-7}
ClueWeb09	4.8B	7.81B	3.27	6.83×10^{-10}

Table 4.1 Characteristics of large graphs used in this work: number of vertices $|V|$, number of edges $|E|$; average node degree; density defined as $|E|/\binom{|V|}{2}$.

	Size		**Temporal statistics**	
dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	$\lvert\mathcal{E}\rvert/\lvert T\rvert$
Wiki-nl	1M	20M	95	148337
Wiki-pl	1M	25M	95	182959
Wiki-it	1.2M	35M	95	250633
Wiki-de	2.1M	86M	95	553257

Table 4.2 Characteristics of dynamic graphs: total number of vertices $|V|$, total number of edges $|E|$; number of timestamps $|T|$; average incoming edges per timestamp $|\mathcal{E}|/|T|$.

Next, we investigate benefits of using SLaQ on dynamic Wikipedia link datasets in 4 different languages: Dutch (`nl`), Polish (`pl`), Italian (`it`), German (`de`). We obtained datasets from [47][4] and generated $|T|$ snapshots for every month in the original dataset.

Last, we verify that SLaQ's improvements in approximation enhance downstream task performance. We use three social network datasets and one from the field of bioinformatics[5]:

- D&D [59, 191, 116] is a dataset of protein structures. Each protein is represented by a graph of amino acids that are connected by an edge if they are less than 6 Ångstroms apart. The prediction task is to classify the protein structures into enzymes and non-enzymes.

[4] We used preprocessed version from KONECT repository, available at konect.uni-koblenz.de/networks/

[5] We obtained them at the graph kernel benchmark collection, available at ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

- COLLAB [234, 129, 116] is a collection of collaboration ego-networks of different researchers derived in [234] from three datasets introduced in [129]. The task is to determine whether the ego-collaboration graph of a researcher belongs to High Energy, Condensed Matter or Astrophysics field.

- Reddit-5k, Reddit-12k [234, 116] are datasets derived from Reddit, an online aggregation and discussion website. Discussions on Reddit are organized into different subcommunities; the task is to determine the community given the structure of the discussion graph.

			Vertices $\|V\|$		
dataset	$\|G\|$	$\|Y\|$	Min.	Avg.	Max.
D&D	1178	2	30	284.32	5748
COLLAB	5000	3	32	74.49	492
Reddit-5k	4999	5	22	508.52	3648
Reddit-12k	22939	11	2	391.41	3782

Table 4.3 Properties of the graph classification datasets used: number of graphs $|G|$; number of labels $|Y|$; minimum, average, and maximum number of nodes in graph collection.

4.4.1 Approximation Accuracy

We proceed with evaluation of SLaQ capacity to approximate matrix functions for graph comparison. We compute full spectrum of 73 small graphs and true values of NetLSD and VNGE and report the relative l_2 approximation error with respect to the true graph descriptor. Figure 4.2 demonstrates that SLaQ offers over 200× reduction in the average relative error for VNGE over FINGER techniques [39] and over 30× improvements over the linear interpolation technique from [211]. Figure 4.3 shows that SLaQ offers 250× reduction over the Taylor expansion and over 22× improvement over the linear interpolation [211].

We also compare how the approximation accuracy changes for NetLSD on the random Erdős-Rényi graphs. We generate random graphs of size 1000 with varying graph density (number of expected edges) p and random graphs of size $100 - 10000$ with the average degree $m/n = 10$. We report the results in the Figure 4.4. We observe that SLaQ's approximation accuracy is stable across the graph size both in terms of number of the nodes and graph density.

4.4.2 Benefits of Non-local Approximation

We verify that our method has a global view of the graph, i.e. is not dominated by only local information. In order to do that, we compute graph descriptors (NetLSD and VNGE) for monthly

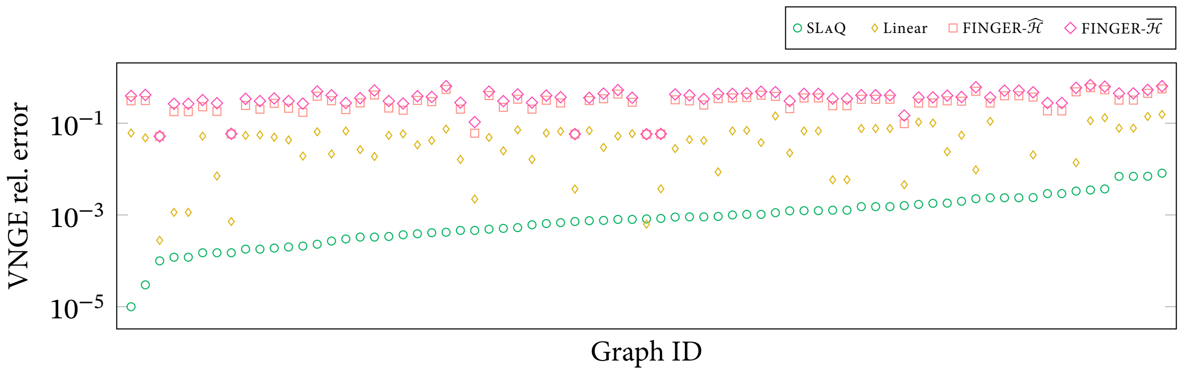


Fig. 4.2 SLaQ offers over **200×** reduction in average error for VNGE over techniques proposed in [39] and over **30×** improvement over the linear approximation from [211].

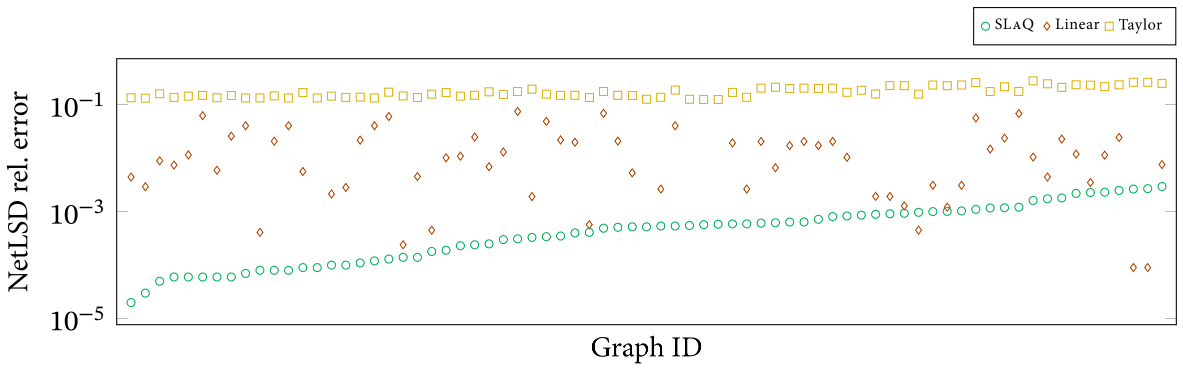


Fig. 4.3 SLaQ offers **22×** reduction in average error for NetLSD over [211] and **250×** over Taylor expansion.

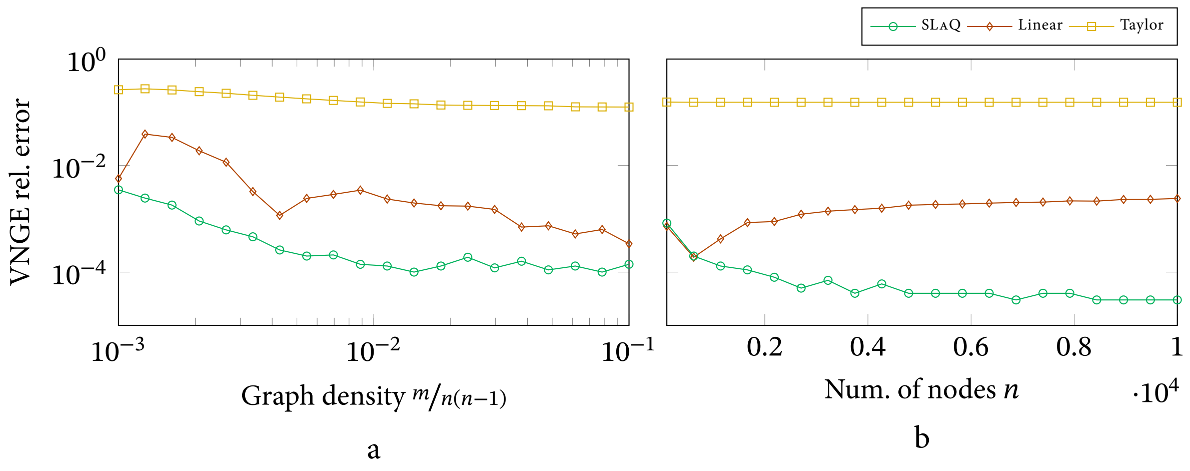


Fig. 4.4 Number of nodes and edges of random Erdős-Rényi graphs does not affect SLaQ's approximation accuracy.

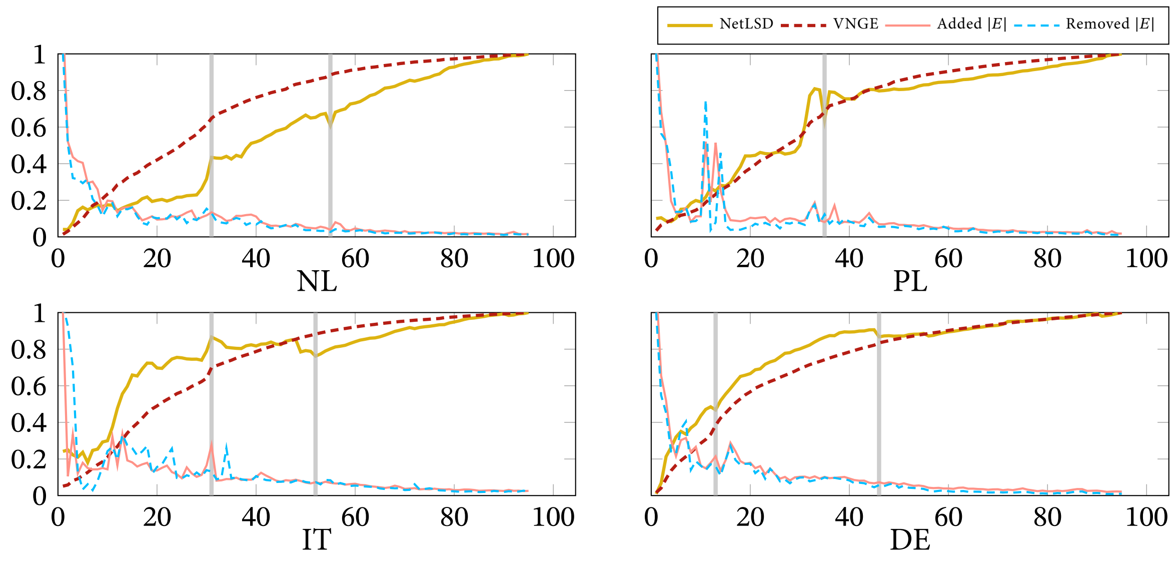


Fig. 4.5 SLaQ approximation of —— NetLSD and - - - - VNGE for Wikipedia graphs across time. Changes that are not explained by local edge differences highlighted in gray.

snapshots of dynamic Wikipedia link datasets from January 2003 to December 2010 (a total of $|T| = 96$ snapshots) and report their change as well as the number of edges added/removed each month.

We plot the proportion of cumulative edge additions/deletions and distances between descriptor pairs of snapshots $(0, i)$, where $i \in 1, \ldots, |T|$. Figure 4.5 reports the distance values for each language as well as the relative number of incoming and outgoing edges per snapshot. We mark exmaples of anomalous spikes in NetLSD and VNGE that can not be explained simply by the edge additions and deletions. In these cases, simple approximations like 2-term Taylor expansion would fail to capture such changes.

4.4.3 Graph Classification Performance

We test our method in the supervised downstream task, by classify graphs in binary and multi-class settings. We compute NetLSD and VNGE descriptors for each of the graphs and use them as feature vectors in classification. Since these graph classification datasets allow direct calculation of the descriptor (maximum number of nodes reported in the Table 4.3 is 5748), we can analyze how approximation affects the downstream accuracy.

We use a non-parametric 1-Nearest Neighbor classification algorithm and repeat classification task using 80/20 training/testing split 1000 times to minimize the biases introduced by the random splitting and the learning algorithm. We report the classification accuracy in the Table 4.4. We believe that this relates to the issues with these datasets pointed out by [189, 36]:

	VNGE					NetLSD			
dataset	FINGER-$\widetilde{\mathcal{H}}$	FINGER-$\widehat{\mathcal{H}}$	Linear	SLaQ	**Exact**	Taylor	Linear	SLaQ	**Exact**
D&D	63.01	66.38	68.13	65.53	**66.40**	67.01	67.98	66.77	**67.24**
COLLAB	64.95	65.10	55.90	49.04	**58.03**	61.81	65.17	58.76	**63.48**
Reddit-5k	30.87	29.85	31.31	31.77	**31.43**	33.67	32.01	35.48	**35.63**
Reddit-12k	16.53	16.20	17.18	17.04	**16.79**	22.67	21.30	25.31	**25.52**

Table 4.4 1-Nearest neighbour graph classification performance on 4 datasets with VNGE and NetLSD. Exact computation results are in bold. Approximations that are close to or better than the exact metric computation are highlighted in green.

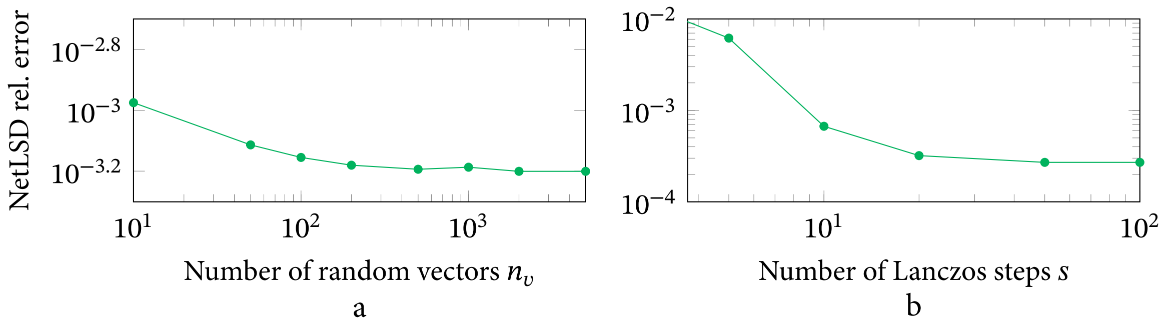


Fig. 4.6 Parameter sensitivity of SLaQ in terms of approximating NetLSD with (a) different number of starting vectors n_v and (b) different number of Lanczos steps s. Error averaged across 73 graphs from the Network Repository.

simple local graph features achieve almost state-of-the-art performance [7]. However, for the Reddit datasets the improvement given by more accurate approximation is as expected due to the task being more sensitive to global structural information rather than simple node-level statistics.

4.4.4 Parameter Sensitivity

We investigate the approximation accuracy of SLaQ with respect to its hyperparameters: number of random starting vectors n_v and the number of Lanczos iterations s. Recall that the error bounds in the Section 4.3 tells us that there are two sources of error in SLaQ: one of the Monte Carlo estimation of the quadratic form and one of the Lanczos process. We measure the relative error ratio on the same 73 medium-sized graphs used in the Section 4.4.1 with respect to the number of random starting vectors n_v and the number of Lanczos iterations s and report the results in the Figure 4.6.

As expected, given enough starting random vectors SLaQ only needs few Lanczos iterations; the default setting of $s = 10$ gives an average error of 6.7×10^{-4}. As for the number of random vectors n_v, we do observe that increasing the number improves performance, but the improvement given by increasing n_v is much slower.

4.4.5 Scalability

We measure the runtime of all approximation techniques on huge graphs with millions of nodes and billions of edges and show that SLaQ is able to process very large graphs on a single machine within reasonable time while offering orders of magnitude better approximation, as measured in the Section 4.4.1. Table 4.5 only reports the results for VNGE, as the results for NetLSD for Linear interpolation and SLaQ approximation are similar to VNGE counterparts, while Taylor approximation works in the same time as FINGER-$\overline{\mathcal{H}}$. As FINGER-$\overline{\mathcal{H}}$ only sums the weights of

dataset	FINGER-$\overline{\mathcal{H}}$	FINGER-$\widehat{\mathcal{H}}$	Linear	SLaQ
DBLP	0.06	0.65	394	28.4
SNAP-Orkut	1.68	70	8863	899
LiveJournal	0.97	15.6	4727	476
Friendster	1.67	71	OOM	900
ClueWeb09	902	OOM	OOM	3447

Table 4.5 Running time (in seconds) of different approximation techniques and SLaQ for VNGE on large graphs.

graphs' edges, it serves as a baseline on how much time it takes to scan the edges of a graph. A more useful comparison is FINGER-$\widehat{\mathcal{H}}$, as it reflects the time to compute a *single* eigenvalue of a graph. SLaQ approximates the *whole* spectrum at the cost of increased time complexity, however, the largest dataset with almost 5 billion nodes is processed in less than an hour.

4.5 Summary

We propose SLaQ, an approximation technique for fast computation of spectral graph distances, VNGE and NetLSD, leveraging state-of-the-art linear algebra methods. We show that faithful approximation of the graph distance is critical for good downstream task performance and those approximation methods previously introduced in the literature do not offer good approximation quality. SLaQ improves approximation errors of such baseline solutions by at least an order of magnitude averaged across 73 real-world graphs. As SLaQ's computation is linear in the number of edges of graphs, scalability of our method is on par with approximation techniques introduced for VNGE and NetLSD. To our knowledge, this is the first work that allows accurate comparison of billion-size graphs on a single machine in less than an hour.

Chapter 5

Anytime Graph Embeddings

In the previous chapters, we used quadratures and numerical linear algebra algorithms for eigenvalue problems to estimate quantities useful across machine learning. In this chapter, we use matrix sketching, another efficient numerical linear algebra tool for compressing information stored in large matrices, for low-dimensional representations of graph's nodes.

We start by describing existing approaches and the trade off between optimality and scalability of node embeddings, establishing their desirable properties. We then argue that matrix sketching algorithms, *anytime* in nature, are more apt for node embeddings and justify this claim by analysing implicit objectives for node embeddings and computational setting typically encountered in graph summarization problems, e.g., streaming data, parallelization and distribution.

5.1 Introduction

Low-dimensional representations, or *embeddings*, of graph's nodes provide a multi-purpose tool for performing data science tasks such as community detection, link prediction, and node classification. Neural embeddings [167, 93, 205, 212], computed by unsupervised representation learning over *nonlinear* transformations, outperform their linear counterparts [161, 245] in task performance, and achieve scalability via sampling a node similarity matrix, such as Personalized PageRank (PPR); however, such neural methods lack theoretically grounded error guarantees with respect to their objectives. The theoretically most well-grounded state-of-the-art method, NetMF [170], performs Singular Value Decomposition (SVD) on a dense matrix of nonlinear node similarities, and achieves the global optimum of its objective by virtue of the properties of SVD.

However, this *optimality* comes to the detriment of *scalability*, as NetMF needs to precompute the similarity matrix and store it in memory at cost quadratic in the number of nodes. An ideal method should achieve both quality and scalability.

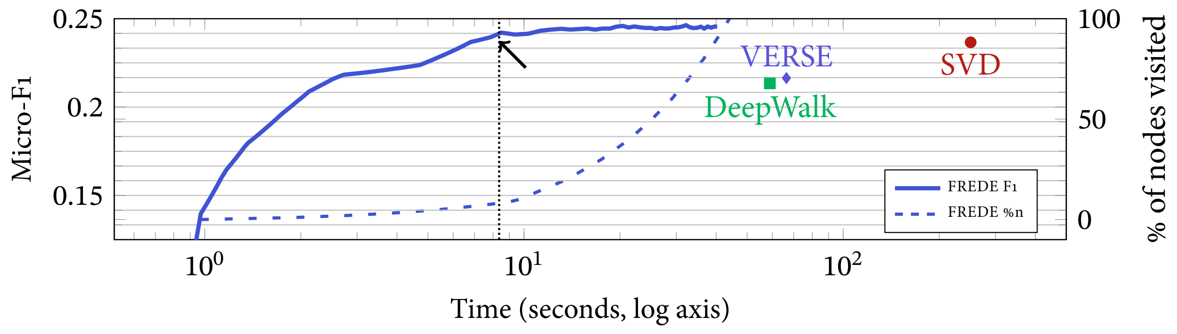


Fig. 5.1 FREDE scalably produces an embedding *at any time*; at the black arrow, it outperforms the SVD of a full PPR-like similarity matrix in 3% of the latter's runtime after processing 10% of matrix rows. (PPI data)

In this chapter, we propose FREDE, the first, to our knowledge, linear-space algorithm that produces embeddings with *quality guarantees* from a nonlinear transform. We observe that factorization-based embeddings effectively strive to preserve the covariance of a similarity matrix, and that a few nodes acting as oracles approximate the distances among all nodes with guarantees [208]. Given these observations, we adapt a covariance-preserving matrix sketching algorithm, *Frequent Directions* (FD) [135, 82], to produce a graph embedding by factorizing, on a per-row basis, a PPR-like node similarity matrix derived by interpreting a state-the-art neural embedding, VERSE [212], as matrix factorization. FREDE can be distributed, as it inherits the *mergeability* property of FD: two embeddings can be computed independently on different node sets and merged to a single embedding, with quality guarantees that hold *anytime* [248], even after accessing a subset of similarity matrix rows. Figure 5.1 shows that FREDE outperforms SVD in a node classification task after processing 10% of nodes represented as similarity matrix rows.

We summarize contributions of this Chapter as follows:

1. We interpret a state-of-the-art graph embedding method, VERSE, as factorizing a transformed PPR similarity matrix;
2. We propose FREDE, an *anytime* graph embedding algorithm that minimizes covariance error on that PPR-like matrix via *sketching*, with space complexity linear in the number of nodes and time linear in the number of processed rows;
3. In a thorough experimental evaluation with real graphs we confirm that FREDE is competitive against the state of the art and scales to large networks.

5.2 Preliminaries and Related Work

Our work builds on the know-how of matrix sketching to derive scalable, anytime graph embeddings for practical data science tasks. Here, we outline previous work on graph embeddings and the fundamentals of matrix sketching.

5.2.1 Problem Setting

The algorithms for local node embeddings rely on the adjacency matrices rather than Laplacians, which we considered in previous chapters for global graph embeddings. Let us first remind some notation. A graph $G = (V, E)$ with n vertices $V = (v_1, \dots, v_n)$, $|V| = n$, and edges $E \subseteq V \times V$, $|E| = m$, is represented by an *adjacency matrix* A for which $\mathrm{A}_{ij} = 1$ if $(i, j) \in E$ is an edge between node i and node j, otherwise $\mathrm{A}_{ij} = 0$. D is the diagonal matrix with the degree of node i as entry $\mathrm{D}_{ii} = \sum_{j=1}^{n} \mathrm{A}_{ij}$. The *normalized adjacency matrix*, $\mathrm{P} = \mathrm{D}^{-1}\mathrm{A}$, represents the transition probability from a node to any of its neighbors. We represent interactions among nodes with a *similarity matrix* $\mathrm{S} \in \mathbb{R}^{n \times n}$ [212, 161, 245]. The row i of A is denoted as A_i. The embedding problem is to find an $n \times d$ matrix W that retains most information in S. The formal criteria for this unsupervised representation learning problem vary from method to method. In what follows we group these methods by their objectives.

	Solution				Computation			Complexity	
method	Nonlinear	Closed-form	Error-bounded	Versatile	Frugal	Anytime	Mergeable	Space	Time
DeepWalk	✔	✘	✘	✘	✔	✘	✘	$\mathcal{O}(dn)$	$\mathcal{O}(dn \log n)$
Node2vec					✘			$\mathcal{O}(n^3)$	$\mathcal{O}(dnb)$
LINE	✔	✘	✘	✘	✔	✘	✘	$\mathcal{O}(dn)$	$\mathcal{O}(dnb)$
HOPE	✘	✔	✔	✔	✔	✘	✘	$\mathcal{O}(dn)$	$\mathcal{O}(d^2m)$
AROPE	✘	✔	✘	✔	✔	✘	✘	$\mathcal{O}(dn)$	$\mathcal{O}(dm+d^2n)$
VERSE	✔	✘	✘	✔	✔	✘	✘	$\mathcal{O}(dn)$	$\mathcal{O}(dnb)$
ApproxPPR	✘	✔	✘	✘	✔	✘	✘	$\mathcal{O}(dn)$	$\mathcal{O}((dm + d^2n) \log n)$
NRP		✘							
NetMF	✔	✔	✔	✔	✘	✘	✘	$\mathcal{O}(n^2)$	$\mathcal{O}(dn^2)$
NetSMF	✔	✔	✘	✔	✘	✘	✘	$\mathcal{O}(Tm \log n)$	$\mathcal{O}(dTm \log n)$
FREDE (ours)	✔	✔	✔	✔	✔	✔	✔	$\mathcal{O}(dn)$	$\mathcal{O}(dn^2)$

Table 5.1 Comparison of works in terms of fulfilled (✔) and missing (✘) desiderata; complexities in terms of number of nodes n and edges m, dimensionality d, context size T, and number of negative samples b, assuming a sparse graph.

5.2.2 Neural Embeddings

Initial works on graph embeddings relied on training a neural network to produce vector representations of a graph's nodes; DeepWalk [167] transferred such methods from words [145, 130] to graphs, utilizing a corpus of random walks. LINE [205] extended DeepWalk by exploiting graph edges rather than walks; Node2vec [93] customized random walk generation; and VERSE [212] generalised this approach to a method that preserves any similarity measure among nodes, with Personalized PageRank (PPR) [163] as the default option. Such neural embeddings leverage paths around a node, reach scalability via sampling, and provide no closed-form solution and no quality guarantees; we call them *positional* embeddings. In another neural approach, *structural* embeddings [5, 176, 181] leverage complex graph structural patterns to improve quality at the expense of scalability; however, positional embeddings outperform their structural counterparts in both quality and scalability. For these reasons, we exclude structural embeddings from our discussion.

5.2.3 Factorization-based Embeddings

Other works cast the problem of embedding a graph's nodes as one of exact or approximate factorization of the node similarity matrix, which is meant to minimize the *reconstruction error* [170]:

Definition 4 (Reconstruction error)**.** *The* reconstruction error *between matrices* S *and* $\tilde{\mathrm{S}}$ *is the Frobenius norm of the difference among the* S *and* $\tilde{\mathrm{S}}$*, i.e.,* $||\mathrm{S}-\tilde{\mathrm{S}}||_F^2 = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(\mathrm{S}_{ij}-\tilde{\mathrm{S}}_{ij})^2}$.

In the case of *symmetric* S, there exists an eigendecomposition $\mathrm{S} = \mathrm{U}\Lambda\mathrm{U}^\top$, and the optimal rank-$k$ approximation of S is $[\mathrm{S}]_k = \mathrm{W}\mathrm{W}^\top$, where the matrix $\mathrm{W} = \mathrm{U}_k\sqrt{\Lambda_k}$ is the product between the matrix of the first k eigenvectors, U_k, and a diagonal matrix of the square roots of the first k eigenvalues, Λ_k. On the other hand, in case S is *asymmetric*, the best rank-k approximation is obtained by the first k singular vectors and values of the Singular Value Decomposition (SVD) $\mathrm{S} = U\Sigma V^\top$, i.e., $[\mathrm{S}]_k = \mathrm{U}_k\Sigma_k\mathrm{V}_k^\top$, where U_k and V_k denotes the first k columns of U and V, respectively.

GraRep [37] applies SVD to factorize a concatenation of dense log-transformed DeepWalk transition probability matrices over different numbers of steps; yet it is neither scalable, nor provides quality guarantees. HOPE [161] overcomes the scalability drawback using a generalised form of SVD on special similarity matrices in the form AB^{-1}; it achieves optimality due to the guarantees of the Eckart–Young–Mirsky theorem, but its overall performance is hindered by the linearity of the underlying transform [212]. AROPE [245] applies spectral filtering on symmetric similarity matrices, forfeiting any guarantees. ApproxPPR [237] applies the randomized block Krylov SVD algorithm [155] on a truncated PPR matrix; NRP [237] iteratively reweights the resulting embedding vectors by coordinate descent; this post-hoc refinement boosts ApproxPPR embeddings [237] performance.

5.2.4 The Neural-Factorization Connection

Recent work has established a connection between neural and factorization-based embeddings. In NetMF [170], Qiu et al. extended an analysis of word embeddings [130] to connect matrix factorization and neural embeddings: under certain probability independence assumptions, DeepWalk, LINE, and Node2vec implicitly apply SVD on dense log-transformed similarity matrices. NetMF proposes novel closed-form solutions to compute such matrices with optimal error guarantees. For example, DeepWalk's objective is equivalent to SVD on the dense similarity matrix

$$\mathrm{S} = \log\left(\frac{m}{bT}\left(\sum_{r=1}^{T}\mathrm{P}^r\right)\mathrm{D}^{-1}\right), \tag{5.1}$$

where m is the number of edges, T is the random walk window size and b is the number of negative samples [170]. The d-dimensional DeepWalk embedding is obtained as $\mathrm{U}_d\sqrt{\Sigma_d}$, where U_d contains the d left singular vectors and Σ_d the first d singular values. However, this NetMF approach requires $\mathcal{O}(n^2)$ space to store S, a prohibitive complexity that hinders its application to graphs with more than 100 000 nodes. In another attempt, NetSMF [171], Qiu et al. sought to mitigate NetMF's scalability drawback by sparsifying the similarity matrix; however, matrix sparsification forfeits optimality guarantees, causing performance deterioration for effectual sparsity levels [171]; besides, the sparsified matrix has $\mathcal{O}(Tm\log n)$ nonzeros, hence it still yields quadratic growth.

5.2.5 Synoptic Overview

Below, we establish desiderata of the *solution*, its *computation*, and time/space *complexity* for graph embeddings, used to compare previous methods in Table 5.1.

- **nonlinear:** using nonlinear transforms; HOPE, AROPE, and ApproxPPR/NRP use linear transforms; nonlinearity is desirable, as linear dimensionality reduction methods fail to confer the advantages of their nonlinear counterparts in general [127, 212].

- **closed-form:** deriving the solution via an explicit formula without relying on heuristic learning components; only NetMF and NetSMF are both closed-form and nonlinear; NRP loses the closed-form character of ApproxPPR due to its performance-boosting post-hoc heuristic reweighting; such reweighting may augment any embedding with additional node degree information, yet it was only applied on ApproxPPR in [237].

- **error-bounded:** affording nontrivial, end-to-end error guarantees with respect to their objective; in principle, error-bounded methods, like HOPE and NetMF, are closed-form;

the reverse is not always the case, as some closed-form methods abandon guarantees for sake of scalability: AROPE by spectral filtering, ApproxPPR by truncating, and NetSMF by sparsifying the similarity matrix.

- **versatile:** accommodating diverse similarity measures; DeepWalk, LINE, Node2vec, and ApproxPPR/NRP lack versatility.
- **frugal** (space-efficient)**:** having worst-case space complexity subquadratic in the number of nodes.
- **anytime:** allowing the computation of a *partial* embedding whose quality improves as more nodes are processed.
- **mergeable:** allowing for a combination of embeddings on two node subsets that retains guarantees, hence enabling distributed computation [4].

5.2.6 Matrix Sketching

An alternative to SVD, *matrix sketching*, finds a low-dimensional matrix, or *sketch*, $W \in \mathbb{R}^{d \times t}$ of a matrix $M \in \mathbb{R}^{s \times t}$ (s elements, t features) that retains most of the information in M without striving for matrix reconstruction. Sketching methods operate in streaming fashion, guaranteeing quality when rows arrive one after another. A popular sketch objective [31, 32, 135, 231] is to preserve the *column covariance* $M^\top M$ of M, i.e., minimize covariance error:

Definition 5 (Covariance error). *The* column covariance error *is the normalized difference between the covariance matrices:*

$$\texttt{ce}_k(M, W) = \frac{\|M^\top M - W^\top W\|_2}{\|M - [M]_k\|_F^2} \geq \frac{\|M^\top M - W^\top W\|_2}{\|M\|_F^2} = \texttt{ce}(M, W).$$

The covariance error accounts for variance loss in each dimension. We use the lower bound $\texttt{ce}(M, W)$ in lieu of $\texttt{ce}_k(M, W)$ to avoid uneccessary parameter. When minimizing reconstruction error by SVD, with $[M]_k = U_k \Sigma_k V_k^\top$, we may also optimal covariance error, which depends on the singular value decay of M, by setting $W = \Sigma_k V_k^\top$. Since SVD is often computationally heavy, sketching algorithms typically provide error guarantees on ce by *row-wise* processing of M.

A desirable sketch property is *mergeability*:

Definition 6. Mergeability. *A sketching algorithm* `sketch` *is* mergeable *if there exists an algorithm* `merge` *that, applied on the $d \times t$ sketches,* $W_1 = \texttt{sketch}(M_1)$ *and* $W_2 = \texttt{sketch}(M_2)$, *of two* $\frac{s}{2} \times t$ *matrices,* M_1, M_2, *with* $\mathtt{ce}(M_1, W_1) \leq \epsilon$ *and* $\mathtt{ce}(M_2, W_2) \leq \epsilon$, *produces a* $d \times t$ *sketch* W *of the concatenated matrix* $M = [M_1; M_2]$, $W = \mathtt{merge}(W_1, W_2) = \mathtt{sketch}(M)$, *that preserves the covariance error bound* ϵ, *i.e.,* $\mathtt{ce}(M, W) \leq \epsilon$.

We now discuss some representative sketching algorithms.

Hashing. We construct a 2-universal hash function $h : [s] \rightarrow [d]$ and a 4-universal hash function $g : [s] \rightarrow \{-1, +1\}$. Starting with a zero-valued sketch matrix W, each row M_i is added to the $h(i)$-th sketch matrix row with sign $g(i)$: $W_{h(i)} = g(i) * M_i$, with complexity linear in matrix size, $\mathcal{O}(st)$. In practice, random assignment of rows is used instead of a hash function. Setting $d = \mathcal{O}(t^2/\epsilon^2)$, hashing achieves $\mathtt{ce} \leq \epsilon$ [231]. This sketch is trivially mergeable: $\mathtt{merge}(W_1, W_2) = W_1 + W_2$.

Random Projections are a fundamental data analysis tool [231]. [32] propose a row-streaming matrix sketching algorithm that randomly combines rows of the input matrix. In matrix form, $\tilde{M} = RM$, where the elements R_{ij} of the $d \times s$ matrix R are uniformly from $\{-1/\sqrt{d}, 1/\sqrt{d}\}$. For each row M_i, the algorithm samples a random vector $\mathbf{r}_i \in \mathbb{R}^d$ with entries in $\{-1/\sqrt{d}, 1/\sqrt{d}\}$ and updates $W = W + \mathbf{r}_i M_i^\top$. This sketch achieves $\mathtt{ce} \leq \epsilon$ with $d = \mathcal{O}(t/\epsilon^2)$, with practical performance exceeding the guarantee [133], and is mergeable with $\mathtt{merge}(W_1, W_2) = W_1 + W_2$.

Sampling. The Column Subset Selection Problem (CSSP) [31] is to select a small column subset of an entire matrix. In the row-update model, a solution is found by sampling scaled rows $M_i/\sqrt{dp_i}$ with probability $p_i = ||M_i||^2/||M||_F^2$. While the norm $||M||_F^2$ is usually unknown in advance, the method can work with d reservoir samplers, where d is the sketch size. This sketch achieves $\mathtt{ce} \leq \epsilon$ with $d = \mathcal{O}(t/\epsilon^2)$, yet the cost of maintaining reservoir samples is non-negligible. The sketch is mergeable if we use distributed reservoir sampling.

Frequent Directions (FD) [135], the current state of the art in sketching, extends the Misra-Gries algorithm [148] from frequent items to matrices and outperforms other methods [48, 31, 32] in quality. FD sketches a matrix by iteratively filling the sketch with incoming rows, performing SVD on it when it cannot add more rows, and shrinking the accumulated vectors with a low-rank SVD approximation. The complexity is $\mathcal{O}(dts)$, due to s/d iterations of computing the $\mathcal{O}(d^2 t)$ SVD decomposition of a $2d \times t$ matrix W with $d \ll t$. This sketch achieves $\mathtt{ce} \leq \epsilon$ when $d = \mathcal{O}(t/\epsilon)$ and is mergeable with

$$\mathtt{merge}(W_1, W_2) = \mathtt{FD}(\mathtt{concatenate}(W_1, W_2)). \tag{5.2}$$

The table below lists the embedding dimension d required to attain error bound $\mathtt{ce} \leq \epsilon \leq 1$ for different algorithms.

We observe that, by putting the node similarity matrix S in the role of the sketched matrix M, we can effectively turn a sketching technique to an embedding method. Indeed, recent

Algorithm	Hashing	RP	Sampling	FD
Dimension d	$\mathcal{O}(t^2/\epsilon^2)$	$\mathcal{O}(t/\epsilon^2)$	$\mathcal{O}(t/\epsilon^2)$	$\mathcal{O}(t/\epsilon)$

Table 5.2 Embedding dimension d required approximation error ϵ for sketching-based embeddings.

work [244] has adapted a sketching algorithm [222, 32] to graph embeddings, yet forfeited[1] its error guarantees. We apply the know-how of state-of-the-art matrix sketching to serve graph embedding purposes, leading to *anytime graph embeddings* with error guarantees.

5.3 Anytime Graph Embeddings

We observe that SVD-based graph embeddings, such as HOPE and NetMF, use only one of the two unitary matrices SVD produces, U and V. For example, NetMF returns $W = U_{:d}\sqrt{\Sigma}_{:d}$, with Σ truncated to d singular values. Therefore, such methods cannot reconstruct matrix S; SVD products U and Σ *may only* reconstruct the *row covariance matrix* $SS^\top = U\Sigma^2U^\top$, as $WW^\top = U\Sigma^2U^\top$, where $W = U\Sigma$; thus, such methods are better understood as implicitly minimizing the covariance error, rather than the reconstruction error [170], in relation to a similarity matrix among graph nodes.

Serendipitously, *sketching algorithms* aim to reconstruct the *column covariance* $S^\top S = V\Sigma^2V^\top$. Given this relationship, we apply a state-of-the-art *matrix sketching algorithm* in lieu of SVD to construct a graph embedding in *anytime* fashion, by *row updates* of any partially materializable similarity matrix S. Unfortunately, the matrix form of DeepWalk (Eq. 5.1) cannot be partially materialized. Next, we propose a partially materializable matrix based on Personalized PageRank (PPR), inspired from the VERSE [212] similarity-based embeddings. This choice attains good quality and time performance verified in the experiments. However, our method carries no prejudice with regard to the partially materializable matrix used; other choices are possible, such as, for example, the Node-Reweighted PageRank (NRP) [237]. Our aim is to illustrate the advantageous application of sketching for embedding purposes, while our framework supports any way of deriving the primary input matrix.

[1]In our experiments, we use a variant of [244] with error guarantees as a baseline.

5.3.1 A Row-wise Computable Similarity Matrix

VERSE [212] is the first similarity-based embedding method that does *not* require the entire matrix as input, as it allows for efficient row-wise computation; in its default version, it uses the PPR similarity measure:

Definition 7. *Given a starting node distribution* s*, damping factor* α*, and the transition probability matrix* P*, the PPR vector* $\mathrm{PPR}_{i\cdot}$ *is defined by the recursive equation:*

$$\mathrm{PPR}_i = \alpha s + (1-\alpha)\mathrm{PPR}_i^{\top}\mathrm{P}. \tag{5.3}$$

To compute PPR_i, we leverage the fact that the probability distribution of a random walk with restart converges to PPR_i vector [163, 19]. Following [130, 170] we show that, under mild assumptions, VERSE with PPR similarity virtually factorizes the log(PPR) matrix up to an additive constant.

Theorem 3. *Let* X *be the matrix of VERSE embeddings. If the terms* $z_{ij} = \mathbf{x}_i^{\top}\mathbf{x}_j$ *are independent, then VERSE factorizes the matrix* $\mathrm{Y} = \log(\mathrm{PPR}) + \log n - \log b = \mathrm{XX}^{\top}$.

Proof. Consider the VERSE objective function for the uniform sampling distribution and PPR similarity:

$$\mathscr{L} = \sum_{i=1}^{n}\sum_{j=1}^{n}\left[\mathrm{PPR}_{ij}\log\sigma(\mathbf{x}_i^{\top}\mathbf{x}_j) + b\mathbb{E}_{j\prime\sim\mathcal{Q}_i}\log\sigma(-\mathbf{x}_i^{\top}\mathbf{x}_{j\prime})\right],$$

where $\sigma(x) = (1+e^{-x})^{-1}$ is the sigmoid, $\mathcal{Q}_i$ is the noise sample distribution, and b the number of noise samples. Since PPR is right-stochastic and $\mathcal{Q}_i$ is uniform, i.e., $\Pr(\mathcal{Q}_i = j) = \frac{1}{n}$, we can separate the two terms as follows:

$$\mathscr{L} = \sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{PPR}_{ij}\log\sigma(\mathbf{x}_i^{\top}\mathbf{x}_j) + \frac{b}{n}\sum_{i=1}^{n}\sum_{j\prime=1}^{n}\log\sigma(-\mathbf{x}_i^{\top}\mathbf{x}_{j\prime}).$$

Then the individual loss term for vertices i and j is:

$$\mathscr{L}_{ij} = \mathrm{PPR}_{ij}\log\sigma(\mathbf{x}_i^{\top}\mathbf{x}_j) + \frac{b}{n}\log\sigma(-\mathbf{x}_i^{\top}\mathbf{x}_j).$$

We substitute $z_{ij} = \mathbf{x}_i^{\top}\mathbf{x}_j$, using our independence assumption, and solving for

$$\frac{\partial\mathscr{L}_{ij}}{\partial z_{ij}} = \mathrm{PPR}_{ij}\sigma(-z_{ij}) - \frac{b}{n}\sigma(z_{ij}) = 0, \tag{5.4}$$

we get $z_{ij} = \log\frac{n\cdot\mathrm{PPR}_{ij}}{b}$, hence $\mathrm{XX}^{\top} = \log(\mathrm{PPR}) + \log n - \log b = \mathrm{Y}$. □

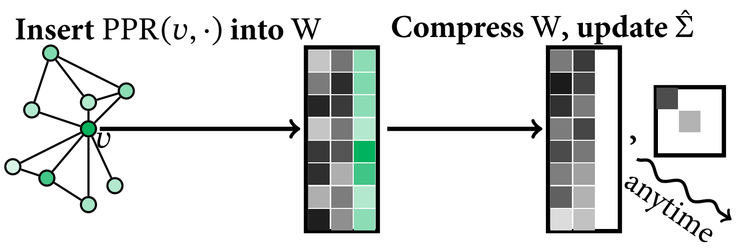


Fig. 5.2 Workflow: FREDE iteratively samples transformed PPR rows, periodically compresses the derived sketch and derives singular values by SVD, and returns an embedding with error guarantees at any time.

Even though this solution is algebraically impossible as it implies approximation of a non-symmetric matrix by a symmetric one, it provides a matrix whose covariance we can sketch.

5.3.2 FREDE Algorithm

Since the matrix $\mathrm{Y} = \mathrm{XX}^\top$ has equal row and column ranks, we rewrite the decomposition commutatively, as $\mathrm{Y} = \log(\mathrm{PPR}) + \log n - \log b = \mathrm{X}^\top\mathrm{X}$. We keep the bias parameter b equal to 1, as in NetMF, and apply Frequent Directions (Section 5.2.6) to obtain a $d \times n$ sketch-based embedding W by processing rows of Y. Algorithm 4 presents the details of FREDE and Figure 5.2 shows its workflow; it computes rows of the PPR matrix, and hence of the transformed Y, by sampling, applies the SVD-based Frequent Directions sketching process periodically with each d rows it processes (Lines 8–12), and returns embeddings with guarantees at any time (Lines 14–15). We keep track of singular values in $\hat{\Sigma}$ alongside the sketch so as to avoid performing SVD upon a request for output; as in [170], we multiply by $\sqrt{\hat{\Sigma}}$ at output time (Line 15), whereas a covariance-oriented sketcher would use $\hat{\Sigma}$. The time to process *all* n nodes with $\mathcal{O}(n/d)$ SVD iterations costing $\mathcal{O}(d^2n)$ is $\mathcal{O}(dn^2)$.

Sketch-based embeddings inherit the covariance error bounds of sketching (Section 5.2.6), which hold *anytime*, even after processing only an arbitrary subset of rows. Thus, FREDE embeddings inherit the *anytime error guarantees* of Frequent Directions, which are valid after materializing only part of the similarity matrix, and superior to those of other sketch-based embeddings; it achieves $\mathtt{ce} \leq \epsilon$ on the submatrix $\mathrm{S}_{[s]}$ built from any size-s subset of processed rows (nodes) when $d = \mathcal{O}(n/\epsilon)$ [82], independently of s. In Section 5.4.8 we show that FREDE outperforms other sketch-based embeddings in anytime node classification.

Algorithm 4 FREDE algorithm

```
1:  function FREDE($G, n, d$)
2:      $W \leftarrow$ zeros($2d, n$)                                   ▷ all zeros matrix $W \in R^{2d\times n}$
3:      $\hat{\Sigma} \leftarrow I(2d)$                                ▷ diagonal identity matrix $\hat{\Sigma} \in R^{2d\times 2d}$
4:      for $v \in V$ do
5:          $x \leftarrow$ PersonalizedPageRank($v$)
6:          $y \leftarrow \log x + \log n$                              ▷ PPR-like similarity row
7:          Insert $y$ into the last zero valued row of W
8:          if W has no zero valued rows then
9:              $U, \Sigma, V^\top \leftarrow \mathrm{SVD}(\hat{\Sigma}W), \sigma \leftarrow \Sigma_{d,d}$
10:             $\hat{\Sigma}_{:d} \leftarrow \sqrt{\max(\Sigma^2_{:d} - \sigma^2 I_d, 0)}$     ▷ set dth row of $\hat{\Sigma}$ to 0
11:             $\hat{\Sigma}_{d:} \leftarrow I_d$                         ▷ set last $d$ entries of $\hat{\Sigma}$ to 1
12:             $W_{:d} \leftarrow V^\top_{:d}, W_{d:} \leftarrow \mathbf{0}_{d\times n}$     ▷ zero last $d$ rows of W
13:     return $\hat{\Sigma}, W_{:d}$
14: function GetEmbedding($k \leq d$)                                 ▷ Anytime
15:     return $\sqrt{\hat{\Sigma}} W_{:k}$                              ▷ first $k$ rows
```

5.3.3 Parallelization and Distribution

The steps of Algorithm 4 may be easily parallelized. In particular, Line 5 could employ approximate PPR [241, 237], and Line 9 efficient SVD calculations [107]. Such speedups trade quality for scalability. Furthermore, FREDE can be efficiently *distributed* across machines for the sake of scalability, with very small communication overhead and *preserving* its quality guarantees. This appealing characteristic, unique among related works on embeddings, follows from the mergeability property that FREDE inherits from Frequent Directions. In each machine m, we may create a partial embedding matrix W based on the subset of the nodes available to m, and then `merge` partial embeddings from t servers, i.e., iteratively sketch their concatenations by Equation 5.2 in hierarchical fashion, incurring a $\log_2 t$ time complexity factor.

5.4 Experiments

The primary advantage of FREDE is its *anytime* character, i.e., its ability to derive embeddings by processing only a fraction of similarity matrix rows. On that front, it may only be compared against other sketch-based emebeddings. Here, we also test FREDE for qualitative performance in data science tasks against other graph embeddings to corroborate its practical impact.

5.4.1 Compared Methods

As previous work [212] has established that nonlinear embeddings outperform those based on linear-transforms, we evaluate FREDE against three representative state-of-the-art graph embeddings based on nonlinear transforms, the classic DeepWalk, factorization-based NetMF, and neural VERSE, three sketching baselines, and exact matrix factorization by SVD:

- DeepWalk[2] [167] learns an embedding by sampling fixed-length random walks from each node and applying word2vec-based learning on those walks; despite intensive research on graph embeddings, DeepWalk remains competitive when used with time-tested default parameters [212]: walk length $t = 80$, number of walks per node $\gamma = 80$, and window size $T = 10$; we use these values.
- VERSE[3] [212] trains a single-layer neural network to learn the PPR similarity measure via sampling, with default parameters $\alpha = 0.85$ and $\texttt{nsamples} = 10^6$.
- NetMF [170] performs SVD on the closed-form DeepWalk matrix. We use the optimal method, NetMF-small; as it is not scalable, we evaluate it on our three smallest datasets, using the same parameters as in DeepWalk, and bias $b = 1$ as in the original paper; NetMF suffices for a quality comparison with NetSMF [171], as the former represents the full version of the latter.

Sketching baselines
We additionally compare with three high-performance baseline sketching methods (Section 5.2.6), namely **Hashing**, **Random Projections** and **Sampling** — computing the sketch and filtering singular value as in FREDE. Our **Random Projections** baseline is a refined variant of [244], substituting a crude higher-order matrix approximation with the row-update random projections sketching algorithm applied on the transformed PPR matrix.

SVD
On the three smallest datasets, we were able to compare against the **exact SVD decomposition** of the nonlinearly tranformed PPR matrix Y with the same parameters as in FREDE.

5.4.2 Parameter Settings

We set embedding dimension $d = 128$ unless indicated otherwise. For SVD, we use the `gesdd` routine in the Intel MKL library. For classification we use LIBLINEAR [69]. We repeat each experiment 10 times and evaluate each embedding 10 times.

[2]https://github.com/xgfs/deepwalk-c
[3]https://github.com/xgfs/verse

	Size			Statistics	
dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathscr{L}\rvert$	Avg. deg.	Density
PPI	4k	77k	50	19.9	5.1×10^{-3}
POS	5k	185k	40	38.7	8.1×10^{-3}
BlogCatalog	10k	334k	39	64.8	6.3×10^{-3}
CoCit	44k	195k	15	8.86	2.0×10^{-4}
CoAuthor	52k	178k	—	6.94	1.3×10^{-4}
VK	79k	2.7M	—	34.1	8.7×10^{-4}
Flickr	80k	12M	195	146.55	1.8×10^{-3}
YouTube	1.1M	3M	47	5.25	9.2×10^{-6}

Table 5.3 Dataset characteristics: number of vertices $|V|$, number of edges $|E|$; number of node labels $|\mathscr{L}|$; average node degree; density defined as $|E|/\binom{|V|}{2}$.

5.4.3 Datasets

We experiment on 8 publicly available real[4,5] datasets.

- PPI [198, 93]: a protein-protein interaction dataset, where labels represent hallmark gene sets of specific biological states.
- POS [140, 93]: a word co-occurrence network built from Wikipedia data. Labels tag parts of speech induced by Stanford NLP parser.
- BlogCatalog [243, 206]: a social network of bloggers from the blogcatalog website. Labels represent self-identified topics of blogs.
- CoCit [1, 212]: a paper citation graph generated from the Microsoft Academic graph, featuring papers published in 15 major data mining conferences. We use conference identifiers as labels.
- CoAuthor [1, 212]: a coauthorsip graph generated from the Microsoft Academic graph. We use snapshots from 2014 and 2016 for link prediction.
- VK [212]: a Russian all-encompassing social network. Labels represent user genders. We use snapshots from November 2016 and May 2017 for link prediction.
- Flickr [243, 206]: a photo-sharing social network, where labels represent user interests, and edges messages between users.

[4] https://github.com/xgfs/verse/tree/master/data
[5] http://leitang.net/code/social-dimension/data/flickr.mat

- YouTube [243, 206]: a video-based social network; labels indicate genre interests.

Table 5.3 summarises the data characteristics. All algorithms are implemented in Python[6] and ran on a 2×20-core Intel E5-2698 v4 CPU machine with 384Gb RAM and a 64Gb memory constraint.

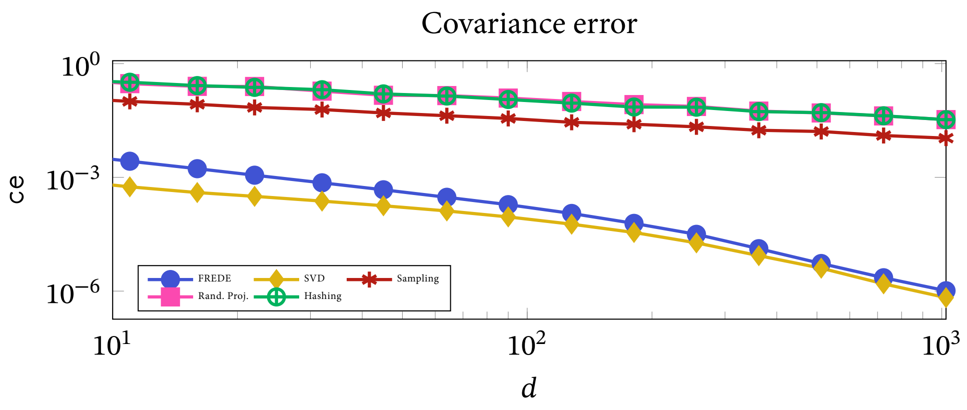


Fig. 5.3 Covariance error vs. dimensionality d; FREDE approaches SVD, which yields optimal covariance error.

5.4.4 Sketching Quality

As a preliminary test, we assess our choice of sketching backbone against other sketching algorithms and the optimal rank-k covariance approximation obtained by SVD on the full similarity matrix, $\tilde{S}^T\tilde{S} = V_d\Sigma_d^2V_d^T$. Figure 5.3 reports the covariance error ce on PPI data, vs. the dimensionality d. FREDE outperforms the other sketching algorithms (Section 5.2.6) by at least 2 orders of magnitude and, as d grows, it converges to the optimal SVD solution. This result reconfirms that the advantages of Frequent Directions versus other sketching methods transfer well to the domain of graph embeddings. For the sake of completeness, we keep comparing to other sketching methods in the rest of our study, as performance may vary depending on the downstream data science task.

5.4.5 PPR Approximation

Figure 5.4 shows the performance of sketching algorithms on a node classification task (predicting correct labels) vs. the number of random walks for PPR approximation. FREDE consistently outperforms sketching baselines and reaches the exact-PPR solution with 10^6 walks. This result

[6] FREDE code repository

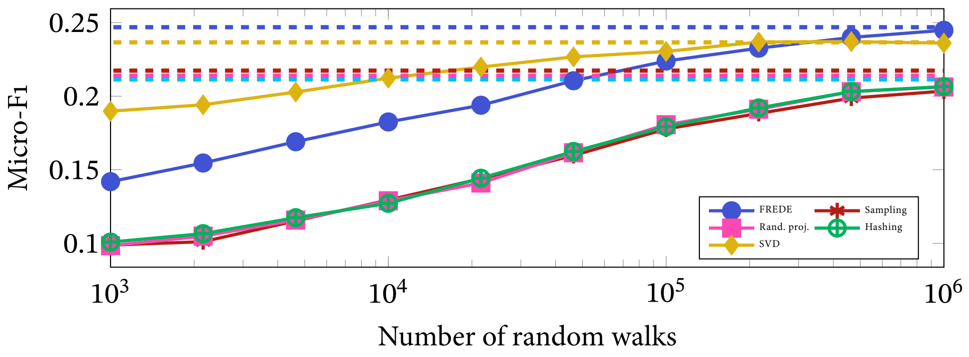


Fig. 5.4 Classification performance of sketching algorithms on PPI data wrt. number of walks to compute PPR.

indicates that we can achieve performance obtained using the exact PPR values in downstream tasks even without computing such PPR values with high precision.

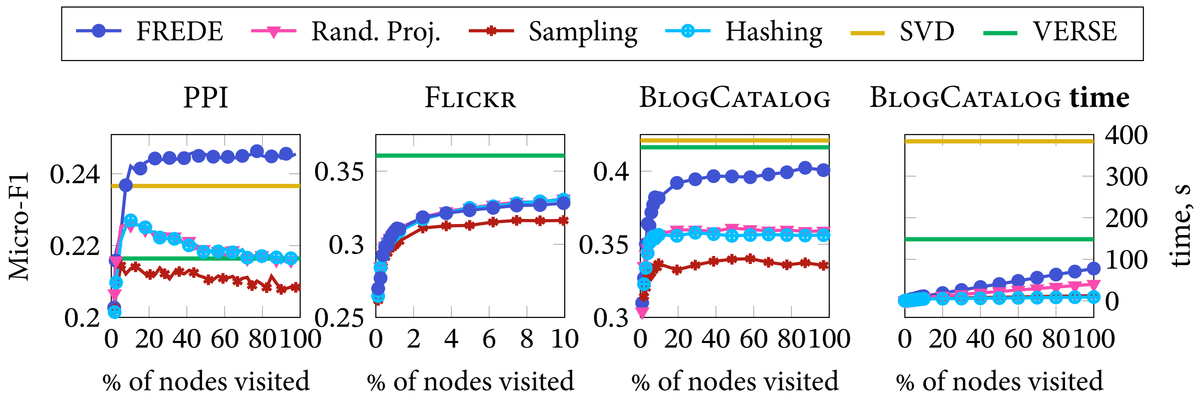


Fig. 5.5 Classification performance of FREDE with varying percentage of the graph as input on three datasets: PPI, FLICKR and BLOGCATALOG.

5.4.6 Node Classification

Tables 5.4 – 5.9 report classification results in terms of the Micro-F1 measure as it is common in the literature [167, 205]; Macro-F1 results are similar and we report them in the Appendix C.1. SVD is featured where it runs within 64Gb. For each dataset, we repeat the experiment 10 times and report the average. Surprisingly, on PPI and POS, FREDE outperforms its exact counterpart, SVD, and consistently supersedes its sketching counterparts across all datasets.

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	16.33	19.74	21.34	22.39	23.38
NetMF	18.58	22.01	23.87	24.65	25.30
VERSE	16.45	19.89	21.64	23.08	23.84
FREDE	19.56	23.11	24.38	25.11	25.52
SVD	18.31	22.12	23.66	25.03	25.78
Rand. Proj.	16.80	19.99	21.45	22.38	23.14
Sampling	16.25	19.55	20.93	21.85	22.68
Hashing	16.73	19.97	21.51	22.43	23.44

Table 5.4 Micro-F1 classification, PPI data.

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	43.42	47.12	48.96	49.86	50.18
NetMF	43.42	46.98	48.52	49.23	49.72
VERSE	40.80	44.70	46.60	47.65	48.24
FREDE	46.59	49.23	50.45	51.02	51.30
SVD	44.69	48.86	50.57	51.53	52.20
Rand. Proj.	40.24	43.87	45.65	46.43	47.18
Sampling	40.35	43.80	45.39	46.30	46.69
Hashing	40.17	43.88	45.44	46.35	46.79

Table 5.5 Micro-F1 classification, POS data.

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	36.22	39.84	41.22	42.06	42.53
NetMF	36.62	39.80	41.05	41.70	42.17
VERSE	35.82	40.06	41.63	42.63	43.14
FREDE	35.69	38.88	39.98	40.54	40.75
SVD	37.60	40.99	42.10	42.66	43.47
Rand. Proj.	30.82	34.43	35.81	36.52	37.16
Sampling	29.44	32.32	33.41	34.04	34.29
Hashing	30.81	34.36	35.82	36.65	37.28

Table 5.6 Micro-F1 classification, BlogCatalog data.

5.4.7 Link Prediction

Link prediction is the task of predicting the appearance of a link between pairs of nodes in a graph. Tables 5.11 and 5.12 report link prediction accuracy (predicting the appearance of a link)

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	37.22	40.34	41.72	42.59	43.16
VERSE	38.95	41.20	42.55	43.41	44.01
FREDE	42.46	44.56	45.39	45.84	46.17
Rand. Proj.	40.89	42.63	43.63	44.32	44.78
Sampling	40.84	42.97	43.93	44.49	44.91
Hashing	40.86	42.66	43.65	44.29	44.83

Table 5.7 Micro-F1 classification, CoCit data.

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	32.39	36.02	37.41	38.15	38.70
VERSE	30.08	34.22	36.06	37.11	37.83
FREDE	30.90	32.98	33.86	34.48	34.88
Rand. Proj.	28.92	32.21	33.82	34.76	35.49
Sampling	28.46	30.97	32.08	32.75	33.24
Hashing	29.07	32.23	33.77	34.75	35.48

Table 5.8 Micro-F1 classification, Flickr data.

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	37.96	40.54	41.75	42.60	43.37
VERSE	38.04	40.50	41.72	42.59	43.33
FREDE	34.51	37.37	38.78	39.40	39.95
Rand. Proj.	33.88	36.10	37.23	37.94	38.38
Sampling	33.97	35.66	36.37	37.19	37.71
Hashing	32.64	35.64	36.92	37.46	38.13

Table 5.9 Micro-F1 classification, YouTube data.

on CoCit and VK by a logistic regression classifier on features derived from embeddings by the rules in Table 5.10. As a baseline, we use common link prediction features (node degree, number of common neighbors, Adamic-Adar index, Jaccard coefficient, and preferential attachment). We represent absent links in the training data by negative sampling, and use 50% of links for training and remaining 50% for testing. FREDE outperforms all methods on CoCit, and all sketching baselines on VK. Surprisingly, sketching baselines perform better than state-of-the-art graph embeddings on CoCit.

Operator	Result
Average	$(\mathbf{a}+\mathbf{b})/2$
Concat	$[\mathbf{a}_1,\dots,\mathbf{a}_d,\mathbf{b}_1,\dots,\mathbf{b}_d]$
Hadamard	$[\mathbf{a}_1 * \mathbf{b}_1,\dots,\mathbf{a}_d * \mathbf{b}_d]$
Weighted L1	$[\lvert\mathbf{a}_1-\mathbf{b}_1\rvert,\dots,\lvert\mathbf{a}_d-\mathbf{b}_d\rvert]$
Weighted L2	$[(\mathbf{a}_1-\mathbf{b}_1)^2,\dots,(\mathbf{a}_d-\mathbf{b}_d)^2]$

Table 5.10 Edge embedding strategies for link prediction, nodes $u, v \in V$ and corresponding embeddings $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$.

method	Average	Concat	Hadamard	L1	L2
DeepWalk	68.97	68.43	66.61	78.80	77.89
VERSE	79.62	79.25	86.27	75.15	75.32
FREDE	81.28	80.95	86.83	81.70	82.37
Rand. Proj.	80.81	80.54	86.73	80.79	81.42
Sampling	80.98	80.74	86.45	79.53	79.51
Hashling	80.84	80.48	86.66	80.59	81.33
Baseline			77.53		

Table 5.11 Link prediction accuracy, CoAuthor data.

method	Average	Concat	Hadamard	L1	L2
DeepWalk	69.98	69.83	69.56	78.42	77.42
VERSE	74.56	74.42	80.94	77.16	77.47
FREDE	74.68	74.59	77.63	74.25	73.60
Rand. Proj.	74.41	74.27	77.01	74.33	74.56
Sampling	74.38	74.27	76.82	72.26	71.95
Hashing	74.36	74.27	76.86	74.30	74.56
Baseline			78.84		

Table 5.12 Link prediction accuracy, VK data.

5.4.8 Anytime Classification

We study anytime operation (Section 5.3.2) on node classification using 50% of nodes for training and processing PPR rows in random order. Figure 5.5 presents results for PPR-based methods on three datasets. FREDE outperforms both VERSE and SVD on PPI, as in Table 5.4, after processing only 10% of similarities; its downstream performance grows with the number of nodes visited on all datasets, while tht of other sketching baselines drops on PPI data; FREDE also performs competitively on Flickr (we examine up to 10% of nodes, as Random Projections was inefficient; SVD did not run within 64Gb) and BlogCatalog, as in Tables 5.6 and 5.8. The rightmost plot in Figure 5.5 shows runtime on BlogCatalog; remarkably, while sketchers' runtime grows linearly, those of one-off methods stand apart. These results also illustrate that embedding merging preserves the downstream embedding quality; as Equation 5.2 shows, merging two embeddings amounts to sketching their concatenation; therefore, the sketch operation Algorithm 4 periodically performs with each new d similarity matrix rows it processes can also be viewed as a `merge` operation.

5.5 Summary

We observed that, since graph embeddings aim to preserve similarity matrix covariance, row-wise sketching techniques are naturally suited to that end. We applied a state-of-the-art sketcher, Frequent Directions, on a matrix factorization interpretation of a state-of-the-art nonlinear transform embedding, VERSE, to craft FREDE: a linear-space graph embedding that allows for scalable data science operations on graph data, as well as for anytime and distributed computation *with error guarantees*. Besides its anytime character, FREDE achieves almost as low covariance error as the exact SVD solution and stands its ground against previous graph embeddings even after processing as little as 10% of similarity matrix rows; therefore, it promises significant practical impact. In the future, we plan to augment FREDE in terms of quality by reweighing the resulting embedding vectors, in the spirit of NRP [237], and in terms of efficiency by using recent enhancements on Frequent Directions [110].

Conclusion and Future Work

Summary of Contributions

In this thesis, we considered efficient numerical linear algebra methods for boosting machine learning algorithms. We will now provide a summary of contributions seasoned with possible future directions for research.

In Chapter 2, we introduced quadrature-based random feature maps and showed that they generalise previous work in random features for kernel approximation. Both widely used random Fourier features and orthogonal random features are special cases of the proposed spherical-radial quadrature rules with degrees (1,1) and (1,3) respectively. We provided error analysis and empirical support for its superiority in both accuracy of kernel approximation and downstream tasks. As we aimed at time and space efficient algorithms, we improved computational complexity by introducing butterfly structure into these feature maps.

In Chapter 3, we proposed a new distance measure, IMD, between unaligned data manifolds by leveraging intrinsic information and stochastic numerical integration technique, namely Stochastic Lanczos Quadrature. SLQ allowed efficient approximation of the spectral descriptor used in the distance computation along with theoretical guarantees that we derived for IMD. We validated the introduced distance on a range of tasks, e.g. quantifying the change of data representations in natural language processing and image processing, evaluation of generative models, and others. Given the ability of IMD to compare diverse manifolds (even with different dimensionality), we hope that our work will pave the way for even more expressive techniques based on geometric insights.

Chapter 4 covers spectral graph distances and their efficient computation for Web-scale graphs. We improved approximation errors of baseline solutions by at least an order of magnitude across a comprehensive collection of real-world graphs. We showed that our method, being linear in the number of edges of graphs, provides an accurate comparison of graphs with billion nodes on a single machine in less than an hour.

In Chapter 5, we interpreted a state-of-the-art nonlinear transform embedding algorithm VERSE as a matrix-factorization problem on a Personalised PageRank similarity matrix and developed an algorithm for graph embeddings. The algorithm can be characterised as anytime and optimal, i.e. it can be stopped anytime to produce embeddings with theoretical error guarantees. Even after processing a fraction of graph similarity matrix rows, it achieves superior or comparable quality with state-of-the-art algorithms that lack FREDE's anytime nature.

All in all, we found that quadrature approximation and fast NLA methods are useful for overcoming computational obstacles and sometimes become a key ingredient in scaling up machine learning to high dimensional spaces and web-scale graphs.

Future work

We hope that our work inspires more application of structured matrices in deep learning, such as the use of butterfly matrices [55]. We also hope that a broader application of numerical methods may open up development of more expressive techniques based on geometric insights. Especially, with recent hardware and software advancements, it should be straightforward to implement methods of this thesis on GPU as they essentially consist of matrix-by-matrix and matrix-by-vector multiplications, e.g. using JAX [17].

This thesis has also left many interesting open questions, which we are excited to explore in the future. Among them, more technical one concerns the error bounds of SLaQ for approximating VNGE graph descriptor. Looking towards unsupervised learning, another direction we may explore is in the adoption of IMD as a loss function in generative models such as generative adversarial networks.

References

[1] Microsoft academic graph - KDD cup 2016. https://kddcup2016.azurewebsites.net/Data, 2016.

[2] Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *JMLR*, 2018.

[3] Ryan P Adams, Jeffrey Pennington, Matthew J Johnson, Jamie Smith, Yaniv Ovadia, Brian Patton, and James Saunderson. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.

[4] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *TODS*, pages 26:1–26:28, 2013.

[5] Nesreen Ahmed, Ryan Anthony Rossi, John Lee, Theodore Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. Role-based graph embeddings. *TKDE*, 2020.

[6] Awad H Al-Mohy and Nicholas J Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM journal on scientific computing*, 2011.

[7] Rami Al-Rfou, Bryan Perozzi, and Dustin Zelle. Ddgk: Learning graph representations for deep divergence graph kernels. In *The World Wide Web Conference*, 2019.

[8] Alexander A. Alemi and Ian Fischer. GILBO: One metric to measure them all. In *NeurIPS*, 2018.

[9] Edward Anderson, Zhaojun Bai, Christian Bischof, Susan Blackford, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, and D Sorensen. *LAPACK Users' guide*. SIAM, 1999.

[10] Theodore W Anderson, Ingram Olkin, and Les G Underhill. Generation of random orthogonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(4):625–629, 1987.

[11] Francis J Anscombe. Graphs in statistical analysis. *The American Statistician*, 1973.

[12] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

[13] Sanjeev Arora and Yi Zhang. Do GANs actually learn the distribution? an empirical study. In *ICLR*, 2018.

[14] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 2019.

[15] Haim Avron and Vikas Sindhwani. High-performance kernel machines with implicit distributed optimization and randomization. *Technometrics*, 58(3):341–349, 2016.

[16] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 2011.

[17] Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Claudio Fantacci, Jonathan Godwin, Chris Jones, Tom Hennigan, Matteo Hessel, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Lena Martens, Vladimir Mikulik, Tamara Norman, John Quan, George Papamakarios, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

[18] Francis Bach. On the equivalence between kernel quadrature rules and random feature expansions. *Journal of Machine Learning Research*, 18(21):1–38, 2017.

[19] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized PageRank. *PVLDB*, 4(3):173–184, 2010.

[20] John A Baker. Integration over spheres and the divergence theorem for balls. *The American Mathematical Monthly*, 104(1):36–47, 1997.

[21] Paul Balister, Béla Bollobás, Amites Sarkar, and Mark Walters. Connectivity of random k-nearest-neighbour graphs. *Advances in Applied Probability*, 2005.

[22] Shane Barratt and Rishi Kant Sharma. A note on the inception score. *CoRR*, abs/1801.01973, 2018.

[23] Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. An estimator for the diagonal of a matrix. *Applied numerical mathematics*, 2007.

[24] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, 2002.

[25] Mikhail Belkin and Partha Niyogi. Convergence of laplacian eigenmaps. In *NIPS*, pages 129–136, 2007.

[26] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123, 2013.

[27] Mikołaj Bińkowski, Dougal J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD gans. In *ICLR*, 2018.

[28] Salomon Bochner. Monotone funktionen, stieltjessche integrale und harmonische analyse. *Mathematische Annalen*, 108(1):378–410, 1933.

[29] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[30] L Bottou, FE Curtis, and J Nocedal. Optimization methods for large-scale machine learning. arxiv 2016. *arXiv preprint arXiv:1606.04838*.

[31] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *SODA*, 2009.

[32] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near optimal column-based matrix reconstruction. *FOCS*, pages 305–314, 2011.

[33] Samuel L Braunstein, Sibasish Ghosh, and Simone Severini. The laplacian of a graph as a density matrix: a basic combinatorial approach to separability of mixed states. *Annals of Combinatorics*, pages 291–317, 2006.

[34] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[35] Thomas Bühler and Matthias Hein. Spectral clustering based on the graph p-laplacian. In *ICML*, pages 81–88, 2009.

[36] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attribute graph classification. In *ICLR, RLGM workshop*, 2019.

[37] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, 2015.

[38] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. In *ICLR*, 2017.

[39] Pin-Yu Chen, Lingfei Wu, Sijia Liu, and Indika Rajapakse. Fast incremental von neumann graph entropy computation: Theory, algorithm, and applications. In *ICML*, 2019.

[40] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[41] Xixian Chen, Haiqin Yang, Irwin King, and Michael R Lyu. Training-efficient feature map for shift-invariant kernels. In *IJCAI*, pages 3395–3401, 2015.

[42] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, pages 342–350, 2009.

[43] Krzysztof Choromanski and Vikas Sindhwani. Recycling randomness with structure for sublinear time kernel expansions. *arXiv preprint arXiv:1605.09049*, 2016.

[44] Krzysztof Choromanski, Mark Rowland, and Adrian Weller. The unreasonable effectiveness of random orthogonal embeddings. *arXiv preprint arXiv:1703.00864*, 2017.

[45] Fan Chung, Linyuan Lu, and Van Vu. The spectra of random graphs with given expected degrees. *Internet Mathematics*, 2004.

[46] Fan RK Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[47] Charles L Clarke, Nick Craswell, and Ian Soboroff. Overview of the TREC 2009 web track. Technical report, DTIC Document, 2009.

[48] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *STOC*, 2013.

[49] Ed S. Coakley and Vladimir Rokhlin. A fast divide-and-conquer algorithm for computing the spectra of real symmetric tridiagonal matrices. *Applied and Computational Harmonic Analysis*, 34(3):379 – 414, 2013.

[50] David Cohen-Steiner, Weihao Kong, Christian Sohler, and Gregory Valiant. Approximating the spectrum of a graph. In *KDD*, 2018.

[51] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 2006.

[52] Amin Coja-Oghlan. On the laplacian eigenvalues of g(n, p). *Combinatorics, Probability and Computing*, 2007.

[53] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

[54] Tri Dao, Christopher M De Sa, and Christopher Ré. Gaussian quadrature for kernel features. In *Advances in Neural Information Processing Systems*, pages 6109–6119, 2017.

[55] Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. *Proceedings of machine learning research*, 97:1517, 2019.

[56] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[57] Emily L. Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015.

[58] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *ICLR*, 2017.

[59] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, pages 771–783, 2003.

[60] Kun Dong, Austin R. Benson, and David Bindel. Network density of states. In *KDD*, 2019.

[61] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, 2011.

[62] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *NIPS*, 2016.

[63] Petros Drineas and Michael W Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6(Dec):2153–2175, 2005.

[64] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[65] Paul Erdös and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 1960.

[66] David Eriksson, Kun Dong, Eric Lee, David Bindel, and Andrew G Wilson. Scaling gaussian process regression with derivatives. In *NIPS*, 2018.

[67] Ernesto Estrada. Characterization of 3d molecular structure. *Chemical Physics Letters*, pages 713–718, 2000.

[68] Ernesto Estrada and Desmond J. Higham. Network properties revealed through matrix functions. *SIAM Review*, pages 696–714, 2010.

[69] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 2008.

[70] Kai-Tai Fang and Run-Ze Li. Some methods for generating both an NT-net and the uniform distribution on a Stiefel manifold and their applications. *Computational Statistics & Data Analysis*, 24(1):29–46, 1997.

[71] X Yu Felix, Ananda Theertha Suresh, Krzysztof M Choromanski, Daniel N Holtmann-Rice, and Sanjiv Kumar. Orthogonal Random Features. In *Advances in Neural Information Processing Systems*, pages 1975–1983, 2016.

[72] Shai Fine and Katya Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2(Dec):243–264, 2001.

[73] Alexander Forrester, Andy Keane, et al. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[74] Jim Gao. Machine learning applications for data center optimization. 2014.

[75] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *NIPS*, 2018.

[76] Alan Genz. Methods for generating random orthogonal matrices. *Monte Carlo and Quasi-Monte Carlo Methods*, pages 199–213, 1998.

[77] Alan Genz and John Monahan. Stochastic integration rules for infinite regions. *SIAM journal on scientific computing*, 19(2):426–439, 1998.

[78] Alan Genz and John Monahan. A stochastic algorithm for high-dimensional integrals over unbounded regions with gaussian weight. *Journal of Computational and Applied Mathematics*, 112(1):71–81, 1999.

[79] Semyon Aranovich Gershgorin. Über die abgrenzung der eigenwerte einer matrix. *Izv. Akad. Nauk.*, (6):749–754, 1931.

[80] Mina Ghashami and Jeff M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *SODA*, 2014.

[81] Mina Ghashami, Edo Liberty, and Jeff M. Phillips. Efficient frequent directions algorithm for sparse matrices. In *KDD*, 2016.

[82] Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. Frequent directions : Simple and deterministic matrix sketching. *SIAM J. Comput.*, 45:1762–1792, 2016.

[83] Gene H Golub and Gérard Meurant. *Matrices, moments and quadrature with applications*. Princeton University Press, 2009.

[84] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, USA, 1996. ISBN 0801854148.

[85] Gene H Golub and John H Welsch. Calculation of gauss quadrature rules. *Mathematics of computation*, 1969.

[86] Gene H Golub and John H Welsch. Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

[87] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013. ISBN 9781421407944. URL https://books.google.ru/books?id=X5YfsuCWpxMC.

[88] Todd R Golub, Donna K Slonim, Pablo Tamayo, Christine Huard, Michelle Gaasenbeek, Jill P Mesirov, Hilary Coller, Mignon L Loh, James R Downing, Mark A Caligiuri, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.

[89] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[90] Diego Granziol, Timur Garipov, Dmitry Vetrov, Stefan Zohren, Stephen Roberts, and Andrew Gordon Wilson. Towards understanding the true loss surface of deep neural networks using random matrix theory and iterative spectral methods. 2019.

[91] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13: 723–773, 2012.

[92] Alexander Grigor'yan. Heat kernels on weighted manifolds and applications. In *Contemp. Math.*, pages 93–191. AMS, 2006.

[93] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[94] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein GANs. In *NIPS*, 2017.

[95] Swaminathan Gurumurthy, Ravi Kiran Sarvadevabhatla, and R Venkatesh Babu. DeLiGAN: Generative adversarial networks for diverse and limited data. In *CVPR*, 2017.

[96] Ivan Gutman and Bo Zhou. Laplacian energy of a graph. *Linear Algebra and its applications*, pages 29–37, 2006.

[97] Lin Han, Francisco Escolano, Edwin R Hancock, and Richard C Wilson. Graph characterizations from von Neumann entropy. *Pattern Recognition Letters*, pages 1958–1967, 2012.

[98] Simon Haykin. *Cognitive dynamic systems: perception-action cycle, radar and radio*. Cambridge University Press, 2012.

[99] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. REGAL: Representation learning-based graph alignment. In *CIKM*, pages 117–126, 2018.

[100] Matthias Hein, Jean-Yves Audibert, and Ulrike Von Luxburg. From graphs to manifolds–weak and strong pointwise consistency of graph laplacians. In *COLT*, 2005.

[101] Matthias Hein, Jean-Yves Audibert, and Ulrike von Luxburg. Graph laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 2007.

[102] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 2005.

[103] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017.

[104] Nicholas J Higham and Awad H Al-Mohy. Computing matrix functions. *Acta Numerica*, 2010.

[105] Marlis Hochbruck and Christian Lubich. On krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 1997.

[106] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalise better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

[107] Michael P Holmes, Jr Isbell, Charles Lee, and Alexander G Gray. Quic-svd: Fast svd using cosine trees. In *NIPS*, pages 673–680, 2009.

[108] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[109] Po-Sen Huang, Haim Avron, Tara N Sainath, Vikas Sindhwani, and Bhuvana Ramabhadran. Kernel methods match deep neural networks on timit. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 205–209. IEEE, 2014.

[110] Zengfeng Huang. Near optimal frequent directions for sketching dense and sparse matrices. In *ICML*, pages 2048–2057, 2018.

[111] Zengfeng Huang. Near optimal frequent directions for sketching dense and sparse matrices. In *ICML*, pages 2048–2057, 2018.

[112] MF Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3): 1059–1076, 1989.

[113] Daniel Jiwoong Im, He Ma, Graham Taylor, and Kristin Branson. Quantitatively evaluating GANs with divergences proposed for training. In *ICLR*, 2018.

[114] Simon Jenni and Paolo Favaro. On stabilizing generative adversarial training with noise. In *CVPR*, 2019.

[115] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[116] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. URL http://graphkernels.cs.tu-dortmund.de.

[117] Valentin Khrulkov and Ivan V. Oseledets. Geometry score: A method for comparing generative adversarial networks. In *ICML*, 2018.

[118] Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan Oseledets. Tensorized embedding layers for efficient model compression. *arXiv preprint arXiv:1901.10787*, 2019.

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

[120] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pages 10215–10224, 2018.

[121] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[122] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019.

[123] Danai Koutra, Joshua T. Vogelstein, and Christos Faloutsos. DELTACON: A principled massive-graph similarity function. In *SDM*, 2013.

[124] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[125] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[126] Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood-approximating kernel expansions in loglinear time. In *Proceedings of the International Conference on Machine Learning*, 2013.

[127] John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[128] Erich L Lehmann and Joseph P Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[129] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, 2005.

[130] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*, 2014.

[131] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[132] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. MMD GAN: Towards deeper understanding of moment matching network. In *NIPS*, 2017.

[133] Ping Li, Trevor J. Hastie, and Kenneth Ward Church. Very sparse random projections. In *KDD*, 2006.

[134] Qunwei Li, Bhavya Kailkhura, Rushil Anirudh, Yi Zhou, Yingbin Liang, and Pramod Varshney. MR-GAN: Manifold regularized generative adversarial networks. *arXiv preprint arXiv:1811.10427*, 2018.

[135] Edo Liberty. Simple and deterministic matrix sketching. In *KDD*, 2013.

[136] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. In *ICLR*, 2017.

[137] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs created equal? a large-scale study. In *NeurIPS*, 2018.

[138] Yueming Lyu. Spherical structured feature maps for kernel approximation. In *International Conference on Machine Learning*, pages 2256–2264, 2017.

[139] Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. A tensorized transformer for language modeling. In *Advances in Neural Information Processing Systems*, pages 2232–2242, 2019.

[140] Matt Mahoney. Large text compression benchmark. http://www.mattmahoney.net/text/text.html, 2011.

[141] Justin Matejka and George Fitzmaurice. Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing. In *CHI*, 2017.

[142] Facundo Mémoli. A spectral notion of gromov–wasserstein distance and related methods. *Applied and Computational Harmonic Analysis*, 2011.

[143] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. In *ICLR*, 2017.

[144] Francesco Mezzadri. How to generate random matrices from the classical compact groups. *arXiv preprint math-ph/0609050*, 2006.

[145] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[146] Giorgia Minello, Luca Rossi, and Andrea Torsello. On the von Neumann entropy of graphs. *Journal of Complex Networks*, 2018.

[147] Alan Mislove. *Online Social Networks: Measurement, Analysis, and Applications to Distributed Information Systems*. PhD thesis, Rice University, Department of Computer Science, May 2009.

[148] Jayaved Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 1982.

[149] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 2003.

[150] John Monahan and Alan Genz. Spherical-radial integration rules for bayesian computation. *Journal of the American Statistical Association*, 92(438):664–674, 1997.

[151] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*, 2018.

[152] Youssef Mroueh, Chun-Liang Li, Tom Sercu, Anant Raj, and Yu Cheng. Sobolev GAN. In *ICLR*, 2018.

[153] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, and Bernhard Schölkopf. Kernel mean embedding of distributions: A review and beyond. *arXiv preprint arXiv:1605.09522*, 2016.

[154] Marina Munkhoeva, Yermek Kapushev, Evgeny Burnaev, and Ivan Oseledets. Quadrature-based features for kernel approximation. In *Advances in Neural Information Processing Systems*, pages 9147–9156, 2018.

[155] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *NeurIPS*, pages 1396–1404, 2015.

[156] Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In *NIPS*, 2010.

[157] Maria CV Nascimento and Andre CPLF De Carvalho. Spectral methods for graph clustering–a survey. *European Journal of Operational Research*, pages 221–231, 2011.

[158] Huda Nassar, Nate Veldt, Shahin Mohammadi, Ananth Grama, and David F Gleich. Low rank spectral network alignment. In *WWW*, pages 619–628, 2018.

[159] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450, 2015.

[160] Augustus Odena, Jacob Buckman, Catherine Olsson, Tom B. Brown, Christopher Olah, Colin A. Raffel, and Ian J. Goodfellow. Is generator conditioning causally related to GAN performance? In *ICML*, 2018.

[161] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, 2016.

[162] Art B Owen. Latin supercube sampling for very high-dimensional simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):71–102, 1998.

[163] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[164] Sung Woo Park and Junseok Kwon. Sphere generative adversarial network based on geometric moment matching. In *CVPR*, 2019.

[165] Jack Parker-Holder, Luke Metz, Cinjon Resnick, Hengyuan Hu, Adam Lerer, Alistair Letcher, Alexander Peysakhovich, Aldo Pacchiano, and Jakob Foerster. Ridge rider: Finding diverse solutions by following eigenvectors of the hessian. *Advances in Neural Information Processing Systems*, 33, 2020.

[166] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572, 1901.

[167] Bryan Perozzi, Rami Al-Rfou', and Steven Skiena. Deepwalk: online learning of social representations. In *KDD*, 2014.

[168] Anish Prabhu, Ali Farhadi, Mohammad Rastegari, et al. Butterfly transform: An efficient fft based neural architecture design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12024–12033, 2020.

[169] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007. ISBN 9780521880688. URL https://books.google.ru/books?id=1aAOdzK3FegC.

[170] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *WSDM*, 2018.

[171] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. Netsmf: Large-scale network embedding as sparse matrix factorization. In *WWW*, 2019.

[172] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training (2018). *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_. pdf*, 2018.

[173] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NIPS*, 2017.

[174] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184, 2008.

[175] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010.

[176] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *KDD*, pages 385–394, 2017.

[177] Eitan Richardson and Yair Weiss. On GANs and GMMs. In *NeurIPS*, 2018.

[178] Karl Ridgeway, Jake Snell, Brett Roads, Richard S. Zemel, and Michael C. Mozer. Learning to generate images with perceptual similarity metrics. In *ICIP*, 2017.

[179] Farbod Roosta-Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. *Foundations of Computational Mathematics*, 2015.

[180] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL http://networkrepository.com.

[181] Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, Sungchul Kim, Anup Rao, and Yasin Abbasi-Yadkori. A structural graph representation learning framework. In *WSDM*, pages 483–491, 2020.

[182] Alessandro Rudi and Lorenzo Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems*, pages 3215–3225, 2017.

[183] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. *Advances in neural information processing systems*, 30:3888–3898, 2017.

[184] Walter Rudin. *Fourier analysis on groups*. Courier Dover Publications, 2017.

[185] Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

[186] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *NIPS*, 2016.

[187] Shibani Santurkar, Ludwig Schmidt, and Aleksander Mądry. A classification-based study of covariate shift in gan distributions. In *ICML*, 2018.

[188] B. Schölkopf, B.S.A.J. Smola, A.J. Smola, F. Bach, MIT Press, and M.D.M.P.I.B.C.T.G.P.B. Scholkopf. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive computation and machine learning. MIT Press, 2002. ISBN 9780262194754. URL https://books.google.ru/books?id=y8ORL3DWt4sC.

[189] Till Schulz and Pascal Welke. On the necessity of graph kernel baselines. In *ECML-PKDD, GEM workshop*, 2019.

[190] Hang Shao, Abhishek Kumar, and P Thomas Fletcher. The riemannian geometry of deep generative models. In *CVPR Workshops*, pages 315–323, 2018.

[191] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 2011.

[192] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *PAMI*, 2000.

[193] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. How good is my GAN? In *ECCV*, 2018.

[194] David E Simmons, Justin P Coon, and Animesh Datta. The von Neumann Theil index: characterizing graph centralization using the von Neumann index. *Journal of Complex Networks*, 2018.

[195] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[196] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[197] Alex J Smola and Bernhard Schölkopf. Sparse greedy matrix approximation for machine learning. 2000.

[198] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. BioGRID: a general repository for interaction datasets. *Nucleic Acids Research*, 2006.

[199] G. W. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis*, 17(3):403–409, 1980. ISSN 00361429. URL http://www.jstor.org/stable/2156882.

[200] J. Stoer, R. Bartels, W. Gautschi, R. Bulirsch, and C. Witzgall. *Introduction to Numerical Analysis*. Texts in Applied Mathematics. Springer New York, 2013. ISBN 9781475722727. URL https://books.google.ru/books?id=YJPkBwAAQBAJ.

[201] Karl Stratos, Michael Collins, and Daniel Hsu. Model-based word embeddings from decompositions of count matrices. In *ACL*, volume 1, pages 1282–1291, 2015.

[202] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*. Wiley Online Library, 2009.

[203] Dougal J Sutherland and Jeff Schneider. On the error of random fourier features. *arXiv preprint arXiv:1506.02785*, 2015.

[204] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[205] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. In *WWW*, 2015.

[206] Lei Tang and Huan Liu. Scalable learning of collective behavior based on sparse social dimensions. In *CIKM*, 2009.

[207] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *ICLR*, 2016.

[208] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *JACM*, 52(1):1–24, 2005.

[209] Daniel Ting, Ling Huang, and Michael Jordan. An analysis of the convergence of graph laplacians. In *ICML*, 2010.

[210] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander M. Bronstein, and Emmanuel Müller. NetLSD: Hearing the shape of a graph. In *KDD*, 2018.

[211] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander M. Bronstein, and Emmanuel Müller. Netlsd: Hearing the shape of a graph. In *KDD*, 2018.

[212] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *WWW*, 2018.

[213] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Spectral graph complexity. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, 2019.

[214] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Alex Bronstein, Ivan Oseledets, and Emmanuel Müller. The shape of data: Intrinsic distance for data distributions. *arXiv preprint arXiv:1905.11141*, 2019.

[215] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Alex Bronstein, Ivan Oseledets, and Emmanuel Müller. The shape of data: Intrinsic distance for data distributions. In *ICLR*, 2020.

[216] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. Frede: Linear-space anytime graph embeddings. *arXiv preprint arXiv:2006.04746*, 2020.

[217] Anton Tsitsulin, Marina Munkhoeva, and Bryan Perozzi. Just slaq when you approximate: Accurate spectral distances for web-scale graphs. In *Proceedings of The Web Conference 2020*, pages 2697–2703, 2020.

[218] Ke Tu, Jianxin Ma, Peng Cui, Jian Pei, and Wenwu Zhu. Autone: Hyperparameter optimization for massive network embedding. In *KDD*, 2019.

[219] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of tr(f(a)) via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 2017.

[220] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016.

[221] Amir Vaxman, Mirela Ben-Chen, and Craig Gotsman. A multi-resolution approach to heat kernels on discrete surfaces. In *TOG*, 2010.

[222] Santosh S Vempala. *The random projection method.* American Math. Soc., 2005.

[223] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, pages 395–416, 2007.

[224] John Von Neumann. *Mathematische grundlagen der quantenmechanik.* Springer-Verlag, 1932.

[225] Liwei Wang, Lunjia Hu, Jiayuan Gu, Zhiqiang Hu, Yue Wu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In *Advances in Neural Information Processing Systems*, 2018.

[226] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, 2009.

[227] Hermann Weyl. Über die asymptotische verteilung der eigenwerte. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1911.

[228] Herbert S Wilf. Mathematics for the physical sciences. 1962.

[229] Christopher KI Williams. Computing with infinite networks. In *Advances in Neural Information Processing Systems*, pages 295–301, 1997.

[230] David P. Woodruff. Low rank approximation lower bounds in row-update streams. In *NIPS*, 2014.

[231] David P Woodruff. Sketching as a tool for numerical linear algebra. *Theoretical Computer Science*, 2014.

[232] D.P. Woodruff. *Sketching as a Tool for Numerical Linear Algebra.* Foundations and Trends(r) in Theoretical Computer Science Series. Now Publishers, 2014. ISBN 9781680830040. URL https://books.google.ru/books?id=ENMOogEACAAJ.

[233] Qiantong Xu, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Q. Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *CoRR*, abs/1806.07755, 2018.

[234] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*, 2015.

[235] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, 2012.

[236] Jiyan Yang, Vikas Sindhwani, Haim Avron, and Michael Mahoney. Quasi-Monte Carlo feature maps for shift-invariant kernels. In *Proceedings of The 31st International Conference on Machine Learning (ICML-14)*, pages 485–493, 2014.

[237] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. Homogeneous network embedding for massive graphs via reweighted personalized pagerank. *PVLDB*, 13(5):670–683, 2020.

[238] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström Method vs Random Fourier Features: A Theoretical and Empirical Comparison. In *Advances in Neural Information Processing Systems*, pages 476–484, 2012.

[239] Zi Yin and Yuanyuan Shen. On the dimensionality of word embedding. In *NeurIPS*, 2018.

[240] Zi Yin and Yuanyuan Shen. On the dimensionality of word embedding. In *Advances in Neural Information Processing Systems*, pages 887–898, 2018.

[241] Minji Yoon, Jinhong Jung, and U Kang. Tpa: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *ICDE*, pages 1132–1143. IEEE, 2018.

[242] Felix X Yu, Sanjiv Kumar, Henry Rowley, and Shih-Fu Chang. Compact nonlinear maps and circulant extensions. *arXiv preprint arXiv:1503.03893*, 2015.

[243] R. Zafarani and H. Liu. Social computing data repository at ASU, 2009. URL http://socialcomputing.asu.edu.

[244] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. Billion-scale network embedding with iterative random projection. In *ICDM*, pages 787–796, 2018.

[245] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. Arbitrary-order proximity preserved network embedding. In *KDD*, 2018.

[246] Sharon Zhou, Mitchell Gordon, Ranjay Krishna, Austin Narcomey, Durim Morina, and Michael S Bernstein. HYPE: Human eYe perceptual evaluation of generative models. In *NeurIPS*, 2019.

[247] Zhiming Zhou, Han Cai, Shu Rong, Yuxuan Song, Kan Ren, Weinan Zhang, Yong Yu, and Jun Wang. Activation maximization generative adversarial nets. In *ICLR*, 2018.

[248] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3): 73–83, 1996.

Appendix A

Derivation of Bounds and Variances

A.1 Error Bounds and Variance for Quadrature Rules

Below is the derivation of the error bounds and variances for the Quadrature-based Features, introduced in Chapter 2.

A.1.1 Variance of the Degree $(3, 3)$ Quadrature Rule

Let us denote $\mathbf{q} = \begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} \in \mathcal{X}^2$, $k(\mathbf{q}) = k(\mathbf{x}, \mathbf{y})$, $h_j(\mathbf{q}) = d\frac{f_{\mathbf{xy}}(-\rho_j \mathbf{Q}\mathbf{v}_j)+f_{\mathbf{xy}}(\rho_j \mathbf{Q}\mathbf{v}_j)}{2\rho_j^2} - k(\mathbf{q}) = s_j(\mathbf{q}) - k(\mathbf{q})$. Then it is easy to see that $\mathbb{E}h_j(\mathbf{q}) = 0$.

Let us denote $I(\mathbf{q}) = SR^{3,3}_{\mathbf{Q}_1,\rho_1}(f_{\mathbf{xy}})$, $g(\mathbf{q}) = I(\mathbf{q}) - k(\mathbf{x}, \mathbf{y})$. Using the above definitions we obtain

$$\begin{aligned}\mathbb{V}g(\mathbf{q}) = \mathbb{V}\left(1 - \sum_{j=1}^{d+1} \frac{d}{(d+1)\rho_j^2}\right) + \mathbb{E}\left(\frac{1}{d+1}\sum_{i=1}^{d+1} h_i(\mathbf{q})\right)^2 \\ +2cov\left(1 - \sum_{j=1}^{d+1} \frac{d}{(d+1)\rho_j^2}, \frac{1}{d+1}\sum_{i=1}^{d+1} h_i(\mathbf{q})\right).\end{aligned} \tag{A.1}$$

Variance of the first term

$$\begin{aligned}\mathbb{V}\left(1-\sum_{j=1}^{d+1}\frac{d}{(d+1)\rho_j^2}\right) &= \mathbb{E}\left(1-\sum_{j=1}^{d+1}\frac{d}{(d+1)\rho_j^2}\right)^2\\ &= \mathbb{E}\left(1-\sum_{j=1}^{d+1}\frac{2d}{(d+1)\rho_j^2}+\left(\sum_{j=1}^{d+1}\frac{d}{(d+1)\rho_j^2}\right)^2\right)\\ &= 1-2+\frac{d}{(d+1)(d-2)}+\frac{d}{d+1}=\frac{2}{(d+1)(d-2)}.\end{aligned} \tag{A.2}$$

Variance of the second term (using independence of $h_i(\mathbf{q})$ and $h_j(\mathbf{q})$ for $i \neq j$)

$$\mathbb{E}\left(\frac{1}{d+1}\sum_{i=1}^{d+1}h_i(\mathbf{q})\right)^2=\mathbb{E}\left(\frac{1}{(d+1)^2}\sum_{i,j=1}^{d+1}h_i(\mathbf{q})h_j(\mathbf{q})\right)=\frac{1}{(d+1)^2}\sum_{i=1}\mathbf{E}h_i(\mathbf{q})^2=\frac{\mathbf{E}h_1(\mathbf{q})^2}{d+1}. \tag{A.3}$$

Variance of the last term (using Cauchy-Schwarz inequality)

$$\begin{aligned}cov\left(1-\sum_{j=1}^{d+1}\frac{d}{(d+1)\rho_j^2},\frac{1}{d+1}\sum_{i=1}^{d+1}h_i(\mathbf{q})\right) &= \mathbb{E}\left[\left(1-\sum_{j=1}^{d+1}\frac{d}{(d+1)\rho_j^2}\frac{1}{d+1}\right)\sum_{i=1}^{d+1}h_i(\mathbf{q})\right]\\ &= -\mathbb{E}\frac{d}{d+1}\sum_{i,j=1}^{d+1}\frac{h_i(\mathbf{q})}{\rho_j^2}\\ &\le \frac{1}{d+1}\sum_{i=1}^{d+1}\sqrt{\mathbb{E}\frac{1}{\rho_i^4}}\sqrt{\mathbb{E}h_i(\mathbf{q})^2}\\ &= \sqrt{\frac{\mathbb{E}h_1(\mathbf{q})^2}{d(d-2)}}.\end{aligned} \tag{A.4}$$

Now, let us upper bound term $\mathbb{E}h_1(\mathbf{q})^2$

$$\mathbb{E}h_1(\mathbf{q})^2=\mathbb{E}\left(\frac{d\phi(\mathbf{w}^\top\mathbf{x})\phi(\mathbf{w}^\top\mathbf{y})}{\rho^2}\right)^2-k(\mathbf{q})^2\le\frac{d\kappa^4}{d-2}.$$

Using this expression and plugging (A.2), (A.3), (A.4) into (A.1) we obtain

$$\begin{aligned}\mathbb{V}\left[\frac{1}{n}\sum_{i=1}^{n} SR^{3,3}_{\mathbf{Q}_i,\rho_i}(f_{\mathbf{xy}})\right] &\leq \frac{2}{n(d+1)(d-2)} + \frac{d\kappa^4}{n(d+1)(d-2)} + \frac{1}{n}\sqrt{\frac{d\kappa^4}{d(d-2)^2}} \leq \\ &\leq \frac{2}{n(d+1)(d-2)} + \frac{d\kappa^4}{n(d+1)(d-2)} + \frac{\kappa^2}{n(d-2)} \leq \frac{2+\kappa^4+\kappa^2}{n(d-2)}.\end{aligned} \tag{A.5}$$

and it concludes the proof.

A.1.2 Error Probability

The proof strategy closely follows that of [203]; we just use Chebyshev-Cantelli ineqaulity instead of Hoeffding's and Bernstein inequalities and all the expectations are calculated according to our quadrature rules.

Let $\mathbf{q} = \begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} \in \mathcal{X}^2$, $\mathcal{X}^2$ is compact set in $\mathbb{R}^{2d}$ with diameter $\sqrt{2}l$, so we can cover it with an ε-net using at most $T = (2\sqrt{2}l/r)^{2d}$ balls of radius r. Let $\{\mathbf{q}_i\}_{i=1}^{T}$ denote their centers, and L_g be the Lipschitz constant of $g(\mathbf{q}) : \mathbb{R}^{2d} \to \mathbb{R}$. If $|g(\mathbf{q}_i)| < \varepsilon/2$ for all i and $L_g < \varepsilon/(2r)$, then $g(\mathbf{q}) < \varepsilon$ for all $\mathbf{q} \in \mathcal{X}^2$.

Regularity Condition

Similarly to [203] (regularity condition section in appendix) it can be proven that $\mathbb{E}\nabla g(\mathbf{q}) = \nabla\mathbb{E}g(\mathbf{q})$.

Lipschitz Constant

Since g is differentiable, $L_g = ||\nabla g(\mathbf{q}^*)||$, where $\mathbf{q}^* = \arg\max_{\mathbf{q}\in\mathcal{X}^2} ||\nabla g(\mathbf{q})||$. Via Jensen's inequality $\mathbb{E}||\nabla h(\mathbf{q})|| \geq ||\mathbb{E}\nabla h(\mathbf{q})||$. Then using independence of $h_i(\mathbf{q})$ and $h_j(\mathbf{q})$ for $i \neq j$

$$\mathbb{E}[L_g]^2 = \mathbb{E}\left[||\nabla I(\mathbf{q}^*) - k(\mathbf{q}^*)||^2\right] = \mathbb{E}\left[\left\|\frac{1}{d+1}\sum_{i=1}^{d+1}\nabla h_i(\mathbf{q}^*)\right\|^2\right] = \mathbb{E}\left[\frac{1}{d+1}||\nabla h_1(\mathbf{q}^*)||^2\right] =$$
$$= \frac{1}{d+1}\mathbb{E}_{\mathbf{q}^*}\left[\mathbb{E}||\nabla s_1(\mathbf{q}^*)||^2 - 2||\nabla k(\mathbf{q}^*)||\mathbb{E}||\nabla s_1(\mathbf{q}^*)|| + ||\nabla k(\mathbf{q}^*)||^2\right] \leq$$
$$\leq \frac{1}{d+1}\mathbb{E}\left[||\nabla s_1(\mathbf{q}^*)||^2 - ||\nabla k(\mathbf{q}^*)||^2\right] \leq \frac{1}{d+1}\mathbb{E}||\nabla s_1(\mathbf{q}^*)||^2 =$$
$$= \frac{1}{d+1}\mathbb{E}\left[||\nabla_{\mathbf{x}^*} s_1(\mathbf{q}^*)||^2 + ||\nabla_{\mathbf{y}^*} s_1(\mathbf{q}^*)||^2\right] \leq \frac{2d^2\kappa^2\mu^2\sigma_p^2}{d+1}\mathbb{E}\frac{1}{\rho_1^2} = \frac{2d\kappa^2\mu^2\sigma_p^2}{d+1},$$

where $|\phi'(\cdot)| \leq \mu$. Then using Markov's inequality we obtain

$$\mathbb{P}(L_g \geq \frac{\varepsilon}{2r}) \leq 8\frac{d}{d+1}\left(\frac{\sigma_p r\kappa\mu}{\varepsilon}\right)^2$$

.

Anchor Points

Let us upper bound the following probability

$$\mathbb{P}\left(\bigcup_{i=1}^{T} |g(\mathbf{q}_i)| \geq \frac{1}{2}\varepsilon\right) \leq T\mathbb{P}\left(|g(\mathbf{q}_i)| \geq \frac{1}{2}\varepsilon\right).$$

Let us rewrite the function $g(\mathbf{q})$

$$g(\mathbf{q}) = 1 - \frac{1}{d+1}\sum_{i=1}^{d+1}\frac{d}{\rho_i^2} + \frac{1}{d+1}\sum_{i=1}^{d+1}\frac{d\phi_{\mathbf{q}}(\rho_i\mathbf{z}_i)}{\rho_i^2} - k(\mathbf{q}) = \frac{1}{d+1}\sum_{i=1}^{d+1}\left(\frac{d(1-\phi_{\mathbf{q}}(\rho_i\mathbf{z}_i))}{\rho_i^2} + 1 - k(\mathbf{q})\right),$$

where $\phi_{\mathbf{q}}(\rho_i\mathbf{z}_i) = \frac{f_{\mathbf{xy}}(-\rho_j\mathbf{Q}\mathbf{v}_j)+f_{\mathbf{xy}}(\rho_j\mathbf{Q}\mathbf{v}_j)}{2\rho_j^2}$. Let us suppose that $\left|\frac{1-\phi_{\mathbf{q}}(\rho\mathbf{z})}{\rho^2}\right| \leq M$. Then we can apply Hoeffding's inequality

$$\mathbb{P}(|g(\mathbf{q})| \geq \frac{1}{2}\varepsilon) \leq 2\exp\left(-\frac{2D\frac{1}{4}\varepsilon^2}{(M-(-M))^2}\right) = 2\exp\left(-\frac{D\varepsilon^2}{8M^2}\right)$$

.

Optimizing over r

Now the probability of $\sup_{\mathbf{q}\in\mathcal{X}^2} |g(\mathbf{q})| \le \varepsilon$ takes the form

$$p = \mathbb{P}\left(\sup_{\mathbf{q}\in\mathcal{X}^2} |g(\mathbf{q})| \le \varepsilon\right) \ge 1 - \kappa_1 r^{-2d} - \kappa_2 r^2,$$

where $\kappa_1 = 2\left(2\sqrt{2}l\right)^{2d} \exp\left(-\frac{D\varepsilon^2}{8M^2}\right)$, $\kappa_2 = \frac{8d}{d+1}\left(\frac{\kappa\mu\sigma_p}{\varepsilon}\right)^2$. Maximizing this probability over r gives us the following bound

$$\mathbb{P}\left(\sup_{\mathbf{q}\in\mathcal{X}^2} |g(\mathbf{q})| \ge \varepsilon\right) \le \left(d^{\frac{-d}{d+1}} + d^{\frac{1}{d+1}}\right) 2^{\frac{6d+1}{d+1}} \left(\frac{d}{d+1}\right)^{\frac{d}{d+1}} \left(\frac{\sigma_p l \kappa\mu}{\varepsilon}\right)^{\frac{2d}{d+1}} \exp\left(-\frac{D\varepsilon^2}{8M^2(d+1)}\right).$$

For RBF kernel $\kappa = \mu = 1, M = \frac{1}{2}$, so we obtain the following bound

$$\mathbb{P}\left(\sup_{\mathbf{q}\in\mathcal{X}^2} |g(\mathbf{q})| \ge \varepsilon\right) \le \left(d^{\frac{-d}{d+1}} + d^{\frac{1}{d+1}}\right) 2^{\frac{6d+1}{d+1}} \left(\frac{d}{d+1}\right)^{\frac{d}{d+1}} \left(\frac{\sigma_p l}{\varepsilon}\right)^{\frac{2d}{d+1}} \exp\left(-\frac{D\varepsilon^2}{2(d+1)}\right).$$

Let us compare it with the bound for RFF

$$\mathbb{P}\left(\sup_{\mathbf{q}\in\mathcal{X}^2} |g(\mathbf{q})| \ge \varepsilon\right) \le \left(d^{\frac{-d}{d+1}} + d^{\frac{1}{d+1}}\right) 2^{\frac{5d+1}{d+1}} 3^{\frac{d}{d+1}} \left(\frac{\sigma_p l}{\varepsilon}\right)^{\frac{2d}{d+1}} \exp\left(-\frac{D\varepsilon^2}{32(d+1)\alpha'_\varepsilon}\right).$$

Appendix B

Additional Implementation Details

Below, we discuss implementation details of algorithms in this thesis.

B.1 Quadrature-based Features for Kernel Approximation

B.1.1 Remarks on Quadrature Rules

Even functions. We note here that for specific functions $f_{\mathbf{xy}}(\mathbf{w})$ we can derive better versions of SR rule by taking on advantage of the knowledge about the integrand. For example, the Gaussian kernel has $f_{\mathbf{xy}}(\mathbf{w}) = \cos(\mathbf{w}^\top(\mathbf{x} - \mathbf{y}))$. Note that f is even, so we can discard an excessive term in the summation in degree $(3, 3)$ rule since $f(\mathbf{w}) = f(-\mathbf{w})$, i.e $SR^{3,3}$ rule reduces to

$$SR^{3,3}_{\mathbf{Q},\rho}(f) = \left(1 - \sum_{j=1}^{d+1} \frac{d}{(d+1)\rho_j^2}\right) f(\mathbf{0}) + \frac{d}{d+1} \sum_{j=1}^{d+1} \frac{f(\rho_j \mathbf{Q}\mathbf{v}_j)}{\rho_j^2}. \tag{B.1}$$

Obtaining a proper ρ. It may be the case when sampling ρ that $1 - \sum_{j=1}^{d+1} \frac{d}{(d+1)\rho_j^2} < 0$ which results in complex a_0 term. Simple solution is just to resample ρ_j to satisfy the non-negativity of the expression. According to central limit theorem $\sum_{j=1}^{d+1} \frac{d}{(d+1)\rho_j^2}$ tends to normal random variable with mean 1 and variance $\frac{1}{d+1}\frac{2}{d-2}$. The probability that this values is non-negative equals $p = \mathbb{P}(1 - \sum_{j=1} \frac{d}{(d+1)\rho^2} \geq 0) \rightsquigarrow \frac{1}{2}$. The expectation of number of resamples needed to satisfy non-negativity constraint is $\frac{1}{p}$ tends to 2.

B.2 Intrinsic Multi-scale Distance

B.2.1 GAN Details

Below, we enlist GAN architectures used in experiments in Chapter 3.

MNIST WGAN

```
ConvGenerator(
  (latent_to_features): Sequential(
    (0): Linear(in_features=100, out_features=512, bias=True)
    (1): ReLU()
  )
  (features_to_image): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4),
          stride=(2, 2), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (3): ConvTranspose2d(64, 32, kernel_size=(4, 4),
          stride=(2, 2), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
    (6): ConvTranspose2d(32, 16, kernel_size=(4, 4),
          stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True)
    (9): ConvTranspose2d(16, 1, kernel_size=(4, 4),
          stride=(2, 2), padding=(1, 1))
    (10): Sigmoid()
  )
)
ConvDiscriminator(
  (image_to_features): Sequential(
    (0): Conv2d(1, 16, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): LeakyReLU(negative_slope=0.2)
    (4): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (5): LeakyReLU(negative_slope=0.2)
    (6): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): Sigmoid()
  )
  (features_to_prob): Sequential(
    (0): Linear(in_features=512, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

MNIST WGAN-GP, FMNIST (WGAN, WGAN-GP)

```
MNISTGenerator(
  (block1): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace)
  )
  (block2): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace)
  )
  (deconv_out): ConvTranspose2d(64, 1, kernel_size=(8, 8), stride=(2, 2))
  (preprocess): Sequential(
    (0): Linear(in_features=128, out_features=4096, bias=True)
    (1): ReLU(inplace)
  )
  (sigmoid): Sigmoid()
)

MNISTDiscriminator(
  (main): Sequential(
    (0): Conv2d(1, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): ReLU(inplace)
    (2): Conv2d(64, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (3): ReLU(inplace)
    (4): Conv2d(128, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (5): ReLU(inplace)
  )
  (output): Linear(in_features=4096, out_features=1, bias=True)
)
```

CIFAR-10 (WGAN, WGAN-GP)

```
CIFARGenerator(
  (preprocess): Sequential(
    (0): Linear(in_features=128, out_features=4096, bias=True)
    (1): BatchNorm1d(4096, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
  )
  (block1): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
  )
  (block2): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
  )
  (deconv_out): ConvTranspose2d(64, 3, kernel_size=(2, 2), stride=(2, 2))
  (tanh): Tanh()
)

CIFARDiscriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.01)
    (2): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): LeakyReLU(negative_slope=0.01)
    (4): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (5): LeakyReLU(negative_slope=0.01)
  )
  (linear): Linear(in_features=4096, out_features=1, bias=True)
)
```

CelebA (WGAN, WGAN-GP)

```
CelebaGenerator(
  (preprocess): Sequential(
    (0): Linear(in_features=128, out_features=8192, bias=True)
    (1): BatchNorm1d(8192, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
  )
  (block1): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(5, 5), stride=(2, 2),
           padding=(2, 2), output_padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
  )
  (block2): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(5, 5), stride=(2, 2),
           padding=(2, 2), output_padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
  )
  (block3): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(5, 5), stride=(2, 2),
           padding=(2, 2), output_padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
  )
  (deconv_out): ConvTranspose2d(64, 3, kernel_size=(5, 5), stride=(2, 2),
           padding=(2, 2), output_padding=(1, 1))
  (tanh): Tanh()
)

CelebaDiscriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): LeakyReLU(negative_slope=0.01)
    (2): Conv2d(64, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (3): LeakyReLU(negative_slope=0.01)
    (4): Conv2d(128, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (5): LeakyReLU(negative_slope=0.01)
    (6): Conv2d(256, 512, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (7): LeakyReLU(negative_slope=0.01)
    (8): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
  )
)
```

Appendix C

Additional Experimental Results

Below, we report some additional experimental results.

C.1 FREDE

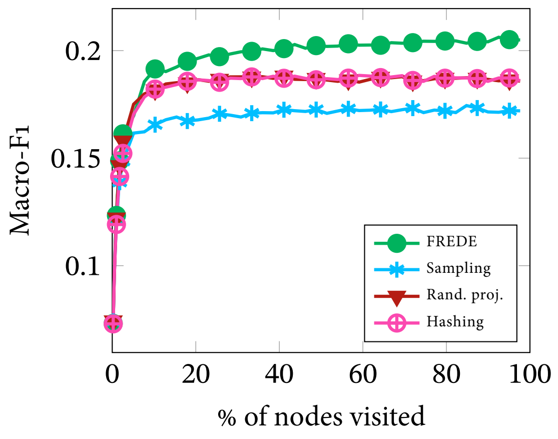


Fig. C.1 Classification performance (Macro-F1) of sketching algorithms on PPI dataset vs number of nodes processed for sketching. Embedding dimensionality $d = 128$.

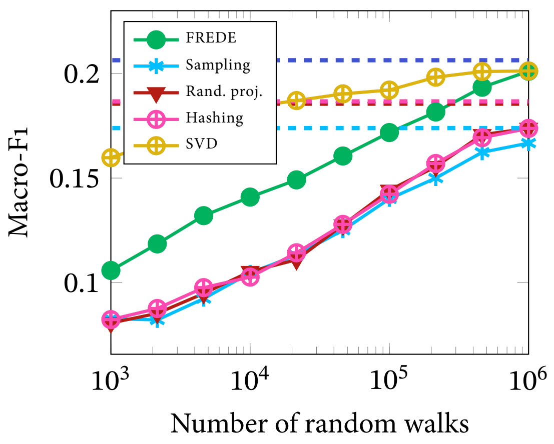


Fig. C.2 Classification performance (Macro-F1) of sketching algorithms on PPI dataset vs number of random walks to generate PPR. Embedding dimensionality $d = 128$.

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	—	—	—	—	—
NetMF	14.23	18.39	19.95	20.94	20.54
VERSE	—	—	—	—	—
FREDE	14.97	19.08	20.42	21.02	20.89
SVD	14.50	18.13	19.49	20.66	21.64
Rand. Proj.	13.32	17.02	18.56	19.30	19.41
Sampling	12.46	15.93	17.25	18.07	18.19
Hashing	13.39	17.15	18.63	19.40	19.61

Table C.1 Macro-F1 classification results in PPI dataset.

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	7.85	9.18	9.90	10.28	10.30
NetMF	6.65	8.07	8.86	9.42	9.25
VERSE	7.27	8.60	9.23	9.62	9.56
FREDE	7.33	8.42	8.79	9.12	9.18
SVD	8.54	10.03	10.72	11.05	10.86
Rand. Proj.	7.31	8.46	8.96	9.16	9.13
Sampling	7.37	8.32	8.73	9.15	9.02
Hashing	7.33	8.42	8.79	9.12	9.18

Table C.2 Macro-F1 classification results in POS dataset.

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	—	—	—	—	—
NetMF	20.17	23.89	25.49	26.18	26.31
VERSE	—	—	—	—	—
FREDE	16.84	20.94	22.55	23.36	23.76
SVD	21.14	25.21	26.30	27.19	27.70
Rand. Proj.	16.52	20.53	22.21	23.00	23.50
Sampling	14.88	17.56	18.46	19.04	19.03
Hashing	16.23	20.26	21.87	22.86	23.44

Table C.3 Macro-F1 classification results in BlogCatalog dataset.

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	26.75	29.31	30.30	30.92	31.24
VERSE	27.99	30.06	31.20	31.89	32.28
FREDE	30.29	31.97	32.67	32.95	33.26
Rand. Proj.	29.40	31.02	31.91	32.47	32.83
Sampling	29.14	30.99	31.70	32.10	32.37
Hashing	29.41	31.01	31.95	32.43	32.86

Table C.4 Macro-F1 classification results in CoCit dataset.

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	13.92	19.65	22.27	23.66	24.76
VERSE	11.72	18.17	21.56	23.49	24.76
FREDE	7.42	9.48	10.81	11.93	12.67
Rand. Proj.	9.00	13.53	16.43	18.35	19.75
Sampling	8.49	11.43	13.06	14.08	14.92
Hashing	9.13	13.64	16.39	18.32	19.77

Table C.5 Macro-F1 classification results in Flickr dataset.

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	29.57	33.81	35.28	36.05	36.75
VERSE	29.55	33.95	35.25	36.06	36.72
FREDE	19.12	23.58	25.68	26.81	27.78
Rand. Proj.	22.05	26.95	28.56	29.48	29.94
Sampling	22.85	26.54	27.72	28.46	29.14
Hashing	20.96	26.42	28.09	29.23	29.75

Table C.6 Macro-F1 classification results in YouTube dataset.