VNiVERSiDAD D SALAMANCA

Doctoral Thesis

Low Computational Cost Machine Learning: Random Projections and Polynomial Kernels

Author:
Daniel López Sánchez

Supervisors:
Juan M. Corchado Rodríguez
Angélica González Arrieta

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy (Ph.D.) in Computer Science at the University of Salamanca

May 17, 2019

Declaration of Authorship

Juan Manuel Corchado Rodríguez, chair of Computer Science and Artificial Intelligence at the University of Salamanca and **Angélica González Arrieta**, full professor of the Computer Science and Artificial Intelligence area at the University of Salamanca,

CERTIFY

That the present document, entitled "*Low Computational Cost Machine Learning: Random Projections and Polynomial Kernels*" has been prepared under their supervision at the Computer Science and Automation Department of the University of Salamanca by Daniel López Sánchez, and constitutes his thesis for the degree of Doctor of Philosophy (Ph.D.) in Computer Science.

Juan M. Corchado Rodríguez

Date:

Angélica González Arrieta

Date:

Daniel López Sánchez

Date:

Abstract

Low Computational Cost Machine Learning:
Random Projections and Polynomial Kernels

by Daniel López Sánchez

According to recent reports, over the course of 2018, the volume of data generated, captured and replicated globally was 33 Zettabytes (ZB), and it is expected to reach 175 ZB by the year 2025. Managing this impressive increase in the volume and variety of data represents a great challenge, but also provides organizations with a precious opportunity to support their decision-making processes with insights and knowledge extracted from massive collections of data and to automate tasks leading to important savings. In this context, the field of machine learning has attracted a notable level of attention, and recent breakthroughs in the area have enabled the creation of predictive models of unprecedented accuracy. However, with the emergence of new computational paradigms, the field is now faced with the challenge of creating more efficient models, capable of running on low computational power environments while maintaining a high level of accuracy. This thesis focuses on the design and evaluation of new algorithms for the generation of useful data representations, with special attention to the scalability and efficiency of the proposed solutions. In particular, the proposed methods make an intensive use of randomization in order to map data samples to the feature spaces of polynomial kernels and then condensate the useful information present in those feature spaces into a more compact representation. The resulting algorithmic designs are easy to implement and require little computational power to run. As a consequence, they are perfectly suited for applications in environments where computational resources are scarce and data needs to be analyzed with little delay. The two major contributions of this thesis are: (1) we present and evaluate efficient and data-independent algorithms that perform Random Projections from the feature spaces of polynomial kernels of different degrees and (2) we demonstrate how these techniques can be used to accelerate machine learning tasks where polynomial interaction features are used, focusing particularly on bilinear models in deep learning.

Acknowledgements

First of all, I must thank my supervisors, Juan Manuel Corchado and Angélica González Arrieta, for their dedication and guidance. This thesis wouldn't have been possible without their valuable support.

Secondly, I want to express my profound gratitude to John A. Lee and the rest of the Machine Learning Group from the Catholic University of Louvain for their overwhelming hospitality during my visits to their lab. The months I spent working with them were of crucial help for the development of this thesis, and made my overall PhD experience far more enriching.

I also want to thank my present and former colleagues at the BISITE research group. It is hard to overstate how grateful I am for their company and friendship. Conferences, coffee breaks, and the long hours of work wouldn't have been nearly as enjoyable without them.

A big thank you goes to Belén Pérez Lancho, for her support and patience in helping me to prepare for my teaching duties in the subject "Theory of Computation". Teaching this subject was a great experience, and provided me with a very interesting perspective on the foundational concepts of computer science.

I gratefully acknowledge the University of Salamanca for providing me with an incomparable frame of rich history, culture and academic life. These years living and working at the city of Salamanca have deeply influenced me, and I will hardly forget all the new friendships, experiences and opportunities that this city has offered me. While it is certainly true that "*Quod natura non dat, Salmantica non præstat*", this city has given me more that I ever thought possible.

Last but not least, I thank my parents, Ángel López López and María Esther Sánchez Casal, for their love, patience and support. From the very beginning of my childhood, my parents encouraged me to explore, learn, and make things, planting the seed of my passion for research. This thesis is dedicated to them.

Daniel López Sánchez
Salamanca, May 17, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Although discrepancies exist in the estimates, all sources agree that Big Data[1] keeps getting bigger year after year. As of 2018, the global Datasphere, which consists of all data created, captured, and replicated in any given year, was estimated to comprise a total of 33 Zettabytes (ZB). Furthermore, forecasts indicate that it will grow to 175 ZB by the year 2025 [85]. In addition to this impressive growth, the Big Data phenomenon has permeated the most diverse spheres of our lives. For instance, a recent report highlighted that, every minute of the day, Internet users post 473,400 tweets, request 1,389 Uber rides, conduct 3,877,140 Google searches, upload 400 hours of video to Youtube, and originate 6,940 Tinder matches [34]. These examples illustrate how the Big Data phenomenon has not only grown steadily, but has come to play a key role in most aspects of human activity. On top of the data generated by the direct interaction of humans with digital systems, a significant fraction of today's information flow is due to the billions of Internet of Things (IoT) devices connected to the network. In fact, these devices are expected to generate over 90 ZB of information in 2025 [85].

This tremendous growth in the volume and variety of data poses a critical challenge for companies, organizations and governments across the globe, but also represents a great opportunity for them to find value in their data. An effective strategy for data collection, storage, and analysis is a crucial tool that supports organizations in their decision-making processes, enabling them to reduce costs, identify consumer needs, improve their services, and reach new markets. In addition, the abundance of data has enabled scientists to create increasingly complex predictive models which are capable of automatizing a wide range of tasks, matching and often surpassing human performance.

In this context, it is easy to understand why the extraction of knowledge from raw data has become a topic of broad and current interest, as scientists are striving to discover new and better methods to analyze large volumes of information. At the core of this technological revolution lies the *machine learning* field, in which recent

[1] According to Gartner's IT Glossary, Big Data consist of "*high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.*"
URL: https://www.gartner.com/it-glossary/big-data (Date accessed: 25/04/2019).

breakthroughs have made predictive models more powerful than ever. Particularly, the ever-growing volume of data, together with the advances in the design of algorithms and the use of specialized hardware, have led to the flourishing of predictive models of unprecedented accuracy, such as those from the *deep learning* paradigm. In fact, most of the digital services that we use on a daily basis rely on some sort of machine learning model and, as we have seen, service providers often need to handle thousands of requests per second. It is therefore no wonder that, among the challenges currently faced by the machine learning community, the scalability and efficiency of algorithms play a crucial role in the technological revolution of Big Data. In addition, the emergence and popularization of new computation paradigms, such as the Internet of Things, have also driven the interest of scientists towards machine learning models capable of running in environments with little computational resources while maintaining low inference times and high accuracy rates.

This thesis focuses on the design and evaluation of new algorithms for the generation of useful data representations, paying special attention to the scalability and efficiency of the proposed solutions. Hence, the proposed methods make intensive use of randomization to map data samples to a richer representation with greater discriminative information. Then, this information is condensed to provide a useful and more compact representation. The resulting algorithmic designs are easy to implement and require little computational resources to run. As a consequence, they are perfectly suited for application in environments where computational resources are scarce and data needs to be analyzed in real time or with little delay.

Before delving into the proposed algorithms, we first introduce the fundamental concepts required to understand both Random Projections and kernel functions, and review the latest advances that bridge these two fields. The main contributions of this thesis are presented in Chapters 3, 4 and 5, which introduce novel algorithmic designs to capture the discriminative information of certain kernel functions in a low dimensional representation. Particularly:

- In Chapter 3 we present an efficient method to approximate Random Projections from the feature space of the homogeneous polynomial kernel of degree two. Thanks to the properties of Random Projections, the resulting data representations approximately preserve the structure of data in the kernel feature space. In turn, this enables us to solve learning problems efficiently by training fast linear methods on the generated representations, while obtaining a boost in accuracy thanks to the discriminative information extracted from the kernel feature space. As opposed to existing kernelized Random Projection approaches, our method is data-independent, meaning that it requires no prior knowledge about the distribution of the data samples that will be transformed at test time. Our experiments focus on the preservation of pairwise distances from the kernel feature space and the classification accuracy of linear models trained on the output representations, evidencing that the proposed method outperforms existing approaches in most cases, while being notably faster.

- Chapter 4 builds upon the ideas presented in the previous chapters. It further aims to improve the generality, efficiency, and effectiveness of the proposed solutions. Particularly, a novel method is introduced which is capable of approximating Random Projections from the feature spaces of polynomial kernels of arbitrary degree. In addition, a new approach to generate the projection vectors in the kernel feature space allows us to reduce the computational cost of the algorithm while improving its performance. The exhaustive experimental results presented in this chapter support our claim that the proposed method is capable of efficiently capturing the structure of data in the feature spaces of polynomial kernels.
- Finally, in Chapter 5 we explore the connection between the kernelized variant of Random Projection presented in Chapter 4 and the popular bilinear Convolutional Neural Network (CNN) architecture, which leverages the bilinear pooling operation. While bilinear CNNs are among the most popular and effective methods for fine-grained image recognition, the dimensionality of the resulting descriptors is a major drawback of these models. Chapter 5 introduces a novel method to efficiently reduce the dimension of bilinear pooling descriptors by approximating their Random Projection. Conveniently, this is achieved without ever computing the high-dimensional descriptors explicitly. Our experimental results evidence that this approach outperforms existing compact bilinear pooling algorithms while running faster on low computational power devices, where efficient extensions of bilinear pooling are most useful.

Chapter 6 closes this thesis by summarizing the major contributions and insights derived from the results presented in the previous chapters.

Chapter 2

Background

This chapter introduces the fundamental concepts and techniques that will be relevant in the chapters that follow. First, we describe Random Projection, an efficient and very popular method for dimensionality reduction that guarantees a low distortion in the distances among data samples. Secondly, we list some relevant properties of kernel functions, with special attention to polynomial kernels. A connection is then drawn between Random Projection and kernel functions through a review of the existing methods to perform Random Projections from kernel feature spaces. Lastly, we discuss bilinear pooling, a technique designed to improve the performance of convolutional neural networks in fine-grained recognition tasks. As we will see, this technique is closely related to the homogeneous polynomial kernel of degree two. This connection will later enable us to adapt the methods introduced in the chapters that follow, making bilinear pooling more efficient.

2.1 The Random Projection algorithm

Random Projection (RP) [98] is a simple yet effective and widely used linear dimensionality reduction technique. Just like any other linear dimensionality reduction method, Random Projection reduces the dimension of samples by applying a linear transformation to input data, so each output component is computed as a linear combination of the original features. However, the main difference with alternative methods is that Random Projection generates the projection matrix from a random distribution. Therefore, as opposed to other approaches where training data is required to select an appropriate projection matrix, Random Projection is a *data-independent* method, since no knowledge about the distribution of data is required to generate the projection matrix. Surprisingly, if an appropriate distribution is used to generate the entries of the projection matrix, the structure of data in the high dimensional input feature space will be mostly preserved after the projection[1].

[1] In this document, the term "projection" is used generically to denote a linear transformation between Euclidean spaces.

In spite of its simplicity, Random Projection has strong theoretical foundations. The main theoretical result that underpins Random Projection is the Johnson–Lindenstrauss (JL) lemma [54], which states that a set of N points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved. Formally, for any $0 < \epsilon < 1$ and $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$ there is a linear map $f : \mathbb{R}^d \to \mathbb{R}^k$ for $k = \mathcal{O}(\epsilon^{-2} \log(N))$ such that:

$$(1-\epsilon)||\mathbf{x}_i - \mathbf{x}_j||^2 \leq ||f(\mathbf{x}_i) - f(\mathbf{x}_j)||^2 \leq (1+\epsilon)||\mathbf{x}_i - \mathbf{x}_j||^2, \tag{2.1}$$

for all $i, j = 1, \ldots, N$. Furthermore, it can be found in randomized polynomial time[2].

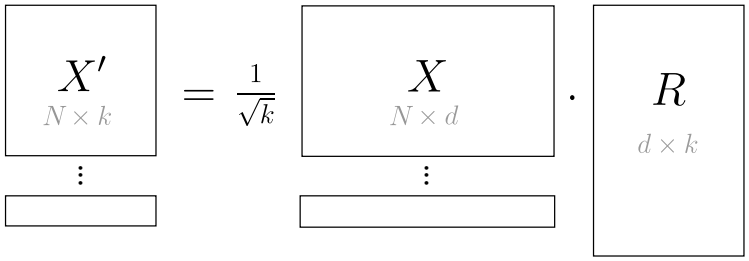


FIGURE 2.1: Schematic view of Random Projection. A dataset with N samples and d features, represented as the input matrix $X \in \mathbb{R}^{N \times d}$, is multiplied by a $d \times k$ projection matrix R, yielding N samples of dimension k, represented as the output matrix $X' \in \mathbb{R}^{N \times k}$ [2].

The original variants of this algorithm performed a projection onto a random k-dimensional subspace, so the map $f(\cdot)$ took the form of a projection onto k random orthonormal vectors [41]. In later versions, the mapping of samples from $\mathbb{R}^d$ to $\mathbb{R}^k$ is done by means of a $d \times k$ projection matrix whose entries are *independently* drawn from a standard normal distribution [52, 8]. Once the $d \times k$ matrix R has been populated, an arbitrary set of N points represented as an $N \times d$ matrix X can be projected from $\mathbb{R}^d$ to $\mathbb{R}^k$ as follows (see Figure 2.1):

$$X' = \frac{1}{\sqrt{k}} XR \ , \text{ where } X' \in \mathbb{R}^{N \times k}. \tag{2.2}$$

However, more recent studies have shown that the projection matrix can also be drawn from much simpler distributions. In particular, Achlioptas showed that if the projection matrix is drawn from the distribution displayed in (2.3) with sparsity term $s = 1$ or $s = 3$, then the JL-lemma will be satisfied [2].

$$R_{ij} = \sqrt{s} \begin{cases} 1 & \text{with prob.} \quad 1/2s \\ 0 & \text{with prob.} \quad 1 - 1/s \\ -1 & \text{with prob.} \quad 1/2s \end{cases}. \tag{2.3}$$

Conveniently, using the distribution proposed by Achlioptas reduces the computational cost of the projection. If the multiplication by $\sqrt{s}$ present in (2.3) is delayed,

[2] A particularly simple constructive proof of this lemma was introduced in [33].

the computation of the projection reduces to aggregate evaluation (i.e., summations and subtractions but no multiplications), which can be efficiently performed in database environments using standard SQL primitives. In addition, the sparsity term s enables further storage and computational savings. For instance, when using $s = 3$ only $1/3$ of the entries of the projection matrix are nonzero. Moreover, it has been shown that using greater sparsity levels in (2.3) is possible with little loss in the preservation of distances. Particularly, in [70] the authors recommended using $s = \sqrt{d}$, leading to a potential $\sqrt{d}$-fold speedup.

Achlioptas also proved that, as long as the entries of the projection matrix are independent and identically distributed (i.i.d.) random variables with zero mean and unit variance, pairwise distances will be preserved in expectation[3] [2]. Moreover, analogous results to the JL-lemma can be derived for projection matrices whose entries are independently drawn from any distribution symmetric about the origin with bounded moments [8], or any distribution with zero mean, unit variance and subgaussian tail [75]. This suggest that the phenomenon described by the JL-lemma is rather robust, as a wide range of distributions can lead to approximate pairwise distance preservation [8]. In fact, the most crucial element which is present in all these variants of Random Projection is that the entries of the projection matrix must be selected *independently*.

For most distributions of the projection matrix, the proof of the JL-lemma follows a similar line of reasoning. First, one proves that the squared length of an arbitrary vector is preserved in expectation after the projection. Proof continues by showing that the squared length after the projection has a low variance, so with high probability the squared length of the vector will not get distorted by more than $(1 \pm \epsilon)$ by the projection. Then, the trivial union bound guarantees that, for $k = \mathcal{O}(\epsilon^{-2} \log(N))$, the probability that the projection matrix produces a relative distortion greater than $(1 \pm \epsilon)$ for any pair among the N samples is lower than $1 - \frac{1}{N}$. Therefore, by generating and evaluating $\mathcal{O}(N)$ projection matrices, one can raise the probability of success to the desired constant, leading to the claimed randomized polynomial time [33].

Another important aspect of Random Projections is whether they are able to preserve inner products as well. In this regard, guarantees for dot product preservation have been historically looser, and some authors suggested that dot products may not be preserved when the angle between the input vectors is obtuse [89]. More recent studies have improved those guarantees, clarifying how relative distortion bounds on dot products under Random Projections depend on the angles between the input vectors, and that in the worst case (when vectors are perpendicular) no relative distortion guarantees are available [56]. Aside from these difficulties in terms of the

[3] It should be noted that, while using i.i.d. random variables with zero mean and unit variance as the entries of the projection matrix guarantees that the pairwise distances will be preserved in expectation, using different distributions can result in different average errors and error tail bounds [70, 8].

theoretical guarantees, Random Projections have been successfully used in practice with a variety of algorithms that heavily rely on dot products [56, 40, 10].

In general, Random Projection is commonly regarded as an efficient, versatile, and simple dimensionality reduction method which approximately preserves the structure of data in the high dimensional space. As such, it has been applied to a variety of machine learning problems, ranging from people counting in images [39], to efficient high dimensional data clustering [19] among many others (for instance see [32, 55, 15, 47]).

2.2 Kernel functions and the kernel trick

Before delving into the specific properties of the polynomial kernel, we present some general ideas about kernel functions, kernel methods and the so-called kernel trick. Formally, a function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a kernel function[4] [76, 46] on $\mathbb{R}^d$ if there is a Hilbert space $\mathcal{H}$ and a possibly non-linear map $\phi : \mathbb{R}^d \to \mathcal{H}$ such that for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{H}}. \tag{2.4}$$

Intuitively, this means that the kernel function evaluates an inner product of the input samples after mapping them to a different feature space. In the literature, $\phi(\cdot)$ and $\mathcal{H}$ are commonly referred to as the *feature map* and *feature space* of the kernel. Figure 2.2 illustrates the relationship between the kernel function, the feature map and the feature space. It is important to note that $\phi(\cdot)$ and $\mathcal{H}$ need not be unique. That is, for a given kernel function, there might be various valid feature maps with their associated feature spaces.

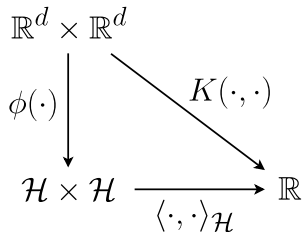


FIGURE 2.2: Provided that $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel function, the above diagram commutes. That is, all directed paths with the same start and endpoints lead to the same result. This illustrates how a kernel function provides a way to compute inner products in its associated feature space without explicitly operating in it.

An immediate consequence of the definition of kernel functions and the symmetry of the inner product is the symmetry of kernel functions:

$$K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x}). \tag{2.5}$$

[4] Kernel functions can be defined on more general input spaces than $\mathbb{R}^d$, but this definition suffices for the purposes of this thesis.

However, it must be noted that not all symmetric functions from $\mathbb{R}^d \times \mathbb{R}^d$ to $\mathbb{R}$ are kernel functions. In this regard, Mercer [76] showed that a symmetric function is a kernel if and only if it is positive definite. In particular, a symmetric function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is positive definite and therefore a kernel on $\mathbb{R}^d$ if and only if for any finite set of data samples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \in \mathbb{R}^d$ and any set of real numbers $\lambda_1, \ldots, \lambda_N$ it satisfies the following inequality:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \geq 0. \tag{2.6}$$

Conveniently, this means that if K is a positive definite function, then (2.4) holds for some feature space $\mathcal{H}$ and feature map $\phi(\cdot)$, even if we ignore their form. However, in order to prove that a given function is a valid kernel, it is sometimes easier to give an explicit feature map for it, or to show that the function itself is a combination of known kernels following some rules [46, 31].

One interesting property of kernel functions is that they can be used to compute Euclidean distances between data samples in the feature space [86], without explicitly evaluating the feature map. This property will be of great help in the following chapters to efficiently assess the distance preservation properties of the proposed algorithms. Formally, due to the linearity of the inner product and the definition of kernel functions:

$$\begin{aligned} ||\phi(\mathbf{x}) - \phi(\mathbf{y})||^2 &= \langle \phi(\mathbf{x}) - \phi(\mathbf{y}), \phi(\mathbf{x}) - \phi(\mathbf{y}) \rangle \\ &= \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle + \langle \phi(\mathbf{y}), \phi(\mathbf{y}) \rangle - 2\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \\ &= K(\mathbf{x}, \mathbf{x}) + K(\mathbf{y}, \mathbf{y}) - 2K(\mathbf{x}, \mathbf{y}). \end{aligned} \tag{2.7}$$

In addition to their importance in the mathematical literature, kernel functions have received a lot of attention from the machine learning community, mainly because they are closely related to some forms of non-linear classification. Specifically, a wide range of non-linear classifiers can be characterized by the following decision function:

$$h(\mathbf{x}) = \langle \phi(\mathbf{x}), \mathbf{w} \rangle + b, \tag{2.8}$$

where $\mathbf{w}$ is the weight vector, b is the intercept or bias, and $\phi(\cdot)$ is a non-linear feature map for some kernel function $K(\cdot, \cdot)$. Intuitively, the idea behind these classification methods is that a non-linearly separable problem in the input feature space might become more linearly separable after data samples are mapped to the feature space of some kernel by the non-linear feature map $\phi(\cdot)$ (see Figure 2.3). However, explicitly evaluating the inner product $\langle \phi(\mathbf{x}), \mathbf{w} \rangle$ can be computationally prohibitive, as $\phi(\mathbf{x})$ is usually very high dimensional, and infinite dimensional in some cases. To address this issue, a common approach [13, 30, 87] is to restrict the weight vector $\mathbf{w}$ to be:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}^{(i)}) \quad \text{for some } \alpha_1, \ldots, \alpha_N \in \mathbb{R}, \tag{2.9}$$

where $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ are the training samples of the classification problem. Conveniently, theoretical results ensure that, under fairly benign conditions, the optimal weight vector admits a representation of this form [7]. In addition, this restriction enables us to rewrite the decision function of the classifiers avoiding any explicit evaluation of $\phi(\cdot)$, using the kernel function instead:

$$h(\mathbf{x}) = \langle \phi(\mathbf{x}), \mathbf{w} \rangle + b = \left\langle \phi(\mathbf{x}), \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}^{(i)}) \right\rangle + b = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}^{(i)}) + b. \quad (2.10)$$

This way of using kernel functions to work in the feature space without explicitly evaluating $\phi(\cdot)$ is often referred to as the *kernel trick*.

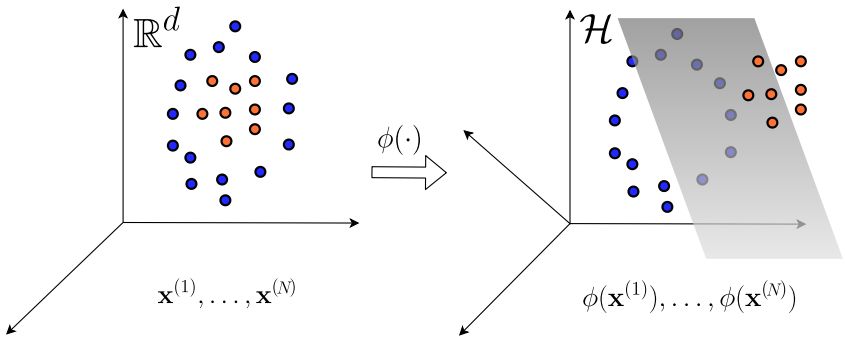


FIGURE 2.3: The core assumption made by kernelized classifiers is that, after being transformed by the feature map $\phi(\cdot)$, classification problems may become more linearly solvable.

While very popular, the kernel trick approach fails to scale to large data scenarios. In particular, note that in order to avoid the explicit use of $\phi(\cdot)$ when evaluating the decision function in (2.10), we have introduced N calls of the kernel function[5]. Assuming that the kernel function can be evaluated in $\mathcal{O}(d)$ time for d-dimensional samples, the complete inference time of such classifiers can be up to $\mathcal{O}(Nd)$, while linear classifiers offer inference times of $\mathcal{O}(d)$.

Scalability issues derived from the use of the kernel trick also arise in the training phase. For instance, the training time of kernelized classifiers such as kernelized support vector machines grows at least like N^2, and in some cases comes close to N^3, where N is the number of training samples [14]. Conversely, efficient algorithms exist for training linear classifiers in linear time [53, 12, 109]. This lack of scalability of algorithms using the kernel trick has even been referred to as the *curse of support* [58], and has motivated researchers to develop more efficient ways of exploiting the discriminative power of kernels, as we will see in Section 2.4.

[5]Usually, this problem is mitigated because an important fraction of $\alpha_1, \ldots, \alpha_N$ are equal to zero [14]. However, this depends on the particular problem at hand, and the number of non-zero alphas often grows linearly in the size of the training set [90].

2.3 Polynomial kernels

Polynomial kernels are a popular family of non-stationary kernels [46], with a great discriminative power and wide applicability [21, 106, 24]. Formally, polynomial kernels are those of the form:

$$K(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + c)^g, \tag{2.11}$$

where the hyperparameters g and c are referred to as the polynomial degree and the constant term. When c is set to a value greater than zero, the result is an inhomogeneous polynomial kernel. If instead the constant term c is set to zero, we obtain an homogeneous polynomial kernel:

$$K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle^g. \tag{2.12}$$

Intuitively, homogeneous polynomial kernels compute the inner product[6] in the feature space spanned by all degree-g monomials on the input features [49]. Conversely, when using an inhomogeneous polynomial kernel (i.e., $c > 0$) all monomials up to degree g are included in the feature space of the kernel.

As opposed to other popular kernel functions, polynomial kernels have finite dimensional feature spaces, and their associated feature maps can be easily determined once g and c are fixed. For instance, we can find an explicit feature map for the homogeneous polynomial kernel of degree two by analyzing the kernel function value:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle^2 &= \left(\sum_{i=1}^{d} \mathbf{x}_i \mathbf{y}_i \right)^2 \\ &= \sum_{i=1}^{d} (\mathbf{x}_i)^2 (\mathbf{y}_i)^2 + \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} (\sqrt{2}\mathbf{x}_i \mathbf{x}_j)(\sqrt{2}\mathbf{y}_i \mathbf{y}_j). \end{aligned} \tag{2.13}$$

Then, it is clear that the following mapping of dimension $d + (d^2 - d)/2$ is a valid feature map for the degree two homogeneous polynomial kernel [88]:

$$\begin{aligned} \phi(\mathbf{x}) = (\mathbf{x}_1^2, \ldots, \mathbf{x}_d^2, \sqrt{2}\mathbf{x}_1\mathbf{x}_2, \ldots, \sqrt{2}\mathbf{x}_1\mathbf{x}_d, \\ \sqrt{2}\mathbf{x}_2\mathbf{x}_3, \ldots, \sqrt{2}\mathbf{x}_2\mathbf{x}_d, \ \ldots, \ \ldots, \ \sqrt{2}\mathbf{x}_{d-1}\mathbf{x}_d). \end{aligned} \tag{2.14}$$

The properties of this feature map will be exploited in Chapter 3. However, it must be noted that the feature map given above is not the only one possible. In fact, an alternative feature map can be given for homogeneous polynomial kernels using

[6] From this point we focus on polynomial kernels, whose feature spaces are finite-dimensional Euclidean spaces. Thus, $\langle \cdot, \cdot \rangle$ shall henceforth denote the standard inner product (i.e., the dot product).

the Kronecker product [96, 82]. In particular, this feature map $\phi : \mathbb{R}^d \to \mathbb{R}^{d^g}$ is:

$$\phi(\mathbf{x}) = \underbrace{\mathbf{x} \otimes \mathbf{x} \cdots \otimes \mathbf{x}}_{g \text{ times}} = \bigotimes_{i=1}^{g} \mathbf{x}, \tag{2.15}$$

where $\otimes$ denotes the Kronecker product and g is the polynomial degree of the kernel. Again, the validity of this feature map for homogeneous polynomial kernels can be proved by analyzing the expression of the kernel itself. For instance, for $g = 2$ we have:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle^2 = \left(\sum_{i=1}^{d} \mathbf{x}_i \mathbf{y}_i \right)^2 = \sum_{i=1}^{d} \sum_{j=1}^{d} \mathbf{x}_i \mathbf{y}_i \mathbf{x}_j \mathbf{y}_j \\ = \sum_{i=1}^{d} \sum_{j=1}^{d} \mathbf{x}_i \mathbf{x}_j \cdot \mathbf{y}_i \mathbf{y}_j = \langle \mathbf{x} \otimes \mathbf{x}, \mathbf{y} \otimes \mathbf{y} \rangle = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle, \end{aligned} \tag{2.16}$$

which means that (2.15) with $g = 2$ is a valid feature map for the degree two homogeneous polynomial kernel[7]. The algorithms presented in Chapters 4 and 5 will take advantage of this feature map based on the Kronecker product.

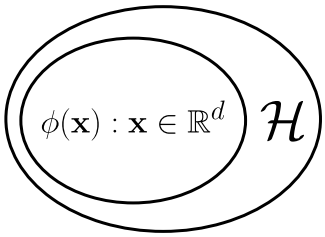
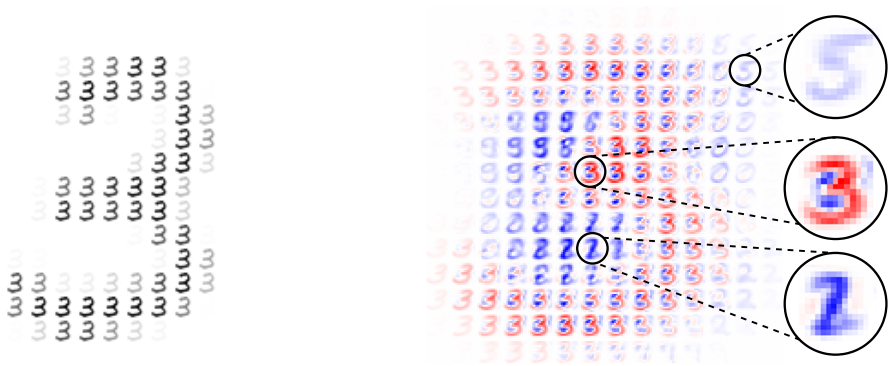


FIGURE 2.4: In the case of polynomial kernels, the feature space $\mathcal{H}$ is larger than the set of all $\phi(\mathbf{x})$ such that $\mathbf{x} \in \mathbb{R}^d$ [48].

Another relevant property of polynomial kernels is that the feature space $\mathcal{H}$ is larger than the set of all $\phi(\mathbf{x})$ (see Figure 2.4). For instance, if the input space is $\mathbb{R}^2$ and we consider the degree two homogeneous polynomial kernel with feature map $\phi(\mathbf{x}) = \mathbf{x} \otimes \mathbf{x} = (\mathbf{x}_1\mathbf{x}_1, \mathbf{x}_1\mathbf{x}_2, \mathbf{x}_2\mathbf{x}_1, \mathbf{x}_2\mathbf{x}_2)$, then one can choose $(-1, 1, -1, -1) \in \mathcal{H}$ but there is no $\mathbf{x} \in \mathbb{R}^2$ such that $\phi(\mathbf{x}) = (-1, 1, -1, -1)$. This illustrates that, while the kernel function provides us with a certain degree of access to the feature space, some regions of it are not directly reachable through the feature map. As a consequence, adapting existing algorithms to work with kernel functions is generally not as simple as naively replacing the inner products present in their formulation by kernel evaluations. This will become abundantly clear throughout the following chapters.

As mentioned before, the intuition behind the polynomial kernel family is that it is often useful to construct new features as the product of the original ones. The polynomial degree g in (2.12) determines the order of monomials composing the

[7] The proof for polynomial degrees greater than two is analogous.



Example of $\phi(\mathbf{x}) \in \mathcal{H}$ for the homogeneous polynomial kernel of degree two

$\mathbf{w}$ for a linear classifier trained directly on the feature space: $h(\mathbf{x}) = \tanh(\langle \phi(\mathbf{x}), \mathbf{w} \rangle + b)$

FIGURE 2.5: Visualization of a 14×14 digit from MNIST in the feature space of the homogeneous polynomial kernel of degree two, and the weights learned by a simple gradient descent linear classifier on that feature space. Positive weights are depicted in red and negative weights in blue. The classifier was trained to detect the digit "3".

feature map. To provide some intuition on the nature of homogeneous polynomial kernels, we can generate the explicit feature-space representation for a number of 14×14 resized images from the MNIST digits dataset[8]. Particularly, here we use the feature map given in (2.15), but rearrange the features to ease visualization. Then, a simple gradient-descent linear classifier is trained on them to distinguish the category "3" from all the others:

$$h(\mathbf{x}) = tanh(\langle \phi(\mathbf{x}), \mathbf{w} \rangle + b). \tag{2.17}$$

Figure 2.5 shows one of those samples in the kernel feature space and the weight vector learned by the linear classifier. As we can see, the classifier appears to be using different "templates" to emit a prediction, depending on the presence/absence of pixel intensity in the different regions of the original digit image. This illustrates how linear classifiers can benefit from polynomial features, as they make it possible for the model to account for more complex interactions between the original features.

Since the feature spaces of polynomial kernels are finite dimensional, some studies have explored the possibility of explicitly operating in them [21], relying on the sparsity of data samples to reduce computation and storage costs. Unfortunately, the dimension of mapped samples $\phi(\mathbf{x}) \in \mathcal{H}$ grows exponentially with the polynomial degree. As a consequence, any algorithm that explicitly operates in the feature space of polynomial kernels will rapidly become intractable as the original dimension of samples d or the polynomial degree g grow.

[8] http://yann.lecun.com/exdb/mnist/ (Date accessed: 10/12/2018).

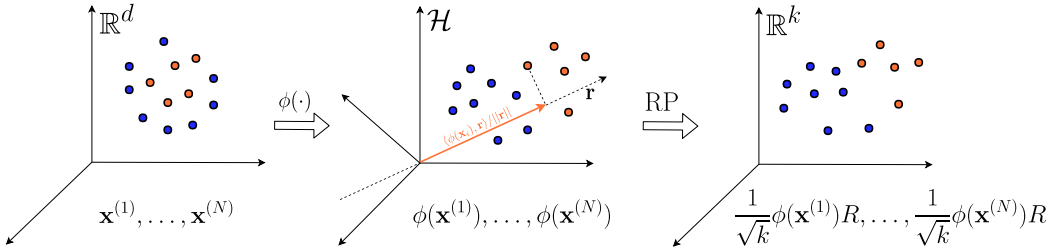


FIGURE 2.6: Schematic view of a Random Projection from the feature space of a kernel. If data is linearly separable in the feature space, an explicit Random Projection from it would approximatively preserve separability [11, 9].

2.4 Random Projections from kernel feature spaces

As we have seen, classifiers using the kernel trick [13, 30] achieve a higher discriminative power by implicitly working in the feature space. However, the use of the kernel trick compromises the scalability of the resulting classifiers both it terms of their training and inference times [109, 14]. This has motivated researchers to design new methods to combine the discriminative power of kernel functions with the efficiency of linear classifiers. A common approach is to design a feature-mapping algorithm which somehow captures the structure of data in the feature space of some kernel, while generating a relatively low-dimensional output representation [110, 5, 82, 69, 58, 97, 103]. This data representation capturing the information present in the kernel feature space is used to train efficient linear classifiers, which approximate the performance of their kernelized counterparts. Similarly, these feature-mapping algorithms can be used in other tasks in addition to classification, as many other problems may benefit from their kernel approximation properties. When designing such algorithms, the following features are desirable:

- Efficiency: The feature-mapping methods must be efficient, since otherwise the scalability of subsequent algorithms applied on the generated representations (e.g., linear classifiers) would be lost.
- Data-independence: Ideally, these algorithms should not require access to training data, or any other information about the distribution of the data samples that will be processed at test time. This makes feature-mapping methods suitable for online-learning scenarios and other applications where training data is not available.
- Kernel-independence: While some feature approximation methods focus on an specific kernel function or family of functions to achieve the aforementioned efficiency and data-independence, it its desirable for these methods to be compatible with any kernel function.

Since the goal of these feature-mapping algorithms is to preserve the properties of data in a very high dimensional space (the kernel feature space) while moving to a low-dimensional representation, a natural option to consider is the use of Random Projections. In some of the earlier works on this topic [11, 9], the authors noted that

if a classification problem is linearly separable by a large margin in the kernel feature space, then a Random Projection from it down to a low-dimensional space will approximately preserve separability (see Figure 2.6). This suggests that, given a classification problem with data samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, one might avoid using an inefficient kernelized classifier by first mapping samples to the kernel feature space and then performing a Random Projection: $\{(1/\sqrt{k})\phi(\mathbf{x}^{(1)})R, \ldots, (1/\sqrt{k})\phi(\mathbf{x}^{(N)})R\}$, to finally train a linear classifier on the resulting low-dimensional representation. Unfortunately, this approach is in most cases infeasible due to the high-dimensional nature of the representation generated by the feature map $\phi(\cdot)$. To avoid this issue, the authors of [11, 9] studied different mappings which avoid any explicit evaluation of $\phi(\cdot)$ and approximately preserve separability by using the kernel function and access to a number of unlabeled training data samples. Their work also addressed the question of whether mappings that approximately preserve linear separability from kernel feature spaces down to low-dimensional spaces can be achieved using only black-box access to the kernel function. Unfortunately, their results were negative, and the authors proved that this is not possible in general for an arbitrary black-box kernel, unless access to the distribution of data via a number of unlabeled data samples is also allowed. However, they left the question open of whether such methods could be developed for specific kernel functions such as the polynomial kernel.

More recently Alavi *et al.* [5, 110] proposed Kernelized Gaussian Random Projection (KG-RP), a general method to perform Random Projections from arbitrary kernel feature spaces. Their findings did not contradict the result described in the previous paragraph since the method they proposed required access to a number of unlabeled training samples in order to work. Interestingly, their algorithm was based on a technique developed to solve a different problem, namely the Kulis-Grauman approach [65]. This technique, originally developed to perform a kernelized variant of Locally Sensitive Hashing, can be used to generate a set of nearly Gaussian vectors in an arbitrary kernel feature space, without any evaluation of the feature map $\phi(\cdot)$. Despite its success, the method has a limitation inherent to its core idea: the approximately Gaussian hyperplanes in the kernel feature space are built as a weighted sum of a subset of the database items, thus making the method data-dependent. Formally, let $\mathbf{z}_t$ be the sum of a set S of t mapped training samples:

$$\mathbf{z}_t = \frac{1}{t} \sum_{i \in S} \phi(\mathbf{x}^{(i)}). \tag{2.18}$$

Then, the central limit theorem (CLT) guarantees that, for a sufficiently large t, the vector $\tilde{\mathbf{z}}_t = \sqrt{t}(\mathbf{z}_t - \mu)$ will be distributed according to the multivariate Gaussian $\mathcal{N}(\mathbf{0}, \Sigma)$ [110]. Therefore, applying a whitening transform, $\tilde{\mathbf{z}}_t \Sigma^{-\frac{1}{2}}$ will be distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The mean and covariance matrix need to be approximated from training data by selecting a set of p data samples where $t < p$. For details about how these computations are performed via the kernel function, see [66].

The contribution of Alavi *et al.* in [5, 110] demonstrated that, with minimal modifications, the Kulis-Grauman approach can be used to perform Random Projections

from the feature spaces of arbitrary kernels. In particular, their method defines each output component of the projection as:

$$\langle \phi(\mathbf{x}),\ \tilde{\mathbf{z}}_t \Sigma^{-\frac{1}{2}} \rangle, \tag{2.19}$$

which for sufficiently large t and p corresponds to a valid Gaussian Random Projection, since $\tilde{\mathbf{z}}_t \Sigma^{-\frac{1}{2}}$ converges in distribution to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Again, the evaluation of (2.19) is performed in practice without any explicit evaluation of the feature map, via the kernel function. However, because we are using database items to generate the projection vectors, this is a data-dependent approach. As a consequence, the quality of the embeddings it produces depends on the number of available training data samples and their variability. In addition, most of the computational efficiency of the original Random Projection method is lost in this version. For example, whereas the training phase in the original method only involves populating a projection matrix from a random distribution, the training phase in this kernelized variant entails expensive computations over training samples (see Section 4.3.4 for more details).

In this context, where existing kernelized variants of Random Projection are data-dependent and computationally demanding, this thesis focuses on the development of new data-independent and efficient methods to perform Random Projections from kernel feature spaces. Following the insights provided in [11, 9], we will focus on an specific kernel family, namely that of polynomial kernels, to preserve the data-independence of the original Random Projection algorithm.

2.5 Bilinear convolutional neural networks

As we have seen, performing Random Projections from the feature spaces of kernels has recently emerged as a promising alternative to the poorly scalable kernelized classifiers and the attempts to explicitly work in the feature spaces. To further motivate the importance of developing new methods to perform these Random Projections in an efficient and data-independent manner, and particularly considering the case of polynomial kernels, we analyze bilinear Convolutional Neural Networks (CNN) [71]. Interestingly, these deep learning models are intimately related to polynomial kernels. In Chapter 5, we will delve deeper into how Random Projection algorithms for polynomial kernels can be adapted to make bilinear models more efficient. In the remainder of this section we present the general properties of bilinear CNNs, and briefly discuss their connection to polynomial kernels.

In essence, bilinear pooling is a method designed to boost the accuracy of classification models in fine-grained visual recognition tasks [26, 37, 95, 6, 91]. CNNs using this technique are sometimes referred to as bilinear CNNs [71]. The bilinear pooling operation itself generates an orderless[9] global descriptor of an image by passing it through two different CNNs and then combining the feature maps generated

[9] An orderless descriptor is an image descriptor that combines the local features extracted from an image without considering the order of the locations at which the features were extracted.

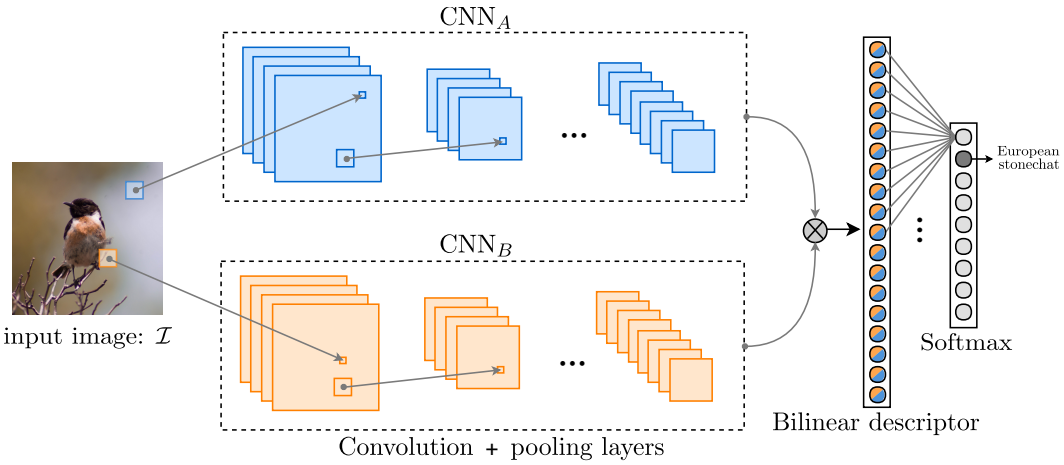


FIGURE 2.7: Structure of a typical bilinear CNN model for image classification. Images go through two different CNNs. The resulting feature maps are combined using the outer product at each location, and pooling is subsequently applied to form the final bilinear descriptor. A linear classifier such as a softmax layer is applied at the end to emit the predictions.

by the two models (see Figure 2.7). Particularly, the features extracted by each of the two CNNs are combined by applying the outer product at each location. The resulting local descriptors are then pooled, typically with sum pooling, to obtain the final descriptor. The intuition behind this approach is that the outer product helps the model to capture pairwise feature interactions in a location-invariant manner, which in turn improves its performance in fine-grained recognition tasks. The bilinear pooling operation can be formalized as follows. Let $\mathcal{I}$ represent an input image and CNN_A, CNN_B be the chosen CNNs. The bilinear descriptor is then computed as follows:

$$\Phi(\mathcal{I}) = \sum_{l \in \mathcal{L}} \text{CNN}_A(\mathcal{I}, l) \otimes \text{CNN}_B(\mathcal{I}, l), \tag{2.20}$$

where $\text{CNN}_A(\mathcal{I}, l)$ and $\text{CNN}_B(\mathcal{I}, l)$ denote the descriptors extracted from image $\mathcal{I}$ at location l by CNNs A and B respectively, $\mathcal{L}$ is the set of valid locations and $\otimes$ denotes the Kronecker product[10]. The final descriptor is usually normalized by first applying an element-wise signed square root operation (i.e., $x \to \text{sgn}(x)\sqrt{|x|}$), followed by L2 normalization.

While notably successful, bilinear CNNs are too computationally demanding for some applications. On the one hand, images have to be processed by two independent CNN models, which is expensive both in terms of memory and computational resources. On the other hand, the use of the outer/Kronecker product to combine the features extracted by the two CNNs results in descriptors of very high dimension. The first problem is often times addressed by making CNN_A and CNN_B be the same network, so only one forward pass is required in practice [71]. For the second issue, recent studies have suggested using various algorithms to condense the discriminative

[10] Here, the Kronecker product is used instead of the outer product to characterize bilinear pooling with the goal of easing the intended analogy with polynomial kernels. Nevertheless, these two operations are in this case equivalent as long as their output is reshaped to have one dimension.

information of the bilinear descriptor into a low-dimensional representation [43].

To draw the analogy between bilinear pooling and polynomial kernels, we need to focus on the scenario where CNN_A and CNN_B are the same network. If this is the case, the bilinear descriptor of image $\mathcal{I}$ becomes the summation of $\text{CNN}(\mathcal{I}, l) \otimes \text{CNN}(\mathcal{I}, l)$ over all locations $l \in \mathcal{L}$. As seen in Section 2.3, the mapping defined as $\phi(\cdot) : \mathbf{x} \rightarrow \mathbf{x} \otimes \mathbf{x}$ is a valid feature map for the degree-two homogeneous polynomial kernel. Therefore, the bilinear pooling operation is basically mapping the local descriptors extracted by the CNN to the feature space of the degree-two homogeneous polynomial kernel, and then summing all of them together. This connection to polynomial kernels will be exploited in Chapter 5 to efficiently approximate a Random Projection of the bilinear descriptor without even having to generate it explicitly. By performing the Random Projection from the kernel feature space in an implicit manner, we will avoid the problems pointed out by Gao *et al.* [43], who first discussed the possibility of using the Random Projection method to reduce the dimensionality of bilinear descriptors.

Chapter 3

Random Projections from the Feature Space of the Degree-Two Homogeneous Polynomial Kernel

Performing a Random Projection from the feature space associated to a kernel function may be useful for two main reasons: (1) As a consequence of the Johnson-Lindenstrauss lemma, the resulting low-dimensional representation will preserve most of the structure of data in the kernel feature space and (2) an efficient linear classifier trained on the projected data might approximate the accuracy of its non-linear counterparts. In this chapter, we present a novel method to approximate Random Projections from the feature space of the homogeneous polynomial kernel of degree two. As opposed to other kernelized Random Projection approaches, our method focuses on a specific kernel family to preserve the beneficial properties of the original Random Projection algorithm, namely its data independence and efficiency. Our experimental results evidence that the proposed method efficiently approximates a Random Projection from the kernel feature space, preserving pairwise distances and enabling a boost in linear classification accuracies.

The contents of this chapter have been adapted from the journal paper: Daniel López-Sánchez, Juan Manuel Corchado and Angélica González Arrieta. "Data-independent Random Projections from the feature-map of the Homogeneous Polynomial Kernel of degree two". In: Information Sciences 436-437C (2018), pp. 214-226.

3.1 Introduction

In this chapter, we present a novel method to efficiently approximate Random Projections from the feature space of the homogeneous polynomial kernel of degree two. By focusing on a specific kernel function, our method overcomes the limitations of previous kernelization attempts of Random Projection, which fail to preserve the data-independence and efficiency of the original algorithm. In addition, our method is compatible with the database-friendly distribution proposed by Achlioptas. Our

experimental results evidence that, by using our method, one can efficiently generate a low-dimensional representation of samples that captures the structure of data in the feature space of the degree-2 homogeneous polynomial kernel, approximately preserving the pairwise distances between samples in that space. Because of the data-independent nature of the proposed method, it can be applied in online-learning scenarios [38] where no data samples are initially available. In addition, this representation can be used to train efficient linear classifiers that approximate the accuracy of their non-linear counterparts.

Intuitively, we propose replacing the dot products that take place during the matrix multiplication in Random Projection, which correspond to the projection of samples in the data matrix X onto the columns of the projection matrix R, by evaluations of the kernel function. By so doing, the columns of the projection matrix and the data samples will be mapped by $\phi(\cdot)$, so the projection will in fact take place in the kernel feature space. Formally, each output component will be computed as:

$$X'_{nc} = \frac{1}{\sqrt{k}} K(\text{row}_n X,\ \text{col}_c R) = \frac{1}{\sqrt{k}} \langle \phi\left(\text{row}_n X\right), \phi(\text{col}_c R) \rangle , \tag{3.1}$$

where $X \in \mathbb{R}^{N \times d}$, $R \in \mathbb{R}^{d \times k}$ and $X' \in \mathbb{R}^{N \times k}$. However, this is not equivalent to explicitly mapping the data points to the kernel feature space by means of the feature map $\phi : \mathbb{R}^d \to \mathcal{H}$ and then performing a classic Random Projection. This is because we cannot guarantee that the distribution of the projection vectors in R will be preserved by the feature map $\phi(\cdot)$. Therefore, our goal will be to define the projection matrix R in such a way that when its columns are mapped by $\phi(\cdot)$ to the implicit, high-dimensional kernel feature space, the result will be a set of valid Random Projection vectors. In this regard, focusing on a specific kernel function will enable us to analyze how its specific feature map affects the columns of the projection matrix.

The rest of this chapter is structured as follows. Section 3.2 provides a brief review of related work. Section 3.3 describes the proposed approach and how it manages to efficiently approximate a Random Projection from the feature space of the homogeneous polynomial kernel of degree two. In Section 3.4, we present empirical evidence that our method approximates a Random Projection from the feature space of the degree-2 homogeneous polynomial kernel, so the pairwise distances between points in the feature space are approximately preserved in the resulting representation. Additional experimental results regarding the suitability of our method for the task of classification are reported in Section 3.5. Finally, Section 3.6 summarizes the conclusions and future research lines.

3.2 Related work

In the previous chapter, we saw that the feature space of the degree-2 homogeneous polynomial kernel consists of all the possible second order monomials of the original

features of a sample. For instance, in a text-processing problem where the features of samples represent the number of occurrences of a word in a document, the features in the kernel feature space correspond to co-occurrences of pairs of words, which may be more informative than individual frequencies. This explains why numerous machine learning problems are more easily solved in the feature space of polynomial kernels [64, 23, 107, 106].

Similarly, the suitability polynomial kernels for various classification tasks is supported by the results presented in [21]. In this work, the authors analyzed the effectiveness of the polynomial kernel of degree two in the context of Support Vector Machine (SVM) classification. Specifically, they apply fast linear-SVM classification methods to data samples explicitly transformed by the feature map associated to the polynomial kernel of degree two. Their results evidence that, using this approach, it is possible to achieve accuracy rates close to those achieved when using highly non-linear kernels (e.g., radial basis kernels) for various datasets. However, explicitly evaluating the feature maps of polynomial kernels is highly inconvenient due to the size of the resulting representations, except for some special cases such as when the original features exhibit a high level of sparsity. To evidence that explicitly mapping data samples to the feature space is in general an impractical approach, the experiments of this chapter evaluate the performance and efficiency of our method as compared to a classic Random Projection [3] from the explicitly computed feature space.

As discussed in the previous chapter, the Kulis-Grauman approach [66] was recently adopted to perform a kernelized version of Random Projection in the context of image classification [5]. Although this version is compatible with any kernel function, it loses much of the computational efficiency and conceptual simplicity of Random Projection. In addition, it requires access to a number of unlabeled training data samples, as opposed to the classic Random Projection algorithm in which the projection matrix is completely data-independent and no training stage is required beyond the random initialization of its entries. Later on, this kernelized version of Random Projection was extended and applied to the problem of image clustering [110]. The authors proposed three versions of the algorithm: Kernelized Gaussian Random Projection (KG-RP), Kernelized Orthonormal Random Projection (KORP), and Kernel Principal Component Analysis Random Projection (KPCA-RP). These three versions mainly differ in the manner in which the projection vectors are generated in the kernel feature space, but the mentioned limitations apply to the three of them. Our experimental results suggest that our method outperforms that of [5] in terms of distance preservation while having a much lower computational cost.

3.3 Proposed method

Given a set of N data samples from $\mathbb{R}^d$ represented as a $N \times d$ matrix X, the goal of our method is to map them to the high-dimensional, implicit feature space $\mathcal{H}$ of

the degree-2 homogeneous polynomial kernel and then perform a Random Projection from $\mathcal{H}$ down to $\mathbb{R}^k$ so the output samples are represented as a $N \times k$ matrix X'. In addition, these operations should be performed implicitly via the kernel function, avoiding any explicit evaluation of the feature map $\phi(\cdot)$. As mentioned before, seeking to make (3.1) approximate a valid Random Projection in the kernel feature space, we will define an appropriate distribution for the projection matrix R by analyzing the properties of the specific kernel function we have chosen. In particular, as seen in Section 2.3 the degree-2 homogeneous polynomial kernel is:

$$K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle^2 \quad \text{with} \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^d, \tag{3.2}$$

and the following mapping with $d + (d^2 - d)/2$ features represents a valid feature map for it (for example see [88]):

$$\begin{aligned} \phi(\mathbf{x}) = (\mathbf{x}_1^2, \ldots, \mathbf{x}_d^2, \sqrt{2}\mathbf{x}_1\mathbf{x}_2, \ldots, \sqrt{2}\mathbf{x}_1\mathbf{x}_d, \\ \sqrt{2}\mathbf{x}_2\mathbf{x}_3, \ldots, \sqrt{2}\mathbf{x}_2\mathbf{x}_d, \quad \ldots, \ \ldots, \quad \sqrt{2}\mathbf{x}_{d-1}\mathbf{x}_d). \end{aligned} \tag{3.3}$$

Since the form of a finite-dimensional feature map associated to the kernel is known, one might consider performing a standard Random Projection from this representation. Following this approach, data samples would be transformed by $\phi(\cdot)$ and then projected to a low-dimensional space with a classic Random Projection. In fact, if the sparse distribution proposed by Achlioptas was used [3, 70], the projection matrix R might be efficiently stored using a sparse matrix implementation. However, explicitly computing $\phi(\mathbf{x})$ might be a very expensive and almost intractable task. This is mainly due to the fact that the dimension of $\phi(\mathbf{x})$ is $\mathcal{O}(d^2)$. As a consequence, trying to explicitly transform a set of samples with $\phi(\cdot)$ requires intensive computations and a significant storage capacity, especially if data samples are dense and sparse matrix routines cannot be used, which is often the case. For instance, we will see that storing the explicit form of $\phi(\mathbf{x})$ for 200 samples from a dataset of color images of size 96×96 would require more than 284 Gigabytes of memory, which is far beyond the current capacity of most devices' main memory. Nevertheless, in Section 3.4 we empirically compare this approach to our proposed method both in terms of performance and efficiency. Henceforth, we will refer to the explicit transformation of data samples by $\phi(\cdot)$ followed by the application of a classic Random Projection as the *explicit approach*.

Turning back to our implicit approach, having found a valid feature map for the studied kernel enables us to define a distribution for the columns of the projection matrix such that, when transformed by $\phi(\cdot)$, their features follow a valid Random Projection distribution, at least when analyzed individually. In this manner, we might be able to perform the Random Projection via the kernel function, thus avoiding any explicit evaluation of $\phi(\cdot)$. Particularly, we will consider what happens if we

populate the entries of the projection matrix R according to the following distribution, where s controls the sparsity level:

$$R_{ij} = \frac{\sqrt{s}}{\sqrt[4]{2}} \begin{cases} 1 & \text{with prob.} \quad 1/2s \\ 0 & \text{with prob.} \quad 1 - 1/s \\ -1 & \text{with prob.} \quad 1/2s \end{cases}. \tag{3.4}$$

Let $\mathbf{r} = (r_1, \ldots, r_d) \in \mathbb{R}^d$ be any of the columns of R. Then, consider the distribution of the entries of $\phi(\mathbf{r})$ (i.e., one of the projection vectors onto which data samples will be projected). When analyzing its entries individually, we observe that two different distributions emerge:

$$\phi(\mathbf{r}) = (\underbrace{r_1^2, \ldots, r_d^2}_{\text{distribution } A}, \underbrace{\sqrt{2}r_1r_2, \ldots, \sqrt{2}r_1r_d, \sqrt{2}r_2r_3, \ldots, \sqrt{2}r_2r_d, \ldots, \sqrt{2}r_{d-1}r_d}_{\text{distribution } B}), \tag{3.5}$$

where distributions A and B are:

$$A := \frac{s}{\sqrt{2}} \begin{cases} 1 & \text{with prob.} \quad \frac{1}{s} \\ 0 & \text{with prob.} \quad 1 - \frac{1}{s} \end{cases}, \qquad B := \sqrt{s^2} \begin{cases} 1 & \text{with prob.} \quad \frac{1}{2s^2} \\ 0 & \text{with prob.} \quad 1 - \frac{1}{s^2} \\ -1 & \text{with prob.} \quad \frac{1}{2s^2} \end{cases}. \tag{3.6}$$

On the one hand, distribution B matches the sparse distribution originally proposed by Achlioptas [3] and later generalized[1] by Li *et al.* [70]. On the other hand, A is not a valid Random Projection distribution, and does not even have a zero mean. Luckily, only the first d entries of $\phi(\mathbf{r})$ follow distribution A, whereas the total number of features in $\phi(\mathbf{r})$ is $d + (d^2 - d)/2$. Therefore, the percentage of components of $\phi(\mathbf{r})$ that follow distribution A tends to zero as d grows, so we might expect the negative impact of these incorrectly distributed entries of the projection vectors to be small in practice, especially for large values of d.

The experimental results reported in Section 3.4 show that pairwise distances between data points in $\mathcal{H}$ are approximately preserved in the reduced representation generated by using (3.1) with the entries of R populated according to (3.4). However, if we compare the distance distortion induced by our method to the distortion induced by the explicit mapping of points to $\mathcal{H}$ and its reduction to $\mathbb{R}^k$ by means of a classic Random Projection (i.e., the explicit approach), we see that the former is slightly higher. To explain this difference, we must consider the requirements that the projection matrix must satisfy in order to have a valid Random Projection. As discussed in Section 2.1, a common requirement among the different variants of Random Projection is that the entries of the projection matrix must be *independently* selected. Thus far, we have focused on analyzing the distribution of the entries of the projection vectors in the feature space *individually*, so we have not considered

[1] Remember that Achlioptas used $s = 1, 3$. However, later studies showed that greater sparsity levels could be used with little loss in performance [70]. See Section 2.1 for more details.

the possible dependence among them. Unfortunately, the entries of our implicit projection vectors do not satisfy the independence condition. Consider for example the entries $\sqrt{2}r_1r_2$, $\sqrt{2}r_2r_3$ and $\sqrt{2}r_1r_3$ of $\phi(\mathbf{r})$, when the sparsity hyperparameter is $s = 1$. Once the sign of the first two entries is known, the sign of the third is completely determined, which evidences that they are not statistically independent. To provide additional insight into the nature of this dependence, we can explicitly generate a number of projection vectors and use some empirical dependence measures. Figure 3.1 shows the correlation matrices of 500 projection vectors generated by a) directly sampling from Achlioptas' distribution in the feature space and b) computing $\phi(\mathbf{r})$ with $\mathbf{r} \in \mathbb{R}^d$ drawn from (3.4). As we can see, except from some random noise, the entries of the projection vectors generated by either method seem to be uncorrelated. However, we know that in the case of $\phi(\mathbf{r})$ the entries are not independent. This means that the dependence among the entries is more subtle than a mere linear correlation. Therefore, to properly visualize this dependence, we need to use a measure capable of detecting non-linear dependence. To this end, we apply the *distance correlation*[2] measure. Figure 3.2 shows the distance correlation matrices for the two types of projection vectors described before. Looking at Figure 3.2.b, we can see that distance correlation successfully reveals the structure of the dependence among the entries of $\phi(\mathbf{r})$. Logically, the dependence mainly exists among pairs of features in $\phi(\mathbf{r})$ which have factors in common (see Figure 3.3). This lack of independence in the entries of the projection vectors causes the mentioned deviation in the distance preservation capabilities with respect to the explicit approach. In the next section we study an alternative way of building the projection vectors to overcome this problem.

3.3.1 Projection vectors and the Central Limit Theorem

This section presents an alternative approach to construct the projection matrix in the implicit feature space of the degree-2 homogeneous polynomial kernel. Most of the entries of this implicit projection matrix will be nearly independent variables drawn from a normal distribution with zero mean and unit variance, which is one of the valid Random Projection distributions (see Section 2.2). To populate this implicit matrix and perform the projection in the implicit feature space without any explicit evaluation of $\phi(\cdot)$, our method relies both on the properties of kernels and the Central Limit Theorem [57].

First, we populate t projection matrices each of dimension $d \times k$. We will refer to the *i-th* projection matrix as $R^{(i)}$. After this, an arbitrary set of N points represented as an $N \times d$ matrix X can be implicitly mapped from $\mathbb{R}^d$ to $\mathcal{H}$ and then projected

[2]Distance correlation is a statistical measure of dependence between random variables. As opposed to Pearson's correlation coefficient, distance correlation takes a value of zero if and only if the variables are statistically independent, as it is capable of detecting non-linear relations [93].

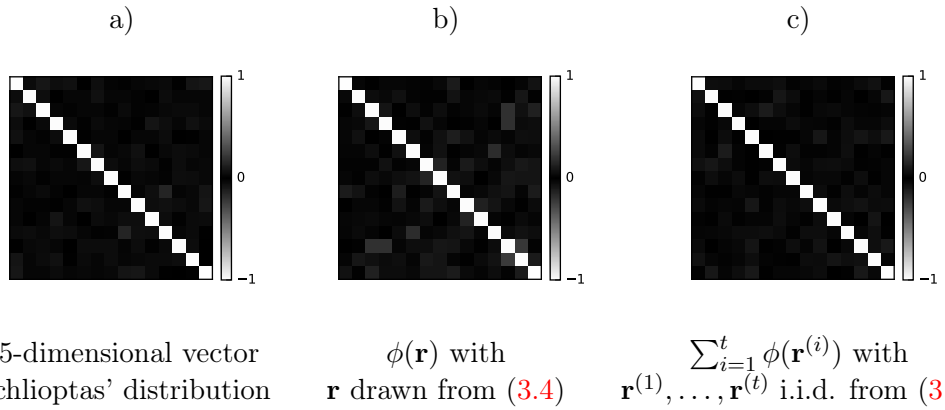


FIGURE 3.1: Correlation matrices computed over 500 15-dimensional samples generated by different methods ($d = 5$, $t = 30$, $s = 3$).

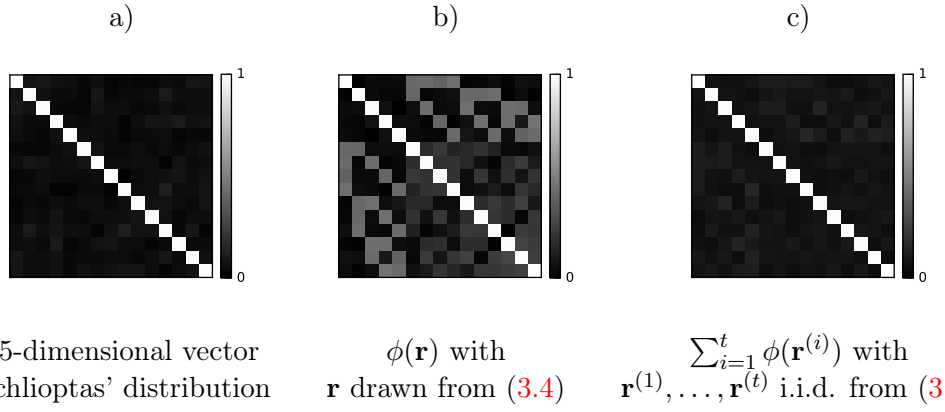
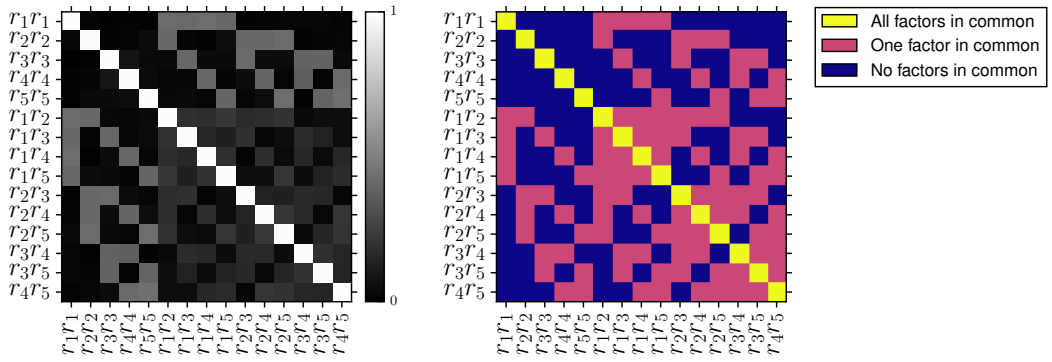


FIGURE 3.2: Distance correlation matrices [93] computed over 500 15-dimensional samples generated by different methods ($d = 5$, $t = 30$, $s = 3$).



$$\mathbf{r} = (r_1, r_2, r_3, r_4, r_5)$$

$$\phi(\mathbf{r}) = (r_1^2, r_2^2, r_3^2, r_4^2, r_5^2, \sqrt{2}r_1r_2, \sqrt{2}r_1r_3, \sqrt{2}r_1r_4, \sqrt{2}r_1r_5, \sqrt{2}r_2r_3, \sqrt{2}r_2r_4, \sqrt{2}r_2r_5, \sqrt{2}r_3r_4, \sqrt{2}r_3r_5, \sqrt{2}r_4r_5)$$

FIGURE 3.3: Distance correlation [93] computed over 1000 15-dimensional vectors of the form $\phi(\mathbf{r})$, with $\mathbf{r} \in \mathbb{R}^5$ drawn from (3.4). The dependence detected by the distance correlation measure can be explained by the number of shared factors among feature pairs.

down to $\mathbb{R}^k$ as follows:

$$X'_{nc} = \frac{1}{\sqrt{k}} \sum_{i=1}^{t} K\left(\text{row}_n X,\ \text{col}_c R^{(i)}\right). \tag{3.7}$$

To show that (3.7) approximates a valid Random Projection by reducing the aforementioned dependence in the entries of the implicit projection vectors, we begin by applying the main property of kernel functions, $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$, to reveal the computations that take place in the feature space. Then, we use the linearity of the inner product to rewrite the equation as a single inner product and use the summation notation to simplify the formula:

$$\begin{aligned} X'_{nc} &= \frac{1}{\sqrt{k}} \sum_{i=1}^{t} K\left(\text{row}_n X,\ \text{col}_c R^{(i)}\right) \\ &= \frac{1}{\sqrt{k}} \sum_{i=1}^{t} \left\langle \phi\left(\text{row}_n X\right), \phi\left(\text{col}_c R^{(i)}\right) \right\rangle \\ &= \frac{1}{\sqrt{k}} \left\langle \phi\left(\text{row}_n X\right), \sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right) \right\rangle. \end{aligned} \tag{3.8}$$

The above equations show that (3.7) corresponds to the mapping of the N-th row of X to the kernel feature space and its projection (dot product) onto a vector of the form $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$. Therefore, in order for (3.7) to compute a valid Random Projection from the kernel feature space, $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$ should follow one of the valid Random Projection distributions (see Section 2.4). In this case, we will use the Central Limit Theorem to ensure that most of the entries of the projection vectors of the form $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$ follow a normal distribution with zero mean and unit variance. To this extent, we define the following distribution to populate the projection matrices $R^{(1)}, \ldots, R^{(t)}$:

$$R_{ij}^{(\cdot)} = \frac{\sqrt{s}}{\sqrt[4]{2t}} \begin{cases} 1 & \text{with prob.} \quad 1/2s \\ 0 & \text{with prob.} \quad 1 - 1/s \\ -1 & \text{with prob.} \quad 1/2s \end{cases}. \tag{3.9}$$

We can again analyze the distribution of entries in the columns of the projection matrices after they are mapped to the feature space. Let $\mathbf{r} = (r_1, \ldots, r_d) \in \mathbb{R}^d$ be a column of any of the t projection matrices populated according to (3.9). One more time, two distributions emerge when we analyze the entries of $\phi(\mathbf{r})$ individually:

$$\phi(\mathbf{r}) = (\underbrace{r_1^2, \ldots, r_d^2}_{\text{distribution } A}, \underbrace{\sqrt{2}r_1r_2, \ldots, \sqrt{2}r_1r_d, \sqrt{2}r_2r_3, \ldots, \sqrt{2}r_2r_d, \ldots, \sqrt{2}r_{d-1}r_d}_{\text{distribution } B}), \tag{3.10}$$

where distributions A and B are:

$$A := \frac{s}{\sqrt{2t}} \begin{cases} 1 & \text{with prob.} \quad \frac{1}{s} \\ 0 & \text{with prob.} \quad 1 - \frac{1}{s} \end{cases}, \quad B := \frac{s}{\sqrt{t}} \begin{cases} 1 & \text{with prob.} \quad \frac{1}{2s^2} \\ 0 & \text{with prob.} \quad 1 - \frac{1}{s^2} \\ -1 & \text{with prob.} \quad \frac{1}{2s^2} \end{cases}. \tag{3.11}$$

Considering B as a discrete random variable, we can determine its mean and variance as follows:

$$\mu = \mathbf{E}[B] = \sum_b b \cdot P(b) = \frac{s}{\sqrt{t}} \cdot \frac{1}{2s^2} - \frac{s}{\sqrt{t}} \cdot \frac{1}{2s^2} = 0, \tag{3.12}$$

$$\sigma^2 = \mathbf{E}[(B-\mu)^2] = \sum_b (b-\mu)^2 \cdot P(b) = \frac{s^2}{t} \cdot \frac{1}{2s^2} + \frac{s^2}{t} \cdot \frac{1}{2s^2} = \frac{1}{t}, \tag{3.13}$$

where $\sum_b$ denotes the sum over all possible values of B and $P(b)$ is the probability of the specific value b in the random distribution of B. Given that the entries of $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$ are the sum of t independent and identically distributed random variables, and that entries following distribution B have zero mean and $1/t$ variance, we can apply the Central Limit Theorem (CLT) [57] to ensure that, except for the first d coordinates, the entries of $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$ will follow a standard normal distribution for a sufficiently large t.

Interestingly, as a side effect of the CLT, the statistical dependence among the entries of $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$ becomes smaller as we increase t, which was our goal from the beginning. In the next chapter, we will delve deeper into the reason why this happens, but for now we will rely on empirical evidence. Figures 3.1.c and 3.2.c show the result of explicitly generating 500 vectors of the form $\sum_{i=1}^{t} \phi(\mathbf{r}^{(i)})$ with $\mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(t)} \in \mathbb{R}^d$ populated according to (3.9) and then computing the correlation and distance correlation matrices for them. As expected, almost no statistical dependence is detected among the entries of the projection vectors when this approach is used, neither by the correlation coefficient nor by the distance correlation measure. This supports our claim that this approach based on the CLT is capable of reducing the dependence among the entries of the implicit projection vectors.

Again, we face the problem that the first d entries of the projection vectors will not follow the desired distribution. Moreover, since the first d components of $\phi\left(\text{col}_c R^{(i)}\right)$ follow distribution A and therefore are never negative, the first d entries of $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$ will grow larger as t increases, having a negative impact in the distance preservation capabilities of our method. To solve this problem, we force t to be an even number and multiply half of the terms in the summation of (3.7) by minus one. By doing so, the first d elements of the projection vectors will at least exhibit a zero mean, and the distribution of the remaining entries will not be modified due to de symmetry about zero of distribution B.

Since by the CLT most of the entries in $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$ are distributed according to $\mathcal{N}(0,1)$, and the dependence among these entries is reduced as t increases, we can expect (3.7) to approximate a valid Random Projection from the kernel feature

space for a sufficiently large t. Note that, due to the fact that the first d features in $\sum_{i=1}^{t} \phi\left(\text{col}_c R^{(i)}\right)$ do not follow a valid Random Projection distribution, we cannot ensure that the JL-lemma will be satisfied. However, as we have seen in the previous section, the percentage of entries in the projection vectors that follow distribution A tends to zero as d grows larger. In fact, our experimental results evidence that the effect of these features is indeed negligible, as in all cases our method was able to induce an average distortion in samples as low as that of the explicit approach by setting a large enough t. Also, Section 3.4.4 reports on the results of statistical tests which ensure that, for a sufficiently large t, there is no statistically significant difference between the distance preservation capabilities of the proposed approach and a classic Random Projection from the explicitly-computed kernel feature space. The effect of choosing different values of t for the proposed method is empirically studied in Section 3.4.

Finally, note that the distribution proposed in this section to populate the projection matrices is conveniently a generalization of the one proposed in the previous section. When t is set to one (that is, when only one projection matrix is used) the distribution proposed in (3.9) is the same as in (3.4). Henceforth, we will refer to our method as Degree-2 Polynomial-Kernel Random Projection[3] (D2PK-RP), indicating in each case the selected value for t. When $t = 1$, data is simply transformed by using (3.1) with the entries of R populated according to (3.4) (i.e., the method described in the previous section). When $t \geq 2$, we transform data samples following the procedure described in this section, which is summarized in Algorithm 1.

Algorithm 1 Degree-2 Polynomial-Kernel Random Projection (D2PK-RP)

Require: A set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ from $\mathbb{R}^d$, the number of projection matrices t which must be an even number, the sparsity level s and the desired output dimension k.

Ensure: Returns a set of output samples $\{\mathbf{x}'_1, \ldots, \mathbf{x}'_N\}$ in $\mathbb{R}^k$ such that pairwise distances between these samples are approximately equal to those of input data samples mapped to the feature space of the homogeneous polynomial kernel of degree two.

Generate t projection matrices $R^{(1)}, \ldots, R^{(t)} \in \mathbb{R}^{d \times k}$,
  with entries $\{\frac{\sqrt{s}}{\sqrt[4]{2t}}, 0, -\frac{\sqrt{s}}{\sqrt[4]{2t}}\}$ w.p. $\{\frac{1}{2s}, 1 - \frac{1}{s}, \frac{1}{2s}\}$
for $n = 1, \ldots, N$ **do** ▷ Iterate over samples
  $\mathbf{x}'_n \leftarrow (0, \ldots, 0) \in \mathbb{R}^k$ ▷ Initialize output vector with zeros
  for $c = 1, \ldots, k$ **do** ▷ Iterate over output dimensions
    for $i = 1, \ldots, t/2$ **do** ▷ Iterate to apply CLT (3.7)
      $\mathbf{x}'_n[c] \leftarrow \mathbf{x}'_n[c] + \langle \mathbf{x}_n, col_c R^{(i)} \rangle^2$
    for $i = t/2 + 1, \ldots, t$ **do** ▷ Flip the sign of half of the terms
      $\mathbf{x}'_n[c] \leftarrow \mathbf{x}'_n[c] - \langle \mathbf{x}_n, col_c R^{(i)} \rangle^2$
  $\mathbf{x}'_n \leftarrow \frac{1}{\sqrt{k}} \cdot \mathbf{x}'_n$ ▷ Final scaling
return $\{\mathbf{x}'_1, \ldots, \mathbf{x}'_N\}$

[3] In the original publication [73], the proposed method was simply referred to as Polynomial Random Projection (P-RP). The change in the name here is to disambiguate with other methods described in the following chapters.

3.3.2 Computational complexity of the proposed approach

The original Random projection algorithm has a remarkably low computational complexity: the training step (i.e., populating the projection matrix R) takes $\mathcal{O}(dk)$ time, and projecting a $N \times d$ data matrix X to $\mathbb{R}^k$ can be done at the cost of $\mathcal{O}(Ndk)$ time [10]. In the proposed variation of the algorithm, the training step consists in populating t projection matrices, thus having a complexity of $\mathcal{O}(tdk)$. Given that the polynomial kernel can be computed in $\mathcal{O}(d)$, the complexity of projecting the $N \times d$ data matrix X to k dimensions by means of (3.7) is of order $\mathcal{O}(Ntdk)$. However, since t is a hyperparameter whose value does not depend on the input data, it can be considered as a constant. By doing this, the complexities of the training and test phases turn out to be the same as in the original RP algorithm. This indicates that the scalability of D2PK-RP in the number of training samples and their dimension is the same as that of the original RP method.

However, it must be noted that the value of the hyperparameter t has a direct impact in the efficiency of D2PK-RP. Since t controls the number of projection matrices used, increasing t results in higher computational and storage costs. An alternative way of reducing computational costs would be re-using some of the random vectors for various output components, and in fact this idea will be applied in the following chapters.

3.4 Experimental results on distance preservation

The proposed technique seeks to implicitly map data samples from $\mathbb{R}^d$ to $\mathcal{H}$ and then project them to $\mathbb{R}^k$ in such a way that pairwise distances between samples in $\mathcal{H}$ are preserved in the resulting representation. To evaluate the distance preservation properties of the different approaches, we compare the squared Euclidean distance between two dimensionality-reduced data samples to their squared Euclidean distance in the kernel feature space. To do so, we use the following measure. Let $\mathbf{x}$ and $\mathbf{y}$ be a couple of samples from $\mathbb{R}^d$ and let $\mathbf{x}', \mathbf{y}'$ be their representation in $\mathbb{R}^k$, generated by some degree-2 homogeneous polynomial kernel approximation method, then:

$$distortion_{\mathbf{x},\mathbf{y}} = \frac{\text{abs}(\ ||\mathbf{x}' - \mathbf{y}'||^2 - ||\phi(\mathbf{x}) - \phi(\mathbf{y})||^2\ )}{||\phi(\mathbf{x}) - \phi(\mathbf{y})||^2}. \tag{3.14}$$

This measure can be easily interpreted. For example, if $distortion_{\mathbf{x},\mathbf{y}} = 0.12$ we can conclude that the distance between both samples in $\mathcal{H}$ suffered a 12% distortion (increase or decrease) in the resulting representation. Note that $||\phi(x) - \phi(y)||^2$ can be calculated without any explicit evaluation of $\phi(\cdot)$, via the kernel function (see Section 2.2):

$$||\phi(\mathbf{x}) - \phi(\mathbf{y})||^2 = K(\mathbf{x},\mathbf{x}) + K(\mathbf{y},\mathbf{y}) - 2K(\mathbf{x},\mathbf{y}). \tag{3.15}$$

To measure the distortion induced by a given method while reducing a set of N samples, the average distortion among all the $\binom{N}{2}$ possible pairs of different samples is computed. We shall use this average distortion measure to compare the different approaches discussed in this chapter. Experiments were carried out using three datasets from different domains, namely artificial vision and speech recognition. Together with D2PK-RP, we also evaluated the effectiveness of alternative methods such as the explicit approach (see Section 3.3) and Kernelized Gaussian Random Projection (KG-RP) [110].

Regarding the parametrization of KG-RP, the hyperparameter p controls the number of samples used to estimate the distribution of data in the kernel feature space, and must be set manually. By its nature, increasing its value will likely cause the performance of the algorithm to improve at the expense of longer running times. The authors of this method suggest using $p = \mathcal{O}(\sqrt{N})$ to achieve a good balance between efficiency and performance [66, 110], where N is the number of training samples. We followed this recommendation in our experiments and evaluated KG-RP with $p = 3\sqrt{N}$, $6\sqrt{N}$, and $9\sqrt{N}$. In addition, we used KG-RP with $t = 10$ in all the experiments. Regarding the sparsity level for the vectors in D2PK-RP, for simplicity we used $s = 1$ in all cases. To support our claims about the scalability of the proposed method, each distance distortion result reported in this section is provided along with the corresponding training/embedding times[4].

3.4.1 Experiments on CIFAR-10

The CIFAR-10 dataset [63] consists of 60000 color images of size 32×32, distributed among 10 different categories (see Figure 3.4). The train/test split is usually arranged with 50000/10000 images respectively. Given that the images are of size 32×32 with three channels, the sample dimension d is 3072. Therefore, the dimensionality of the feature space for the homogeneous polynomial kernel of degree two is 4,720,128. In this case, the explicit approach (i.e., the explicit mapping of the samples by means of $\phi(\cdot)$ and their reduction with a classic Random Projection) is still tractable. However, it is extremely demanding both in terms of computational power and memory. For example, storing 200 samples from $\mathcal{H}$ as a matrix of 32-bit floats requires approximately 3.5 Gigabytes of memory.

To compare the different approaches, 200 samples were selected at random from the whole dataset. Then, the samples were transformed by means of the different methods discussed in this chapter and the average pairwise distance distortion was measured. Figure 3.5.a shows the average distortion induced by the different methods as the resulting dimension k grows.

[4] In the experiments, KG-RP training was performed by invoking the original Matlab implementation from Python. Slightly better training times might be achieved by entirely porting the Matlab implementation to Python, to avoid overheads.

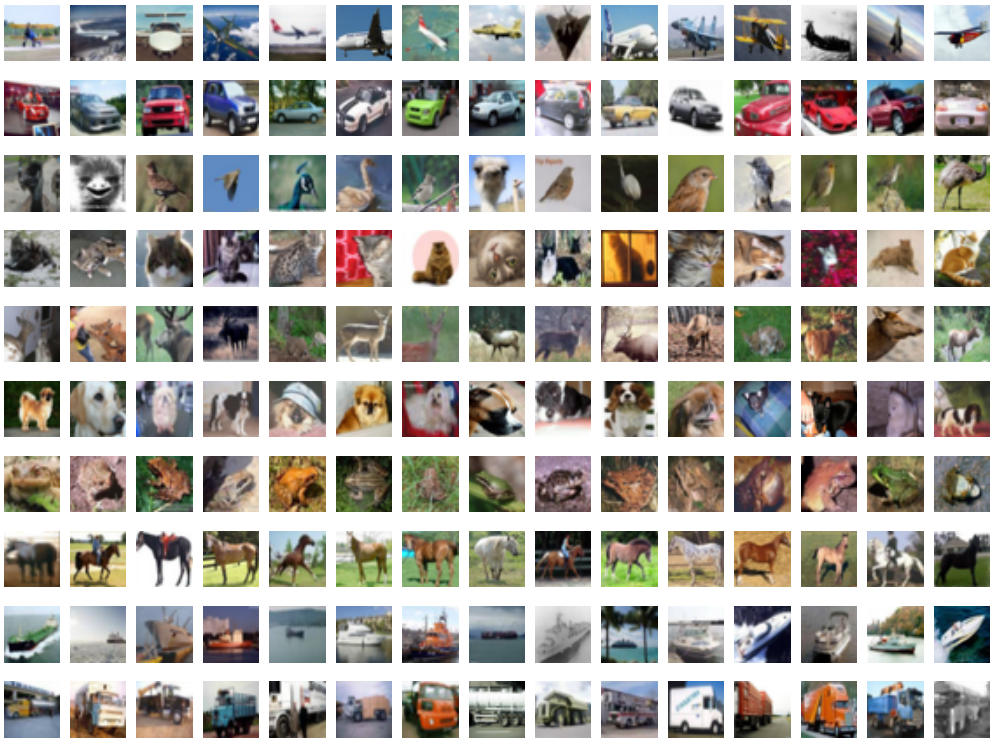


FIGURE 3.4: Some random examples from the different categories of CIFAR-10.

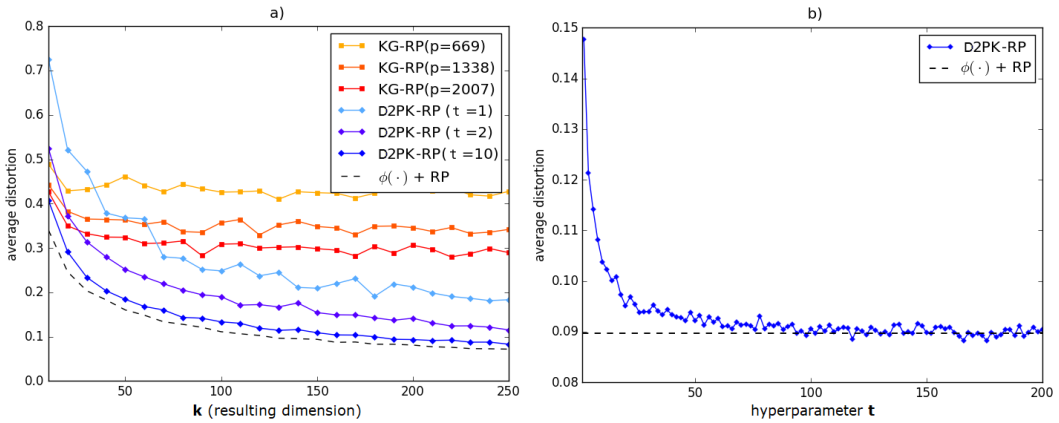


FIGURE 3.5: a) Average distortion induced on CIFAR10 samples by using different methods as the resulting dimension grows. b) Effect of different values for the hyperparameter t on CIFAR10 samples transformed by D2PK-RP. The resulting dimension was fixed to 160.

As we can se in Figure 3.5.a, the method proposed in Section 3.3 (i.e., D2PK-RP with $t = 1$) provides a reasonably low distortion given its simplicity. However, as explained before, a significant difference exits between the effectiveness of this method and the explicit approach. On the other hand, the method proposed in Section 3.3.1 to overcome that limitation (i.e., D2PK-RP with $t > 1$) shows a rapid decrease in the induced distortion as the hyperparameter t grows. Table 3.1 compiles the resulting average distortions obtained using different methods and various values of k and t. To mitigate the stochastic nature of the evaluated methods, each experiment was

TABLE 3.1: Average pairwise distance distortion induced by different methods and hyperparameters on 200 samples from CIFAR10. Training/embedding times are also provided.

Method	**k=40**	**k=80**	**k=120**	**k=160**
$\phi(\cdot) + RP$	0.177 ± 0.007 7433.7/4837.7 ms	0.122 ± 0.003 9974.4/5263.4 ms	0.103 ± 0.004 18288.6/6142.6 ms	0.090 ± 0.003 36351.8/9226.9 ms
D2PK-RP (t=1)	0.386 ± 0.040 3.6/0.41 ms	0.277 ± 0.022 5.6/0.61 ms	0.239 ± 0.035 10.6/0.81 ms	0.239 ± 0.033 16.6/0.95 ms
D2PK-RP (t=2)	0.294 ± 0.025 2.14/0.81 ms	0.212 ± 0.018 5.05/1.29 ms	0.170 ± 0.009 6.85/1.36 ms	0.147 ± 0.006 12.5/1.66 ms
D2PK-RP (t=10)	0.204 ± 0.008 11.6/3.58 ms	0.145 ± 0.007 28.7/3.66 ms	0.119 ± 0.006 38.9/7.8 ms	0.101 ± 0.005 47.4/8.11 ms
D2PK-RP (t=30)	0.188 ± 0.016 35.2/9.9 ms	0.132 ± 0.005 70.0/16.9 ms	0.109 ± 0.004 116.8/18.5 ms	0.091 ± 0.002 148.7/23.4 ms
KG-RP (p=669)	0.438 ± 0.027 3311.0/5.02 ms	0.431 ± 0.030 3344.5/5.44 ms	0.430 ± 0.019 3430.0/6.59 ms	0.424 ± 0.010 3527.6/6.86 ms
KG-RP (p=1338)	0.357 ± 0.019 13473.4/9.38 ms	0.348 ± 0.024 13634/9.41 ms	0.335 ± 0.020 13878.2/11.61 ms	0.339 ± 0.013 14156.4/11.83 ms
KG-RP (p=2007)	0.323 ± 0.027 31532.1/15.52 ms	0.305 ± 0.018 32111.9/15.63 ms	0.292 ± 0.014 32607.4/16.31 ms	0.291 ± 0.015 33442.0/18.0 ms

performed ten times. The average result and the standard deviation are reported. Regarding the ruining times, D2PK-RP is by far the fastest alternative, especially considering training times, which in the case of D2PK-RP are solely due to the initialization of the random matrices. As expected, the explicit approach reports the longest running times due to the expensive explicit evaluation of the kernel feature map.

Finally, we analyze the effect of the hyperparameter t on the average distortion induced by D2PK-RP. To this extent, the dimension of the resulting space k was fixed to 160 and the average distortion was evaluated for $t = 2, 4, 6, \ldots, 200$. The results were compared to the average distortion induced by the explicit approach, also with $k = 160$. This comparison is shown in Figure 3.5.b. The results suggest that using values of t greater than 100 will ensure a performance of D2PK-RP nearly equivalent to that of the explicit approach. In addition, it is even possible to obtain a close approximation to this performance with much lower values of t.

3.4.2 Experiments on ISOLET

The ISOLET Spoken Letter Database consists of letters from the English alphabet pronounced by native speakers under controlled conditions, with two realizations of each letter by a total of 150 subjects. The dataset contains a total of 7,800 samples, each corresponding to the features extracted from the pronunciation of a letter. For more details about the feature extraction process, see [36]. Since each sample consist of 617 features, the dimensionality of the associated implicit feature space $\mathcal{H}$ is 190,653. Hence, the storage of 200 samples from $\mathcal{H}$ as a matrix of 32-bit floats requires approximately 145.5 Megabytes of memory.

To compare the different approaches, 200 samples were selected at random from the whole dataset. Then, the samples were processed by means of the different methods compared in this chapter and the average pairwise distance distortion was

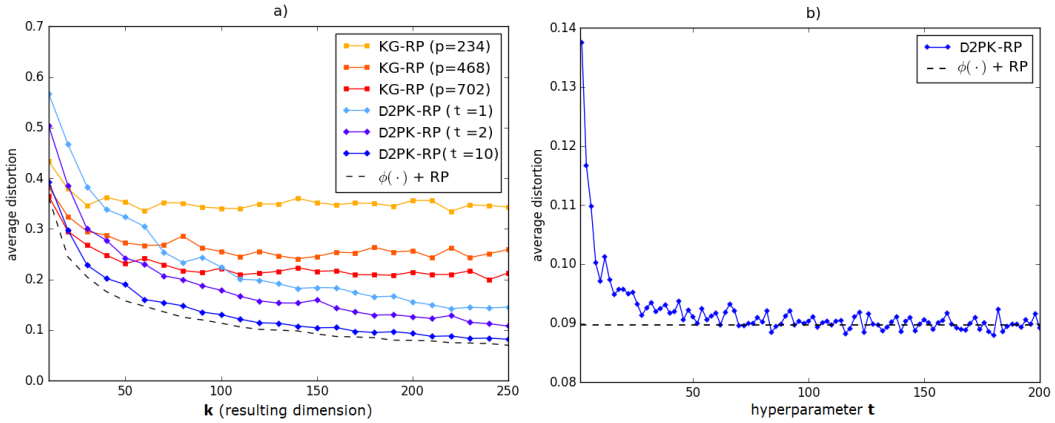


FIGURE 3.6: a) Average distortion induced on ISOLET samples by using different methods as the resulting dimension grows. b) Effect of different values for the hyperparameter t on ISOLET samples transformed by D2PK-RP. The resulting dimension was fixed to 160.

TABLE 3.2: Average pairwise distance distortion induced by different methods and hyperparameters on 200 samples from ISOLET. Training/embedding times are also provided.

Method	**k=40**	**k=80**	**k=120**	**k=160**
$\phi(\cdot) + RP$	0.174 ± 0.004 226.6/665.3 ms	0.123 ± 0.007 428.2/692.6 ms	0.102 ± 0.005 594.1/723.5 ms	0.089 ± 0.008 881.6/732.1 ms
D2PK-RP (t=1)	0.361 ± 0.086 0.9/0.09 ms	0.245 ± 0.041 1.4/0.14 ms	0.215 ± 0.028 1.8/0.19 ms	0.179 ± 0.038 2.7/0.25 ms
D2PK-RP (t=2)	0.279 ± 0.022 0.3/0.17 ms	0.192 ± 0.017 0.6/0.34 ms	0.155 ± 0.008 1.2/0.4 ms	0.139 ± 0.019 1.6/0.57 ms
D2PK-RP (t=10)	0.196 ± 0.010 2.2/0.84 ms	0.139 ± 0.009 5.2/1.4 ms	0.114 ± 0.007 9.3/1.96ms	0.100 ± 0.007 9.7/2.59 ms
D2PK-RP (t=30)	0.181 ± 0.008 7.1/2.48 ms	0.132 ± 0.009 16.8/4.0 ms	0.107 ± 0.004 27.0/6.09 ms	0.092 ± 0.003 31.5/4.86 ms
KG-RP (p=234)	0.344 ± 0.026 448.7/0.8 ms	0.349 ± 0.028 454.8/0.86 ms	0.348 ± 0.017 482.6/0.96 ms	0.335 ± 0.02 492.9/1.25 ms
KG-RP (p=468)	0.283 ± 0.029 1668.9/1.45 ms	0.250 ± 0.017 1768.5/2.05 ms	0.257 ± 0.018 1845.9/2.13 ms	0.269 ± 0.02 1896.1/2.82 ms
KG-RP (p=702)	0.243 ± 0.025 3588.1/2.49 ms	0.222 ± 0.019 3730.68/2.56 ms	0.219 ± 0.017 3820.1/3.33 ms	0.204 ± 0.022 4012.4/3.41 ms

measured. Figure 3.6.a shows the average distortion induced by the different methods as the resulting dimension k grows.

Table 3.2 compiles the resulting average distortions obtained using different methods and various values of k and t. Each experiment was performed ten times, so the average result and standard deviation of those ten runs are reported. To analyze the effect of the hyperparameter t on the average distortion induced by D2PK-RP, the dimension of the resulting space k was fixed to 160 and the average distortion was evaluated for a wide range of values of t. The results were compared to the average distortion induced by the explicit approach also with $k = 160$, and are shown in Figure 3.6.b.

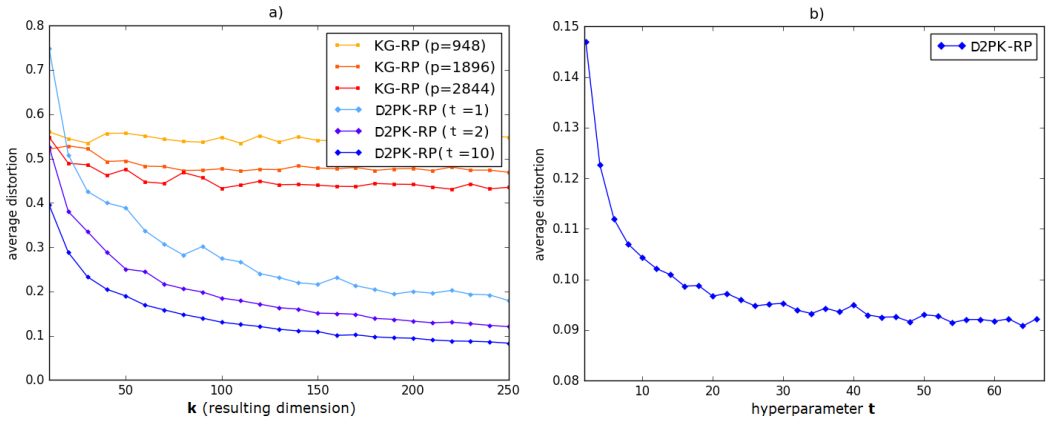


FIGURE 3.7: a) Average distortion induced on 500 STL-10 samples by using different methods as the resulting dimension grows. b) Effect of different values for the hyperparameter t on STL-10 samples transformed by D2PK-RP. The resulting dimension was fixed to 160.

TABLE 3.3: Average pairwise distance distortion induced by different methods and hyperparameters on 500 samples from STL-10. Training/embedding times are also provided.

Method	**k=40**	**k=80**	**k=120**	**k=160**
D2PK-RP (t=1)	0.360 ± 0.086 38.9/7.65 ms	0.254 ± 0.041 63.6/10.88 ms	0.215 ± 0.028 88.6/10.76 ms	0.179 ± 0.038 118.08/15.69 ms
D2PK-RP (t=2)	0.279 ± 0.022 21.58/13.09 ms	0.192 ± 0.017 51.7/22.05 ms	0.155 ± 0.008 68.09/23.2 ms	0.139 ± 0.019 89.75/32.11 ms
D2PK-RP (t=10)	0.196 ± 0.010 113.33/53.71 ms	0.139 ± 0.009 222.9/92.04 ms	0.114 ± 0.007 330.2/123.96 ms	0.100 ± 0.007 434.1/174.08 ms
D2PK-RP (t=30)	0.181 ± 0.008 316.2/183.58 ms	0.132 ± 0.009 654.5/304.2 ms	0.107 ± 0.004 968.8/396.21 ms	0.092 ± 0.003 1295.6/487.22 ms
KG-RP (p=948)	0.559 ± 0.025 6934.6/215.1 ms	0.546 ± 0.018 6976.3/220.8 ms	0.544 ± 0.018 7123.3/224.5 ms	0.541 ± 0.004 7277.3/225.8 ms
KG-RP (p=1896)	0.495 ± 0.024 29368.1/439.7 ms	0.485 ± 0.010 29545/444.6 ms	0.479 ± 0.013 29956.4/449.0 ms	0.482 ± 0.008 30524.9/454.9 ms
KG-RP (p=2844)	0.469 ± 0.020 69502.2/634.3 ms	0.452 ± 0.017 70184.2/643.5 ms	0.445 ± 0.011 71195.8/644.2 ms	0.434 ± 0.012 72354.2/647.9 ms

3.4.3 Experiments on STL-10

The STL-10 dataset [28], inspired by the CIFAR-10 dataset, is another very popular benchmark for image categorization. The two major differences with respect to CIFAR-10 are the much lower number of labeled images per class and the size of the images. The dataset consists of 500/800 train/test images per class, and 100,000 additional unlabeled images for unsupervised learning. Each image is of size 96×96 with three color channels, so each sample contains 27,648 features. The dimensionality of the implicit feature space $\mathcal{H}$ is then 382,219,776. Consequently, storing 200 samples from $\mathcal{H}$ as a matrix of 32-bit floats requires approximately 284.7 Gigabytes of memory, which is far beyond the current capacity of personal computers and a challenging volume even for high-end computing systems. This illustrates how rapidly the explicit approach becomes intractable when the dimensionality of data samples grows, as working with this sample size renders the explicit approach intractable. However, we could still measure the effectiveness of the remaining methods. To compare the different approaches, 500 samples were selected at random from the whole

dataset. Then, the samples were reduced by means of the different methods discussed in this chapter and the average pairwise distance distortion was measured. Figure 3.7.a shows the average distortion induced by different methods as the resulting dimension k grows. Table 3.3 compiles the resulting average distortions obtained using different methods and various values of k and t. Again, each experiment was performed ten times, so the average result and standard deviation are reported. Figure 3.7.b analyzes the effect of the hyperparameter t on the average distortion induced by D2PK-RP (with $k = 160$).

3.4.4 Friedman test and post-hoc tests

This section reports on the results of statistical tests supporting our claim that the proposed method approximates a Random Projection from the feature space of the degree-2 homogeneous polynomial kernel. We applied the Friedman method[5] with post-hoc tests as described in [45, 44]. For all tests, the performance measure used was $1 - avg.\ distortion$. Intuitively, higher values of this performance measure correspond to small induced distortions in pairwise distances.

First of all, we analyzed whether a significant difference existed in the performance of the compared methods over the different datasets evaluated. Under the null-hypothesis, the Friedman test states that all the algorithms are equivalent, so a rejection of this hypothesis implies the existence of differences among the performances of the different methods. Using the Friedman statistic over the performance results previously reported in this section resulted in a value of 80.805 (distributed according to chi-square with 7 degrees of freedom) and a corresponding p-value of 4.325×10^{-11}. As a consequence, we can reject the null-hypothesis and conclude that significant differences exist between the performances of the compared methods. Hence, a post-hoc statistical analysis must be performed. We selected the best performing method, namely $\phi(\cdot) + RP$ (i.e., the explicit approach), as the control. Then, the Bonferroni-Dunn, Holm and Hochbergs tests [45] were used to find whether the control method presented statistically significant differences when compared to the remaining approaches in terms of performance. The adjusted p-values for these tests are reported in Table 3.4, stressing in bold those methods that were worse than the control considering a level of significance $\alpha = 0.05$.

As we can see, while KG-RP and D2PK-RP with $t \leq 2$ perform worse that the control approach, no significant differences were detected by the tests when comparing the distance preservation performance of our method using $t \geq 10$ and the explicit approach $\phi(\cdot) + RP$ (with a level of significance $\alpha = 0.05$). This supports our claim that, when a big enough value is selected for t, our method approximates a Random Projection from the feature space of the degree-2 homogeneous polynomial kernel.

[5] Particularly, we used the CONTROLTEST package developed at the University of Granada. URL: https://sci2s.ugr.es/sicidm (Date accessed: 15/09/2018).

TABLE 3.4: Adjusted p-values for the comparison of the control algorithm ($\phi(\cdot) + RP$) with the remaining algorithms (Bonferroni-Dunn, Holm and Hochberg tests).

Algorithm	unadjusted p	p_{Bonf}	p_{Holm}	p_{Hoch}
KG-RP (p=$3 \cdot \sqrt{N}$)	4.62394×10^{-12}	3.23676×10^{-11}	3.23676×10^{-11}	3.23676×10^{-11}
KG-RP (p=$6 \cdot \sqrt{N}$)	5.43308×10^{-9}	3.80316×10^{-8}	3.25985×10^{-8}	3.25985×10^{-5}
KG-RP (p=$9 \cdot \sqrt{N}$)	3.06125×10^{-6}	2.142877×10^{-5}	1.53062×10^{-5}	1.53062×10^{-5}
D2PK-RP (t=1)	6.79534×10^{-6}	4.75674×10^{-5}	2.71813×10^{-5}	2.71813×10^{-5}
D2PK-RP (t=2)	0.00204	0.014328	0.00614	0.00614
D2PK-RP (t=10)	0.04550	0.31850	0.09100	0.09100
D2PK-RP (t=30)	0.31731	2.22117	0.31731	0.31731

3.5 Experimental results on classification accuracy

Although Radial Basis Function (RBF) is the most widely used type of kernel in the context of Support Vector Machine (SVM) classification, it suffers from some limitations. Mainly, the implicit feature map $\phi_{RBF}(\cdot)$ associated to the RBF kernel is infinite dimensional, which enforces the application of the kernel trick to train RBF-SVMs. As a result, RBF-SVMs are inefficient and poorly scalable as compared to their linear counterparts [109].

As an alternative, Chang *et al.* [21] proposed mapping the data samples by the feature transformation associated to the polynomial kernel of degree two as a prior step to fast linear-SVM classification. Their experimental results evidence that, using this method on some datasets, one may achieve accuracy rates close to those of highly non-linear kernels. Unfortunately, as we have mentioned before, the dimensionality of $\mathcal{H}$ grows rapidly as the original dimension of samples increases. Therefore, the method proposed in [21] is not convenient when training samples have a significant number of features and those features are not sparse.

Conversely, the method proposed in this chapter can be used to efficiently condense the structure of a dataset in $\mathcal{H}$ to a low-dimensional representation of the samples. This representation can be used to train efficient linear classifiers that obtain accuracy rates almost as good as those trained on samples explicitly mapped by means of the embedding $\phi : \mathbb{R}^d \to \mathcal{H}$. To support our claim, we reproduced the experimental protocol developed in [21]. The accuracies obtained by using D2PK-RP as a feature extractor prior to linear classification[6] were compared to the results presented by Chang *et al.* Table 3.6 compiles the experimental results obtained on various datasets by various dimensionality reduction and classification methods[7]. For details on the characteristics of the different datasets refer to Table 3.5.

These results evidence that the proposed method can be used as a previous step to linear classification, boosting the capabilities of linear classifiers and approximating, in some cases, the performance attained by highly non-linear classification methods. Interestingly, the effect of increasing the hyperparameter t of D2PK-RP has a more

[6] We used the linear SVM implementation provided by LIBSVM [35]

[7] Note that the results concerning L-SVM, RBF-SVM and $\phi(\cdot)$ + L-SVM where directly taken from [21], where the authors used the feature map for the polynomial kernel $K(\mathbf{x}, \mathbf{y}) = (\gamma\langle \mathbf{x}, \mathbf{y}\rangle + 1)^2$, setting γ to the appropriate value on each dataset.

TABLE 3.5: Feature number, kernel feature space size, training sample number and test sample number of the datasets used in Section 3.5.

Dataset	**sample dim.** d	**size of** $\mathcal{H}$	**# train**	**# test**
ijcnn1	22	253	49,990	91,701
MNIST38	784	307,720	11,982	1,984
covtype	54	1,485	464,809	116,203
webspam	254	32,385	280,000	70,000

TABLE 3.6: Classification accuracy on various datasets obtained by using: a linear SVM over the original features, a Gaussian-kernel SVM, a linear SVM over the embedding defined by the polynomial kernel of degree two, a linear SVM trained on D2PK-RP features and a linear SVM trained on KG-RP features.

Methods	**IJCNN**		**MNIST38**		**covtype**		**webspam**	
	param.	**acc.**	**param.**	**acc.**	**param.**	**acc.**	**param.**	**acc.**
raw features Linear SVM	C=32	92.21%	C=0.03125	96.82%	C=0.0625	76.35%	C=32	93.15%
raw features RBF-SVM	C=32	98.69%	C=2	99.70%	C=32	96.08%	C=8	99.20%
$\phi(\cdot) : \mathbb{R}^d \to \mathcal{H}$ Linear SVM	C=0.125	97.84%	C=2	99.29%	C=2	80.09%	C=8	98.44%
D2PK-RP (t=1) Linear SVM	C=4 k=250	97.24% ±0.05	C=4 k=2000	98.63% ±0.17	C=0.1 k=500	79.77% ±0.04	C=0.05 k=2000	97.77% ±0.14
D2PK-RP (t=2) Linear SVM	C=4 k=250	97.29% ±0.08	C=4 k=2000	98.80% ±0.19	C=0.1 k=500	79.72% ±0.01	C=0.05 k=2000	97.56% ±0.13
D2PK-RP (t=10) Linear SVM	C=4 k=250	97.31% ±0.07	C=4 k=2000	98.90% ±0.2	C=0.1 k=500	79.72% ±0.03	C=0.05 k=2000	97.60% ±0.06
D2PK-RP (t=20) Linear SVM	C=4 k=250	97.32% ±0.09	C=4 k=2000	98.94% ±0.24	C=0.1 k=500	79.73% ±0.01	C=0.05 k=2000	97.65% ±0.14
D2PK-RP (t=30) Linear SVM	C=4 k=250	97.33% ±0.06	C=4 k=2000	98.95% ±0.2	C=0.1 k=500	79.73% ±0.02	C=0.05 k=2000	97.66% ±0.09
KG-RP (p=$9\sqrt{N}$) Linear SVM	C=4 k=250	97.36% ±0.10	C=0.5 k=2000	98.68% ±0.07	C=0.1 k=500	79.43% ±0.10	C=0.05 k=2000	97.76% ±0.04

subtle effect in the classification accuracy than it had in the results concerning distance preservation in the previous section. Conveniently, high accuracy rates can be achieved using very small values of t. In fact, no significant accuracy improvements where registered using $t > 20$. Moreover, the best results with D2PK-RP for webspam and covtype datasets were achieved by using $t = 1$. In addition, we can see that KG-RP achieved similar or slightly lower accuracies than our method on all datasets.

3.6 Conclusions and future work

In this chapter, a novel non-linear dimensionality reduction method has been presented. The proposed algorithm makes it possible to implicitly approximate a Random Projection from the feature space associated to the polynomial kernel of degree two to an Euclidean space of the desired dimension. This projection is conveniently performed without any explicit evaluation of the feature map $\phi : \mathbb{R}^d \to \mathcal{H}$. As opposed to previous techniques to perform Random Projections from kernel feature spaces [5, 110], our method preserves the data-independence and efficiency properties of the original Random Projection algorithm. In fact, the training stage only involves the population of a number of random matrices, thus preserving much of

the simplicity of the original Random Projection method. Moreover, the proposed method is compatible with the database-friendly distribution proposed by Achlioptas [2], thus allowing its implementation in terms of aggregate evaluation. This can be achieved by delaying the floating-point multiplication present in (3.9).

Our experimental results show that the proposed method approximates the distance preservation properties of Random Projection, so the generated representations approximately preserve the pairwise distances between samples in the kernel feature space. Because of the data-independent nature of the proposed method, it could be applied in online-learning scenarios [38], where no data samples are initially available. In addition, the method proposed in this chapter can be used to train efficient linear classifiers that approximate the performance of their kernelized counterparts.

The major shortcoming of the proposed method is that it was designed to work solely with a specific kernel function. In this regard, the efficiency and data-independence of our method was achieved at the expense of generality, as this method is only compatible with the homogeneous polynomial kernel of degree two. Nevertheless, the wide applicability and popularity of this kernel function [21, 64, 23, 107, 106] justifies this design decision. We also believe that it would be possible to develop similar methods for other specific kernel functions, but this possibility will be examined in future work. In addition, the suitability of the proposed method for other machine learning tasks (e.g., regression, clustering and document retrieval) could be evaluated. Finally, we believe a similar approach to the one proposed in this chapter could be used to develop a data-independent form of kernelized Locality-Sensitive Hashing (LSH) [66] with the polynomial kernel, as this task is also based on the projection of samples onto random vectors in the kernel feature space.

Chapter 4

Random Projections from the Feature Spaces of Arbitrary-Degree Polynomial Kernels

The results presented in the previous chapter have shown that it is possible to approximate a Random Projection from the feature space of the degree-two homogeneous polynomial kernel in a relatively efficient and data-independent manner. In the following pages, we build upon the ideas of Chapter 3 to improve the generality, efficiency and effectiveness of our kernelized Random Projection approach. Particularly, we introduce a novel method to efficiently perform Random Projections from the feature spaces of homogeneous polynomial kernels of arbitrary degree. Extensive experimental results evidence that this new algorithm outperforms alternative approaches in terms of distance preservation, while being more efficient. Furthermore, results show that the proposed method can be applied to boost the accuracy of linear classifiers, approximating in some cases the effectiveness of kernelized classifiers.

The contents of this chapter have been adapted from the journal paper: Daniel López-Sánchez, Angélica González Arrieta and Juan M. Corchado. "Data independent Random Projections from the feature-space of the Homogeneous Polynomial Kernel". In: Pattern Recognition (2018).

4.1 Introduction

As evidenced by the results presented in the previous chapter, it is possible to approximate a Random Projection from the feature space of the degree-two homogeneous polynomial kernel by carefully choosing the distribution of the projection matrix and applying the Central Limit Theorem in conjunction with the properties of kernels. Focusing on a specific kernel function with fixed hyperparameters allowed us to consider a concrete feature map and analyze its effect on the entries of the projection

matrix when inner products in the Random Projection algorithm were replaced by kernel function evaluations. While being reasonably effective, this approach presented some limitations. The fist and most obvious drawback was that, by design, our method was constrained to only work with the degree two homogeneous polynomial kernel. In addition, the fact that the projection vectors were the result of transforming low-dimensional vectors with the kernel feature map caused the theoretically unpleasant problem of having two different distributions for the entries of the projection vectors in the kernel feature space, one of which was not even a valid Random Projection distribution. Fortunately, the number of entries of the projection vectors following that invalid distribution was small, and some algorithmic tricks together with the application of the Central Limit Theorem enabled our method to approximate the performance of a true Random Projection from the kernel feature space. Nevertheless, improvements are still possible in terms of the effectiveness, efficiency and generality of the algorithm.

In this chapter, we propose an improved method to efficiently perform Random Projections from the feature spaces of homogeneous polynomial kernels of arbitrary degree. Again, focusing on the family of homogeneous polynomial kernels allows us to preserve the data-independence and efficiency of the original Random Projection method. However, by introducing a new manner of generating the projection vectors, we manage to extend our method to polynomial degrees greater than two. In addition, this new method is compatible with both the Gaussian distribution [8] and the database-friendly distribution proposed by Achlioptas [3] for the projection vectors. Our experimental results evidence that this improved algorithm outperforms alternative approaches in terms of pairwise distance preservation, while requiring significantly less computational resources. We also present results evidencing that the generated feature representations can be used to achieve a higher linear classification accuracy, approximating the effectiveness of nonlinear classifiers in some datasets.

The rest of this chapter is structured as follows. Section 4.2 reviews some of the most relevant works that have studied the possible kernelization of the Random Projection algorithm. Section 4.3 introduces our proposed algorithm and analyzes its compatibility with the database-friendly distribution proposed by Achlioptas. This section also contains a detailed analysis of the computational complexity of our algorithm and other alternative approaches. Section 4.4 presents the results of extensive experiments, which evidence the properties of our kernelized variant of Random Projection. Finally, in Section 4.5 we present the conclusions of this work and propose some promising future lines of research.

4.2 Related work

As discussed in Section 2.4, the problem of developing a kernelized variant of the Random Projection algorithm has already been addressed in the literature. The interest in such kernelized algorithms is motivated by two main reasons:

1. A kernelized variant of the Random Projection algorithm would provide a tool to generate low dimensional representations where relative distances between data points would be approximately equal to those in the kernel feature space. This could have numerous applications in machine learning tasks such as clustering and information retrieval.

2. While theoretical guarantees for the preservation of inner products under Random Projections have been historically looser that those regarding Euclidean distances, improved bounds on the preservation of dot products have been recently proved [56]. This gives support to the idea that an efficient technique which performs a Random Projection from a kernel feature space could be used as a representation-generator prior to a linear classifier, which would benefit from the non-linearity of the feature space and approximate the accuracy of non-linear classifiers while being significantly more efficient [109].

Motivated by these possibilities, the authors of [11, 9] analyzed whether it would be possible to formulate an algorithm capable of performing a Random Projection from the feature space of an arbitrary kernel function, by just having black-box access to the kernel function but no unlabeled training samples (i.e., without access to the distribution of input data). Unfortunately, their results were negative, and the authors proved that this is not possible for an arbitrary black-box kernel. However, they left the question open of whether such methods could be developed for specific kernel functions such as the polynomial kernel.

Years later, Alavi *et al.* [5, 110] proposed a general method to perform Random Projections from arbitrary kernel feature spaces, named Kernelized Gaussian Random Projection (KG-RP). Their findings did not contradict the result described in the previous paragraph since the method they proposed required access to a number of unlabeled training samples to work. As a consequence, KG-RP is a data-dependent method, and the quality of the embeddings it produces depends on the amount of training samples available and their variability. In addition, most of the computational efficiency of the original Random Projection method is lost in this version, as we will see in Section 4.3.4.

Following a diametrically opposite approach, Chang *et al.* [21] proposed explicitly computing the feature map of low-rank polynomial kernels to train efficient linear classifiers. They exploited the fact that, as opposed to other popular kernel functions, the feature spaces associated to polynomial kernels are known and of finite dimension. They also took advantage of the sparse nature of some datasets to reduce the time and storage requirements of explicitly computing the mapped data samples. Although their results evidenced the potential of polynomial kernels, this approach is generally too demanding in terms of storage and computation. This is especially true when working with polynomial degrees greater than two, as in the case of polynomial kernels the dimension of the feature space grows exponentially with the degree.

The results presented in Chapter 3 have shown that, rather than explicitly computing the feature space of a polynomial kernel, it is possible to implicitly perform

a Random Projection from it. Particularly, a data-independent algorithm named D2PK-RP was introduced to approximate Random Projections from the feature space of the degree-two homogeneous polynomial kernel. Unfortunately, the applicability of this method is limited by its exclusive compatibility with the second degree homogeneous polynomial kernel. In addition, it requires populating a number of complete projection matrices, thus incurring in significant computational overheads. Hence, there is much room for improvement in the generality and efficiency of this method.

Finally, it is worth noticing that, during the past decade, a lot of effort has been put into designing methods to efficiently approximate dot products in different kernels' feature spaces [97, 84, 82]. Formally, given a kernel function $K(\cdot,\cdot)$, the goal of such methods is to find an approximated feature map $h(\cdot)$ such that:

$$K(\mathbf{x},\mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{H}} \approx \langle h(\mathbf{x}), h(\mathbf{y}) \rangle_{\mathbb{R}^k}, \tag{4.1}$$

where $h(\cdot)$ can be computed efficiently and the feature space it generates is sufficiently low-dimensional or sparse [97]. Note that these methods are designed to approximate dot products between samples rather than Euclidean distances (i.e., they are not directly related to Random Projections or the JL-lemma). Nevertheless, we selected one of the most popular and generally applicable methods of this class, namely the Nyström method [103], and included it in our comparisons.

4.3 Proposed method

In this section, we introduce the proposed method and provide a simple algorithmic description to ease its implementation. Afterwards, the compatibility of our technique with the database-friendly distribution proposed by Achlioptas [2] is explored. We also analyze the possibility of using our method with inhomogeneous polynomial kernels. Finally, the computational complexity of our algorithm in both train and test phases is analyzed and compared to alternative approaches.

As mentioned before, our method is specifically designed to efficiently perform Random Projections from the feature spaces of homogeneous polynomial kernels. We focused on this family of kernel functions due to their simplicity, proven power [21] and special properties, which will allow us to perform the Random Projection efficiently and without any knowledge of the distribution of the data to be projected. As discussed in Section 2.3, homogeneous polynomial kernels are those of the form:

$$K(\mathbf{x},\mathbf{y}) = \langle \mathbf{x},\mathbf{y} \rangle^g \quad \text{with} \quad \mathbf{x},\mathbf{y} \in \mathbb{R}^d. \tag{4.2}$$

In addition, the following mapping from $\mathbb{R}^d$ to $\mathbb{R}^{d^g}$ represents a valid feature map for the homogeneous polynomial kernel of degree g:

$$\phi(\mathbf{x}) = \underbrace{\mathbf{x} \otimes \mathbf{x} \cdots \otimes \mathbf{x}}_{g \text{ times}} = \bigotimes_{i=1}^{g} \mathbf{x}, \tag{4.3}$$

where $\otimes$ denotes the Kronecker product.

4.3.1 Random Projection for homogeneous polynomial kernels

Our goal is to perform a Random Projection from the kernel feature space onto a lower-dimensional Euclidean space $\mathbb{R}^k$, while avoiding any explicit computation of the feature map $\phi(\cdot)$. In this regard, each output component must be generated as the inner product between the mapped data point and a random vector whose entries are independently drawn from a valid Random Projection distribution. It must be emphasized that, in some cases, the entries of a random vector might follow a valid Random projection distribution when analyzed individually. However, if they are not mutually independent, the result of using such vectors to project data samples will not be a valid Random Projection, so the Johnson-Lindesstrauss lemma will not be applicable to guarantee the preservation of pairwise distances after the projection.

Before introducing our algorithm, let us present a fundamental property of homogeneous polynomial kernels. Let $\mathbf{x}$ and $\mathbf{r}_1, \ldots, \mathbf{r}_g$ be arbitrary vectors in $\mathbb{R}^d$, then it holds that:

$$\prod_{j=1}^{g} \langle \mathbf{x}, \mathbf{r}_j \rangle_{\mathbb{R}^d} = \langle \phi(\mathbf{x}),\ \textstyle\bigotimes_{j=1}^{g} \mathbf{r}_j \rangle_{\mathcal{H}}, \tag{4.4}$$

where $\phi : \mathbf{x} \to \bigotimes_{i=1}^{g} \mathbf{x}$ is the feature map given in (4.3) for the degree-g homogeneous polynomial kernel, and $\mathcal{H}$ is its associated feature space. To prove this, it suffices to rewrite the product of inner products in the left-hand side of the equation as a product of summations. Then, rearranging we arrive at the desired expression. For instance, let $\mathbf{x}, \mathbf{y}, \mathbf{z}$ be three arbitray vectors in $\mathbb{R}^d$, then for $g = 2$ we have:

$$\begin{aligned} \langle \mathbf{x}, \mathbf{y} \rangle \langle \mathbf{x}, \mathbf{z} \rangle &= \left(\sum_{i=1}^{d} \mathbf{x}_i \mathbf{y}_i \right) \left(\sum_{j=1}^{d} \mathbf{x}_j \mathbf{z}_j \right) \\ &= \sum_{i=1}^{d} \sum_{j=1}^{d} \mathbf{x}_i \mathbf{x}_j \mathbf{y}_i \mathbf{z}_j \\ &= \langle \mathbf{x} \otimes \mathbf{x}, \mathbf{y} \otimes \mathbf{z} \rangle = \langle \phi(\mathbf{x}), \mathbf{y} \otimes \mathbf{z} \rangle_{\mathcal{H}} . \end{aligned} \tag{4.5}$$

The proof for $g > 2$ is analogous.

At this point, one might attempt to use (4.4) to perform a Random Projection in the feature space of the homogeneous polynomial kernel without explicitly operating in it. To do this, the first step would be to select a distribution for $\mathbf{r}_1, \ldots, \mathbf{r}_g$ such that the projection vector in the feature space, $\bigotimes_{j=1}^{g} \mathbf{r}_j$, follows a valid Random Projection distribution. For instance, if $\mathbf{r}_1, \ldots, \mathbf{r}_g$ are independently drawn from $\mathcal{N}_d(\mathbf{0}, \mathbf{I})$, the individual entries of $\bigotimes_{j=1}^{g} \mathbf{r}_j$ will be the product of independent standard normal variables and thus follow a symmetric distribution with zero mean and unit variance[1] (the normal product distribution [100, 102], in particular). As mentioned before, as long as the entries of the projection vectors are i.i.d. with zero mean and unit variance, pairwise distances will be preserved in expectation [3]. When analyzed individually, the entries of $\bigotimes_{j=1}^{g} \mathbf{r}_j$ exhibit the desired zero mean and unit variance. However, due to the manner in which they are computed, they are not mutually independent. Particularly, one can see that each entry in $\bigotimes_{j=1}^{g} \mathbf{r}_j$ is the product of g independent factors, but also that each individual factor appears in multiple entries of $\bigotimes_{j=1}^{g} \mathbf{r}_j$. Moreover, this dependence in the entries of vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$ will appear regardless of the selected distribution for $\mathbf{r}_1, \ldots, \mathbf{r}_g$. As a consequence, (4.4) cannot be directly used to perform a valid Random Projection from the kernel feature space.

To provide insight into the dependence of the entries in $\bigotimes_{j=1}^{g} \mathbf{r}_j$, we explicitly generated a number of such projection vectors. To keep the computations tractable, we considered a polynomial degree of two ($g = 2$) and an input feature space of dimension 5 ($\mathbf{r}_1, \mathbf{r}_2 \sim \mathcal{N}_5(\mathbf{0}, \mathbf{I})$). Hence, the dimension of the vectors generated in this manner was 25. For comparison, we also generated a set of projection vectors by directly sampling from $\mathcal{N}_{25}(\mathbf{0}, \mathbf{I})$. Figures 4.1.a and 4.1.b show the correlation matrices for the vectors generated by either method. Interestingly, correlation matrices of both sets of vectors look quite similar. This suggest that the dependence among the entries of vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$ is not a mere linear correlation. To actually visualize this phenomenon, we need to use a more sophisticated measure of statistical dependence, namely the distance correlation [93]. Figures 4.2.a and 4.2.b show the distance correlation matrices for the projection vectors generated by directly sampling from $\mathcal{N}_{25}(\mathbf{0}, \mathbf{I})$ and evaluating $\bigotimes_{j=1}^{g} \mathbf{r}_j$ respectively. In this case, the matrix corresponding to the vectors generated by using $\bigotimes_{j=1}^{g} \mathbf{r}_j$ shows a clear deviation from the identity matrix. This indicates that, as expected, a certain degree of dependence exists among entries. The structure of this dependence is further explored in Figure 4.3.

To overcome the problem of dependence among the entries of the projection vectors in the kernel feature space, we propose applying the Central Limit Theorem (CLT) [57]. This classical result states that the sum of independent random variables with finite, non-null variance is approximately distributed according to a normal distribution. In particular, consider the multidimensional version of the Central Limit Theorem [17] which can be formulated in the following manner: Let $\mathbf{x}_1, \ldots, \mathbf{x}_t$

[1] This comes from the fact that if A and B are two independent random variables, then their product has expectation $\mathrm{E}[AB] = \mathrm{E}[A]\mathrm{E}[B]$, and variance $\mathrm{Var}[AB] = \mathrm{Var}[A]\mathrm{Var}[B] + \mathrm{Var}[A](\mathrm{E}[B])^2 + \mathrm{Var}[B](\mathrm{E}[A])^2$ [77, 18].

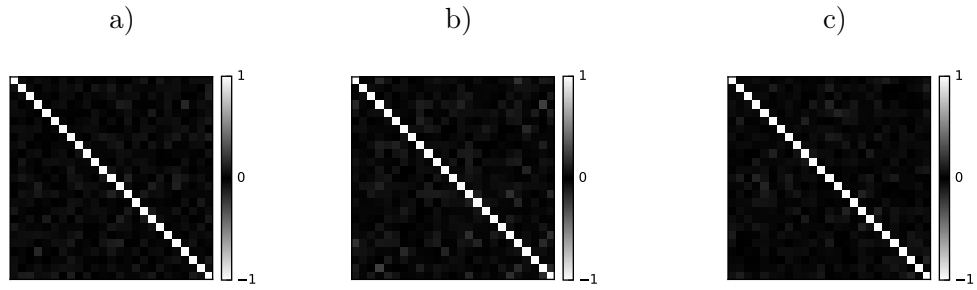


FIGURE 4.1: Correlation matrices computed over 500 25-dimensional samples generated by different methods ($d = 5$, $g = 2$, $t = 30$).

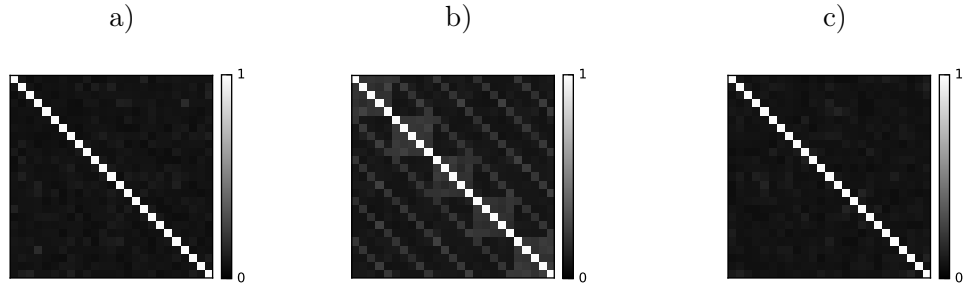
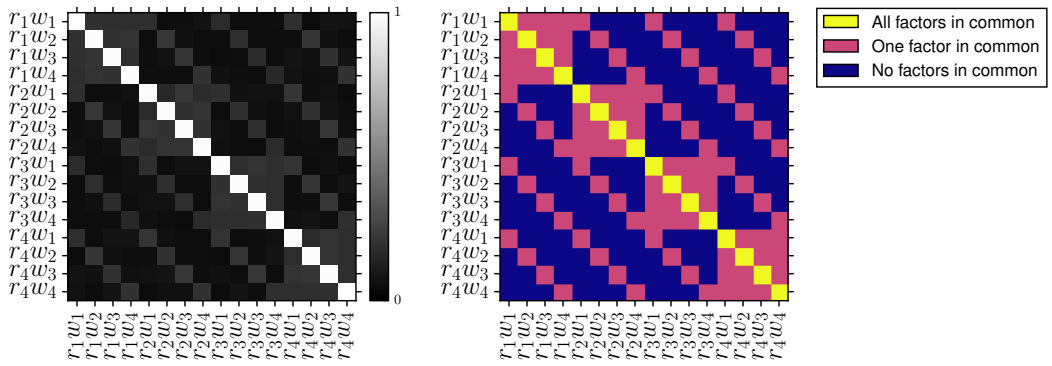


FIGURE 4.2: Distance correlation matrices [93] computed over 500 25-dimensional samples generated by different methods ($d = 5$, $g = 2$, $t = 30$).



$$\mathbf{r} \bigotimes \mathbf{w} = (r_1, r_2, r_3, r_4) \bigotimes (w_1, w_2, w_3, w_4)$$
$$= (r_1w_1, r_1w_2, r_1w_3, r_1w_4, r_2w_1, r_2w_2, r_2w_3, r_2w_4, r_3w_1, r_3w_2, r_3w_3, r_3w_4, r_4w_1, r_4w_2, r_4w_3, r_4w_4)$$

FIGURE 4.3: Distance correlation [93] computed over 1000 16-dimensional samples of the form $\mathbf{r} \bigotimes \mathbf{w}$, where $\mathbf{r}$ and $\mathbf{w}$ are i.i.d. from $\mathcal{N}_4(\mathbf{0}, \mathbf{I})$. The dependence detected by the distance correlation measure can be explained by the number of shared factors among entry pairs.

be t i.i.d. random vectors drawn from a distribution with zero means and finite covariance matrix $\mathbf{\Sigma}$. Then, the sum of these vectors scaled by $1/\sqrt{t}$ converges in distribution to a multivariate normal distribution with zero means and $\mathbf{\Sigma}$ covariance, as t goes to infinity:

$$\frac{\mathbf{x}_1 + \cdots + \mathbf{x}_t}{\sqrt{t}} \xrightarrow{\mathcal{D}} \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}). \tag{4.6}$$

To exploit this classical theorem, our method generates the projection vectors as the scaled sum of t vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$:

$$\sum_{i=0}^{t-1} \left(\frac{1}{\sqrt{t}} \bigotimes_{j=1}^{g} \mathbf{r}_{gi+j} \right) \quad \text{where} \quad \mathbf{r}_1, \ldots, \mathbf{r}_{gt} \text{ are } gt \text{ i.i.d. vectors from } \mathcal{N}_d(\mathbf{0}, \mathbf{I}). \tag{4.7}$$

Since the vectors inside the summation of (4.7) are i.i.d. with zero means, the multidimensional CLT applies and we can guarantee that, as t goes to infinity, the generated projection vectors converge in distribution to $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$. Note that this is true regardless of the previously discussed dependence among the entries in vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$, since the multidimensional CLT requires the independence of the summed random vectors but not among the entries that form each vector. Furthermore, taking a closer look at the entries of vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$ with $\mathbf{r}_j \sim \mathcal{N}_d(\mathbf{0}, \mathbf{I})$, we can see that they have an identity covariance matrix[2]. Therefore, by the multivariate CLT, the projection vectors generated according to (4.7) converge in distribution to a multivariate normal with zero means and identity covariance as t goes to infinity:

$$\sum_{i=0}^{t-1} \left(\frac{1}{\sqrt{t}} \bigotimes_{j=1}^{g} \mathbf{r}_{gi+j} \right) \xrightarrow{\mathcal{D}} \mathcal{N}_{d^g}(\mathbf{0}, \mathbf{I}), \tag{4.8}$$

$$\text{given that } \mathbf{r}_1, \ldots, \mathbf{r}_{gt} \text{ are } gt \text{ i.i.d. vectors from } \mathcal{N}_d(\mathbf{0}, \mathbf{I}).$$

Note that, conveniently, variables following a multivariate normal distribution with diagonal covariance matrix are mutually independent [50]. Hence, by using projection vectors generated as described in (4.7) with a sufficiently large value for t, we are fulfilling the necessary conditions to obtain a valid Gaussian Random Projection [8]. To empirically assess the independence of the entries of the projection vectors generated by (4.7), we explicitly generated a number of them (note that the final version of the algorithm will never compute these vectors explicitly). Figures 4.1.c and 4.2.c show the correlation and distance correlation matrices of the generated vectors. As desired, the distance correlation matrix approximates the identity except

[2]This is because the entries in vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$ are the product of g factors and, conveniently, any pair of entries from $\bigotimes_{j=1}^{g} \mathbf{r}_j$ shares at most $g-1$ factors. Since the differing factors randomly flip the sign of the shared ones, vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$ will have an identity covariance matrix as long as the entries of $\mathbf{r}_1, \ldots, \mathbf{r}_g \in \mathbb{R}^d$ are i.i.d. from a distribution with zero mean and unit variance.

for some random noise. This indicates that applying the CLT effectively mitigated the dependence among the entries of the projection vectors[3].

At this point, we can present how our proposed method computes each component of the k-dimensional output representation for a given data point $\mathbf{x} \in \mathbb{R}^d$. For computational reasons, instead of creating gtk unique random vectors, our method generates a set of p vectors and uses random subsets sampled from it for each output component. Of course, this breaks the theoretical requirement of independence among the entries of the projection matrix in Random Projection. However, we found that this relaxation produces good results in practice while enabling important computational savings. Formally, let S be a set of p i.i.d. random vectors drawn from $\mathcal{N}_d(\mathbf{0}, \mathbf{I})$. Then, for each output component we form $S_c = \{\mathbf{r}_1, \ldots, \mathbf{r}_{gt}\}$, a subset of gt vectors chosen at random from S (i.e., $S_c \subset S$)[4]. Afterwards, the c-th component in the output representation of $\mathbf{x}$ is computed as follows:

$$f_c(\mathbf{x}) = \left\langle \phi(\mathbf{x}),\ \sum_{i=0}^{t-1} \left(\frac{1}{\sqrt{t}} \bigotimes_{j=1}^{g} \mathbf{r}_{gi+j} \right) \right\rangle_{\mathcal{H}}, \tag{4.9}$$

which, for a sufficiently large t, corresponds to the projection of the mapped data point $\phi(\mathbf{x})$ onto a random vector following a multivariate normal distribution with zero means and identity covariance. Conveniently, (4.9) can be rewritten to avoid any explicit evaluation of the feature map or the Kronecker product:

$$\begin{aligned} f_c(\mathbf{x}) &= \left\langle \phi(\mathbf{x}), \sum_{i=0}^{t-1} \left(\frac{1}{\sqrt{t}} \bigotimes_{j=1}^{g} \mathbf{r}_{gi+j} \right) \right\rangle_{\mathcal{H}} \\ &= \sum_{i=0}^{t-1} \left(\frac{1}{\sqrt{t}} \left\langle \phi(\mathbf{x}),\ \bigotimes_{j=1}^{g} \mathbf{r}_{gi+j} \right\rangle_{\mathcal{H}} \right) \\ &= \sum_{i=0}^{t-1} \left(\frac{1}{\sqrt{t}} \prod_{j=1}^{g} \langle \mathbf{x}, \mathbf{r}_{gi+j} \rangle \right). \end{aligned} \tag{4.10}$$

Then, the output representation for sample $\mathbf{x}$ is formed by concatenating the k components and multiplying them by the corresponding scaling factor (see [2]):

$$f(\mathbf{x}) = \frac{1}{\sqrt{k}}\ (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))\,. \tag{4.11}$$

[3] It should be noted that our experiment with the distance correlation measure only assessed the pairwise independence among the entries of vectors, rather than the stronger condition of mutual independence. However, the latter is theoretically guaranteed for a sufficiently large t by the convergence to a multivariate normal with identity covariance.

[4] In practice, in order to save storage resources, the subsets $S_1, \ldots, S_k$ store the indexes to the selected vectors from S, rather than duplicated copies of them. See Appendix A for a low level, efficient specification of the algorithm.

In practice, the most effective strategy to transform a sample involves precomputing the inner products of that sample with the p random vectors in S. By so doing, (4.10) can be evaluated without any further dot product evaluation (i.e., with a computational complexity independent of d). The steps required to transform a number of samples with the proposed method are summarized in Algorithm 2. Henceforth, we will refer to the proposed method as Polynomial Kernel Random Projection[5] (PK-RP). Note that, for the sake of clarity, Algorithm 2 does not include the computational trick we just described, as it is rather a high level description of our method than a pseudo-code specification. See Appendix A for a more implementation-oriented description of this algorithm.

Algorithm 2 Polynomial Kernel Random Projection (PK-RP)

Require: A set of points $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ from $\mathbb{R}^d$, the desired degree g for the polynomial kernel, the total number p of random vectors generated, the number of vectors t summed for the CLT and the desired output dimension k.

Ensure: Returns a set of output samples $\{\mathbf{x}'_1, \dots, \mathbf{x}'_N\}$ in $\mathbb{R}^k$ such that pairwise distances between these samples are approximately equal to those of input data samples mapped to the feature space of the homogeneous polynomial kernel of degree g.

$S \leftarrow \{\mathbf{r}_1, \dots, \mathbf{r}_p\}$ where $\mathbf{r}_1, \dots, \mathbf{r}_p \sim \mathcal{N}_d(\mathbf{0}, \mathbf{I})$ ▷ Generate p random vectors
Sample S to form $S_1, \dots, S_k \subset S$, each of size gt ▷ Form $S_1, \dots, S_k \subset S$
for $n = 1, \dots, N$ **do** ▷ Iterate over samples
  $\mathbf{x}'_n \leftarrow (0, \dots, 0) \in \mathbb{R}^k$ ▷ Initialize output vector with zeros
  for $c = 1, \dots, k$ **do** ▷ Iterate over output dimensions
    for $i = 0, \dots, t-1$ **do** ▷ Iterate to apply CLT
      $temp \leftarrow \frac{1}{\sqrt{t}}$ ▷ Initialize temp. variable to hold the product
      for $j = 1, \dots, g$ **do** ▷ Iterate over the polynomial degree
        $temp \leftarrow temp \cdot \langle \mathbf{x}_n, S_c[gi + j] \rangle$ ▷ Product of inner products (4.10)
      $\mathbf{x}'_n[c] \leftarrow \mathbf{x}'_n[c] + temp$ ▷ Summation for CLT (4.10)
  $\mathbf{x}'_n \leftarrow \frac{1}{\sqrt{k}} \cdot \mathbf{x}'_n$ ▷ Final scaling
return $\{\mathbf{x}'_1, \dots, \mathbf{x}'_N\}$

4.3.2 Compatibility with sparse Random Projection distributions

Thus far, we have assumed that the random vectors used in our method have to be drawn from a standard normal distribution. However, as mentioned before, the projection vectors used in the classic Random Projection algorithm can be drawn from the much simpler distribution proposed by Achlioptas ($s = 1, 3$) [2, 70]:

$$R_{ij} = \sqrt{s} \begin{cases} 1 & \text{with prob.} \quad 1/2s \\ 0 & \text{with prob.} \quad 1 - 1/s \\ -1 & \text{with prob.} \quad 1/2s \end{cases}. \tag{4.12}$$

[5] In the original publication [72], the proposed method was simply referred to as Kernelized Random Projection (KRP). The change in the name is to disambiguate with the other methods described in this thesis.

Surprisingly, the method presented in the previous section is directly compatible with the sparse distribution proposed by Achlioptas [2]. Let us consider (4.4) again: we want to analyze the distribution of projection vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$ when the random vectors $\mathbf{r}_1, \ldots, \mathbf{r}_g \in \mathbb{R}^d$ used in this equation are drawn from the distribution defined in (4.12). It is clear that the entries of $\bigotimes_{j=1}^{g} \mathbf{r}_j$ are the product of g independent and discrete random variables drawn from (4.12). As a consequence, the marginal distribution of those entries can be determined using the properties of discrete random variables. Particularly, given g independent and identically distributed random variables $V_1, \ldots, V_g$ with support $\mathcal{V}$ (i.e., the set of realizations that have a strictly positive probability of being observed), the distribution of the product $V_1 \cdots V_g$ can be computed as follows:

$$P(V_1 \cdots V_g = c) = \sum_{\substack{v_1,\ldots,v_g \in \mathcal{V} \\ \text{s.t. } v_1 \cdots v_g = c}} P(V_1 = v_1) \cdots P(V_g = v_g). \tag{4.13}$$

Looking at (4.12) we can see that, in our case, the support is $\mathcal{V} = \{-1, 0, 1\}$, with associated probabilities $\frac{1}{2s}, 1 - \frac{1}{s}$ and $\frac{1}{2s}$. Applying (4.13) we get that, when analyzed individually, the entries of $\bigotimes_{j=1}^{g} \mathbf{r}_j$ are distributed according to:

$$\sqrt{s^g} \begin{cases} 1 & \text{with prob.} \quad 1/2s^g \\ 0 & \text{with prob.} \quad 1 - 1/s^g \\ -1 & \text{with prob.} \quad 1/2s^g \end{cases}, \tag{4.14}$$

which is a valid sparse Random Projection distribution[6] [2, 70]. However, just like when we used the normal distribution, the entries in $\bigotimes_{j=1}^{g} \mathbf{r}_j$ are not independent from each other. Fortunately, as the above distribution has zero mean and unit variance, the multidimensional CLT can be applied just like in the Gaussian case. Particularly, since vectors of the form $\bigotimes_{j=1}^{g} \mathbf{r}_j$ with the entries of $\mathbf{r}_1, \ldots, \mathbf{r}_g$ independently drawn according to (4.12) have zero mean and identity covariance matrix, the scaled sum of such vectors converges in distribution to $\mathcal{N}_{d^g}(\mathbf{0}, \mathbf{I})$ by virtue of the multidimensional CLT. As a consequence, the method proposed in the previous section is directly compatible with the discrete distribution proposed by Achlioptas. In fact, one might draw the random vectors of (4.10) from Achliptas' distribution and the result would still approximate a valid Gaussian Random Projection [8] from the feature space for a sufficiently large t. This claim is also supported by the experimental results presented in Section 4.4.

To use this sparse variant of the random vectors with our method, it suffices to modify step 1 of Algorithm 2. Instead of generating the random vectors in S by sampling $\mathcal{N}_d(\mathbf{0}, \mathbf{I})$, they can be populated following the sparse distribution described

[6] To see this, simply substitute s by s^g in (4.12).

in (4.12). Formally, step 1 of Algorithm 2 becomes:

$$S \leftarrow \{\mathbf{r}_1, \ldots, \mathbf{r}_p\}, \quad \text{where the entries of } \mathbf{r}_1, \ldots, \mathbf{r}_p \in \mathbb{R}^d \text{ are i.i.d. from:}$$
$$\Pr(x) = \begin{cases} 1/2s & , \quad x = \sqrt{s} \\ 1 - 1/s & , \quad x = 0 \\ 1/2s & , \quad x = -\sqrt{s} \end{cases}, \tag{4.15}$$

where the hyperparameter s controls the sparsity level of the vectors. Conveniently, the subsequent steps of the algorithm remain exactly the same. Also note that, apart from the sparseness of the vectors, a major advantage of Achlioptas' distribution is the fact that the projection of samples onto the random vectors (Algorithm 2, step 9) can be implemented solely in terms of aggregate evaluation (i.e., summations and subtractions) by delaying the multiplication by $\sqrt{s}$ present in Achlioptas' distribution. This implementation trick can be an advantage in structured database environments, as the projections can be implemented with standard SQL primitives.

4.3.3 Extension to inhomogeneous polynomial kernels

Thus far, we have focused solely on the family of homogeneous polynomial kernels. However, in some cases it might be useful to consider inhomogeneous polynomial kernels. As seen before, inhomogeneous polynomial kernels are those of the form:

$$K(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + c)^g, \text{ with } c > 0. \tag{4.16}$$

Using $c > 0$ basically includes lower-degree polynomial interactions of the original features in the kernel feature space, which might be beneficial in some applications.

As noted by Pham *et al.* [82], once a method to approximate homogeneous polynomial kernels is available, it can be easily extended to work with inhomogeneous polynomial kernels. Specifically, this is possible thanks to the following property of polynomial kernels[7]:

$$\left\langle \phi(\mathbf{x} \| \sqrt{c}), \phi(\mathbf{y} \| \sqrt{c}) \right\rangle = \left\langle \mathbf{x} \| \sqrt{c}, \mathbf{y} \| \sqrt{c} \right\rangle^g = (\langle \mathbf{x}, \mathbf{y} \rangle + c)^g, \tag{4.17}$$

where $\phi(\mathbf{x}) = \bigotimes_{i=1}^{g} \mathbf{x}$ is the feature map of the homogeneous polynomial kernel of degree g, and the operator $\|$ denotes feature concatenation. In essence, this means that $\mathbf{x} \to \phi(\mathbf{x} \| \sqrt{c})$ is a valid feature map for the inhomogeneous polynomial kernel of constant c and degree g. Conveniently, $\phi(\cdot)$ itself is the feature map of the homogeneous polynomial kernel. Therefore, by applying our kernelized Random Projection method on $\mathbf{x} \| \sqrt{c}$ we can approximate a Random Projection of $\phi(\mathbf{x} \| \sqrt{c})$, and in turn a Random Projection of the mapping of $\mathbf{x}$ into the feature space of the inhomogeneous polynomial kernel with constant c and degree g.

[7] The first equality simply applies the definition of the homogeneous polynomial kernel and its feature map $\phi(\cdot)$. The second one uses the definition of the dot product and the fact that both $\mathbf{x} \| \sqrt{c}$ and $\mathbf{y} \| \sqrt{c}$ have $\sqrt{c}$ as their last feature.

4.3.4 Computational complexity analysis

This section compares the computational complexities of the different methods considered in this chapter both in training and test phases. First, we analyze the computational complexity of the KG-RP method. Most of the computations involved in this method correspond to the calculations performed by the Kulis-Grauman approach [65]. However, the computational complexities reported vary slightly due to the different optimizations applied. Let us analyze step by step the computations performed by taking the pseudo-code implementation presented in [110] as the reference. In the training stage we have to:

1. Compute the $p \times p$ kernel Gram matrix $K_{\mathcal{S}}$ among the p selected training points. Assuming the kernel computation takes $\mathcal{O}(d)$ for samples in $\mathbb{R}^d$, this step requires $\mathcal{O}(dp^2)$ time.
2. Compute $K_{\mathcal{S}}^{-1/2}$ by means of eigendecomposition, which requires $\mathcal{O}(p^3)$ time.
3. Form the weight vector for each output component $\mathbf{w}_1, \ldots, \mathbf{w}_k$. Since each vector is computed as $\mathbf{w}_i = \sqrt{\frac{p-1}{t}} K_{\mathcal{S}}^{-1/2} e_{\mathcal{S}}$, and $e_{\mathcal{S}}$ is a p-dimensional column vector, this step has a complexity of $\mathcal{O}(kp^2)$.

By combining the different steps, the total complexity of training KG-RP is $\mathcal{O}(dp^2 + p^3 + kp^2)$. Note that, in the case of KG-RP, p is a hyperparameter which controls the number of training samples used to estimate the mean and covariance matrix of data in the kernel feature space. The authors suggested the heuristic rule of setting $p = \mathcal{O}(\sqrt{N})$, where N is the number of available training samples. Regarding the test phase, the following computations must be performed to transform a single sample:

1. Compute the kernel Gram matrix K between the test sample and the p points in $\mathcal{S}$. Assuming the kernel computation takes $\mathcal{O}(d)$ for samples in $\mathbb{R}^d$, this step requires $\mathcal{O}(pd)$ operations.
2. Generate the final representation of the test sample as KW, where $W = [\mathbf{w}_1, \ldots, \mathbf{w}_k]$. This can be done at the cost of $\mathcal{O}(pk)$ time.

Therefore, transforming a single test sample with KG-RP is $\mathcal{O}(pd + pk)$.

Now we analyze the proposed algorithm, PK-RP. The computations needed to initialize/train the algorithm (steps 1-2 of Algorithm 2) are the following:

1. The set S is populated with p random vectors drawn from $\mathcal{N}_d(\mathbf{0}, \mathbf{I})$ (or alternatively using Achlioptas' distribution as described in Section 4.3.2), where d is the dimension of data samples. This can be done in $\mathcal{O}(pd)$ time.
2. The set S is sampled at random to form $S_1, \ldots, S_k \subset S$, each with cardinality gt. This takes $\mathcal{O}(gtk)$ time, where $gt < p$.

This shows that the training stage of the proposed method has a computational complexity of $\mathcal{O}(pd + gtk)$. To transform a test sample, each output component is computed by using (4.9) or equivalently, executing steps 3-12 of Algorithm 2. In any

case, this computation requires a time of $\mathcal{O}(gtkd)$. As mentioned before, this complexity can be reduced by pre-computing the inner products between the test sample and the p vectors in S. By so doing, the computational complexity of transforming a sample by means of the proposed method ends up being $\mathcal{O}(pd+gtk)$. It is also worth comparing the complexity of our method with that of D2PK-RP, presented in [73]. As explained in the previous chapter, D2PK-RP uses a number t of $d \times k$ projection matrices. From the analysis presented in the previous chapter, populating the projection matrices for D2PK-RP takes $\mathcal{O}(dtk)$ time, and transforming one sample requires $\mathcal{O}(dtk)$ operations. As evidenced by our experimental results, this multiple projection matrix approach is relatively inefficient, often leading to computing times one order of magnitude higher than those of PK-RP, while exhibiting an equal or worse performance. Also note that the complexity of D2PK-RP is independent of the polynomial degree g, because this method is only compatible with $g = 2$.

Finally, the Nyström method works by generating a low-rank approximation of the kernel matrix by sampling a number of columns [103]. Although some alternative sampling methods have been studied, the original method, where a fixed random distribution is used to select the columns from the kernel matrix, continues being one of the most widely used approaches [67]. For our analysis and experiments, we focus on the standard Nyström algorithm as implemented in [80]. The computations involved in training this algorithm are the following:

1. Compute the $k \times k$ reduced kernel Gram matrix W among the samples corresponding to the k selected columns from the full kernel matrix. Assuming the kernel computation takes $\mathcal{O}(d)$ for samples in $\mathbb{R}^d$, this step takes $\mathcal{O}(k^2 d)$.
2. Compute $W^{-\frac{1}{2}}$ by means of Singular Value Decomposition, which requires $\mathcal{O}(k^3)$ time.

In summary, the training stage of Nyström requieres a time of $\mathcal{O}(dk^2 + k^3)$. The following operations are performed in the test phase to transform each data sample:

1. Compute the kernel Gram matrix K between the test sample and the k samples selected during training. Assuming the kernel computation takes $\mathcal{O}(d)$ for samples in $\mathbb{R}^d$, this step requires $\mathcal{O}(dk)$ operations.
2. Generate the output representation for the test sample as $KW^{-\frac{1}{2}}$. Since $W^{-\frac{1}{2}}$ is of size $k \times k$, this can be done in $\mathcal{O}(k^2)$ time.

From the combination of these complexities we obtain that transforming a test sample by means of the Nyström method has as a time complexity of $\mathcal{O}(dk + k^2)$.

Our analysis shows that the proposed algorithm exhibits a better computational complexity than the alternative methods. Concerning the training phase, the time required by KG-RP increases as the cube of p, and also requires p^2 evaluations of the kernel function. Similarly, Nyström's training time grows as the cube of k, and involves k^2 kernel evaluations. For its part, our proposed method has a training time which grows linearly with respect to p and k. In addition, thanks to its data-independent nature, it requires no evaluations of the kernel function at training time.

TABLE 4.1: Computational complexities of different methods, and meaning of the variables. (*) Complexity obtained by pre-computing inner products between test samples and the p random vectors in S.

Method	**Train**	**Transform**
PK-RP	$\mathcal{O}(pd + gtk)$	$\mathcal{O}(dgtk)$
PK-RP*	$\mathcal{O}(pd + gtk)$	$\mathcal{O}(pd + gtk)$
KG-RP [5]	$\mathcal{O}(p^2d + p^3 + kp^2)$	$\mathcal{O}(pd + pk)$
Nyström [103]	$\mathcal{O}(dk^2 + k^3)$	$\mathcal{O}(kd + k^2)$
D2PK-RP [73]	$\mathcal{O}(dtk)$	$\mathcal{O}(dtk)$
$\phi(\cdot)$+RP	$\mathcal{O}(d^g k)$	$\mathcal{O}(d^g k)$

Variable	**Meaning**
d	input sample dimension
k	output sample dimension
g	polynomial kernel degree
t (D2PK-RP)	number of projection matrices (CLT)
t (PK-RP)	number of summations to form each projection vector (CLT)
p (KG-RP)	number of training samples used
p (PK-RP)	number of random vectors used

Our method is also very competitive in terms of testing-time complexity. Provided that $gt < p$, the complexity of our method is lower than that of KG-RP.

The train and test computational complexities of the different methods analyzed in this section are summarized in Table 4.1.

4.4 Experimental results

This section presents extensive experimental results validating the ability of the proposed method to (1) generate a low-dimensional representation where the distances between points are approximately equal to distances in the feature space of homogeneous polynomial kernels; and (2) boost the accuracy of linear classifiers by generating a data representation where they can approximate the accuracy of their non-linear counterparts.

The methods evaluated are D2PK-RP (see Chapter 3), PK-RP, KG-RP [5], and the Nyström method [103]. We also compare these methods with the explicit approach $\phi(\cdot) + RP$, which involves explicitly transforming data with the feature map $\phi(\cdot)$ followed by a classic Random Projection. Of course, this approach is highly inefficient, but we use it to measure how well D2PK-RP, KG-RP and PK-RP approximate a Random Projection from the kernel feature space.

To evaluate the first property (i.e., the pairwise distance preservation), we compare the squared Euclidean distance between two transformed data samples to their squared Euclidean distance in the kernel feature space. Formally, let $\mathbf{x}, \mathbf{y}$ be a couple of data samples from $\mathbb{R}^d$ and let $f(\mathbf{x}), f(\mathbf{y})$ be their k-dimensional representation generated by any of the methods evaluated in this section, then:

$$distortion_{\mathbf{x},\mathbf{y}} = \frac{abs(\ ||f(\mathbf{x}) - f(\mathbf{y})||^2 - ||\phi(\mathbf{x}) - \phi(\mathbf{y})||^2\ )}{||\phi(\mathbf{x}) - \phi(\mathbf{y})||^2}. \tag{4.18}$$

For example, if $distortion_{\mathbf{x},\mathbf{y}} = 0.11$, we can conclude that the distance between $\mathbf{x}$ and $\mathbf{y}$ in $\mathcal{H}$ suffered a 11% increase/decrease in the resulting representation. Also note that distances between samples in the feature space can be calculated without any explicit evaluation of $\phi(\cdot)$, via the kernel function [86]:

$$||\phi(\mathbf{x}) - \phi(\mathbf{y})||^2 = -2K(\mathbf{x},\mathbf{y}) + K(\mathbf{x},\mathbf{x}) + K(\mathbf{y},\mathbf{y}). \tag{4.19}$$

To measure the distortion induced by a given method while transforming a set of N samples, the average distortion among the $\binom{N}{2}$ possible pairs of different samples is computed. We then use the average distortion measure to compare the different approaches evaluated in this section in terms of their distance preservation capabilities. To score them, the different methods were first provided with the corresponding training set if needed. Next, 500 samples from the test-set of each dataset were selected at random and transformed by means of each competing method. The induced distortion was then computed and averaged for the $\binom{500}{2}$ possible pairs of samples.

As stated above, we also evaluated to what extent the different methods can be used to boost the accuracy of linear classifiers by generating a data representation where they can approximate the accuracy of their non-linear counterparts. To this extent, each kernel approximation method was provided, if needed, with the corresponding training set. Next, it was used to transform both the training set and the complete test set. A linear SVM was then trained[8] on the resulting representation and its classification accuracy was obtained. To evaluate the improvement in the classification accuracy, we also provide the resulting accuracy of training a linear SVM directly on the original features of each dataset.

To mitigate the stochastic nature of some of the evaluated methods, the above described evaluation protocol was executed ten times. All the results reported in this section consist of the average and standard deviation of the corresponding metric over those ten runs. For a fair comparison, all the experiments in this chapter were carried out on the same machine, equipped with an Intel i7-6700K processor and 16GB of DDR4 RAM. To ease the visualization of results in tables, each cell is colored according to the reported score (lighter is better in all tables).

4.4.1 MNIST dataset

The database used for the first set of experiments is MNIST [108]. This database consists of a collection of images of handwritten digits and has been profusely used in optical character recognition and machine learning research. It contains a total of 70,000 images, each of size 28×28 (see Figure 4.4). The digits are size-normalized and centered on the center of gravity of the intensity in the image. A predefined split is normally used with 60,000 images for training and 10,000 for testing.

[8] We used the linear SVM implementation of Liblinear [35]. An appropriate value of C for the SVM was determined by performing cross-validation over the training data for each dataset. Data standardization methods were used prior to classification in all experiments.

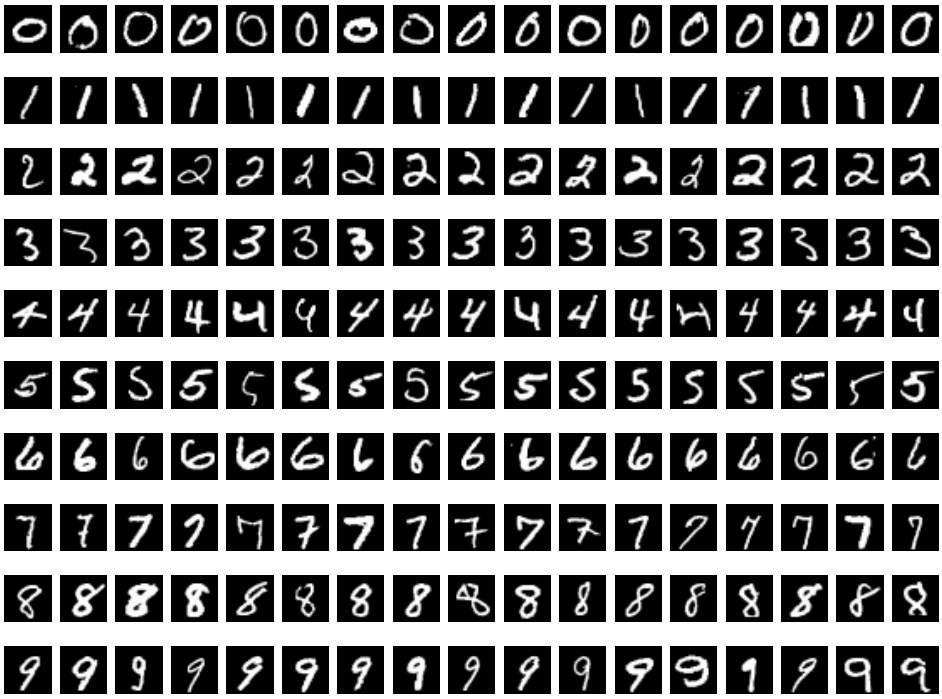


FIGURE 4.4: Some random examples from the different categories of MNIST.

TABLE 4.2: Results on distance preservation from the homogeneous polynomial kernel of degree two ($g = 2$) for 500 samples from MNIST.

Method	**Parameters**	**200 output dim.**		**500 output dim.**		**1000 output dim.**	
		avg. dist.	time	avg. dist.	time	avg. dist.	time
$\phi(\cdot)$+RP	Gaussian	0.079±0.002	4.416s	0.050±0.002	10.124s	0.035±0.001	19.724s
$\phi(\cdot)$+RP	Sparse, s=1	0.080±0.003	4.314s	0.051±0.002	9.893s	0.036±0.001	19.338s
$\phi(\cdot)$+RP	Sparse, s=3	0.080±0.002	3.616s	0.050±0.002	8.899s	0.036±0.001	19.495s
Nyström [103]	-	0.259±0.002	0.064s	0.155±0.001	0.102s	0.101±0.001	0.309s
D2PK-RP [73]	s=1	0.085±0.002	0.121s	0.054±0.002	0.311s	0.038±0.001	0.651s
PK-RP	Gaussian, p=16000	0.082±0.004	0.583s	0.053±0.002	0.611s	0.038±0.002	0.609s
PK-RP	Gaussian, p=8000	0.083±0.003	0.299s	0.055±0.003	0.315s	0.040±0.002	0.334s
PK-RP	Gaussian, p=3000	0.087±0.003	0.122s	0.056±0.002	0.134s	0.046±0.005	0.152s
PK-RP	Gaussian, p=976	0.092±0.007	0.039s	0.079±0.018	0.054s	0.059±0.007	0.079s
PK-RP	Sparse, s=1, p=976	0.094±0.004	0.038s	0.068±0.008	0.050s	0.059±0.008	0.075s
PK-RP	Sparse, s=3, p=976	0.098±0.007	0.038s	0.072±0.009	0.051s	0.060±0.008	0.074s
KG-RP [110]	p=976	0.141±0.013	7.525s	0.127±0.011	8.602s	0.130±0.006	10.509s
PK-RP	Gaussian, p=488	0.106±0.008	0.025s	0.102±0.029	0.036s	0.095±0.038	0.058s
PK-RP	Sparse, s=1, p=488	0.101±0.004	0.023s	0.088±0.012	0.037s	0.082±0.017	0.058s
PK-RP	Sparse, s=3, p=488	0.113±0.015	0.022s	0.095±0.019	0.037s	0.092±0.021	0.061s
KG-RP [110]	p=488	0.207±0.019	1.957s	0.211±0.005	2.378s	0.210±0.004	3.136s
PK-RP	Gaussian, p=244	0.120±0.006	0.015s	0.106±0.018	0.028s	0.112±0.024	0.053s
PK-RP	Sparse, s=1, p=244	0.132±0.018	0.015s	0.126±0.028	0.027s	0.101±0.020	0.050s
PK-RP	Sparse, s=3, p=244	0.122±0.010	0.016s	0.116±0.027	0.030s	0.118±0.027	0.055s
KG-RP [110]	p=244	0.329±0.014	0.546s	0.329±0.006	0.714s	0.328±0.005	1.018s
PK-RP	Gaussian, p=122	0.177±0.052	0.011s	0.151±0.027	0.026s	0.135±0.014	0.051s
PK-RP	Sparse, s=1, p=122	0.152±0.028	0.011s	0.136±0.013	0.024s	0.159±0.038	0.045s
PK-RP	Sparse, s=3, p=122	0.177±0.046	0.012s	0.150±0.033	0.024s	0.159±0.027	0.045s
KG-RP [110]	p=122	0.503±0.012	0.180s	0.497±0.008	0.268s	0.499±0.005	0.427s

Distance preservation on MNIST

First, we evaluate the different algorithms in terms of pairwise distance preservation. We do so for the two most frequently used polynomial degrees, namely $g = 2$ and $g = 3$. We also measured the time required to train each algorithm and transform 500

TABLE 4.3: Results on distance preservation from the homogeneous polynomial kernel of degree three ($g = 3$) for 500 samples from MNIST.

Method	**Parameters**	**200 output dim.** avg. dist.	time	**500 output dim.** avg. dist.	time	**1000 output dim.** avg. dist.	time
Nyström [103]	-	0.372±0.002	0.065s	0.255±0.001	0.116s	0.187±0.001	0.334s
PK-RP	Gaussian, p=976	0.119±0.019	0.045s	0.095±0.023	0.060s	0.092±0.031	0.089s
PK-RP	Sparse, s=1, p=976	0.109±0.009	0.040s	0.092±0.018	0.055s	0.080±0.016	0.088s
PK-RP	Sparse, s=3, p=976	0.114±0.020	0.039s	0.117±0.057	0.057s	0.088±0.032	0.082s
KG-RP [110]	p=976	0.215±0.008	7.578s	0.212±0.008	8.708s	0.214±0.006	10.576s
PK-RP	Gaussian, p=488	0.153±0.031	0.029s	0.131±0.040	0.043s	0.108±0.046	0.072s
PK-RP	Sparse, s=1, p=488	0.148±0.035	0.024s	0.127±0.041	0.042s	0.110±0.024	0.071s
PK-RP	Sparse, s=3, p=488	0.134±0.031	0.027s	0.124±0.026	0.041s	0.131±0.055	0.067s
KG-RP [110]	p=488	0.298±0.012	2.001s	0.306±0.006	2.415s	0.305±0.008	3.145s
PK-RP	Gaussian, p=244	0.164±0.025	0.018s	0.141±0.036	0.034s	0.168±0.066	0.062s
PK-RP	Sparse, s=1, p=244	0.192±0.075	0.018s	0.162±0.053	0.033s	0.159±0.053	0.063s
PK-RP	Sparse, s=3, p=244	0.209±0.097	0.018s	0.169±0.040	0.034s	0.149±0.049	0.061s
KG-RP [110]	p=244	0.421±0.011	0.560s	0.424±0.004	0.731s	0.427±0.007	1.020s
PK-RP	Gaussian, p=122	0.249±0.064	0.013s	0.250±0.112	0.032s	0.241±0.083	0.059s
PK-RP	Sparse, s=1, p=122	0.253±0.080	0.013s	0.247±0.119	0.030s	0.241±0.090	0.059s
PK-RP	Sparse, s=3, p=122	0.232±0.093	0.013s	0.213±0.038	0.029s	0.271±0.172	0.057s
KG-RP [110]	p=122	0.587±0.009	0.180s	0.587±0.009	0.277s	0.581±0.008	0.424s

test samples, reporting the average time required by each method. For both KG-RP and our method, the hyperparameter p must be manually selected. Recall that, for KG-RP, p controls the number of samples used by the underlaying Kulis-Grauman method to estimate the mean and covariance matrix of data in the kernel feature space (for more details see [65]). Meanwhile, in our method p controls the number of random vectors used to populate S (see Section 4.3.1). The reason for comparing KG-RP and our method with equal values of p while they have different meanings is that, for both algorithms, the value of p determines the number of evaluations of inner products involving the d-dimensional data samples during the test phase (see Section 4.3.4). Furthermore, in both cases p controls the accuracy/efficiency tradeoff of the algorithm. Due to the way these algorithms were designed, we know that increasing p will likely result in better results at the expense of higher processing times. For this reason, we empirically evaluated the accuracy/efficiency tradeoff that occurs when different values of p are chosen. In particular, we experimented with various values for p following the heuristic criterion proposed in [65]. There, the authors advise using $p = \mathcal{O}(\sqrt{N})$, where N is the number of training samples available. Accordingly, we experimented with $p = \frac{1}{2}\sqrt{N}, \sqrt{N}, 2\sqrt{N}$ and $4\sqrt{N}$. The hyperparameter t, which controls the number of samples used by the CLT, was set to the typical value of 30 (see [65]). The results for the polynomial degrees 2 and 3 can be found in Tables 4.2 and 4.3 respectively.

Note that the method involving the explicit computation of $\phi(\cdot)$ for each test sample was not evaluated for $g = 3$. In the case of the MNIST dataset, storing the explicit form of test samples in the kernel feature space for $g = 2$ required approximately 1.14 GBs of free memory[9]. Doing so for the homogeneous polynomial kernel of degree 3 would have required almost a terabyte of main memory, which is nearly intractable even for specialized high-performance computing systems.

[9]Since $x \in \mathbb{R}^{784}$, $\phi(\mathbf{x}) \in \mathcal{H}$ is 784^g-dimensional. In the case of the homogeneous polynomial kernel of degree 2, $\mathcal{H}$ is 614656-dimensional. As a consequence, the storage of 500 samples, assuming that a 4-byte float format is used, takes about 1.14 GBs of memory.

Not surprisingly, the results obtained using $\phi(\cdot)$+RP are the best in all cases. However, the explicit computation of the feature map comes at a great cost. Looking at the computation times of this approach we see that, in all cases, transforming 500 MNIST samples took longer that one minute. In addition, as previously explained, this approach becomes intractable for polynomial degrees greater than two. For its part, Nyström seeks to preserve inner products rather than pairwise distances. However, a close relation between both metrics exists. As a consequence, our experiments show that Nyström was able to approximately preserve the pairwise distances, but induced a significantly higher distortion than the other methods is most cases. In addition, the distance preservation capabilities of Nyström seem to be highly dependent on the output dimension, which is an important drawback since Nyström's time complexity scales polynomially with this hyperparameter.

Finally, D2PK-RP, KG-RP and PK-RP try to approximate $\phi(\cdot)$+RP while avoiding the expensive computation of the feature map. KG-RP exhibited a high sensitivity to variations in the value of p. Unfortunately, the computational cost of KG-RP grows fast with the value of this hyperparameter (see Section 4.3.4). Moreover, even if high values of p are used, KG-RP's distance preservation results are significantly worse than those of our proposed method. PK-RP provided the best approximation of $\phi(\cdot)$+RP with a very low computational cost. As expected, if a sufficiently high p is used, PK-RP induces an average distortion in pairwise distances almost as small as the explicit approach. Conveniently, this approximation is achieved with a very small computational cost (e.g., it only took 70ms to train the algorithm and project 500 MNIST samples to $\mathbb{R}^{1000}$). In this case, D2PK-RP seems to slightly outperform our proposed method when $p < 1000$, at the cost of much greater computation times. However, increasing p above that threshold enables our method to match the accuracy of D2PK-RP, sacrificing some of its efficiency. Also note that D2PK-RP was only evaluated for $g = 2$, as it is only compatible with the second degree polynomial kernel.

Classification on MNIST

Here we evaluate to which extent the different methods can be used to boost the accuracy of linear classifiers. In this case, we experimented with different values of t and p[10]. The resulting classification accuracies and their standard deviations can be found in Table 4.4, where we also provide the computation times required to train each method and to use it to transform the MNIST training set. It is worth noticing that the accuracy of a linear SVM classifier trained on the original MNIST samples is 91.81% (using the implementation of Liblinear [35] with C=0.5) and the accuracy of a degree-2 polynomial-kernel SVM is 97.84% (C=0.5). For completeness, we trained a SVM with the Gaussian kernel, which achieved a 98.56% accuracy ($C = 5$, $\gamma = 0.02$).

[10]Note that p and t hyperparameters are only used by two of the evaluated methods, namely KG-RP and PK-RP (see Section 4.3 for more details).

TABLE 4.4: Classification accuracies on MNIST obtained by a linear SVM trained on the representations generated by different methods to approximate the feature space of the homogeneous polynomial kernel of degree two.

F. Extraction	Parameters	1500 output dim.			2000 output dim.		
		Acc. (%)	Train	Transform	Acc. (%)	Train	Transform
Nyström [103]	-	97.67±0.05	0.957s	3.510s	97.96±0.02	2.434s	5.394s
$\phi(\cdot)$+RP	Gaussian,	96.98±0.09	22.6s	4h 38m	97.40±0.07	34.8s	6h 10m
D2PK-RP [73]	t=10	96.96±0.04	0.172s	18.65s	97.33±0.12	0.302s	45.19s
PK-RP	Gaussian, t=10, p=488	96.98±0.11	0.012s	3.162s	97.31±0.05	0.013s	4.073s
PK-RP	Sparse, s=1, t=10, p=488	96.97±0.11	0.010s	3.165s	97.37±0.07	0.012s	4.067s
PK-RP	Sparse, s=3, t=10, p=488	97.03±0.20	0.011s	3.165s	97.30±0.14	0.012s	4.067s
KG-RP [110]	t=10, p=488	96.15±0.04	3.918s	1.652s	96.27±0.11	4.577s	2.036s
PK-RP	Gaussian, t=1, p=488	96.95±0.12	0.012s	1.409s	97.31±0.13	0.011s	1.736s
PK-RP	Sparse, s=1, t=1, p=488	96.90±0.07	0.010s	1.390s	97.33±0.08	0.010s	1.738s
PK-RP	Sparse, s=3, t=1, p=488	97.00±0.08	0.011s	1.404s	97.23±0.08	0.010s	1.729s
KG-RP [110]	t=1, p=488	95.95±0.07	3.833s	1.652s	96.09±0.08	4.626s	2.018s
PK-RP	Gaussian, t=10, p=244	97.01±0.08	0.006s	2.950s	97.31±0.09	0.008s	3.859s
PK-RP	Sparse, s=1, t=10, p=244	96.91±0.11	0.006s	2.958s	97.30±0.06	0.006s	3.872s
PK-RP	Sparse, s=3, t=10, p=244	96.93±0.13	0.006s	2.956s	97.21±0.13	0.006s	3.861s
KG-RP [110]	t=10, p=244	94.19±0.26	1.295s	1.121s	94.44±0.23	1.586s	1.374s
PK-RP	Gaussian, t=1, p=244	96.84±0.09	0.006s	1.204s	97.20±0.09	0.006s	1.511s
PK-RP	Sparse, s=1, t=1, p=244	96.86±0.11	0.005s	1.186s	97.17±0.12	0.005s	1.513s
PK-RP	Sparse, s=3, t=1, p=244	96.85±0.16	0.006s	1.208s	97.24±0.06	0.005s	1.504s
KG-RP [110]	t=1, p=244	94.48±0.13	1.294s	1.122s	94.46±0.25	1.583s	1.372s
PK-RP	Gaussian, t=10, p=122	96.78±0.12	0.003s	2.869s	96.96±0.06	0.004s	3.750s
PK-RP	Sparse, s=1, t=10, p=122	96.70±0.17	0.003s	2.903s	97.00±0.20	0.003s	3.752s
PK-RP	Sparse, s=3, t=10, p=122	96.79±0.09	0.003s	2.834s	96.93±0.05	0.003s	3.752s
KG-RP [110]	t=10, p=122	91.96±0.17	0.572s	0.827s	92.10±0.10	0.714s	1.044s
PK-RP	Gaussian, t=1, p=122	96.65±0.11	0.004s	1.096s	96.82±0.14	0.003s	1.418s
PK-RP	Sparse, s=1, t=1, p=122	96.59±0.11	0.003s	1.073s	96.85±0.12	0.003s	1.422s
PK-RP	Sparse, s=3, t=1, p=122	96.59±0.10	0.003s	1.088s	96.91±0.11	0.003s	1.414s
KG-RP [110]	t=1, p=122	92.06±0.20	0.565s	0.815s	92.19±0.23	0.707s	1.051s

As we can see, the highest accuracies for both 1500 and 2000 output dimensions were achieved with Nyström. The explicit $\phi(\cdot)$+RP approach yielded slightly lower accuracies, with the gap being smaller when using 2000 output features. As expected, the computational time required by this approach was several orders of magnitude higher than that of the other methods. As both PK-RP and KG-RP try to approximate the computations performed in the explicit approach, we can not expect them to outperform Nyström in this case. Regarding PK-RP, when a reasonably high p was used, our method achieved the same classification accuracy as the explicit approach. The accuracies achieved by D2PK-RP were similar to those of PK-RP, but again with a computational cost one order of magnitude larger.

We found that the impact of t on the classification accuracy is almost negligible, and thus we recommend using a small value. In this regard, figure 4.5 shows the effect of varying t while keeping the remaining hyperparameters fixed. As in the previous set of experiments, our method was the most efficient alternative, especially in the training phase where it was several orders of magnitude faster than some alternative methods. In this case, KG-RP fails to approximate the accuracies obtained by the explicit approach, even for the highest values of p evaluated.

4.4.2 Webspam dataset

The *webspam* dataset [101] compiles thousands of web pages categorized as spam or legitimate. The goal of its creators was to facilitate research on web spam detection

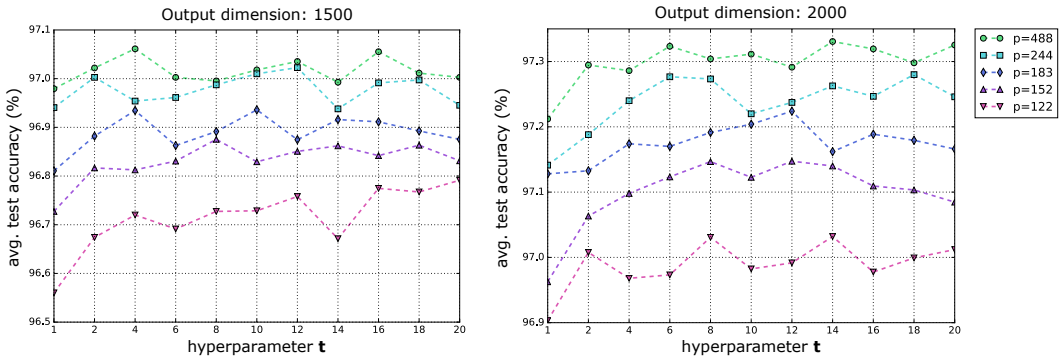


FIGURE 4.5: Effect on the classification accuracy of varying the value of t while using a fixed output dimension and p hyperparameter value. The accuracies were obtained by applying the proposed method with $g = 2$ followed by a linear SVM on the MNIST dataset, averaging the resulting accuracies of each experiment over 15 runs.

algorithms by providing a large-scale, publicly available dataset. A refined version of this dataset, used in [21], can be found at the *LIBSVM tools* web page [20]. This subset consists of uni-gram count features for 350,000 websites. Each sample was normalized to unit-length and the number of features for each sample is 254. As opposed to MNIST, this dataset does not come with predefined training and testing sets. For this reason, we used a 80/20 random split for training and testing, as done in [21]. Hence, the training and testing datasets consist of 280,000 and 70,000 samples respectively.

Distance preservation on Webspam

Tables 4.5 and 4.6 compile the results concerning the average distance distortion obtained when transforming 500 samples from the Webspam dataset with $g = 1$ and $g = 2$ respectively. A value of $t = 30$ was used for KG-RP and PK-RP in all cases.

Interestingly, in this case Nyström provided the best results in terms of pairwise distance preservation from the kernel feature space. It even outperformed the explicit approach of $\phi(\cdot)$ + RP. The success of this method when evaluated on the Webspam dataset contrasts with the results obtained in our experiments with other datasets, where Nyström never outperformed $\phi(\cdot)$ + RP or PK-RP. Nevertheless, the previously mentioned limitation regarding the scalability of Nyström holds.

The results obtained using $\phi(\cdot)$+RP were as good as expected. However, once again computational costs render this approach impractical. Even with the relatively low original dimension of Webspam samples, the explicit approach consumes up to 9 seconds to initialize its projection matrices and transform 500 samples.

Regarding KG-RP and PK-RP, our results suggest that they are evenly matched when it comes to approximating the pairwise-distance preservation capabilities of the explicit approach. However, if computational requirements are considered, our proposed approach offers a significantly better option, as in this case it can provide

TABLE 4.5: Results on distance preservation from the homogeneous polynomial kernel of degree two ($g = 2$) for 500 samples from Webspam.

Method	Parameters	200 output dim.		500 output dim.		1000 output dim.	
		avg. dist.	time	avg. dist.	time	avg. dist.	time
$\phi(\cdot)$+RP	Gaussian	0.078±0.007	7.358s	0.052±0.005	7.961s	0.036±0.004	8.991s
$\phi(\cdot)$+RP	Sparse, s=1	0.079±0.008	7.349s	0.049±0.004	7.915s	0.034±0.003	8.927s
$\phi(\cdot)$+RP	Sparse, s=3	0.084±0.007	7.265s	0.052±0.007	7.830s	0.036±0.004	8.798s
Nyström [103]	-	0.037±0.003	0.087s	0.014±0.000	0.124s	0.006±0.000	0.296s
D2PK-RP [73]	s=1	0.170±0.016	0.038s	0.159±0.021	0.132s	0.145±0.008	0.288s
PK-RP	Gaussian, p=2116	0.082±0.011	0.034s	0.060±0.009	0.047s	0.051±0.015	0.068s
PK-RP	Sparse, s=1, p=2116	0.093±0.014	0.031s	0.057±0.005	0.046s	0.051±0.012	0.067s
PK-RP	Sparse, s=3, p=2116	0.086±0.010	0.034s	0.060±0.006	0.049s	0.049±0.005	0.069s
KG-RP [110]	p=2116	0.081±0.006	36.440s	0.054±0.005	41.336s	0.041±0.006	48.975s
PK-RP	Gaussian, p=1058	0.094±0.011	0.020s	0.067±0.011	0.034s	0.052±0.008	0.056s
PK-RP	Sparse, s=1, p=1058	0.088±0.007	0.019s	0.073±0.022	0.035s	0.054±0.009	0.055s
PK-RP	Sparse, s=3, p=1058	0.095±0.018	0.019s	0.066±0.004	0.035s	0.068±0.026	0.055s
KG-RP [110]	p=1058	0.085±0.006	8.683s	0.063±0.009	10.010s	0.047±0.006	12.305s
PK-RP	Gaussian, p=529	0.113±0.020	0.015s	0.086±0.013	0.027s	0.068±0.010	0.049s
PK-RP	Sparse, s=1, p=529	0.103±0.019	0.015s	0.075±0.012	0.028s	0.074±0.034	0.048s
PK-RP	Sparse, s=3, p=529	0.105±0.019	0.013s	0.085±0.018	0.027s	0.076±0.020	0.048s
KG-RP [110]	p=529	0.099±0.017	2.243s	0.082±0.011	2.754s	0.066±0.011	3.557s
PK-RP	Gaussian, p=264	0.114±0.016	0.010s	0.096±0.018	0.025s	0.088±0.012	0.047s
PK-RP	Sparse, s=1, p=264	0.113±0.044	0.011s	0.095±0.015	0.024s	0.080±0.009	0.045s
PK-RP	Sparse, s=3, p=264	0.138±0.034	0.011s	0.111±0.024	0.025s	0.104±0.025	0.045s
KG-RP [110]	p=264	0.140±0.013	0.620s	0.141±0.012	0.813s	0.131±0.013	1.121s

TABLE 4.6: Results on distance preservation from the homogeneous polynomial kernel of degree three ($g = 3$) for 500 samples from Webspam.

Method	Parameters	200 output dim.		500 output dim.		1000 output dim.	
		avg. dist.	time	avg. dist.	time	avg. dist.	time
Nyström [103]	-	0.067±0.001	0.097s	0.032±0.001	0.154s	0.018±0.001	0.379s
PK-RP	Gaussian, p=2116	0.098±0.014	0.040s	0.075±0.009	0.057s	0.064±0.016	0.083s
PK-RP	Sparse, s=1, p=2116	0.100±0.012	0.035s	0.086±0.044	0.055s	0.064±0.029	0.082s
PK-RP	Sparse, s=3, p=2116	0.096±0.011	0.039s	0.096±0.045	0.056s	0.081±0.032	0.082s
KG-RP [110]	p=2116	0.087±0.008	37.241s	0.061±0.013	41.357s	0.047±0.004	50.742s
PK-RP	Gaussian, p=1058	0.114±0.017	0.027s	0.079±0.020	0.042s	0.080±0.021	0.070s
PK-RP	Sparse, s=1, p=1058	0.102±0.014	0.023s	0.095±0.030	0.042s	0.085±0.036	0.070s
PK-RP	Sparse, s=3, p=1058	0.112±0.019	0.023s	0.082±0.013	0.039s	0.088±0.029	0.072s
KG-RP [110]	p=1058	0.092±0.009	8.867s	0.069±0.011	10.138s	0.057±0.004	12.319s
PK-RP	Gaussian, p=529	0.163±0.051	0.019s	0.108±0.026	0.037s	0.123±0.059	0.063s
PK-RP	Sparse, s=1, p=529	0.126±0.020	0.017s	0.141±0.076	0.034s	0.097±0.027	0.062s
PK-RP	Sparse, s=3, p=529	0.147±0.042	0.017s	0.094±0.024	0.035s	0.114±0.050	0.062s
KG-RP [110]	p=529	0.111±0.010	2.320s	0.086±0.011	2.778s	0.083±0.007	3.598s
PK-RP	Gaussian, p=264	0.198±0.132	0.013s	0.175±0.069	0.032s	0.189±0.098	0.062s
PK-RP	Sparse, s=1, p=264	0.166±0.077	0.014s	0.172±0.066	0.029s	0.137±0.056	0.061s
PK-RP	Sparse, s=3, p=264	0.209±0.067	0.013s	0.146±0.049	0.032s	0.120±0.033	0.062s
KG-RP [110]	p=264	0.172±0.017	0.658s	0.157±0.016	0.836s	0.162±0.009	1.141s

similar distance preservation results while keeping computation times under 80 ms (while KG-RP times range from half a second to almost one minute). In this case D2PK-RP performed poorly, while also being significantly more computationally expensive than PK-RP.

Classification on Webspam

As with the previous dataset, we experimented with different values of p and t. The resulting classification accuracies and their standard deviations are shown in Table 4.7. This table also provides the computation times required to train each method and to use it to transform the Webspam training set. Note that the accuracy of a linear SVM classifier trained on the original Webspam dataset is 92.55% (with

TABLE 4.7: Classification accuracies on Webspam obtained by a linear SVM trained on the representations generated by different methods to approximate the feature space of the degree-2 homogeneous polynomial kernel.

F. Extraction	Parameters	1500 output dim.			2000 output dim.		
		Acc. (%)	Train	Transform	Acc. (%)	Train	Transform
Nyström [103]	-	97.99±0.04	3.071s	11.803s	98.23±0.02	3.745s	19.466s
$\phi(\cdot)$+RP	Gaussian	97.90±0.03	2.47s	2h 11m	98.15±0.03	3.17s	2h 54m
D2PK-RP [73]	s=1,t=10	97.80±0.01	0.0631s	50.92s	98.08±0.01	0.080s	154.85s
PK-RP	Gaussian, t=10, p=1058	97.81±0.03	0.009s	8.232s	98.02±0.03	0.009s	10.656s
PK-RP	Sparse, s=1, t=10, p=1058	97.80±0.04	0.009s	8.204s	98.02±0.07	0.007s	10.652s
PK-RP	Sparse, s=3, t=10, p=1058	97.79±0.02	0.009s	8.215s	98.01±0.05	0.007s	10.678s
KG-RP [110]	t=10, p=1058	97.66±0.02	14.901s	11.321s	97.64±0.07	17.328s	14.485s
PK-RP	Gaussian, t=1, p=1058	97.79±0.06	0.009s	3.825s	97.98±0.03	0.009s	5.061s
PK-RP	Sparse, s=1, t=1, p=1058	97.70±0.09	0.009s	3.844s	98.04±0.06	0.007s	5.079s
PK-RP	Sparse, s=3, t=1, p=1058	97.83±0.06	0.009s	3.832s	98.03±0.04	0.007s	5.032s
KG-RP [110]	t=1, p=1058	97.27±0.13	14.915s	10.620s	97.42±0.07	17.844s	14.488s
PK-RP	Gaussian, t=10, p=529	97.82±0.05	0.006s	7.750s	98.03±0.06	0.007s	10.067s
PK-RP	Sparse, s=1, t=10, p=529	97.82±0.03	0.008s	7.692s	98.04±0.05	0.008s	10.071s
PK-RP	Sparse, s=3, t=10, p=529	97.79±0.03	0.007s	7.686s	98.01±0.04	0.008s	10.095s
KG-RP [110]	t=10, p=529	96.75±0.10	4.852s	6.404s	96.72±0.02	5.529s	9.058s
PK-RP	Gaussian, t=1, p=529	97.80±0.03	0.005s	3.319s	98.05±0.02	0.006s	4.251s
PK-RP	Sparse, s=1, t=1, p=529	97.77±0.04	0.006s	3.319s	98.01±0.06	0.007s	4.264s
PK-RP	Sparse, s=3, t=1, p=529	97.80±0.04	0.006s	3.295s	98.02±0.05	0.006s	4.234s
KG-RP [110]	t=1, p=529	96.63±0.13	4.860s	6.511s	96.69±0.09	5.486s	9.534s
PK-RP	Gaussian, t=10, p=264	97.81±0.03	0.005s	7.420s	98.00±0.05	0.006s	9.924s
PK-RP	Sparse, s=1, t=10, p=264	97.80±0.01	0.008s	7.426s	98.00±0.02	0.009s	9.891s
PK-RP	Sparse, s=3, t=10, p=264	97.79±0.06	0.008s	7.410s	98.02±0.02	0.009s	9.850s
KG-RP [110]	t=10, p=264	95.15±0.11	1.662s	5.047s	95.24±0.06	2.006s	6.603s
PK-RP	Gaussian, t=1, p=264	97.76±0.03	0.005s	3.070s	98.01±0.02	0.005s	3.988s
PK-RP	Sparse, s=1, t=1, p=264	97.75±0.10	0.007s	2.986s	97.94±0.06	0.009s	3.972s
PK-RP	Sparse, s=3, t=1, p=264	97.81±0.06	0.007s	3.007s	98.00±0.02	0.008s	3.961s
KG-RP [110]	t=1, p=264	95.24±0.07	1.677s	4.913s	95.21±0.13	1.968s	6.098s

C=4) and the accuracy of a degree-2 polynomial-kernel SVM is 98.4% ($C = 512$). A SVM with the Gaussian kernel achieves a 99.23% accuracy ($C = 8$, $\gamma = 32$).

Again, the highest accuracies for both 1500 and 2000 output dimensions were achieved with by the Nyström method. However, in this case the accuracy difference between $\phi(\cdot)$+RP and Nyström was almost negligible ($\approx 0.09\%$). As expected, both PK-RP and KG-RP approximate the accuracy achieved by the explicit approach, which brings them very close to the accuracy obtained by the winning method. For instance, using PK-RP with the Gaussian distribution, $k = 1500$, $p = 529$ and $t = 1$, one can achieve a linear classification accuracy of 97.80%, which is only 0.19% below the accuracy of Nyström for than same number of output dimensions. However, PK-RP achieves this with a training time two orders of magnitude lower and by using one-third the time to transform the samples. The accuracies obtained with KG-RP are slightly lower than those of PK-RP, and the difference increased when using lower values of p. Also, the computation times of KG-RP are significantly greater. D2PK-RP performed comparably to PK-RP, but as in the previous experiments this performance came with a computational cost approximately ten times that of PK-RP.

4.4.3 W8a dataset

The *w8a* dataset [83] is a widely used [27, 81] web-classification dataset in the context of machine learning research. Conveniently, it is publicly available and can be

TABLE 4.8: Results on distance preservation from the homogeneous polynomial kernel of degree two ($g = 2$) for 500 samples from w8a.

Method	Parameters	200 output dim.		500 output dim.		1000 output dim.	
		avg. dist.	time	avg. dist.	time	avg. dist.	time
$\phi(\cdot)$+RP	Gaussian	0.079±0.003	9.693s	0.051±0.002	10.532s	0.036±0.001	11.951s
$\phi(\cdot)$+RP	Sparse, s=1	0.077±0.002	9.684s	0.049±0.001	10.474s	0.035±0.001	11.890s
$\phi(\cdot)$+RP	Sparse, s=3	0.079±0.001	9.561s	0.050±0.001	10.351s	0.036±0.001	11.687s
Nyström [103]	-	0.772±0.005	0.024s	0.633±0.008	0.056s	0.507±0.006	0.200s
D2PK-RP [73]	s=1	0.131±0.005	0.077s	0.112±0.003	0.334s	0.107±0.003	0.331s
PK-RP	Gaussian, p=892	0.105±0.005	0.021s	0.084±0.004	0.034s	0.078±0.002	0.054s
PK-RP	Sparse, s=1, p=892	0.103±0.002	0.019s	0.080±0.004	0.035s	0.071±0.005	0.055s
PK-RP	Sparse, s=3, p=892	0.107±0.003	0.021s	0.085±0.004	0.033s	0.079±0.005	0.053s
KG-RP [110]	p=892	0.554±0.009	6.744s	0.547±0.007	8.286s	0.548±0.007	10.891s
PK-RP	Gaussian, p=446	0.126±0.009	0.015s	0.109±0.005	0.029s	0.099±0.005	0.049s
PK-RP	Sparse, s=1, p=446	0.126±0.005	0.015s	0.104±0.005	0.026s	0.098±0.008	0.047s
PK-RP	Sparse, s=3, p=446	0.127±0.006	0.014s	0.107±0.006	0.027s	0.101±0.005	0.048s
KG-RP [110]	p=446	0.677±0.007	1.738s	0.676±0.007	2.241s	0.674±0.005	3.045s
PK-RP	Gaussian, p=223	0.157±0.008	0.012s	0.141±0.005	0.023s	0.142±0.008	0.045s
PK-RP	Sparse, s=1, p=223	0.152±0.009	0.010s	0.135±0.006	0.027s	0.132±0.010	0.047s
PK-RP	Sparse, s=3, p=223	0.154±0.008	0.010s	0.151±0.015	0.025s	0.138±0.005	0.047s
KG-RP [110]	p=223	0.791±0.007	0.485s	0.791±0.006	0.652s	0.788±0.006	0.950s
PK-RP	Gaussian, p=111	0.218±0.031	0.009s	0.203±0.016	0.024s	0.191±0.008	0.044s
PK-RP	Sparse, s=1, p=111	0.195±0.009	0.009s	0.183±0.008	0.021s	0.179±0.008	0.044s
PK-RP	Sparse, s=3, p=111	0.218±0.017	0.009s	0.194±0.014	0.022s	0.200±0.009	0.046s
KG-RP [110]	p=111	0.887±0.008	0.160s	0.883±0.007	0.251s	0.883±0.004	0.394s

downloaded from the *LIBSVM tools* web page [20]. Each sample in the dataset consist of a number of binary features which represent the presence/absence of a set of keywords in the web page associated to the sample. The dataset contains a total of 64,000 samples, with 300 features each. Predefined training and testing sets are usually used for evaluation, with 49,749 and 14,951 samples respectively. As opposed to the other datasets used in this chapter, *w8a* exhibits a significant imbalance in the distribution of class labels, which makes it much more challenging for algorithms which rely on correctly estimating the distribution of data to operate.

Distance preservation on W8a

Tables 4.8 and 4.9 list the results concerning the average distance distortion obtained when transforming 500 samples from the *w8a* dataset with $g = 2$ and $g = 3$ respectively. A value of $t = 30$ was used for KG-RP and PK-RP in all cases.

In this case, both the KG-RP and Nyström methods failed to preserve pairwise distances. This is probably due to the fact that, as opposed to PK-RP, both these methods depend on training data, so the imbalance exhibited by the *w8a* dataset affected them in a negative manner. One more time, $\phi(\cdot)$+RP produced the best results regarding distance preservation, at the cost of large processing times. Finally, the approach proposed in this chapter approximated the distance preservation properties of $\phi(\cdot)$+RP reasonably well, while keeping computational times always below 70ms. Again, D2PK-RP performed poorly, while also being significantly more computationally expensive than PK-RP.

TABLE 4.9: Results on distance preservation from the homogeneous polynomial kernel of degree three ($g = 3$) for 500 samples from w8a.

Method	**Parameters**	**200 output dim.** avg. dist.	time	**500 output dim.** avg. dist.	time	**1000 output dim.** avg. dist.	time
Nyström [103]	-	0.915±0.006	0.025s	0.846±0.009	0.066s	0.773±0.005	0.226s
PK-RP	Gaussian, t=30, p=892	0.135±0.009	0.022s	0.127±0.013	0.039s	0.112±0.006	0.067s
PK-RP	Sparse, s=1, t=30, p=892	0.133±0.008	0.022s	0.114±0.005	0.040s	0.105±0.004	0.063s
PK-RP	Sparse, s=3, t=30, p=892	0.144±0.007	0.021s	0.121±0.006	0.040s	0.111±0.007	0.067s
KG-RP [110]	t=30, p=892	0.802±0.007	6.681s	0.798±0.009	8.239s	0.798±0.007	10.862s
PK-RP	Gaussian, t=30, p=446	0.170±0.009	0.016s	0.161±0.011	0.034s	0.158±0.008	0.061s
PK-RP	Sparse, s=1, t=30, p=446	0.161±0.009	0.016s	0.145±0.004	0.032s	0.140±0.009	0.060s
PK-RP	Sparse, s=3, t=30, p=446	0.171±0.010	0.016s	0.162±0.010	0.033s	0.150±0.009	0.061s
KG-RP [110]	t=30, p=446	0.871±0.007	1.720s	0.867±0.007	2.236s	0.871±0.007	3.031s
PK-RP	Gaussian, t=30, p=223	0.241±0.020	0.013s	0.224±0.026	0.032s	0.219±0.024	0.058s
PK-RP	Sparse, s=1, t=30, p=223	0.213±0.010	0.013s	0.209±0.014	0.029s	0.199±0.009	0.056s
PK-RP	Sparse, s=3, t=30, p=223	0.228±0.012	0.012s	0.221±0.012	0.030s	0.218±0.024	0.059s
KG-RP [110]	t=30, p=223	0.924±0.005	0.487s	0.923±0.007	0.640s	0.924±0.006	0.941s
PK-RP	Gaussian, t=30, p=111	0.320±0.030	0.012s	0.302±0.021	0.028s	0.310±0.028	0.056s
PK-RP	Sparse, s=1, t=30, p=111	0.290±0.020	0.013s	0.284±0.025	0.028s	0.298±0.040	0.057s
PK-RP	Sparse, s=3, t=30, p=111	0.336±0.033	0.011s	0.296±0.016	0.030s	0.294±0.014	0.056s
KG-RP [110]	t=30, p=111	0.959±0.004	0.166s	0.962±0.004	0.257s	0.959±0.004	0.414s

Classification on W8a

Finally, we present the classification results of the different methods on the *w8a* dataset. Again, we experimented with different p and t hyperparameter values. Due to the class imbalance in *w8a*, the raw accuracy is not an appropriate metric to measure the performance of classification methods on this dataset. Instead, we used the F_1-score:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{4.20}$$

We also report the standard deviation in the F_1-score over multiple runs of the experiments, and the computation times required to train each method and transform *w8a*'s entire training set (see Table 4.10). Note that the F_1-score of a linear SVM classifier trained on the original *w8a* samples is 0.7343 (with C=1), which corresponds to a classification accuracy of 98.66%. The classification F_1-score of a degree-2 polynomial-kernel SVM is 0.9020 (with $C = 1$). For comparison, a SVM with the Gaussian kernel achieves a 0.9037 F_1-score ($C = 200$, $\gamma = 0.05$).

As opposed to what happened in the experiments with MNIST and Webspam, the classification performance with Nyström was not the best. Instead, this method was largely outperformed by $\phi(\cdot)$+RP, with the gap between their scores being lower when using 2000 output dimensions. Similarly, KG-RP performed poorly in this dataset, supporting our hypothesis that these methods are largely affected by the imbalance of the dataset.

For its part, PK-RP performed remarkably well in this dataset, achieving F_1-scores similar to those of $\phi(\cdot)$+RP but with a computational-time lower by orders of magnitude. For instance, by using PK-RP with the Gaussian distribution, $k = 2000$, $p = 446$ and $t = 1$, one can achieve a linear classification F_1-score of 0.8973, which is only 0.0018 below the score of $\phi(\cdot)$+RP for the same number of output dimensions. In this case D2PK-RP performed poorly also for classification, in spite of being significantly more computationally expensive than PK-RP.

TABLE 4.10: Classification accuracies on w8a obtained by a linear SVM trained on the representations generated by different methods to approximate the feature space of the degree-2 homogeneous polynomial kernel.

F. Extraction	Parameters	1500 output dim.			2000 output dim.		
		F_1-score	Train	Transform	F_1-score	Train	Transform
Nyström [103]	-	.7849±.015	0.743s	2.117s	.8596±.008	2.127s	3.480s
$\phi(\cdot)$+RP	Gaussian	.8738±.003	3.65s	32m 34s	.8991±.002	4.83s	44m 48s
D2PK-RP [73]	s=1, t=10	.7679±.005	0.068s	8.713s	.8496±.001	0.087s	11.935s
PK-RP	Gaussian, t=10, p=446	.8721±.011	0.004s	2.273s	.9003±.004	0.005s	2.927s
PK-RP	Sparse, s=1, t=10, p=446	.8702±.007	0.004s	2.255s	.8972±.003	0.004s	2.917s
PK-RP	Sparse, s=3, t=10, p=446	.8785±.012	0.004s	2.265s	.8988±.002	0.004s	2.938s
KG-RP [110]	t=10, p=446	.4223±.061	3.815s	1.054s	.4423±.048	4.692s	1.352s
PK-RP	Gaussian, t=1, p=446	.8667±.004	0.004s	0.795s	.8973±.002	0.004s	1.014s
PK-RP	Sparse, s=1, t=1, p=446	.8722±.005	0.003s	0.803s	.8999±.003	0.004s	1.016s
PK-RP	Sparse, s=3, t=1, p=446	.8695±.006	0.003s	0.805s	.8972±.002	0.004s	1.004s
KG-RP [110]	t=1, p=446	.4916±.030	3.819s	1.056s	.4865±.031	4.625s	1.336s
PK-RP	Gaussian, t=10, p=223	.8690±.002	0.003s	2.167s	.8982±.002	0.003s	2.879s
PK-RP	Sparse, s=1, t=10, p=223	.8715±.006	0.002s	2.180s	.8984±.001	0.002s	2.903s
PK-RP	Sparse, s=3, t=10, p=223	.8663±.005	0.002s	2.179s	.9008±.004	0.002s	2.888s
KG-RP [110]	t=10, p=223	.3070±.033	1.208s	0.724s	.3652±.031	1.498s	0.934s
PK-RP	Gaussian, t=1, p=223	.8637±.011	0.002s	0.717s	.8966±.004	0.002s	0.932s
PK-RP	Sparse, s=1, t=1, p=223	.8585±.006	0.002s	0.719s	.8960±.002	0.002s	0.923s
PK-RP	Sparse, s=3, t=1, p=223	.8650±.007	0.002s	0.728s	.8985±.003	0.002s	0.933s
KG-RP [110]	t=1, p=223	.2924±.048	1.210s	0.727s	.3337±.045	1.481s	0.949s
PK-RP	Gaussian, t=10, p=111	.8656±.006	0.002s	2.126s	.8978±.002	0.002s	2.828s
PK-RP	Sparse, s=1, t=10, p=111	.8622±.006	0.002s	2.133s	.8986±.003	0.002s	2.838s
PK-RP	Sparse, s=3, t=10, p=111	.8667±.011	0.002s	2.128s	.8983±.003	0.002s	2.823s
KG-RP [110]	t=10, p=111	.1242±.063	0.539s	0.566s	.1757±.044	0.679s	0.744s
PK-RP	Gaussian, t=1, p=111	.8434±.014	0.001s	0.670s	.8884±.005	0.001s	0.883s
PK-RP	Sparse, s=1, t=1, p=111	.8375±.009	0.002s	0.664s	.8863±.004	0.002s	0.888s
PK-RP	Sparse, s=3, t=1, p=111	.8361±.009	0.002s	0.664s	.8874±.004	0.001s	0.882s
KG-RP [110]	t=1, p=111	.1572±.037	0.542s	0.555s	.1542±.073	0.665s	0.746s

4.4.4 Polynomial kernel degree selection

As mentioned before, due to their function, increasing the hyperparameters t and p for our method will likely result in higher accuracies at the expense of a greater computational cost. However, determining the best polynomial degree is not that simple. While in some contexts the best performing polynomial kernel degree is known based on expert knowledge, experimentation is usually needed to determine the best value for this hyperparameter. In this section, we show how the right polynomial degree for PK-RP can be determined by using a standard hyperparameter selection strategy. In particular, given a desired output dimension and the value of p and t, the most appropriate kernel degree can be determined by performing a Cross-Validation on the training set with different polynomial kernel degrees. Then, the best performing value of g according to the Cross-Validation accuracies is selected and evaluated on the test set.

For the experiments in this section, we used the binary version of the Covertype dataset [29]. The task with this dataset is to predict the forest cover-type from cartographic variables (e.g., elevation, slope, soil type, etc.). In particular, we used the pre-processed version of the dataset available at the LIBSVM web page[11]. It contains a total of 581,012 samples each of dimension 54. Since no predefined train/test split exists for this dataset, for our experiments we randomly sampled 20% of the data to

[11] www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets (Date accessed: 03/02/2018).

TABLE 4.11: Classification accuracies for 5-fold Cross-Validation on the training set and for the test set of Covertype. The results show that, given an output dimension and the the desired value of p, the most suitable polynomial degree g for PK-RP can be selected by using a standard hyperparameter selection strategy.

Parameters	**1000 output dim.**		**2000 output dim.**		**3000 output dim.**	
	CV Acc. (%)	Test Acc. (%)	CV Acc. (%)	Test Acc. (%)	CV Acc. (%)	Test Acc.
g=2, p=500	79.21±0.28	79.55±0.03	79.33±0.25	79.56±0.03	79.37±0.21	79.57±0.03
g=3, p=500	78.27±0.25	79.17±0.26	79.85±0.34	80.45±0.15	80.45±0.17	81.17±0.09
g=4, p=500	69.34±0.71	70.39±0.51	72.08±0.23	73.10±0.25	73.64±0.70	74.25±0.42
g=2, p=1000	79.26±0.22	79.54±0.04	79.34±0.32	79.55±0.02	79.32±0.33	79.55±0.03
g=3, p=1000	78.51±0.14	78.93±0.28	79.82±0.11	80.63±0.13	80.34±0.16	81.13±0.11
g=4, p=1000	69.83±0.75	69.85±0.62	72.22±0.78	72.92±0.23	73.69±0.25	74.29±0.28

form the test set (116,202 samples) and 10% to form the training set (58,101 samples). Table 4.11 shows the 5-fold Cross-Validation accuracies and the corresponding test accuracies for different polynomial degrees, values of p and output dimensions on the Covertype dataset. Hyperparameter t as fixed to 10. Looking at the table we can se that, for each output dimension and selected p combination, the best performing polynomial degree in the Cross-Validation process over the training set matches the best performing kernel as evaluated on the test set. This suggests that the most appropriate kernel degree for a specific application can be successfully determined with the above described hyperparameter selection scheme. In addition, it must be noted that the best performing polynomial degree for PK-RP need not be the same as the best polynomial degree for a conventional kernel-SVM using a polynomial kernel. Since our method is implicitly performing a Random Projection from the kernel feature space, higher polynomial degrees might require a bigger output dimension to fully capture their discriminative information, as the dimension of the implicit kernel feature space grows with the degree. Therefore, the optimal polynomial degree to be used with our method depends on the selected output dimension. For instance, we can see that in this case our method performed best using $g = 3$ only when the output dimension was at least 2000.

4.4.5 Repetition minimization

Before concluding this chapter, we present a small modification of the proposed algorithm which slightly improves the performance with little to no cost. As explained before, our algorithm generates a set S containing p i.i.d. random vectors. Afterwards, k random subsets of S are selected, each containing the random vectors that will be used for one of the output components. This enables us to pre-compute the inner product of data samples with the p vectors in S, reducing the total number of d-dimensional inner products evaluated by our algorithm from gtk to p. However, this approach results in an uneven usage of the vectors in S. That is, after selecting the k random subsets $S_1, \ldots, S_k$ from S, some of the original vectors might be present in more of the subsets than others. In fact, it is even possible that a given vector in S does not appear in any of the random subsets, in which case we would be wasting computational resources for nothing. To solve this, we introduce an alternative procedure to generate the collections of vectors used for each output

a) Without repetition minimization

b) With repetition minimization

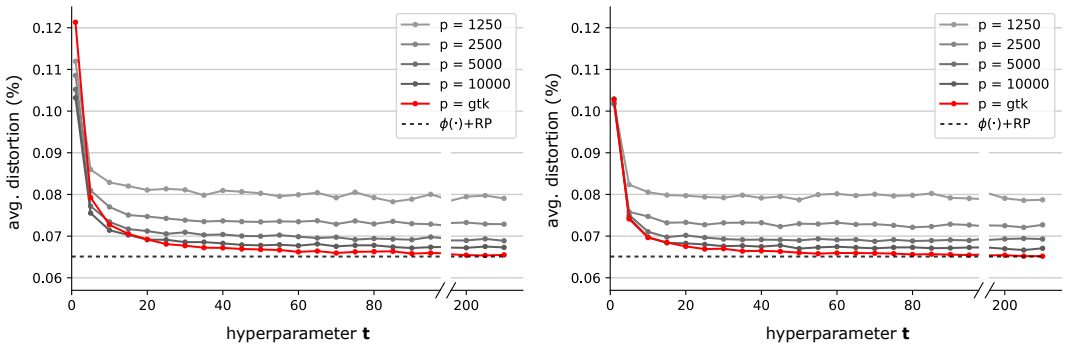


FIGURE 4.6: Comparison of the proposed method for a fixed output dimension of $k = 300$, $g = 2$, different values of t and p, and with/without repetition minimization. Average distortion is evaluated for pairwise distances among 500 MNIST test samples.

component. The goal here will be to achieve a more even usage of the p vectors in S. First, we generate a collection P containing the elements of S repeated the necessary number of times to ensure $|P| = gtk$:

$$P = \underbrace{S \cup \cdots \cup S}_{\lfloor gtk/p \rfloor} \cup S[1 : \text{gtk mod p}]. \tag{4.21}$$

Then, we sample P without replacement to form k collections $S_1, \ldots, S_k$, each with gt vectors. The gt vectors in collection S_c are as usual used in the computation of the c-th output component. In addition to ensuring that vectors in S are more evenly used, this method guarantees that if we select p to be equal to gtk, no vector repetition will occur, so we will no longer be violating the theoretical requirement of independence among the output components of the Random Projection. We will refer to this alternative approach to populate $S_1, \ldots, S_k$ as *repetition minimization*. Figure 4.6 compares the performance of our algorithm with and without repetition minimization, measuring the average distortion of pairwise distances among 500 test samples from MNIST. As expected, small improvements were registered, especially for lower values of t. This trick will be further exploited in the following chapter.

4.5 Conclusions and future work

This chapter introduced a novel method to approximate Random Projections from the feature spaces of homogeneous polynomial kernels. As opposed to previous kernelization attempts of the Random Projection algorithm [110, 111], our approach preserves the data-independence and low computational complexity of the original Random Projection method. As a drawback, this was achieved by sacrificing the generality of the method, focusing on a specific kernel family. Nevertheless, the chosen kernel family, homogeneous polynomial kernels, is one of the most popular choices and has been successfully applied to a wide range of classification and clustering

problems. In addition, our method is compatible with homogeneous polynomial kernels of arbitrary degree, and can also be used to approximate Random Projections from the feature spaces of inhomogeneous polynomial kernels, via the trick described in section 4.3.3.

Our work is closely related to the theoretical work of Balcan *et al.* [9, 11], who demonstrated that it is in general not possible to perform a valid Random Projection from the feature space of an arbitrary kernel, given only black-box access to the kernel function and without access to the distribution of data. However, they hypothesized that such methods could be developed for specific natural kernel families. The proposed method confirms their hypothesis, since it can approximate a Random Projection from the feature space of the homogeneous polynomial kernel, without ever computing the explicit form of the feature space or considering the distribution of data samples being processed.

Our theoretical analysis of computational complexities showed that the time required by the proposed approach grows linearly with respect to the dimensionality of samples and the desired output dimension. Also, the training time of PK-RP is independent of the number of training samples as opposed to KG-RP, which requires $\mathcal{O}(p^3)$ training time where p must be set considering the number of available training samples[12]. Our method also compares favorably to the Nyström algorithm, whose training and testing times are $\mathcal{O}(k^3)$ and $\mathcal{O}(k^2)$ respectively. Our theoretical analysis regarding the time complexities of the different methods is supported by the experimental measurements, where our approach consistently resulted in the lowest execution times.

The experimental results presented in Section 4.4 evidence the performance of the proposed method both in terms of distance preservation and generation of useful representations for linear classification. Regarding distance preservation, the proposed approach outperformed alternative methods in most of our experiments. In terms of classification accuracy, PK-RP showed its ability to approximate the results obtained with the explicit $\phi(\cdot)$+RP approach, while being orders of magnitude faster.

Apart from the above mentioned advantages of the proposed method, it is also worth noticing that it works in a completely data-independent manner. That is, the algorithm can be initialized without access to any training sample, and the properties of the algorithm do not depend on estimating the distribution of input data. As a consequence, our method is well suited to work in online/incremental learning scenarios [38], where data samples arrive in a sequential manner. Lastly, we showed that the kernelization approach proposed in this chapter is directly compatible with the database-friendly distribution proposed by Achlioptas [2]. This property can be used to ease the implementation of this algorithm in SQL environments, as the projection of data samples over the random vectors can be done in terms of aggregate

[12] In the case of KG-RP, the hyperparameter p controls the number of training samples used by the underlying Kulis-Grauman method to estimate the distribution of data. As a consequence, the authors of this method [65] recommended setting $p = \mathcal{O}(\sqrt{N})$, where N is the number of training samples available.

evaluation. In this regard, our experimental results show no significant accuracy loss when using Achlioptas' distribution with $s = 1, 3$ instead of the Gaussian distribution. Using greater levels of sparsity in the projection vectors could be explored as a way of reducing computational costs even more.

As for future lines of research, we propose exploring the development of similar kernelized variants of Random Projection for other kernel families. While in this chapter we have focused on the polynomial kernel family, it would be interesting to compare kernel feature space approximation methods for different kernel families. In addition, we intend to investigate the applicability of the proposed approach in different clustering, classification and information retrieval tasks, especially in domains where a limited computational power is available.

Chapter 5

Compact Bilinear Pooling via Kernelized Random Projection

Bilinear convolutional neural networks, which include the bilinear pooling operation as their key feature, are among the most popular and effective models for fine-grained image recognition. However, a major drawback of models using bilinear pooling is the dimensionality of the resulting descriptors, which typically consist of several hundred thousand features. Even when generating the descriptor itself is feasible, its dimension renders any subsequent operations impractical and often results in huge computational and storage costs. In this chapter, we introduce a novel method to efficiently reduce the dimension of the bilinear pooling descriptor by performing a Random Projection. Conveniently, this is achieved without ever computing the high-dimensional descriptor explicitly. Experimental results show that our technique outperforms existing compact bilinear pooling algorithms in most cases. In addition, it runs faster that alternative methods on low computational power devices, where efficient extensions of bilinear pooling are most useful.

The contents of this chapter have been adapted from the journal paper: Daniel López-Sánchez, Angélica González Arrieta and Juan M. Corchado. "Compact Bilinear Pooling via Kernelized Random Projection for Fine-Grained Image Categorization on Low Computational Power Devices". In: Neurocomputing (In press).

5.1 Introduction

The term fine-grained recognition is generally applied to describe classification tasks with a relatively large number of very similar categories. Examples of this include animal and plant species classification [99, 59, 78], automobile and plane model identification [61, 74], or scene recognition [112], among others. These classification tasks tend to be quite challenging, partly because of the high intra-class variability they exhibit, combined with a low inter-class variability. In other words, the small variations that contain the information needed to distinguish between classes

can be easily overwhelmed by non-informative factors such as pose, orientation or illumination conditions.

In the recent years, many different approaches have been proposed to address the challenge of fine-grained recognition, and accuracies have risen steadily [16, 105, 62]. One of the most effective and widely adopted approaches is the use of bilinear Convolutional Neural Networks (CNN) [26, 37, 95, 6, 91], originally proposed by Lin *et al.* [71]. As explained in Section 2.5, bilinear CNNs build an image feature descriptor by first applying two CNNs as feature extractors. Then, the two descriptors generated are combined at each location by using the outer product. Finally, the resulting descriptor is pooled across locations to obtain a global descriptor of the input image. A classic linear classifier (e.g., softmax, logistic regression, etc.) is then applied on the global descriptor. This approach enables bilinear CNNs to capture pairwise feature interactions in a location-invariant manner, which in turn produces a boost in fine-grained classification accuracies.

In spite of their success and popularity, models using bilinear pooling have a major drawback. As a consequence of using the outer product, the generated descriptor is extremely high-dimensional. For instance, the bilinear descriptor used in [71] had more than $250,000$ features. As a consequence, even a simple linear classifier will have millions of parameters when trained on such a high-dimensional descriptor, or even hundreds of millions if the number of classes is large. This results in high computation and storage overheads, and makes machine learning models more prone to over-fitting.

To mitigate this problem, methods which try to compress the discriminative information of the bilinear descriptor into low-dimensional representations have been developed. Most notably, Gao *et al.* proposed an approach called compact bilinear pooling [43], which uses polynomial kernel approximation techniques to achieve this. In addition, the authors of [43] also discussed the possibility of using the Random Projection algorithm [3] to reduce the dimension of the bilinear descriptor, but discarded this option after noting that such an approach would require storing a huge projection matrix and explicitly computing the bilinear descriptor before the projection.

In this chapter, we further develop the idea of using Random Projection to reduce the dimension of the bilinear pooling descriptor. In particular, we propose adapting the kernelized variant of Random Projection presented in the previous chapter to efficiently project bilinear descriptors to a lower dimension, without ever having to explicitly compute the high-dimensional bilinear descriptors themselves. We also derive back-propagation for our algorithm, so that it can be included as a building block in end-to-end trainable models. As a practical application of the proposed approach, we study the task of fine-grained image classification on low computational power devices. We focus on this application scenario because, as pointed out by Gao *et al.* [43], methods for making bilinear pooling more efficient are most useful for low power devices, where computational resources are scarce. Our experimental results

suggest that the proposed algorithm generates a better compacted representation of the bilinear descriptor in most cases, while being significantly faster than alternative approaches.

5.2 Related work

Bilinear models were originally proposed in [94], where the authors used them to separately model the style and content of images. More recently, Lin *et al.* [71] explored their applicability in the context of deep learning for fine-grained image categorization, showing that bilinear Convolutional Neural Networks could be used to achieve state-of-the-art results in various fine-grained image categorization datasets.

In [43], the authors applied two polynomial kernel approximation techniques to make bilinear CNNs less computationally demanding, especially in terms of the memory required for parameter storage. This approach emerged from the notion that bilinear features are fundamentally related to the feature space of the homogeneous polynomial kernel of degree two, so kernel approximation methods can also be used to approximate bilinear pooling descriptors. This approach is known as compact bilinear pooling, since it reduces the dimension of the bilinear descriptor proposed in [71]. In addition, back-propagation was derived for both methods in [43], making the proposed models end-to-end trainable.

The first kernel approximation technique applied in [43] was Random Maclaurin (RM) [58]. In essence, Random Maclaurin builds a randomized feature map which, when approximating the degree-two homogeneous polynomial kernel, takes the form $Z : \mathbb{R}^d \rightarrow \mathbb{R}$, $Z : \mathbf{x} \rightarrow \langle \mathbf{x}, \mathbf{w}_1 \rangle \langle \mathbf{x}, \mathbf{w}_2 \rangle$ where $\mathbf{x} \in \mathbb{R}^d$ is the input data sample and $\mathbf{w_1}, \mathbf{w_2} \in \mathbb{R}^d$ are i.i.d. random Rademacher vectors. Conveniently, for two arbitrary data samples $\mathbf{x}$ and $\mathbf{y}$, it can be proven that $\mathbb{E}[Z(\mathbf{x})Z(\mathbf{y})] = \langle \mathbf{x}, \mathbf{y} \rangle^2$. That is, the randomized feature map preserves inner products from the kernel feature space in expectation. Of course, the quality of this feature map can be improved by using more than one entry in the output representation, thus reducing the variance of the estimator. While this approach performed well in the experiments of [43], it has the inherent limitation of requiring a significant amount of memory to store the Rademacher vectors used for the map.

The second kernel approximation method used in [43] was Tensor Sketch (TS) [82]. Introduced a few years later than Random Maclaurin, Tensor Sketch obtains a Count Sketch [22] of the outer product of two vectors in an efficient manner, which can be used to approximate polynomial kernels and in turn the bilinear descriptor. In particular, instead of explicitly computing the outer product, TS computes the Count Sketch of the vectors and then uses polynomial multiplication via the Fast Fourier Transform to compute the Count Sketch of their outer product. Using this method to achieve a compact bilinear pooling typically results in higher accuracies, while requiring much less memory for parameter storage than RM.

As mentioned above, Gao *et al.* [43] also suggested the possibility of using the Random Projection algorithm [3] to reduce the dimensionality of bilinear descriptors. Thanks to the Johnson-Lindenstrauss lemma [33] that underpins Random Projection, pairwise distances between bilinear descriptors would be approximately preserved in the resulting representation. However, they discarded this idea because directly applying a Random Projection to the bilinear descriptors would require storing a large projection matrix and explicitly computing the bilinear descriptors in the first place. However, recent advances in the intersection of kernel functions and Random Projections [110, 5, 73, 72] have made it possible to efficiently perform Random Projections from the feature spaces of different kernel functions in an efficient manner. In particular, an efficient method to approximate Random Projections from the feature spaces of polynomial kernels was introduced in the previous chapter (also see [72]). Here we adapt the ideas presented in Chapter 4 to make bilinear pooling less computationally demanding by approximating a Random Projection of the bilinear descriptor.

5.3 Proposed method

Bilinear pooling [71] computes a global descriptor for an image $\mathcal{I}$ by computing the outer product of local descriptors and then applying average pooling over locations. In the context of this chapter, the local descriptors are generated by means of an arbitrary CNN (see Figure 5.1). Formally, the global bilinear descriptor is:

$$\Phi(\mathcal{I}) = \sum_{l \in \mathcal{L}} \text{CNN}(\mathcal{I}, l) \otimes \text{CNN}(\mathcal{I}, l), \tag{5.1}$$

where $\text{CNN}(\mathcal{I}, l)$ denotes the descriptor extracted from image $\mathcal{I}$ at location l by the chosen CNN[1], $\mathcal{L}$ is the set of existing locations and $\otimes$ denotes the Kronecker product[2]. For instance, if the CNN generates feature maps of dimension $H \times W$ with d channels, there will be HW locations in $\mathcal{L}$, and each local descriptor $\text{CNN}(\mathcal{I}, l)$ will be of size d. As a consequence, the final bilinear descriptor $\Phi(\mathcal{I})$ will be of dimension d^2, which is the main cause of the inefficiency of this approach. The descriptor is typically normalized by first applying an element-wise signed square root operation (i.e., $x \to \text{sgn}(x)\sqrt{|x|}$), followed by L2 normalization.

One possible approach to mitigate the issue of the high dimensionality of the bilinear descriptor is to perform a Random Projection to reduce its size. In practice, explicitly performing this Random Projection would involve multiplying the bilinear descriptor $\Phi(\mathcal{I}) \in \mathbb{R}^{d^2}$ by a projection matrix $R \in \mathbb{R}^{d^2 \times k}$ whose entries are independently drawn from a suitable distribution [3]. Formally, a Random Projection of the

[1] Note that, like in [43], we focus on the case were the same feature-extraction CNN is used in both sides of the Kronecker/outer product.

[2] We use the Kronecker product rather than the outer product to characterize bilinear pooling for the sake of consistency with the notation in [72], but these operations are in this case equivalent when their output is reshaped.

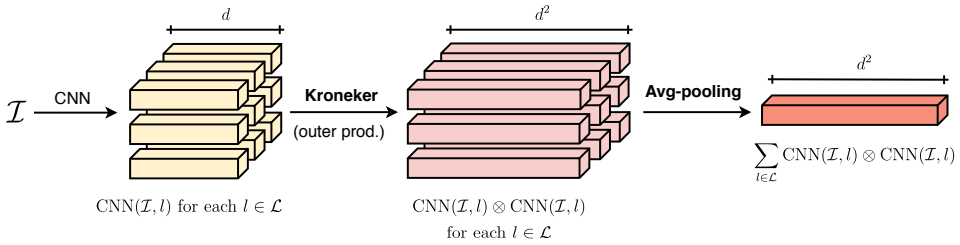


FIGURE 5.1: Schematic view of Bilinear Pooling [71] for an input image $\mathcal{I}$ and a Convolutional Neural Network (CNN) which produces an output feature map with d channels. First, the Kronecker product is applied at each location of the feature maps generated by the CNN. Then, the resulting bilinear descriptors are averaged to form the final global bilinear descriptor.

bilinear descriptor would be:

$$\frac{1}{\sqrt{k}}\Phi(\mathcal{I})R = \frac{1}{\sqrt{k}}\left(\sum_{l\in\mathcal{L}} \mathrm{CNN}(\mathcal{I}, l) \otimes \mathrm{CNN}(\mathcal{I}, l)\right) R, \tag{5.2}$$

which results in a k-dimensional descriptor. Intuitively, we can think of each output feature from this operation as the projection (inner product) of the bilinear descriptor onto one of the columns of the projection matrix. As discussed in Section 2.2, several options have been proposed over the years for the distribution of the entries of R. Originally, the standard normal distribution was used [52, 8]. Later on, studies demonstrated that projection matrices could be drawn from much simpler distributions. Notably, Achlioptas showed that the entries of the projection matrix can be instead drawn from a discrete and sparse distribution [3]. In particular, Achlioptas' work proved that if the entries of R are drawn from the distribution shown in (5.3) with sparsity term $s = 1$ or $s = 3$, then the result will be a valid Random Projection.

$$R_{ij} = \sqrt{s}\begin{cases} 1 & \text{with prob.} \quad 1/2s \\ 0 & \text{with prob.} \quad 1 - 1/s \\ -1 & \text{with prob.} \quad 1/2s \end{cases}. \tag{5.3}$$

A crucial point, however, is that the entries of the projection matrix must be chosen independently, as we have seen in the previous chapters.

It is worth to keep in mind that using the distribution proposed by Achlioptas reduces the computational cost of the projection. For instance, when using $s = 3$, only $1/3$ of the entries of the projection matrix are nonzero. Moreover, it has been suggested that using greater sparsity levels in (5.3) is possible with little loss in accuracy. In particular, some studies recommend using $s = \mathcal{O}(\sqrt{d})$, where d is the dimension of data samples [70].

However, as pointed out by Gao *et al.* [43], even if a sparse projection matrix is used, the d^2-dimensional bilinear descriptor needs to be computed before performing the projection in (5.2), incurring in much of the inefficiency of standard bilinear pooling. Luckily, various methods have been recently introduced to efficiently perform

Random Projections from kernel feature spaces. In particular, the kernelized algorithm proposed in Chapter 4 can be used to approximate Random Projections from the feature spaces of homogeneous polynomial kernels. Moreover, the same ideas can be used to apply Random Projection to bilinear descriptors in an efficient manner. To show this, we begin by examining the following property of the Kronecker product. Let $\mathbf{r}_1, \mathbf{r}_2$ and $\mathbf{x}$ be three arbitrary vectors from $\mathbb{R}^d$. Then the following equality holds:

$$\langle \mathbf{x}, \mathbf{r}_1 \rangle \langle \mathbf{x}, \mathbf{r}_2 \rangle = \langle \mathbf{x} \otimes \mathbf{x},\ \mathbf{r}_1 \otimes \mathbf{r}_2 \rangle. \tag{5.4}$$

A more general version of this equality was proved and used in Chapter 4 to perform operations in the feature space of homogeneous polynomial kernel without ever evaluating it explicitly. Note that $\phi(\cdot) : \mathbf{x} \to \mathbf{x} \otimes \mathbf{x}$ is a valid feature map for the homogeneous polynomial kernel of degree two, so the inner product in the right-hand side of the above equation can be thought as taking place in the feature space of that kernel. Conveniently, the inner products in the left-hand side of the equation are in $\mathbb{R}^d$, which enables us to evaluate the expression in an efficient manner.

At this point, one might attempt to exploit (5.4) to perform a Random Projection of $\mathbf{x} \otimes \mathbf{x}$, as a first step towards our goal of projecting the bilinear descriptor. To achieve this, $\mathbf{r}_1$ and $\mathbf{r}_2$ should be chosen in such a way that the entries of $\mathbf{r}_1 \otimes \mathbf{r}_2$ follow one of the valid Random Projection distributions reviewed in Section 2.1, so $\mathbf{r}_1 \otimes \mathbf{r}_2$ can play the role of one of the columns of the projection matrix. For instance, if we draw the entries of $\mathbf{r}_1$ and $\mathbf{r}_2$ according to (5.3) with $s = 1$, then the entries of $\mathbf{r}_1 \otimes \mathbf{r}_2$ will appear to also follow this distribution when analyzed individually. However, the entries of $\mathbf{r}_1 \otimes \mathbf{r}_2$ are not mutually independent, which as mentioned before is a crucial requirement for achieving a valid Random Projection.

As shown in Chapter 4, one possible solution to this problem is to apply the multidimensional Central Limit Theorem (CLT) [17]. This classical result states that the sum of t i.i.d. random vectors with zero means and $\mathbf{\Sigma}$ covariance, scaled by $1/\sqrt{t}$, converges in distribution to a multivariate normal with zero means and $\mathbf{\Sigma}$ covariance as t goes to infinity. As a consequence, given $2t$ i.i.d. zero-mean random vectors $\mathbf{r}_1, \ldots, \mathbf{r}_{2t}$, we can ensure that

$$\sum_{i=0}^{t-1} \left(\frac{\mathbf{r}_{2i+1} \otimes \mathbf{r}_{2i+2}}{\sqrt{t}} \right) \tag{5.5}$$

converges in distribution to a multidimensional normal distribution with zero means. Moreover, if vectors we are summing have identity covariance matrix, then (5.5) converges in distribution to a multidimensional normal with zero means and identity covariance, which is one of the valid distributions for the Random Projection matrix[3] [8]. Conveniently, the desired identity covariance for vectors in the summation

[3]Note that drawing the columns of the projection matrix from a multidimensional normal with zero means and identity covariance is equivalent to independently drawing the individual entries from the unidimensional standard normal, as done in [8].

of (5.5) can be achieved by independently drawing the entries of $\mathbf{r}_1, \ldots, \mathbf{r}_{2t}$ from Achlioptas' distribution, displayed in (5.3). Note that, by definition, the individual variables in a multidimensional normal with identity covariance are independent [50], so the dependence among the entries of vectors formed following (5.5) gradually vanishes as t grows.

Therefore, if we use projection vectors generated as in (5.5), with $\mathbf{r}_1, \ldots, \mathbf{r}_{2t} \in \mathbb{R}^d$ populated according to (5.3), then for a sufficiently large t we will be performing a valid Random Projection. Formally, each component $\mathbf{y}_c$ of the output representation will be:

$$\mathbf{y}_c = \frac{1}{\sqrt{k}} \left\langle \Phi(\mathcal{I}),\ \sum_{i=0}^{t-1} \left(\frac{\mathbf{r}_{2i+1} \otimes \mathbf{r}_{2i+2}}{\sqrt{t}} \right) \right\rangle_{\mathbb{R}^{d^2}} . \tag{5.6}$$

As shown in Chapter 4, even if the selected value for t is not big enough to make the entries of the resulting projection vectors follow a perfect normal distribution, the summation in (5.5) has the effect of reducing the statistical dependence among the entries of the projection vectors, resulting in a better approximation of a valid Random Projection.

However, directly using (5.6) to compute the Random Projection of the bilinear descriptor involves explicitly generating the descriptor and the projection vectors, resulting in the same inefficiencies as directly applying a classic Random Projection. Luckily, the inner product between the bilinear descriptor and our projection vector can be conveniently rewritten to avoid working in the d^2-dimensional space. This is achieved by using (5.4) along with some elementary properties of inner products:

$$\begin{aligned}
\mathbf{y}_c &= \frac{1}{\sqrt{k}} \left\langle \Phi(\mathcal{I}),\ \sum_{i=0}^{t-1} \left(\frac{\mathbf{r}_{2i+1} \otimes \mathbf{r}_{2i+2}}{\sqrt{t}} \right) \right\rangle \\
&= \frac{1}{\sqrt{tk}} \sum_{i=0}^{t-1} \langle \Phi(\mathcal{I}),\ \ \mathbf{r}_{2i+1} \otimes \mathbf{r}_{2i+2} \rangle \\
&= \frac{1}{\sqrt{tk}} \sum_{l \in \mathcal{L}} \sum_{i=0}^{t-1} \langle \mathrm{CNN}(\mathcal{I}, l) \otimes \mathrm{CNN}(\mathcal{I}, l),\ \mathbf{r}_{2i+1} \otimes \mathbf{r}_{2i+2} \rangle \\
&= \frac{1}{\sqrt{tk}} \sum_{l \in \mathcal{L}} \sum_{i=0}^{t-1} \langle \mathrm{CNN}(\mathcal{I}, l), \mathbf{r}_{2i+1} \rangle \langle \mathrm{CNN}(\mathcal{I}, l), \mathbf{r}_{2i+2} \rangle .
\end{aligned} \tag{5.7}$$

Conveniently, the inner products appearing in the last expression are in $\mathbb{R}^d$, avoiding the explicit generation of the bilinear descriptor and the d^2-dimensional projection vectors. The complete output representation generated by our algorithm is obtained by repeating this projection k times, each with a different set of random vectors $\mathbf{r}_1, \ldots, \mathbf{r}_{2t}$:

$$\mathbf{y} = [\mathbf{y}_1, \ldots, \mathbf{y}_k] . \tag{5.8}$$

Algorithm 3 Compact Bilinear Pooling via Kernelized Random Projection

Require: Descriptors for some image $\mathcal{I}$ at each location: $\text{CNN}(\mathcal{I}, l)$ for $l \in \mathcal{L}$. The total number of vectors p and their sparsity level s, the number t of vectors used for the Central Limit Theorem and the desired output dimension k.

Ensure: Returns a k-dimensional vector which approximates a Random Projection of the full bilinear pooling descriptor.

$S \leftarrow \{\mathbf{r}_1, \dots, \mathbf{r}_p\}$ where each entry of $\mathbf{r}_i \in \mathbb{R}^d$ is $\{-\sqrt{s}, 0, \sqrt{s}\}$ w.p. $\{\frac{1}{2s}, 1 - \frac{1}{s}, \frac{1}{2s}\}$ ▷ Generate vectors

$P = \underbrace{S \cup \cdots \cup S}_{\lfloor 2tk/p \rfloor} \cup S[\,1 : 2\text{tk mod p}]$, so that $|P| = 2tk$ ▷ Generate a redundant collection

Sample P w/o replacement to form $S_1, \dots, S_k$, where $|S_i| = 2t$ ▷ Form k collections of $2t$ vectors

$\mathbf{y} \leftarrow (0, \dots, 0) \in \mathbb{R}^k$ ▷ Initialize output vector

for $l \in \mathcal{L}$ **do** ▷ Iterate over locations

  for $c = 1, \dots, k$ **do** ▷ Iterate over output components

    for $i = 0, \dots, t-1$ **do**

      $\mathbf{y}[c] \leftarrow \mathbf{y}[c] + \frac{1}{\sqrt{t}} \langle \text{CNN}(\mathcal{I}, l), S_c[2\text{i}+1] \rangle \cdot \langle \text{CNN}(\mathcal{I}, l), S_c[2\text{i}+2] \rangle$ ▷ Apply equation (5.7)

$\mathbf{y} \leftarrow \frac{1}{\sqrt{k}} \cdot \mathbf{y}$ ▷ Scale to compensate for the dimensionality reduction

return $\mathbf{y}$

Regarding the selection of the hyperparameter t, the results in Chapter 4 suggest that while relatively high values of t are required for a good pairwise-distance preservation after the projection, classification accuracies do not benefit much from using values of t greater than two. In fact, the results presented there seem to suggest that it is a good idea to use small values of t in classification scenarios in order to reduce computational cost.

5.3.1 Reusing vectors for improved efficiency

Up to this point, we have assumed that each of the output components $\mathbf{y}_c$ of the representation generated by our algorithm uses a completely different set of random vectors $\mathbf{r}_1, \dots, \mathbf{r}_{2t}$. This ensures that the projection vectors generated using (5.5) for different output components are independent of each other, which is required to achieve a valid Random Projection. Unfortunately, this also forces us to maintain a total of $2tk$ d-dimensional vectors in memory, which in some cases can be challenging. However, as shown in Chapter 4, this requirement can be relaxed in practice. In particular, instead of using $2tk$ different vectors, one can generate a set $S = \{\mathbf{r}_1, \dots, \mathbf{r}_p\}$ containing p i.i.d. vectors, and then use a random subset $S_c \subset S$ for each output component. This approach produced good results in practice, while enabling substantial computational savings [72].

A similar approach is taken in this chapter. First, a set containing p i.i.d. random vectors $S = \{\mathbf{r}_1, \dots, \mathbf{r}_p\}$ is generated with the entries of each vector following the distribution defined in (5.3) and $2t < p \leq 2tk$. Then, $2t$ of those vectors are selected for each output component $\mathbf{y}_1, \dots, \mathbf{y}_k$. However, rather than simply selecting k random subsets of S as done in Chapter 4, we make sure that each individual vector is used the lowest number of times possible. In contrast, randomly selecting k subsets of S would result in some vectors being used more often than others, as discussed in Section 4.4.5. To achieve a more even usage of the vectors in S, we first generate a

collection P containing the elements of S repeated the necessary number of times to ensure $|P| = 2tk$:

$$P = \underbrace{S \cup \cdots \cup S}_{\lfloor 2tk/p \rfloor} \cup S[1 : 2\text{tk mod p}]. \tag{5.9}$$

Afterwards, we sample P without replacement to form k collections $S_1, \ldots, S_k$, each with $2t$ vectors. The $2t$ vectors in collection S_c are then used in the computation of the output component $\mathbf{y}_c$, using (5.7).

In practice, $S_1, \ldots, S_k$ store references to the original vectors is S rather than copies of them, so no extra memory needs to be allocated. Algorithm 3 provides a self-contained high-level description of the proposed method. A lower-level description of the algorithm is provided in Appendix A. Throughout the following sections, we will refer to this algorithm as Compact Bilinear Pooling via Kernelized Random Projection (CBP-KRP).

5.3.2 Computational complexity and implementation tricks

Analyzing the different steps in Algorithm 3, it is possible to determine both the time complexity and storage requirements of the proposed method. Steps 1-3 correspond to the instantiation of the algorithm, and contain the initialization of the parameters of the model. Most of the memory cost comes from storing S, which contains p vectors of dimension d. Luckily, these vectors are drawn from Achlioptas' sparse distribution, so using an appropriate sparse matrix implementation the zero-valued entries need not be stored. Therefore, only $\mathcal{O}(dp/s)$ parameters need to be stored to represent S.

Regarding the collections P and $S_1, \ldots, S_k$, as mentioned before, they can be implemented in such a way that they only store references to the original vectors in S, so the memory requirements are reduced significantly. In addition, note that P is only temporarily used to form $S_1, \ldots, S_k$. In total, $S_1, \ldots, S_k$ contain $2tk$ references[4] that need to be stored after the initialization of the algorithm, together with the set of vectors S. Therefore, the complete model requires storing $\mathcal{O}(dp/s) + 2tk$ parameters.

To assess the computational complexity, we separately consider the initialization phase (steps 1-3) and the projection of the bilinear descriptor (steps 4-10). Regarding the initialization, the computational cost is $\mathcal{O}(dp + tk)$, where the $\mathcal{O}(dp)$ comes from forming S and the $\mathcal{O}(tk)$ from the sampling of P to form $S_1, \ldots, S_k$. In practice, these initialization steps only have to be executed once and require a time in the order of seconds at most.

For the projection of the bilinear descriptor (steps 4-10), a more detailed analysis is required. As we can see, these steps consist of a series of nested loops that, at the innermost operation, perform two inner products between d-dimensional vectors.

[4] Depending on the implementation, these references can take the form of integer indexes, memory pointers, etc. In any case, storing one of these references has a similar memory cost as storing a floating point parameter.

TABLE 5.1: Comparison of descriptor dimension, memory usage and time complexity for the different approaches and networks considered in this chapter. Variables d, L and C represent the number of channels before the pooling operation, the number of locations at which the CNN is applied (i.e., height times width of the feature maps), and the number of classes respectively. Hyperparameter k corresponds to the desired output dimension for CBP-KRP, TS and RM. Hyperparameters t, p and s control the behavior of CBP-KRP. Numeric results are for $C = 200$, $k = 5000$, p=5000, $t = 2$ and $s = 100$, using *float32* precision.

		FB	**CBP-KRP**	**TS [43]**	**RM [43]**
Theoretical	Descriptor Size	d^2	k	k	k
	Parameters	0	$\mathcal{O}(dp/s) + 2tk$	$4d$	$2dk$
	Classifier Param.	Cd^2	Ck	Ck	Ck
	Computation	$\mathcal{O}(Ld^2)$	$\mathcal{O}(L(pd + tk))$	$\mathcal{O}(L(d + k \log k))$	$\mathcal{O}(Ldk)$
SqueezeNet [51]	Descriptor Size	**262,144**	5,000	5,000	5,000
Network size: 4.8 MB	Parameters	0 B	280 KB	8 KB	**19.5 MB**
@ *fire9* ($13 \times 13 \times 512$)	Classifier Param.	**200 MB**	3.8 MB	3.8 MB	3.8 MB
GoogLeNet [92]	Descriptor Size	**692,224**	5,000	5,000	5,000
Network size: 25.7 MB	Parameters	0 B	406 KB	13 KB	**31.7 MB**
@ *incept-4e* ($14 \times 14 \times 832$)	Classifier Param.	**528 MB**	3.8 MB	3.8 MB	3.8 MB

Therefore, considering the number of iterations of each loop and the cost of these inner products, we can conclude that the complexity of these steps is $\mathcal{O}(Lktd)$, where L is the number of local descriptors $L = |\mathcal{L}|$. However, one may notice that most of the inner products computed are redundant, since $S_1, \ldots, S_k$ only contain references to p unique vectors and we are computing $2tk$ inner products for each local descriptor $\text{CNN}(\mathcal{I}, l)$. As shown in Chapter 4, a much more efficient strategy would be precomputing the inner products of the L local descriptors with the p vectors in S before steps 4-5. With these inner products precomputed, the expression in step 8 can be evaluated in $\mathcal{O}(1)$ time. Therefore, applying this implementation trick, the total time complexity of the proposed algorithm simplifies to $\mathcal{O}(L(pd + tk))$, where the $\mathcal{O}(Lpd)$ comes from precomputing the inner products and the $\mathcal{O}(Ltk)$ from executing steps 4-10. It is also important to note that, thanks to the sparse nature of the vectors in S, the computation of the inner products can be accelerated by using sparse matrix multiplication routines, available in most linear algebra packages.

Table 5.1 compares the number of parameters and time complexity of the proposed method with those of the Full Bilinear (FB) descriptor [71] and existing compact bilinear pooling methods [43]. In addition to the number of parameters needed to compute the final descriptor in each case, the table also shows the number of parameters of a one-vs-all linear classifier trained on the resulting descriptor, which in the case of the full bilinear descriptor is the main source of inefficiency. Some empirical values obtained for the particular hyperparameters and CNNs used in Section 5.4 are also provided.

5.3.3 Back-propagation for the proposed method

One of the main benefits of existing compact bilinear pooling methods [43] is their compatibility with the back-propagation algorithm, which makes them suitable for end-to-end training schemes. The fact that the partial derivative of the output of

these algorithms with respect to their input can be easily computed makes it possible to include them as intermediate layers in deep learning models, as the gradient of the loss function can be back-propagated towards the first layers using the chain rule.

In this section, we derive back-propagation for the proposed method, thus showing that it is also compatible with end-to-end training schemes. First, let $\mathfrak{L}$ denote the selected loss function. To keep the notation simple, we will denote the local descriptor $\text{CNN}(\mathcal{I}, l)$ as $\mathbf{x}_l$. Therefore, the input to the proposed algorithm is the set of d-dimensional local descriptors $\{\mathbf{x}_l\}_{l\in\mathcal{L}}$. The output of the algorithm is the k-dimensional projection of the bilinear descriptor, $\mathbf{y} \in \mathbb{R}^k$. Back-propagation for CBP-KRP can then be written as follows:

$$\frac{\partial \mathfrak{L}}{\partial \mathbf{x}_l} = \sum_{c=1}^{k} \frac{\partial \mathfrak{L}}{\partial \mathbf{y}_c} \frac{\partial \mathbf{y}_c}{\partial \mathbf{x}_l}, \tag{5.10}$$

$$\frac{\partial \mathbf{y}_c}{\partial \mathbf{x}_l} = \frac{1}{\sqrt{tk}} \sum_{i=0}^{t-1} \left(\langle \mathbf{x}_l, S_c[2i+1] \rangle S_c[2i+2] + \langle \mathbf{x}_l, S_c[2i+2] \rangle S_c[2i+1] \right).$$

The first equability is derived by simply applying the chain rule, and the second one is the partial derivative of the c-th feature in $\mathbf{y}$ with respect to one of the input local descriptors. A more detailed description of the derivation process can be found in Appendix B.

With this two equations, one can propagate the gradient of the loss function across our algorithm to layers closer to the input of the model. While it might be possible to derive the gradient with respect to the vectors in $S_1, \ldots, S_k$ to also update them during training, this is not recommended because (1) the sparsity of the vectors would be lost, and (2) we would no longer be approximating a Random Projection of the bilinear descriptor, as the distribution of the projection vectors would be altered. Section 5.4.4 presents experimental results on the fine-tuning of CNNs with CBP-KRP as an intermediate layer.

5.4 Experimental results and discussion

In this section, we present experimental results regarding both the efficiency and accuracy achieved by the proposed algorithm as compared with existing approaches. As mentioned before, our inference-time results focus on low computational power devices. As shown in [43], when compact bilinear pooling is executed on specialized hardware such as GPUs, the high level of parallelism in such devices makes bilinear pooling reasonably fast, to the point that compact bilinear pooling methods can even be slower[5]. In addition, the dominant factor in most cases is the forward pass of the convolution layers, so improvements in the efficiency of bilinear pooling might not have a significant impact in the total inference time of the entire model.

[5]For instance, [43] reported that full bilinear pooling and TS compact bilinear pooling required 0.77 and 5.03 ms respectively, while the time required for a forward pass of their CNN was 312 ms.

TABLE 5.2: Main features of the two Raspberry devices used for the inference-time experiments.

Raspberry Model	CPU model	Cores & Freq.	RAM	Release	Price
Pi 3 Model B+	BCM2837B0 (Cortex-A53)	4 @ 1.4 GHz	1 GB	14/03/18	$35
Pi Zero W	BCM2835 (ARM1176JZF-S)	1 @ 1 GHz	512 MB	28/02/17	$10

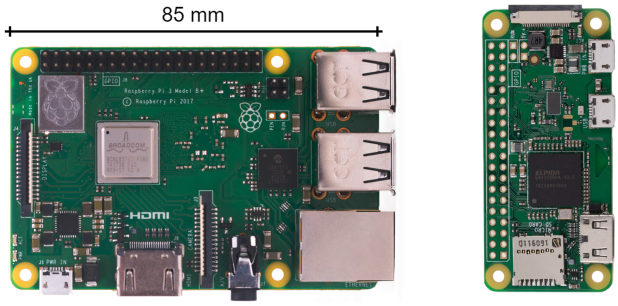


FIGURE 5.2: Raspberry Pi Model 3 B+ (Left) and Zero W (Right).

In such scenarios, the main advantage of compact bilinear pooling methods is the reduction in the number of parameters of the model, obtained as a consequence of the reduced dimensionality of the final descriptor. Conversely, when running on low computational power devices, compact bilinear pooling methods can make a huge difference both in terms of memory requirements and total inference times.

We perform inference-time experiments on two devices from one of most popular low-cost hardware platforms. In particular, we used the Raspberry Pi 3 Model B+, the latest version of the classic Raspberry series, and the Raspberry Pi Zero W, the smallest Raspberry computer to date[6]. Table 5.2 highlights some of the most important features of these devices, and Figure 5.2 shows their relative sizes. Given their widespread use, some of the most popular deep learning tools such as Tensorflow [1] now include support for installation on devices of the Raspberry Pi ecosystem. This reflects the growing interest of the community in running deep learning models on low cost and low power devices.

5.4.1 Evaluated methods

Since our experiments focus on inference-times for low power devices, we selected two relatively lightweight pretrained CNNs to make sure that the models would fit in memory. In particular, we used SqueezeNet v1.1 [51] and GoogLeNet [92] CNNs. On the one hand, SqueezeNet is a recently proposed architecture specifically designed for efficiency. Notably, the weights of this CNN only require 4.8 MB of storage, and even less if weight compression techniques are applied. Version v1.1 of this model achieves a similar accuracy as the original one while being twice as fast[7]. On the other hand, GoogLeNet is a slightly heavier model with a size of 25.7 MB, which was the winning architecture on the ImageNet 2014 challenge. Conveniently, public

[6] https://www.raspberrypi.org/products/ (Date accessed: 10/02/2019).
[7] https://github.com/DeepScale/SqueezeNet (Date accessed: 15/10/2018).

implementations exist for both models[8], based on the Keras [25] and Tensorflow [1] Python libraries[9]. The cut-off layers at which the bilinear pooling operation was performed were *fire9* for SqueezeNet and *inception (4e)* for GoogLeNet. The different evaluated approaches were as follows:

- Baseline: The CNN model is chopped at the specified cut-off layer. Then, a signed square root operation is applied followed by L2 normalization of the features. A one-vs-rest linear SVM classifier [35] is then trained directly on those features.
- Full bilinear pooling (FB): The CNNs model is chopped at the specified cut-off layer. Then, the bilinear pooling descriptor is generated [71] for the feature maps at the cut-off layer, followed by a signed square root operation and L2 normalization. A one-vs-rest linear SVM classifier [35] is then trained on the full bilinear descriptors.
- Compact bilinear pooling via Kernelized Random Projection (CBP-KRP): The CNN model is chopped at the specified cut-off layer. Then, Algorithm 3 is applied on the feature maps at the cut-off layer to compute a compact version of the bilinear descriptor, followed by a signed square root operation and L2 normalization. A one-vs-rest linear SVM classifier [35] is trained on the resulting descriptors. For CBP-KRP, we used $p = 5000$, $t = 2$ and $s = 100$ in all the experiments. The CBP-KRP algorithm itself was implemented in Python, using the standard linear algebra libraries [79] and numba [68] to accelerate loops where possible.
- Compact bilinear pooling via Random Maclaurin (RM): The CNN model is chopped at the specified cut-off layer. Random Maclaurin [43, 58] is used to generate a compact representation of the outer product of each local descriptor, and the resulting descriptors are average-pooled. Then, a signed square root operation is applied, followed by L2 normalization. A one-vs-rest linear SVM classifier [35] is then trained on the resulting descriptors. The original Matlab implementation of RM was rewritten in Python, using the standard linear algebra packages [79].
- Compact bilinear pooling via Tensor Sketch (TS): The CNN model is chopped at the specified cut-off layer. Tensor Sketch [43, 82] is used to generate a compact representation of the outer product of each local descriptor, and the resulting descriptors are average-pooled. Then, a signed square root operation is applied, followed by L2 normalization. A one-vs-rest linear SVM classifier [35] is then trained on the resulting descriptors. The original Matlab implementation of TS was rewritten in Python, using the standard linear algebra packages [79] and numba [68] to accelerate loops where possible[10].

[8] `https://github.com/rcmalli/keras-squeezenet` (Date accessed: 15/10/2018), `https://github.com/fchollet/deep-learning-models/pull/59` (Date accessed: 15/10/2018).

[9] We used Keras version 2.1.1 and Tensorflow version 1.9.0 in all our experiments.

[10] We used the Fast Fourier Transform implementation from: `https://docs.scipy.org/doc/scipy/reference/fftpack.html` (Date accessed: 15/10/2018).

As done in [71], we use $C_{\text{svm}} = 1$ to train the linear SVMs in all experiments.

5.4.2 Datasets used in the experiments

For our experiments, we used three well known fine-grained image categorization datasets, all of which include predefined train/test splits:

- Caltech UCSD Birds-200-2011 [99] (CUB). Animal species recognition dataset with 200 bird species, which extends the earlier CUB-200 dataset by increasing the number of images per class. The dataset contains a total of 11,788 images, with a standard split of 5,994 images for training and 5,794 for testing. The number of images per class ranges from 41 to 60. Part annotations and bounding boxes are provided for all the images.
- Stanford Cars Dataset (CARS) [61] Car model recognition dataset with 196 categories. Classes include the model and year of the car, for example "2012 Tesla Model S" or "2012 BMW M3". The dataset contains a total of 16,185 images, with a standard split of 8,144 images for training+validation and 8,041 for testing. Bounding boxes are provided for all the images.
- 102 Category Flower Dataset [78] (Flowers). Plant species recognition dataset with 102 flower species commonly occurring in the United Kingdom. The dataset contains a total of 8,189 images, with a standard split of 2,040 images for training+validation and 6,149 for testing. The number of images per class ranges from 40 to 258. Segmentation data is provided for the images.

Training and test images were preprocessed as follows. First, bounding boxes were used for CUB and CAR datasets to extract the relevant region of the images. In the case of the Flower dataset, bounding boxes are not explicitly provided, so the entire images were kept. Secondly, the resulting images were padded with zeros to make them square, and resized to the appropriate size depending on the CNN used in the experiment[11]. Finally, color preprocessing was applied as required[12].

5.4.3 Classification accuracy and inference-time

Tables 5.3 and 5.4 compare the accuracies and inference-times achieved by the different approaches described in Section 5.4.1, using SqueezeNet and GoogLeNet respectively. To compensate for the stochastic nature of some of the methods evaluated, each experiment was executed ten times. Average accuracies are reported together with their standard deviations. Regarding inference-time results, times are reported in the format $T_1/T_2/T_3$, where T_1 represents the time required for the image to be

[11] By default, SqueezeNet and GoogLeNet have input sizes of 227×227 and 224×224 respectively.

[12] The GoogLeNet implementation used requires pixel values in the range [-1,1]. SqueezeNet requires conversion from RGB to BGR and color zero-centering with respect to the ImageNet dataset.

TABLE 5.3: Comparison of compact bilinear pooling methods using SqueezeNet v1.1 chopped at *fire9* [51] as the base network. Inference time results are for the CUB dataset (i.e., 200 categories).

Method	**Descript. size (k)**	**Acc. (%)** CUB [99]	**Acc. (%)** CARS [61]	**Acc. (%)** Flowers [78]	**Time (ms)** Pi 3 Model B+	**Time (ms)** Pi Zero W
Baseline	$(13, 13, 512)$	46.46	49.07	71.83	156/0/119 Total: 273	1989/0/490 Total: 2,481
FB	512^2	66.05	63.42	83.34	149/22/360 Total: 539	1996/1042/1493 Total: 4,540
CBP-KRP	2,000	60.78±0.29	54.36±0.39	80.75±0.19	156/105/2 Total: 265	1962/490/11 Total: 2,461
TS	2,000	60.17±0.22	54.13±0.24	80.60±0.30	154/340/2 Total: 500	1968/1162/11 Total: 3,152
RM	2,000	59.51±0.28	53.12±0 .35	79.48±0.29	154/483/2 Total: 644	1950/1865/11 Total: 3,828
CBP-KRP	3,500	62.26±0.24	57.28±0.36	81.63±0.20	155/113/4 Total: 276	1966/582/20 Total: 2,573
TS	3,500	61.80±0.25	57.35±0.30	81.48±0.24	148/747/4 Total: 908	1950/2920/20 Total: 4,895
RM	3,500	60.63±0.25	55.84±0.31	80.17±0.27	147/859/4 Total: 1,021	1967/3268/20 Total: 5,257
CBP-KRP	5,000	62.94±0.16	58.68±0.42	82.04±0.29	155/123/6 Total: 287	1965/697/28 Total: 2,690
TS	5,000	62.85±0.29	58.84±0.26	81.95±0.20	152/916/6 Total: 1,077	1960/3791/28 Total: 5,797
RM	5,000	61.11±0.17	56.97±0.25	80.49±0.19	152/1224/6 Total: 1,388	1951/4668/28 Total: 6,665

TABLE 5.4: Comparison of compact bilinear pooling methods using GoogLeNet chopped at *inception (4e)* [51] as the base CNN. Inference time results are for the CUB dataset (i.e., 200 categories).

Method	**Descript. size (k)**	**Acc. (%)** CUB [99]	**Acc. (%)** CARS [61]	**Acc. (%)** Flowers [78]	**Time (ms)** Pi 3 Model B+	**Time (ms)** Pi Zero W
Baseline	$(14, 14, 832)$	47.03	56.05	77.49	542/0/223 Total: 770	11629/0/968 Total: 12,684
FB	832^2	74.83	75.46	89.78	545/74/951 Total: 1,571	11499/3083/49374 Total: 100,280
CBP-KRP	2,000	68.68±0.27	62.88±0.32	88.28±0.23	537/156/2 Total: 704	11440/740/12 Total: 12,174
TS	2,000	67.44±0.22	61.31±0.26	87.90±0.21	534/396/2 Total: 944	11770/1437/12 Total: 13,155
RM	2,000	67.56±0.33	62.17±0.34	87.82±0.21	539/820/2 Total: 1,359	11358/3417/12 Total: 14,781
CBP-KRP	3,500	70.14±0.45	65.46±0.34	89.02±0.24	537/171/4 Total: 712	11627/862/20 Total: 12,553
TS	3,500	69.61±0.35	64.20±0.24	88.73±0.23	536/872/4 Total: 1,426	11478/3636/20 Total: 15,142
RM	3,500	69.20±0.27	64.57±0.22	88.39±0.20	532/1477/4 Total: 2,020	11514/6084/20 Total: 17,614
CBP-KRP	5,000	71.04±0.22	66.84±0.32	89.24±0.15	546/182/7 Total: 731	11481/985/30 Total: 12,504
TS	5,000	70.52±0.27	65.68±0.35	89.05±0.17	526/1074/6 Total: 1,619	11452/4867/29 Total: 16,363
RM	5,000	69.87±0.30	65.71±0.28	88.56±0.17	539/2100/6 Total: 2,651	11554/8682/29 Total: 20,243

passed through the CNN, T_2 is the time needed to generate the final descriptor (either by full bilinear pooling or the corresponding compact bilinear pooling method), and T_3 is the time taken by the final linear classifier to emit a prediction. Note that unlike T_1 and T_2, T_3 is affected by the number of classes in the dataset. The timings reported in the tables are for a training dataset with 200 categories (e.g., the CUB dataset). Total inference times are also reported[13].

Looking at the accuracies in Tables 5.3 and 5.4, we can see that CBP-KRP outperformed the alternative compact bilinear pooling methods is most cases, providing

[13]Small discrepancies exist between total inference times reported and the sum of T_1, T_2 and T_3. This is because total inference times were measured independently and not computed as $T_1+T_2+T_3$. All the timings reported correspond to the lowest execution time among ten runs.

TABLE 5.5: Accuracies obtained using CBP-KRP with SqueezeNet and GoogLeNet, fine tuning all layers of the pretrained base network on the target dataset. Reported accuracies are the average over five runs, together with the average improvement with respect to the same experiment without fine-tuning.

Method	**Base Network**	**Descript. size (k)**	**Acc. (%)** CUB [99]	**Acc. (%)** CARS [61]	**Acc. (%)** Flowers [78]
CBP-KRP fine-tuned	SqueezeNet v1.1 at *fire9*	2,000	68.50±0.13 (7.72 ↑)	66.50±0.18 (12.14 ↑)	84.99±0.27 (4.24 ↑)
CBP-KRP fine-tuned	SqueezeNet v1.1 at *fire9*	3,500	69.53±0.42 (7.24 ↑)	68.59±0.12 (11.31 ↑)	85.47±0.10 (3.84 ↑)
CBP-KRP fine-tuned	SqueezeNet v1.1 at *fire9*	5,000	69.92±0.20 (6.98 ↑)	69.66±0.11 (10.98 ↑)	85.65±0.08 (3.61 ↑)
CBP-KRP fine-tuned	GoogLeNet at *inception (4e)*	2,000	80.13±0.51 (11.45 ↑)	82.08±0.35 (19.20 ↑)	92.61±0.16 (4.33 ↑)
CBP-KRP fine-tuned	GoogLeNet at *inception (4e)*	3,500	80.71±0.06 (10.57 ↑)	83.21±0.22 (17.75 ↑)	92.72±0.20 (3.70 ↑)
CBP-KRP fine-tuned	GoogLeNet at *inception (4e)*	5,000	81.11±0.10 (10.07 ↑)	83.79±0.20 (16.95 ↑)	92.94±0.10 (3.70 ↑)

the closest approximation to the accuracy of full bilinear pooling. Notably, this is achieved while maintaining much lower total inference times. For instance, using SqueezeNet and $k = 5,000$ on the Raspberry Pi 3 Model B+, the total inference time with CBP-KRP as the compact bilinear pooling method is 287 ms, while with TS and RM inference times break the one second mark. In addition, using CBP-KRP also results in lower inference times when compared with the full bilinear approach. In fact, CBP-KRP inference times were about half those of full bilinear pooling on the Pi 3 Model B+, and up to eight times lower on the Pi Zero W. This efficiency is in part achieved thanks to the sparse nature of the vectors used by CBP-KRP, which enables using fast sparse matrix multiplication routines for the projection. This supports our claim that, when considering low computational power devices, compact bilinear pooling methods can be useful not only to reduce models' memory requirements but to achieve lower inference times.

Another important aspect to consider when analyzing these results is the final model size achieved when using the different methods. As mentioned before, Table 5.1 shows some useful figures in this respect. Both CNNs used have a relatively low initial model size with 4.8 MB for SqueezeNet and 25.7 MB for GoogLeNet. In our experimental setup, using full bilinear pooling increases model sizes by 200 and 528 MB respectively, as a consequence of the high number of parameters of a linear classifier trained on 512^2 or 832^2 features, with 200 classes and a one-vs-all scheme. This of course is a problem if we want our models to run on devices with as little as 512 MB of main memory, which might also have other running processes competing for resources. Model size is also a problem when using compact bilinear pooling via RM, as the parameters needed by RM itself can require an important amount of memory. For instance, when using SqueezeNet, RM required 19.5 MB of additional memory, making the final model five times as heavy as the base CNN. Conversely, compact bilinear pooling via TS and CBP-KRP have a low memory footprint. As an example, consider the case were we use GoogLeNet. With TS, only 13 KB of additional memory are required to store its parameters. With CBP-KRP, 406 KB are required for the same purpose. This difference in the memory requirements of

TS and CBP-KRP is significant, but has a limited impact in the final model size given the 25 MBs of the base network and the 3.8 MBs of the linear classifier, which do not vary depending on whether TS or CBP-KRP are used.

5.4.4 Results with fine-tuning

As explained in Section 5.3.3, one interesting feature of the proposed algorithm is its compatibility with the back-propagation algorithm, which makes it possible to include CBP-KRP as an intermediate layer of end-to-end trainable models. In our experiments so far, we have focused on a simple transfer learning use case where only the final layer of the model is trained (i.e., the linear SVM), while the weights of the remaining layers are fixed. However, it is also common, if enough training data is available, to fine-tune the weights of the entire model by running some iterations of gradient descent with a low learning rate. Conveniently, the inference-time and size of the model do not change with this process. Therefore, models can be fine-tuned on computers with specialized hardware and then deployed in low power devices, obtaining a potential boost in accuracy with no increase in inference times. In this section, we show that CBP-KRP is compatible with this fine-tuning approach and how it can improve the performance with respect to a transfer learning strategy without fine-tuning.

We adopted a two step fine-tuning procedure similar to the one used in [71]. The process begins by chopping the pre-trained CNN model, keeping the layers before the selected cutoff point. After this, CBP-KRP is initialized and appended to the CNN as a layer in the model[14]. Then, a softmax layer is added as the final layer of the model. The first step in the training procedure consist in training this softmax layer alone, without altering the rest of the weights of the model. Then, with the model assembled and the final layer already trained, all the weights in the model are fine-tuned by executing a number of iterations of gradient descent. As explained in Section 5.3.3, the parameters of CBP-KRP are excluded from this fine-tuning process in order to preserve their sparsity. For our experiments, we used Adam [60] as the optimizer, and set the learning rate to 0.001 with a learning rate decay of 0.1. Batch size was set to 32 and the number of epochs to 20.

Table 5.5 shows the accuracies obtained by applying this approach with both SqueezeNet and GoogLeNet as the base CNN, and different output dimensions for CBP-KRP. As we can see, accuracies improved in all cases as a result of fine-tuning. The improvements in the accuracy ranged from 3.61 to 19.20 points, with the higher improvements occurring for the GoogLeNet CNN. These results evidence the potential of fine-tuning models which include compact bilinear pooling as an intermediate layer, and the compatibility of CBP-KRP with this approach.

[14] In order to include CBP-KRP as a layer in CNN models, we had to re-implement it using Keras and Tensorflow primitives.

TABLE 5.6: Accuracies obtained by CBP-KRP on the three datasets studied, with SqueezeNet and GoogLeNet as the base network and using different values for hyperparameters t and s. Hyperparameters p and k were fixed to 5000 and 2000 respectively. The best result for each dataset and base network is stressed in bold.

CBP-KRP Hyperparam.	**Accuracy with SqueezeNet (%)** CUB [99]	CARS [61]	Flowers [78]	**Accuracy with GoogLeNet (%)** CUB [99]	CARS [61]	Flowers [78]
$t = 2,\ s = 50$	60.22±0.26	54.15±0.36	80.55±0.30	68.07±0.27	62.26±0.37	88.13±0.21
$t = 4,\ s = 50$	59.86±0.38	53.78±0.37	80.22±0.24	67.58±0.32	61.72±0.44	87.88±0.21
$t = 6,\ s = 50$	59.66±0.32	53.42±0.36	80.00±0.26	67.37±0.33	61.15±0.40	87.73±0.21
$t = 2,\ s = 100$	**60.75**±0.35	**54.30**±0.39	**80.73**±0.26	68.73±0.36	62.88±0.43	88.25±0.20
$t = 4,\ s = 100$	60.39±0.35	54.16±0.39	80.57±0.28	68.29±0.34	62.31±0.35	88.19±0.19
$t = 6,\ s = 100$	60.15±0.40	54.06±0.35	80.49±0.27	67.88±0.33	61.74±0.44	88.06±0.26
$t = 2,\ s = 200$	60.41±0.38	52.65±0.46	80.25±0.34	68.72±0.39	**63.21**±0.48	88.15±0.28
$t = 4,\ s = 200$	60.74±0.33	53.86±0.43	80.60±0.39	**68.77**±0.41	62.86±0.37	**88.30**±0.31
$t = 6,\ s = 200$	60.63±0.44	54.24±0.36	80.70±0.21	68.55±0.42	62.46±0.36	88.17±0.22

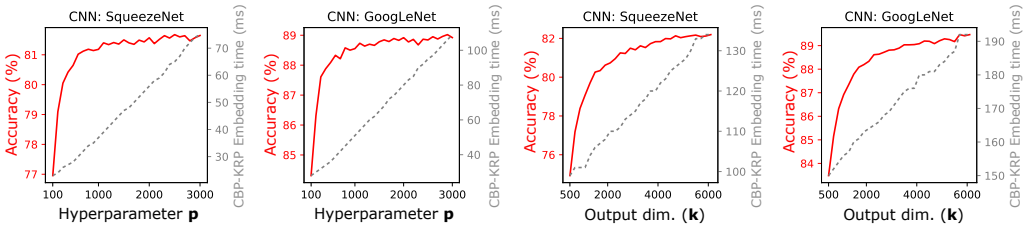


FIGURE 5.3: Effect of using different values for the hyperparameter p and the output dimension k. Experiments are for the Flowers [78] dataset. When exploring different values of p, k was fixed to 3500. Similarly, p was fixed to 5000 when exploring the effect of k. Embedding times are for the Raspberry Pi 3 Model B+.

5.4.5 Hyperparameter selection

One possible drawback of the proposed method is that the end-user must specify the value of a number of hyperparameters, which can be challenging when the underlying effects of these hyperparameters are not known. This subsection tries to mitigate this problem by providing a detailed description of the different hyperparameters of CBP-KRP, and exploring the effect of modifying each of them.

Looking at Algorithm 3, we can see that CBP-KRP has four hyperparameters whose values must be provided. These are the total number of random vectors generated (p), their sparsity level (s), the number of vectors summed to form each projection vector in the feature space (t), and the desired output dimension (k).

The hyperparameter p, which controls the number of unique random vectors generated by the algorithm, was introduced in Chapter 4 to reduce the computational cost of the kernelized Random Projection. As explained in Section 5.3.1, instead of using $2t$ distinct vectors for each output component, our algorithm generates a collection with p vectors, and reuses some of them in order to reduce costs. Therefore, p must be set to be $2tk \geq p > 2t$. Larger values of p reduce the re-usage of vectors, improving performance at the cost of longer running times. If p is set to $2tk$, no vector repetition will occur at all. Similarly, lower values of p sacrifice some performance to achieve a faster execution. Therefore, this hyperparameter can be used to control the performance/efficiency trade-off, without modifying the dimension of the output representation, which may have further implications. Figure 5.3 illustrates the effect

in accuracy and execution times of using different values for p. Conveniently, we can see that the accuracy grows rapidly with p, while as explained in Section 5.3.2 embedding times with CBP-KRP are linear in p.

For is part, k is a common hyperparameter in most kernel approximation methods which controls the number of features generated to approximate the kernel. Therefore, the hyperparameter k also defines a trade-off between accuracy and efficiency. The main difference is that, as opposed to p, k determines the dimensionality of the resulting descriptors, which might have implications for subsequent steps in the processing chain (e.g., for the final linear classifier in our case). Again, Figure 5.3 explores this trade-off, showing that the accuracy grows quickly as k increases.

The hyperparameter t determines the number of random vectors summed to form the projection vectors in the feature space. As explained in Section 5.3, forming the projection vectors as the sum of t random vectors results in a reduced dependence among their entries, which as shown in Chapter 4 is key for the distance-preservation properties of Random Projection. However, the same study revealed that the effect of t in classification accuracies is limited, and recommended using small values of this hyperparameter when the generated representations are intended for classification.

Finally, hyperparameter s determines the degree of sparsity of the generated random vectors. In particular, the entries of these vectors are zero with probability $1 - 1/s$. Therefore, using a relatively large s enables us to reduce computational and storage costs. Furthermore, using projection vectors with a certain degree of sparsity does not necessary have a negative impact in the classification accuracy, as sparse Random Projections are known to perform well in practice [70]. Moreover, sparsity has been shown to be a powerful tool in the context of deep learning, as it can contribute to mitigate over-fitting [104].

It must be noted, however, that since the projection vectors in the kernel feature space are built as the sum of t vectors, the sparsity level of the final projection vectors will also be affected by t, and not only by s. Hence, t and s should be jointly selected. Table 5.6 shows the accuracies obtained by CBP-KRP on the three datasets studied, using different values for hyperparameters t and s. Luckily, the results suggest that the proposed method is fairly robust to the selection of these hyperparameters. Particularly, the combination used in the comparisons of the previous section, $t = 2$ and $s = 100$, resulted in either the best or the second best result in all experiments. In some cases, a slight improvement in the accuracy was achieved when increasing the sparsity by setting $s = 200$ and using $t = 4$ or $t = 6$.

5.5 Conclusions and future work

This chapter has built upon the ideas of [43, 72] to propose CBP-KRP, a novel method to create compact feature descriptors which capture most of the power of full bilinear pooling descriptors [71]. Following the insights provided by [43], we

proposed an efficient method to approximate a Random Projection of the full bilinear descriptor, mostly preserving its discriminative information while greatly reducing the dimension of the final descriptor. This was achieved by adapting the ideas from Chapter 4, and exploiting the close relation that exists between the bilinear pooling operation and the degree-two homogeneous polynomial kernel. We also derived back-propagation for the proposed algorithm, showing that it can be used as a building block in end-to-end trainable models.

Our experimental results show that, for three popular fine-grained image categorization datasets, our method produces the best approximation to the accuracy of full bilinear pooling, outperforming existing compact bilinear pooling methods. Moreover, this is achieved while running significantly faster than TS and RM-based compact bilinear pooling on low computational power devices such as those from the Raspberry Pi ecosystem, and also faster than full bilinear pooling. In addition, the number of parameters used by our algorithm is relatively low, solving the memory issues that emerge when using full bilinear descriptors. As a consequence, our algorithm could be useful in embedded systems or other low computational power scenarios where tight computation and memory constraints exist.

Following previous studies on the topic of compact bilinear pooling, we focused on the case where a single CNN is used to form the bilinear descriptors [43]. However, extending CBP-KRP to the case where bilinear descriptors are formed as the outer product of the descriptors extracted by two different CNNs would be an interesting line for future work, as this could have applications in the domain of multi-modal problems [42]. In addition, we would like to explore the applicability of our algorithm in areas such as Internet of Things, Wearable technology or Embedded Systems [4], where efficient fine-grained image understanding methods could be of great use.

Chapter 6

Conclusion

As the amount of data generated by humans and devices continues to rise, machine learning has become an essential tool for data analysis, forecasting and task automation. Most of the success of modern machine learning models has been thanks to the availability of massive collections of data from which they learn, and the advances in hardware which have made it possible to train models with hundreds of millions of parameters. However, a growing need exists for efficient machine learning solutions, capable of running in low computational power environments while maintaining the standards of accuracy and reliability of existing methods.

A notable example of the importance of efficiency and scalability in machine learning is the case of kernel approximation techniques. As discussed in Chapter 2, a lot of effort has been put into the development of methods that efficiently approximate the properties of data in different kernel feature spaces, avoiding the scalability issues of traditional kernelized classifiers.

This thesis has explored the applicability of the Random Projection method to approximate the structure of data in the feature spaces of polynomial kernels. Specifically, the algorithms presented in Chapters 3 and 4 can be used to approximate a Random Projection from the feature space of a polynomial kernel without ever working in the feature space in an explicit manner. As opposed to existing solutions, the methods proposed in this thesis are data-independent, meaning that they don't require any knowledge about the distribution of data in order to operate. As a consequence, their training phase boils down to the initialization of some random matrices, which makes our methods more efficient and scalable than alternative approaches. This data-independence was achieved mainly by focusing on a specific kernel family, which enabled us to analyze the explicit form of the feature maps and develop more efficient methods to approximate Random Projections from kernel feature spaces.

Our experimental results focused on the preservation of pairwise distances from the kernel feature space down to a low dimensional representation. In addition, we studied the the classification accuracy obtained by linear classifiers in the resulting representations. Generally speaking, experimental results evidenced that the proposed methods succeed in approximately preserving the structure of data from

the feature spaces of polynomial kernels, which opens the door to applications in scenarios where polynomial interactions between features play an important role. Particularly, in Chapter 5 we explored the applicability of the methods proposed in the previous chapters to make bilinear deep learning models more efficient. Our results evidenced that bilinear Convolutional Neural Networks can be accelerated by applying the ideas presented in Chapter 4, leading to important time and memory savings when models run in low computational power environments.

In the future, we would like to explore the applicability of the proposed methods in other domains beyond categorization. While the experiments in this thesis have focused on the problem of classification, the information present in the feature space of polynomial kernels may likely be useful in other machine learning and data analysis tasks such as information retrieval, regression or clustering. Moreover, the ideas presented in this thesis could be used to accelerate existing machine learning methods which rely on polynomial kernels or, more generally, on the polynomial interaction of features.

In closing, Random Projections from polynomial kernel feature spaces have proven to be an excellent approach for the efficient generation of compact representations of data which capture the useful information offered by the kernel, thus providing a powerful tool for the creation of effective and scalable machine learning applications.

Appendix A

Low-level Algorithmic Specification of the Proposed Methods

This appendix contains a low-level description of the algorithms presented in Chapters 4 and 5. While the algorithmic descriptions given in the corresponding chapters focus on readability and clarity, the pseudo-code descriptions provided here try to give a self-contained description of how to implement the proposed algorithms in an efficient manner.

Pseudocode 1 gives an implementation-oriented description of Algorithm 2 (Chapter 4). Similarly, Pseudocode 2 provides the implementation-oriented description of Algorithm 3 (Chapter 5). In both cases, the main differences between the original algorithms' descriptions and these pseudocodes are (1) the inner products between the data samples and the projection vectors are pre-computed, and (2) the subsets/collections of projection vectors $S_1, \ldots, S_k$ are replaced by a matrix of integers, which indexes the matrix K of pre-computed inner products. Depending on the languague and environment selected to implement the algorithms, further implementation tricks might be applied to accelerate computations (e.g., sample-level parallelism, sparse matrix multiplication routines, etc.)

Pseudocode 1 Polynomial Kernel Random Projection (PK-RP)

Require: A $N \times d$ matrix X representing input data samples, the desired degree g for the polynomial kernel, the total number of random vectors p, the number t of summed vectors for the CLT, the desired output dimension k and if using Achlioptas' distribution the desired sparsity level s.

Ensure: Returns a $N \times k$ matrix representing output samples such that pairwise distances between these samples are approximately equal to those of input data samples mapped to the feature space of the homogeneous polynomial kernel of degree g.

Initialize a matrix $S \in \mathbb{R}^{d \times p}$ with entries i.i.d. from $\mathcal{N}(0,1)$
    or with entries independently drawn from $\{-\sqrt{s}, 0, \sqrt{s}\}$ with prob. $\{\frac{1}{2s}, 1-\frac{1}{s}, \frac{1}{2s}\}$
Generate a matrix of indexes $I \in \mathbb{N}^{k \times gt}$ where each row
    consists of gt distinct random integers from the range $[1, p]$
$K \leftarrow XS$
$X' \leftarrow \{0\}^{N \times k}$
for $n = 1, \cdots, N$ **do**
    for $c = 1, \cdots, k$ **do**
        for $i = 0, \cdots, t-1$ **do**
            $temp \leftarrow K[n, I[c, gi+1]]$
            for $j = 2, \cdots, g$ **do**
                $temp \leftarrow temp \cdot K[n, I[c, gi+j]]$
            $X'[n, c] \leftarrow X'[n, c] + temp$
$X' \leftarrow \frac{1}{\sqrt{tk}} X'$
return X'

Pseudocode 2 Compact Bilinear Pooling via Kernelized Random Projection

Require: Descriptors for some image $\mathcal{I}$ at each location: $\text{CNN}(\mathcal{I}, l)$ for $l \in \mathcal{L}$, represented as a $|\mathcal{L}| \times d$ matrix X. The total number of random vectors p and their sparsity level s, the number t of vectors used for the CLT and the desired output dimension k.

Ensure: Returns a k-dimensional vector which approximates a Random Projection of the full bilinear pooling descriptor.

Initialize a matrix $S \in \mathbb{R}^{d \times p}$ with entries independently drawn
    from $\{-\sqrt{s}, 0, \sqrt{s}\}$ with prob. $\{\frac{1}{2s}, 1-\frac{1}{s}, \frac{1}{2s}\}$ respectively.
$P = \underbrace{\{1, \cdots, p\} \cup \cdots \cup \{1, \cdots, p\}}_{\lfloor 2tk/p \rfloor} \cup \{1, \cdots, 2tk \bmod p\}$, so that $|P| = 2tk$
Shuffle the contents of P and use them to populate a matrix of integers $I \in \mathbb{N}^{k \times 2t}$
$K \leftarrow XS$ ▷ Accelerate using sparse matrix multiplication
$\mathbf{y} \leftarrow (0, \cdots, 0) \in \mathbb{R}^k$
for $l = 1, \cdots, |\mathcal{L}|$ **do**
    for $c = 1, \cdots, k$ **do**
        for $i = 0, \cdots, t-1$ **do**
            $\mathbf{y}[c] \leftarrow \mathbf{y}[c] + K[l, I[c, 2i+1]] \cdot K[l, I[c, 2i+2]]$
$\mathbf{y} = \frac{1}{\sqrt{tk}} \mathbf{y}$
return $\mathbf{y}$

Appendix B

Back-propagation for CBP-KRP

This appendix contains the details for the derivation of back-propagation for the CBP-KRP algorithm, considered as a layer inside an end-to-end trainable model. The CBP-KRP algorithm takes the descriptors $\mathrm{CNN}(\mathcal{I}, l)$ for each $l \in \mathcal{L}$ as the input, and computes a k-dimensional vector $\mathbf{y}$ as the output. Each component of the output vector $\mathbf{y}$ is computed based on a collection of randomly initialized and fixed vectors $\mathbf{r}_1, \cdots, \mathbf{r}_{2t}$, applying the following formula:

$$\mathbf{y}_c = \frac{1}{\sqrt{tk}} \sum_{l \in \mathcal{L}} \sum_{i=0}^{t-1} \langle \mathrm{CNN}(\mathcal{I}, l), \mathbf{r}_{2i+1} \rangle \langle \mathrm{CNN}(\mathcal{I}, l), \mathbf{r}_{2i+2} \rangle . \tag{B.1}$$

First, we find the partial derivative of the output features that form $\mathbf{y}$ with respect to the inputs of the algorithm. For the sake of simplicity, we will denote the descriptor $\mathrm{CNN}(\mathcal{I}, l)$ of location l as $\mathbf{x}_l \in \mathbb{R}^d$. Therefore, we are interested in the following derivative:

$$\frac{\partial \mathbf{y}_c}{\partial \mathbf{x}_l} = \frac{\partial}{\partial \mathbf{x}_l} \frac{1}{\sqrt{tk}} \sum_{i=0}^{t-1} \langle \mathbf{x}_l, \mathbf{r}_{2i+1} \rangle \langle \mathbf{x}_l, \mathbf{r}_{2i+2} \rangle \tag{B.2}$$

$$= \frac{1}{\sqrt{tk}} \sum_{i=0}^{t-1} \frac{\partial}{\partial \mathbf{x}_l} \langle \mathbf{x}_l, \mathbf{r}_{2i+1} \rangle \langle \mathbf{x}_l, \mathbf{r}_{2i+2} \rangle . \tag{B.3}$$

Note that the summation over locations in $\mathcal{L}$ disappears from the expression of $\mathbf{y}_c$ when taking the derivative with respect to the descriptor from a particular location l, since only one term in the summation is a function of $\mathbf{x}_l$. The second line follows from the linearity of differentiation.

Now, let us consider three arbitrary d-dimensional vectors $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{v}^{(3)}$. Given the product of inner products $\langle \mathbf{v}^{(1)}, \mathbf{v}^{(2)} \rangle \langle \mathbf{v}^{(1)}, \mathbf{v}^{(3)} \rangle$, we calculate its derivative with

respect to the entry at position m of vector $\mathbf{v}^{(1)}$:

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{v}_m^{(1)}} \left\langle \mathbf{v}^{(1)}, \mathbf{v}^{(2)} \right\rangle \left\langle \mathbf{v}^{(1)}, \mathbf{v}^{(3)} \right\rangle \\
&= \frac{\partial}{\partial \mathbf{v}_m^{(1)}} \left(\mathbf{v}_1^{(1)} \mathbf{v}_1^{(2)} + \cdots + \mathbf{v}_d^{(1)} \mathbf{v}_d^{(2)} \right) \cdot \left(\mathbf{v}_1^{(1)} \mathbf{v}_1^{(3)} + \cdots + \mathbf{v}_d^{(1)} \mathbf{v}_d^{(3)} \right) \\
&= \mathbf{v}_m^{(2)} \cdot \left(\mathbf{v}_1^{(1)} \mathbf{v}_1^{(3)} + \cdots + \mathbf{v}_d^{(1)} \mathbf{v}_d^{(3)} \right) + \mathbf{v}_m^{(3)} \cdot \left(\mathbf{v}_1^{(1)} \mathbf{v}_1^{(2)} + \cdots + \mathbf{v}_d^{(1)} \mathbf{v}_d^{(2)} \right) \\
&= \mathbf{v}_m^{(2)} \left\langle \mathbf{v}^{(1)}, \mathbf{v}^{(3)} \right\rangle + \mathbf{v}_m^{(3)} \left\langle \mathbf{v}^{(1)}, \mathbf{v}^{(2)} \right\rangle .
\end{aligned} \tag{B.4}$$

From this, it is easy to compute the partial derivative of $\langle \mathbf{v}^{(1)}, \mathbf{v}^{(2)} \rangle \langle \mathbf{v}^{(1)}, \mathbf{v}^{(3)} \rangle$ with respect to $\mathbf{v}^{(1)}$, which of course is a d-dimensional vector:

$$\frac{\partial}{\partial \mathbf{v}^{(1)}} \left\langle \mathbf{v}^{(1)}, \mathbf{v}^{(2)} \right\rangle \left\langle \mathbf{v}^{(1)}, \mathbf{v}^{(3)} \right\rangle = \mathbf{v}^{(2)} \left\langle \mathbf{v}^{(1)}, \mathbf{v}^{(3)} \right\rangle + \mathbf{v}^{(3)} \left\langle \mathbf{v}^{(1)}, \mathbf{v}^{(2)} \right\rangle . \tag{B.5}$$

The formula we just derived can be conveniently used to complete the derivation of (B.3) as follows:

$$\frac{\partial \mathbf{y}_c}{\partial \mathbf{x}_l} = \frac{1}{\sqrt{tk}} \sum_{i=0}^{t-1} \left(\mathbf{r}_{2i+1} \langle \mathbf{x}_l, \mathbf{r}_{2i+2} \rangle + \mathbf{r}_{2i+2} \langle \mathbf{x}_l, \mathbf{r}_{2i+1} \rangle \right). \tag{B.6}$$

Now that we have the derivative of each output value of our algorithm with respect to the inputs, we can use the chain rule to obtain the partial derivative of the final loss function of the entire model with respect to our algorithm's inputs, which enables the propagation of the gradient through it. Formally, let $\mathfrak{L}$ be the loss function of the model. Then the partial derivative of this loss with respect to one of the input descriptors is:

$$\frac{\partial \mathfrak{L}}{\partial \mathbf{x}_l} = \sum_{c=1}^{k} \frac{\partial \mathfrak{L}}{\partial \mathbf{y}_c} \frac{\partial \mathbf{y}_c}{\partial \mathbf{x}_l}. \tag{B.7}$$

Finally, as explained in Chapter 5, CBP-KRP uses a set S containing p unique vectors which is sampled to form the collections $S_1, \cdots, S_k$ that contain the vectors used to compute each of the k output features of $\mathbf{y}$. Taking this into consideration, (B.6) can be rewritten as:

$$\frac{\partial \mathbf{y}_c}{\partial \mathbf{x}_l} = \frac{1}{\sqrt{tk}} \sum_{i=0}^{t-1} \left(\langle \mathbf{x}_l, S_c[2i+1] \rangle S_c[2i+2] + \langle \mathbf{x}_l, S_c[2i+2] \rangle S_c[2i+1] \right),$$

which together with (B.7) completes the derivation of the back-propagation rules for CBP-KRP.

It is also possible to calculate the derivative of the loss function with respect to the vectors in the set S, to update them as part of the training of the model. However, this is not recommend because (1) this would make them dense, as opposed

to their initial sparse nature which enables important computational and memory savings; and (2) it would alter the distribution of the resulting projection vectors in the feature space, so we would not be approximating a Random Projection anymore.

Appendix C

Thesis Summary in Spanish

Este anexo recoge la traducción al Español del **título**, **resumen**, **agradecimientos**, **índice**, **introducción** y **conclusiones** del trabajo de tesis doctoral "*Low Computational Cost Machine Learning: Random Projections and Polynomial Kernels*", de Daniel López Sánchez, así como un **resumen significativo** de los capítulos restantes. El trabajo original, escrito en inglés, describe los resultados de forma más detallada.

Fdo. Juan M. Corchado Rodríguez

Fecha:

Fdo. Angélica González Arrieta

Fecha:

Fdo. Daniel López Sánchez

Fecha:

Resumen

Aprendizaje automático de bajo coste computacional: proyecciones aleatorias y kernels polinómicos

por Daniel López Sánchez

Según estudios recientes, el volumen de datos creados, capturados y replicados globalmente durante el año 2018 fue de 33 Zettabytes (ZB), y se espera que alcance 175 ZB para el año 2025. Manejar este impresionante crecimiento en el volumen y variedad de los datos representa un gran reto, pero también supone una valiosa oportunidad para que las organizaciones den soporte a sus procesos de toma de decisiones mediante el conocimiento extraído de enormes colecciones de datos, y automaticen tareas obteniendo importantes ahorros. En este contexto, el campo del aprendizaje automático ha atraído un notable nivel de atención, y los avances recientes en el campo han permitido crear modelos predictivos con una precisión sin precedentes. Sin embargo, con la emergencia de nuevos paradigmas de computación, el campo del aprendizaje automático debe afrontar el reto de crear modelos más eficientes, capaces de funcionar en entornos con una baja potencia de cómputo, manteniendo al mismo tiempo un nivel alto de precisión. Esta tesis se centra en el diseño y evaluación de nuevos algoritmos para la generación de representaciones de datos útiles, con especial atención a la escalabilidad y eficiencia de los métodos propuestos. En particular, los métodos propuestos hacen un uso intensivo de la aleatorización con el fin de mapear las muestras de datos al espacio de características de kernels polinómicos, para luego condensar la información útil presente en esos espacios de características a una representación compacta. Los diseños algorítmicos resultantes son fáciles de implementar, y requieren solo una baja potencia de cómputo para ser ejecutados. Como consecuencia, están perfectamente adaptados para ser aplicados en entornos donde los recursos computacionales son escasos, y los datos deben ser analizados con muy poco margen de tiempo. En particular, las dos contribuciones principales de esta tesis son: (1) se presentan y evalúan algoritmos eficientes para realizar Proyecciones Aleatorias desde los espacios de características de kernels polinómicos de diferente grado y (2) se demuestra la aplicabilidad de estas técnicas para acelerar tareas de aprendizaje automático donde la interacción polinómica de las características es importante, centrándonos en el caso particular de los métodos bilineales en el aprendizaje profundo.

Agradecimientos

En primer lugar, doy las gracias a mis directores, Juan Manuel Corchado y Angélica González Arrieta, por su dedicación y consejo. Esta tesis nunca habría sido posible sin su valioso apoyo.

En segundo lugar, quiero expresar mi profunda gratitud a John A. Lee y al resto del Machine Learning Group de la Universidad Católica de Lovaina por su extraordinaria hospitalidad durante mis estancias en su laboratorio. Los meses que pasé trabajando con ellos fueron de crucial ayuda para el desarrollo de la tesis e hicieron mi experiencia como doctorando mucho más enriquecedora.

También quiero dar las gracias a mis compañeros presentes y pasados del grupo de investigación BISITE. Es difícil expresar con palabras lo agradecido que les estoy por su compañía y amistad. Conferencias, descansos y las largas horas de trabajo no habrían sido ni remotamente tan agradables sin ellos.

Mi gratitud va también dirigida a Belén Pérez Lancho, por su apoyo y paciencia en la preparación de mis labores docentes en la asignatura de Informática Teórica. Enseñar esta asignatura ha sido una experiencia magnífica, y me ha proporcionado una perspectiva muy interesante sobre los conceptos fundacionales de la ciencia de la computación.

Quiero de igual forma mostrar mi gratitud con la Universidad de Salamanca, por proporcionarme un marco incomparable de historia, cultura y vida académica. Estos años viviendo y trabajando en Salamanca me han marcado profundamente, y difícilmente podré olvidar todas las amistades, experiencias y oportunidades que esta ciudad me ha ofrecido. Si bien es cierto que "Quod natura non dat, Salmantica non præstat", esta ciudad me ha dado más de lo que nunca habría pensado.

Por último, doy las gracias a mis padres Ángel López López y Esther Sánchez Casal, por su amor, paciencia y apoyo. Desde los albores de mi infancia, mis padres me animaron a explorar, aprender y hacer, plantando así la semilla de mi pasión por la investigación. Esta tesis está dedicada a ellos.

Daniel López Sánchez
Salamanca, 17 de Mayo de 2019

Índice general

(Los números de página se refieren al documento principal y no a la versión de este anexo)

Capítulo 1

Introducción

Aunque existen discrepancias en las estimaciones, todas las fuentes están de acuerdo en que el fenómeno del Big Data[1] sigue creciendo. A finales de 2018, la *global Datasphere*, que se compone de todos los datos creados, capturados y replicados, tenía un volumen de 33 Zettabytes (ZB). De hecho, las proyecciones indican que crecerá hasta los 175 ZB para el año 2025 [85]. Además de este impresionante crecimiento, el fenómeno del Big Data ha permeado las más diversas esferas de la actividad humana. Por ejemplo, un estudio reciente estimó que, cada minuto, los usuarios de Internet publican 473.400 tweets, solicitan 1.389 viajes en Uber, realizan 3.877.130 búsquedas en Google, suben 400 horas de vídeo a Youtube y originan 6.940 coincidencias en Tinder. Esto ejemplifica cómo el fenómeno del Big Data no solo ha crecido de forma sostenida, sino que se encuentra ya presente en casi todas las facetas de nuestra vida cotidiana. Además de la información generada como resultado de la interacción directa de los seres humanos con diversos sistemas digitales, una fracción importante del flujo de información actual se debe a los cientos de miles de dispositivos del *Internet of Things* (IoT) conectados a la red. De hecho, se espera que estos dispositivos generen 90 ZB de información en 2025 [85].

Este enorme crecimiento en el volumen y variedad de los datos supone un reto crítico para las compañías, organizaciones y gobiernos de todo el mundo, pero al mismo tiempo representa una gran oportunidad para que estos encuentren valor en sus datos. Una estrategia efectiva de recolección, almacenamiento y procesamiento de datos puede proporcionar a las organizaciones una herramienta crucial para apoyar sus procesos de toma de decisiones, permitiéndoles reducir costes, identificar necesidades de los consumidores, mejorar sus servicios y alcanzar nuevos mercados. Además, la abundancia de datos ha permitido a los científicos crear modelos predictivos cada vez más complejos, capaces de automatizar un amplio rango de tareas, igualando y a menudo superando el desempeño humano.

En este contexto, es fácil entender por qué la extracción de conocimiento a partir de los datos en bruto se ha convertido en un tema de investigación candente en los

[1] De acuerdo con el Gartner's IT Glossary, el Big Data consiste en "*activos de información de alto volumen, velocidad o variedad, que requieren formas de procesamiento de la información eficientes e innovadoras que habiliten una mejor extracción de conocimiento, toma de decisiones y automatización de procesos.*"
URL: https://www.gartner.com/it-glossary/big-data (Fecha de acceso: 24/04/2019).

últimos años, a medida que los científicos han competido para descubrir nuevos y mejores métodos para analizar grandes volúmenes de información. En el centro de esta revolución tecnológica reside el campo del *aprendizaje automático*, donde los avances recientes han hecho que los modelos predictivos se vuelvan más efectivos que nunca. En particular, el volumen de datos en continuo crecimiento junto con los avances en el diseño de los algoritmos y el uso de hardware especializado, han dado lugar a la proliferación de modelos predictivos cada vez más precisos tales como los del paradigma del aprendizaje profundo.

De hecho, la mayoría de los servicios digitales que usamos a diario dependen de algún tipo de modelo de aprendizaje automático, y como hemos visto, estos proveedores de servicios a menudo deben dar respuesta a miles de peticiones por segundo. Por tanto, no es ninguna sorpresa que de entre los retos afrontados actualmente por la comunidad del aprendizaje automático, la escalabilidad y eficiencia de los algoritmos jueguen un papel crucial en la revolución tecnológica del Big Data. Además, la emergencia y posterior popularización de nuevos paradigmas de computación tales como el Internet of Things también han contribuido a dirigir el interés de los investigadores hacia modelos de aprendizaje automático capaces de ejecutarse en entornos con escasos recursos computacionales, manteniendo al mismo tiempo tiempos de inferencia bajos.

Esta tesis se centra en el diseño y evaluación de nuevos algoritmos para la generación de representaciones útiles para los datos, con especial atención a la escalabilidad y eficiencia de las soluciones propuestas. En particular, los métodos propuestos hacen un uso intensivo de la aleatorización para mapear las muestras de datos a una representación más rica, de mayor información discriminativa, y después condensar esta información útil en una representación compacta. Los diseños experimentales resultantes son fáciles de implementar y requieren pocos recursos computacionales para ser ejecutados. Como consecuencia, están perfectamente preparados para ser aplicados en entornos donde los recursos computacionales son escasos, y los datos deben ser analizados en tiempo real o con un retardo pequeño.

Antes de profundizar en los métodos propuestos, presentamos algunos aspectos fundamentales relacionados con esta tesis y revisamos los avances más recientes en la intersección de las Proyecciones Aleatorias y las funciones de kernel. Las contribuciones principales de este trabajo se presentan en los Capítulos 3, 4 y 5, que detallan nuevos diseños algorítmicos para capturar la información descriptiva de ciertas funciones de kernel en una representación de baja dimensión. En particular:

- En el Capítulo 3, presentamos un método eficiente para aproximar una Proyección aleatoria desde el espacio de características del kernel polinómico de grado dos. Las representaciones de datos resultantes preservan de forma aproximada la estructura de los datos en ese espacio de características, lo que nos permite resolver problemas de aprendizaje de forma eficiente. En particular, esto se logra usando clasificadores lineales sobre la versión proyectada de los

datos, garantizando así la eficiencia del proceso al tiempo que mejora la tasa de acierto, gracias a la información discriminativa extraída del espacio de características del kernel. Al contrario que otros enfoques para la realización de Proyecciones Aleatorias en espacios de características de kernels, nuestro método es independiente de los datos, lo que significa que no necesita ningún conocimiento previo sobre la distribución de las muestras de datos que serán procesadas en la fase de test. Presentamos resultados experimentales sobre la preservación de distancias y la tasa de acierto de clasificación de clasificadores lineales entrenados en las representaciones de salida, evidenciando que el método propuesto supera a los métodos existentes en la mayoría de casos, siendo además notablemente más rápido.

- El Capítulo 4 desarrolla y completa las ideas presentadas en el Capítulo 3 para mejorar la generalizad, eficiencia y efectividad del método propuesto. Particularmente, se presenta un nuevo método que es capaz de aproximar una Proyección aleatoria desde el espacio de características de kernels polinómicos de grado mayor que dos. Además, un nuevo método para generar los vectores de proyección en el espacio de características del kernel nos permite reducir el coste computacional de nuestro algoritmo al tiempo que mejoramos su eficacia. De nuevo, los resultados experimentales dan soporte a nuestra tesis de que el método propuesto es capaz de condensar la estructura de los datos en el espacio de características del kernel, preservando aproximadamente las distancias entre las muestras y mejorando la tasa de acierto de los clasificadores lineales.
- Finalmente, el Capítulo 5 explora la conexión entre la versión kernelizada del algoritmo de Proyección aleatoria presentada en el Capítulo 4 y la popular arquitectura de las *bilinear Convolutional Neural Networks* (CNN), caracterizadas por el uso de la operación conocida como *bilinear pooling*. Si bien las redes de tipo *bilinear CNN* se encuentran entre los métodos más efectivos y populares para el reconocimiento de imágenes de grado fino, la dimensionalidad de los descriptores generados por estas redes suponen una limitación importante para estos modelos, consistiendo a menudo en varios cientos de miles de características. El Capítulo 5 presenta un nuevo método para reducir la dimensión de los descriptores de tipo *bilinear pooling* de forma eficiente, mediante la realización de una Proyección aleatoria. Convenientemente, esto se logra sin necesidad de computar el descriptor de alta dimensión de forma explícita. Nuestros resultados evidencian que este enfoque supera los enfoques conocidos como *compact bilinear pooling* en la mayoría de los casos, ejecutándose en menor tiempo en dispositivos de baja potencia de cómputo, donde estas extensiones eficientes del *bilinear pooling* son más necesarias.

El Capítulo 6 cierra esta tesis, resumiendo las contribuciones principales de la misma, así como los principales resultados obtenidos en los capítulos anteriores.

Capítulo 2

Fundamentos

Este capítulo introduce los conceptos y técnicas fundamentales que serán relevantes para los siguientes capítulos. En primer lugar, se describe el algoritmo Random Projection, un método de reducción de dimensionalidad muy popular y de una gran eficiencia que garantiza una baja distorsión en las distancias entre las muestras de datos. En segundo lugar, se enumeran algunas de las propiedades más importantes de las funciones de kernel, con especial atención a los kernels polinómicos. Después, se dibuja una conexión entre el algoritmo Random Projection y las funciones de kernel, revisando los algoritmos existentes para la realización de proyecciones aleatorias desde los espacios de características de diversas funciones de kernel. Finalmente, se repasan las propiedades de bilinear pooling, una técnica diseñada para mejorar el desempeño de las redes neuronales convolucionales en tareas de clasificación de grano fino. Como veremos, esta técnica está íntimamente conectada con el kernel polinómico de grado dos. Tal conexión nos permitirá más adelante adaptar los métodos presentados en los Capítulos 3 y 4, haciendo que la aplicación de bilinear pooling sea mucho más eficiente.

2.1. El algoritmo Random Projection

El algoritmo *Random Projection* (RP) es un método simple pero efectivo, y muy utilizado para la reducción de dimensionalidad lineal. Al igual que cualquier otro método de reducción de dimensionalidad lineal, *Random Projection* reduce la dimensión de las muestras aplicándoles una transformación lineal, de forma que cada componente de salida se computa como una combinación lineal de las características originales. Sin embargo, la mayor diferencia con métodos alternativos es que *Random Projection* genera la matriz de proyección a partir de una distribución aleatoria. Por tanto, a diferencia de otros métodos de reducción de dimensionalidad que requieren acceso a datos de entrenamiento para generar una matriz de proyección apropiada, *Random Projection* es un método independiente de los datos, dado que no necesita información sobre la distribución de los mismos para generar la matriz de proyección. Sorprendentemente, si se usa una distribución apropiada para generar las entradas de la matriz de proyección, la estructura de los datos será aproximadamente preservada en la representación resultante.

A pesar de su simplicidad, *Random Projection* tiene una sólida fundamentación teórica. El resultado teórico más importante que subyace a RP es el lema de Johnson–Lindenstrauss (JL) [54], que garantiza que N puntos en un espacio de alta dimensionalidad pueden ser proyectados a un espacio de mucha menor dimensionalidad de tal forma que las distancias entre los puntos sean aproximadamente preservadas.

Las variantes originales de este algoritmo realizaban una proyección sobre un subespacio aleatorio de dimensión k, por lo que la proyección aleatoria tomaba la forma de una proyección sobre k vectores ortonormales aleatorios [41]. En versiones posteriores, la proyección de las muestras tiene lugar por medio de una matriz de proyección de dimensión $d \times k$, cuyas entradas se seleccionan de forma independiente a partir de una distribución normal estándar [52, 8].

Sin embargo, estudios más recientes han mostrado que la matriz de proyección puede ser generada a partir de una distribución mucho más simple. En particular, Achlioptas demostró que si la matriz de proyección se genera a partir de una distribución discreta y dispersa apropiada, el lema de Johnson–Lindenstrauss será igualmente satisfecho [2].

Convenientemente, usar la distribución propuesta por Achlioptas reduce el coste computacional de la proyección. Si la multiplicación por $\sqrt{s}$ presente en la distribución propuesta por Achlioptas se retrasa, es posible evaluar la proyección haciendo uso únicamente de sumas y restas (y no multiplicaciones) lo que permite implementar esta operación de forma sencilla en entornos de bases de datos, usando primitivas SQL. Además, la naturaleza dispersa de la distribución de Achlioptas permite ahorros computacionales adicionales, ya que permite controlar la fracción de entradas de la matriz de proyección que valen cero. Convenientemente, estudios posteriores sugieren que se puede usar una matriz de proyección altamente dispersa, con un bajo coste en términos de efectividad [70].

A lo largo de los años, numerosos estudios han demostrado que el fenómeno descrito por el lema de Johnson–Lindenstrauss es bastante robusto, ya que numerosas distribuciones para la matriz de proyección pueden dar lugar a una preservación aproximada de las distancias entre los puntos una vez estos son proyectados [2, 8, 75]. Sin embargo, el elemento crucial presente en todas estas versiones de *Random Projection* es que las entradas de la matriz de proyección deben ser seleccionadas de forma *independiente*.

2.2. Funciones de kernel y el kernel trick

Intuitivamente, las funciones de kernel nos permiten evaluar de forma eficiente un producto interno tras mapear las muestras a un espacio de características diferente. Este espacio de características alternativo asociado a la función de kernel se conoce como el *feature space* del kernel, y la función que mapea las muestras de entrada

a este espacio se denomina *feature map*. Es importante señalar que pueden existir varios *feature spaces* y *feature maps* válidos para una misma función de kernel. Por otro lado, una característica interesante de las funciones de kernel es que las mismas pueden ser usadas para calcular distancias entre muestras en el *feature space*, sin necesidad de trabajar en el mismo de forma explícita.

Además de su importancia en la literatura matemática, las funciones de kernel han recibido una gran atención por parte de la comunidad del aprendizaje automático, principalmente por estar íntimamente relacionadas con algunas forma de clasificación no lineal. En particular, un gran número de clasificadores no lineales se caracterizan por transformar las muestras por medio de un *feature map* no lineal, tras el cual se aplica un clasificador lineal tradicional. Sin embargo, esto suele ser computacionalmente muy costoso dada la alta dimensionalidad de las representaciones generadas por el *feature map*.

Para resolver este problema, un enfoque común consiste en forzar que el vector de pesos del clasificador lineal final sea una combinación lineal de la versión mapeada de las muestras de entrenamiento [13, 30, 87]. Afortunadamente, existen resultados teóricos que garantizan que, bajo condiciones razonables, el vector de pesos óptimo admite una representación de esta forma [7]. Además, esta restricción permite reescribir la función de decisión de los clasificadores evitando cualquier evaluación explícita del *feature map*, usando la función de kernel en su lugar. Sin embargo, este enfoque conocido como el *kernel trick* introduce problemas de escalabilidad tanto en fase de entrenamiento como de test. Esta falta de escalabilidad de los algoritmos que usan funciones de kernel ha motivado a los científicos para buscar formas más eficientes de combinar la eficacia discriminativa de las funciones de kernel con la escalabilidad de los clasificadores lineales.

2.3. Kernels polinómicos

Los kernels polinómicos son una popular familia de funciones de kernel no estacionarias [46], con un gran poder de discriminación y de gran aplicabilidad [21, 105, 23]. De forma intuitiva, las funciones de kernel polinómicas computan productos internos en el espacio de características formado por todos los posibles monomios de grado g sobre las características originales [49]. Estas interacciones polinómicas entre las características originales resultan a menudo de gran utilidad para resolver problemas de clasificación no linealmente separables en el espacio original. A diferencia de otras funciones de kernel populares, los kernels polinómicos tienen un *feature space* asociado de dimensión finita, que puede ser fácilmente determinado una vez que se fija el grado y constante del kernel.

Dado que los *feature spaces* de los kernels polinómicos son de dimensión finita, algunos estudios han explorado la posibilidad de operar de forma explícita en ellos [21], aprovechando la naturaleza dispersa de algunos datasets para reducir los costes computacionales. Sin embargo, la dimensión de las muestras mapeadas crece

de forma exponencial con el grado del kernel polinómico. Como consecuencia, un algoritmo que opere en el *feature space* de forma explícita se volverá rápidamente intratable a medida que crezca la dimensión original de las muestras o el grado del kernel polinómico seleccionado.

2.4. Proyecciones aleatorias desde feature spaces de funciones de kernel

Como hemos visto, los clasificadores que usan el *kernel trick* obtienen una mayor capacidad discriminativa, trabajando de forma implícita en el *feature space* de algún kernel. Sin embargo, el uso del *kernel trick* compromete la escalabilidad de los clasificadores resultantes tanto en términos de su tiempo de entrenamiento como de inferencia [109, 14]. Esto ha motivado a los investigadores para diseñar nuevos métodos que combinen el poder discriminativo de las funciones de kernel con la eficiencia de los clasificadores lineales. Un enfoque frecuente es el de diseñar un algoritmo de mapeo de características que de alguna forma capture la estructura de los datos en el espacio de características de algún kernel, al tiempo que genere una representación de salida de dimensión relativamente reducida [110, 5, 82, 69, 58, 97, 103]. Esta representación reducida es usada para entrenar clasificadores lineales, que consiguen así aproximar la eficacia de sus versiones no lineales. Cuando se diseñan algoritmos de este tipo, se suelen buscar las propiedades de eficiencia, independencia de datos y independencia del kernel.

Dado que el objetivo de estos algoritmos de mapeo de características es preservar las propiedades de los datos al tiempo que estos se transforman a una representación de menor dimensionalidad, una opción natural es considerar el uso del algoritmo *Random Projection*. En algunos de los trabajos más tempranos sobre este tema, los autores notaron que si un problema de clasificación es linealmente separable con un amplio margen en el espacio de características de un kernel, una proyección aleatoria desde el mismo preservaría la separabilidad de los datos [11, 9]. Sin embargo, en estos estudios quedó demostrado que no es posible realizar una proyección aleatoria a través del uso de una función de kernel arbitraria, salvo que también se permita acceso a la distribución de los datos. Esta posibilidad quedó sin embargo abierta para funciones de kernel específicas.

Más recientemente Alavi *et al.* [5, 110] propusieron *Kernelized Gaussian Random Projection* (KG-RP), un método general para realizar proyecciones aleatorias desde los *feature spaces* de funciones de kernel arbitrarias. Sus hallazgos no contradijeron los resultados descritos con anterioridad ya que, en efecto, su método requiere de acceso a la distribución de los datos por medio de un número de muestras de entrenamiento. Esto hace que el método sea dependiente de los datos, y que su fase de entrenamiento sea más costosa y menos sencilla que el el caso del algoritmo *Random Projection* original.

En este contexto, donde las versiones kernelizadas de *Random Projection* son dependientes de datos y computacionalmente costosas, esta tesis se centra en el desarrollo de nuevos algoritmos independientes de datos para la computación eficiente de proyecciones aleatorias desde los *feature spaces* de kernels polinómicos. Siguiendo la intuición proporcionada en [11, 9], nos centraremos en una familia de kernels específica para preservar la eficiencia e independencia de datos del algoritmo *Random Projection* original.

2.5. Redes neuronales convolucionales bilineales

Como hemos visto, la realización de proyecciones aleatorias desde los espacios de características de kernels ha emergido recientemente como una prometedora alternativa a la pobre escalabilidad de los clasificadores kernelizados tradicionales. Un ejemplo de cómo este enfoque puede resultar de interés es el de los modelos que usan *bilinear pooling*, que como veremos más adelante está íntimamente conectado con los kernels polinómicos. En esencia, *bilinear pooling* es un método diseñado para aumenta la tasa de acierto de los modelos de clasificación en tareas de reconocimiento visual de grano fino [26, 37, 95, 6, 91]. Este aumento en la tasa de acierto se logra calculando el producto de Kronecker de los vectores de características locales extraídos por dos modelos, agrupando después los vectores resultantes para obtener un descriptor global. Las Redes Neuronales Convolucionales (CNNs) que usan esta técnica suelen conocerse como CNNs bilineales [71]. El Capítulo 5 profundiza en cómo los algoritmos de proyección aleatoria para kernels polinómicos presentados en esta tesis pueden también utilizarse para hacer que los modelos que usan *bilinear pooling* sean más eficientes.

Capítulo 3

Proyecciones aleatorias desde el feature space del kernel polinómico de grado dos

Realizar una proyección aleatoria desde el feature space asociado a una función de kernel puede ser de utilidad por dos motivos principales: (1) Como consecuencia del lema de Johnson-Linden- strauss, la representación de baja dimensión resultante preservará gran parte de la estructura de los datos en el feature space del kernel y (2) un clasificador lineal eficiente entrenado sobre los datos proyectados puede aproximar la tasa de acierto de los clasificadores no lineales. En este capítulo, presentamos un nuevo método para aproximar proyecciones aleatorias desde el feature space del kernel polinómico de grado dos. Al contrario que otros enfoques de proyección aleatoria para kernels, nuestro método se centra en una familia de kernels particular para preservar las ventajas del algoritmo Random Projection original, tales como su independencia de los datos y su eficiencia. Los resultados experimentales presentados en este capítulo evidencian que el método propuesto consigue aproximar de forma eficiente la eficacia de una proyección aleatoria desde el feature space del kernel, preservando las distancias entre las muestras y permitiendo un incremento en las tasas de acierto de los clasificadores lineales.

Los contenidos de este capítulo son una adaptación del artículo de revista: Daniel López-Sánchez, Juan Manuel Corchado and Angélica González Arrieta. "Data-independent Random Projections from the feature-map of the Homogeneous Polynomial Kernel of degree two". In: Information Sciences 436-437C (2018), pp. 214-226.

3.1. Resumen del capítulo

Este capítulo presenta un nuevo algoritmo que permite aproximar de manera eficiente una proyección aleatoria de nuestros datos desde el *feature space* del kernel polinómico homogéneo de grado dos. De forma intuitiva, nuestro algoritmo se basa en la idea de reemplazar los productos escalares que aparecen en la multiplicación

de matrices del algoritmo *Random Projection* original por evaluaciones de la función de kernel. Esto equivale a mapear las muestras de datos y las columnas de la matriz de proyección al *feature space* del kernel, para calcular el producto escalar de las mismas en este espacio. Sin embargo, en general esto no es equivalente a realizar una proyección aleatoria desde el *feature space* de forma explícita, dado que no podemos garantizar que la distribución de los vectores de proyección (las columnas de la matriz de proyección) sea preservada tras es mapeo al *feature space*. Elegir una función de kernel particular nos permitirá estudiar como afecta este paso al *feature space* de los vectores de proyección, tomando las medidas necesarias para garantizar que el resultado es un set de vectores de proyección con la distribución apropiada para el algoritmo *Random Projection*.

En primer lugar, analizamos qué sucede con las entradas de los vectores de proyección cuando estos son transformados por el *feature map* del kernel polinómico homogéneo de grado dos. Gracias a las propiedades de este kernel, resulta posible diseñar una distribución para los vectores de proyección tal que, cuando estos son transformados por el *feature map*, la mayoría de sus entradas presentan una distribución apropiada para el algoritmo *Random Projection*, al menos cuando las analizamos de forma individual. Esto nos permite proponer un primer método extremadamente simple para aproximar la proyección aleatoria desde el *feature space* del kernel, que implica reemplazar los productos escalares en *Random Projection* por evaluaciones de la función de kernel, y el uso de una distribución especial para la matriz de proyección.

Aunque este primer enfoque resulta conveniente por su extrema sencillez, produce resultados significativamente inferiores a los de una proyección aleatoria explícita desde el *feature space*. Nuestros análisis muestran que esto se debe a que, aunque las entradas de los vectores de proyección mapeados al *feature space* siguen una distribución apropiada cuando se analizan individualmente, no respetan en requisito de ser estadísticamente independientes entre sí, lo que afecta negativamente a la eficacia del algoritmo. Para solucionar este problema, se propone generar los vectores de proyección en el *feature space* como la suma de t vectores independientes. De esta forma, el Teorema del Límite Central (CLT) [57] nos permite asegurar que para un valor de t lo suficientemente grande, las entradas de estos vectores seguirán una distribución normal, que está entre las distribuciones válidas para la matriz de proyección de *Random Projection*. Además, el uso de este enfoque consigue reducir la dependencia entre las entradas de los vectores de proyección, lo que permite una mejora en la eficacia de nuestro algoritmo en términos de su capacidad para preservar las distancias entre las muestras tras la proyección.

En cuanto a la complejidad computacional del método propuesto, el análisis presentado en este capítulo muestra que es de $\mathcal{O}(tdk)$ para el entrenamiento o inicialización y de $\mathcal{O}(Ndk)$ para transformar o proyectar N muestras, donde t es un hiperparámetro del algoritmo, d es la dimensión de las muestras y k es la dimensión del espacio de salida. Esto quiere decir que la escalabilidad del método propuesto en el número de muestras a transformar, su dimensión y la dimensión de salida es

similar a la del algoritmo *Random Projection* original, a diferencia de las versiones existentes de *Random Projection* para kernels que necesitan de costosas fases de entrenamiento para las que requieren acceso a la distribución de los datos.

En cuanto a los resultados experimentales, estos se centran en la capacidad del algoritmo propuesto de preservar las distancias entre las muestras de datos en el *feature space* del kernel. Se compara la eficacia del método propuesto con los algoritmos existentes para la realización de proyecciones aleatorias desde *feature spaces* de kernels, así como con el desempeño del algoritmo *Random Projection* original aplicado sobre las muestras de datos tras ser mapeadas de forma explícita al *feature space* del kernel. Además de la preservación de las distancias, nuestros resultados miden el tiempo de ejecución medio de los distintos algoritmos, tanto en su fase de inicialización o entrenamiento como en la de proyección de muestras. Para los experimentos, se usan tres datasets públicos: CIFAR-10, un dataset de imágenes de tamaño 32×32 distribuidas en diez categorías; ISOLET, un dataset donde las muestras consisten en características sonoras extraídas de la pronunciación de las letras del alfabeto por diferentes individuos; y STL-10, similar en naturaleza a CIFAR-10 pero con imágenes de mucho mayor tamaño.

En general, los resultados muestran que el método propuesto en este capítulo es la mejor alternativa, al proporcionar una buena aproximación de las propiedades del algoritmo *Random Projection* al tiempo que permite mantener unos tiempos de ejecución muy bajos. Por el contrario, los enfoques alternativos requieren de tiempos de ejecución mucho mayores, o resultan en una peor preservación de las distancias entre las muestras.

Además de los resultados sobre la preservación de distancias entre las muestras, estudiamos la tasa de acierto obtenida por clasificadores lineales sobre las representaciones resultantes de aplicar los distintos algoritmos. Como ya se ha visto, una proyección aleatoria desde el espacio de características de un kernel debería capturar la información discriminativa presente en este, permitiendo una mejora en las tasas de acierto de los clasificadores lineales, sin incurrir en los problemas de escalabilidad de los clasificadores de kernel tradicionales. Para este conjunto de experimentos, reproducimos el protocolo experimental de Chang *et al.* [21], añadiendo a la comparativa nuestro enfoque basado en la aproximación de una proyección aleatoria desde el *feature space*. Los resultados indican que nuestro método obtiene la mejor aproximación de la tasa de acierto de los clasificadores no lineales en varios de los datasets, a pesar de ser notablemente más eficiente, y mantener la independencia de datos del algoritmo *Random Projection* original.

En definitiva, los resultados experimentales confirman que el método propuesto consigue aproximar de forma eficiente las propiedades de una proyección aleatoria desde el *feature space* del kernel polinómico homogéneo de grado dos. Posiblemente, la mayor limitación de este enfoque sea su rigidez, al estar diseñado para trabajar de forma exclusiva con esta función de kernel. Sin embargo, la gran aplicabilidad y popularidad de esta función de kernel justifica el interés de nuestra propuesta.

Además, las ideas presentadas en este capítulo sirven como base para proponer métodos más eficientes y de aplicabilidad más general en los siguientes capítulos.

Capítulo 4

Proyecciones aleatorias desde los feature spaces de kernels polinómicos de grado arbitrario

Los resultados presentados en el capítulo anterior han mostrado que es posible aproximar una proyección aleatoria desde el featrue space del kernel polinómico homogéneo de grado dos de una forma eficiente e independiente de los datos. En las siguientes páginas, desarrollamos las ideas del Capítulo 3 para mejorar la generalidad, eficiencia y eficacia de nuestro enfoque para las proyecciones aleatorias con kernels. En particular, presentamos un nuevo método para realizar proyecciones aleatorias desde los espacios de características de kernels polinómicos de grado arbitrario. Los numerosos resultados experimentales presentados en este capítulo muestran que este nuevo algoritmo supera a los métodos alternativos en términos de su capacidad para preservar las distancias entre las muestras, además de ser más eficiente. Adicionalmente, los resultados muestran que el método propuesto puede ser usado para mejorar la tasa de acierto de los clasificadores lineales, aproximando en algunos casos la eficacia de los clasificadores que usan el kernel trick.

Los contenidos de este capítulo son una adaptación del artículo de revista: Daniel López-Sánchez, Angélica González Arrieta and Juan M. Corchado. "Data independent Random Projections from the feature-space of the Homogeneous Polynomial Kernel". In: Pattern Recognition (2018).

4.1. Resumen del capítulo

Tal como evidencian los resultados presentados en el Capítulo 3, es posible aproximar una proyección aleatoria desde el *feature space* del kernel polinómico homogéneo de grado dos mediante una elección cuidadosa de la distribución de la matriz de proyección y el uso del Teorema del Límite Central, así como de las propiedades de las funciones de kernel. Centrarnos en una función de kernel específica nos permitió analizar el efecto de su *feature map* al ser aplicado sobre los vectores de proyección

cuando los productos escalares presentes en la formulación de *Random Projection* se remplazan por evaluaciones de la función de kernel. Aunque este enfoque ha resultado ser razonablemente efectivo, presenta algunas limitaciones. La primera es la restricción de trabajar en exclusiva con una única función de kernel. Además, el hecho de que los vectores de proyección en el *feature space* se generasen mediante el mapeo por medio del *feature map* de las columnas de la matriz de proyección derivaba en algunos problemas, como la aparición de un número de entradas que no seguían una distribución válida para *Random Projection*. Por este motivo, es posible mejorar esta propuesta algorítmica en términos de su generalidad, eficiencia y efectividad.

En esencia, el método propuesto en este capítulo se basa en la noción de que los vectores de proyección en el *feature space* no tienen que ser necesariamente generados mediante la transformación de un único vector mediante el *feature map* del kernel. Por el contrario, es posible generar estos vectores como el producto de Kronecker de g vectores diferentes, donde g es el grado del kernel polinómico utilizado. Convenientemente, si los vectores de proyección se generan de esta forma, es posible calcular el producto escalar de los mismos con las muestras de datos en el *feature space* de una forma eficiente y sin necesidad de trabajar en este espacio de forma explícita. Al generar los vectores de proyección con el producto de Kronecker de varios vectores distintos, se elimina el problema de la aparición de elementos en los vectores finales con una distribución distinta a la deseada, lo que nos permite trabajar con kernels polinómicos de grado mayor que dos. Sin embargo, persiste el problema de la dependencia estadística entre las entradas de los vectores de proyección, que debe ser de nuevo mitigada mediante la aplicación del Teorema del Límite Central. De nuevo, los vectores de proyección finales se formarán mediante la suma de varios vectores en el *feature space*, cada uno de los cuales consiste en el producto de Kronecker de g vectores. Una cuidadosa formulación de estas operaciones nos permite evaluar el producto escalar de las muestras de datos mapeadas al *feature space* con nuestros vectores de proyección finales sin necesidad de operar de forma explícita en este espacio, lo que por su alta dimensionalidad implicaría un alto coste computacional.

Adicionalmente, este capítulo presenta una serie de trucos algorítmicos para reducir aún más el coste computacional de las operaciones realizadas por nuestro algoritmo. En particular, el algoritmo propuesto hace un uso intensivo del producto escalar entre las muestras de datos a proyectar y un conjunto de vectores aleatorios. En su formulación más simple, el algoritmo usa un total de gtk vectores aleatorios distintos, donde g es el grado del kernel polinómico, t controla el número de vectores sumados para formar los vectores de proyección finales y k determina el número de componentes de salida. Para reducir el coste computacional, es posible reutilizar algunos de estos vectores aleatorios, en lugar de usar vectores distintos para cada componente de salida. En particular, se propone generar un set con p vectores aleatorios, de forma que se seleccione un subset aleatorio de los mismos para el cálculo de cada componente de salida. Esto significa que para transformar una muestra,

nuestro algoritmo deberá evaluar p productos escalares en lugar de gtk. Por tanto, controlando el valor de p, resulta posible reducir el coste computacional y los requisitos de memoria del algoritmo propuesto, con un muy reducido impacto en su eficacia. Además, nuestro análisis muestra que los vectores aleatorios pueden ser generados tanto a partir de la clásica distribución normal estándar como usando la distribución discreta propuesta por Achlioptas [2].

De nuevo, los experimentos se centran en la capacidad de nuestro algoritmo para preservar las distancias entre las muestras de datos en el *feature space* del kernel, y en la mejora de la tasa de acierto de los clasificadores lineales al entrenarse sobre la representación generada por nuestro algoritmo. En esta ocasión, se evalúan estas propiedades tanto para el kernel polinómico homogéneo de grado dos como el de grado tres. Además, para cada experimento se mide tanto el resultado obtenido por el algoritmo correspondiente como su tiempo de ejecución en fase de inicialización y de transformación de muestras. Para los experimentos, se usan tres datasets públicos de diferentes dominios. En particular, los datasets elegidos son MNIST [108], Webspam [101] y W8a [83].

En general, los resultados experimentales evidencian que el método propuesto ofrece la mejor relación eficacia-eficiencia de entre los métodos evaluados. En particular, destaca su eficiencia en la fase de inicialización. Gracias a la independencia de datos del método propuesto, su fase de inicialización se reduce a la generación de un número de vectores aleatorios, mientras que los métodos alternativos suelen necesitar de un conjunto de datos de entrenamiento sobre el que realizan costosas operaciones. Como resultado, nuestro método reporta unos tiempos de inicialización insignificantes en comparación con los de otros métodos. La eficiencia de nuestro algoritmo queda patente también a la hora de transformar muestras, donde suele obtener resultados comparables o superiores a los métodos alternativos, manteniendo generalmente tiempos de ejecución inferiores. Estos resultados confirman también la mejora del enfoque propuesto en este capítulo respecto al descrito en el Capítulo 3, que resulta más ineficiente en la mayoría de los experimentos. Además, la posibilidad de controlar el balance eficiencia/eficacia de nuestro método mediante el ajuste de sus hiperparámetros le dota de una gran versatilidad, pudiendo seleccionarse los valores más apropiados para cada aplicación.

En definitiva, este capítulo presenta un método eficiente para la realización de proyecciones aleatorias desde los *feature spaces* de kernels polinómicos de grado arbitrario. Los resultados experimentales muestran su capacidad para aproximar las propiedades de preservación de la distancia entre muestras del algoritmo *Random Projection* original. Además, también confirman la posibilidad de usar este enfoque para generar representaciones de los datos sobre las cuales los clasificadores lineales son capaces de aproximar la tasa de acierto de los clasificadores que usan el *kernel trick*, evitando sus problemas de escalabilidad asociados. Además, las técnicas descritas en este capítulo pueden ser adaptadas para acelerar las computaciones de otros algoritmos que dependan del cálculo de características de interacción polinómica, como es el caso de las redes neuronales convolucionales bilineales. Se deja además

la puerta abierta a la posibilidad de diseñar métodos similares para la proyección aleatoria de datos desde los *feature spaces* de otras funciones de kernel populares.

Capítulo 5

Bilinear pooling compacto mediante proyecciones aleatorias kernelizadas

Las redes neuronales convolucionales bilineales, que incluyen la operación bilinear pooling como su característica principal, se encuentran entre los modelos más populares y efectivos para el reconocimiento de imagen de grano fino. Sin embargo, una de las principales desventajas de los modelos que usan bilinear pooling es la dimensión de los descriptores que generan, que suelen contener cientos de miles de características. Incluso cuando generar el descriptor en sí es computacionalmente asequible, su alta dimensionalidad hace que cualquier operación posterior se vuelva ineficiente, resultando a menudo en grades costes computacionales y de almacenamiento. En este capítulo, presentamos un nuevo método para reducir la dimensionalidad del descriptor de bilinear pooling de forma eficiente, mediante la realización de una proyección aleatoria del mismo. Convenientemente, esto se logra sin necesidad de generar explícitamente el descriptor de alta dimensionalidad. Los resultados experimentales evidencian que nuestra técnica supera a los enfoques de tipo bilinear pooling compacto en la mayoría de casos. Además, se ejecuta de forma más rápida que los métodos alternativos en dispositivos de baja potencia de cómputo, donde las variantes eficientes de bilinear pooling son más necesarias.

Los contenidos de este capítulo son una adaptación del artículo: Daniel López-Sánchez, Angélica González Arrieta and Juan M. Corchado. "Compact Bilinear Pooling via Kernelized Random Projection for Fine-Grained Image Categorization on Low Computational Power Devices". In: Neurocomputing (In press).

5.1. Resumen del capítulo

El nombre "reconocimiento de grano fino" suele usarse para referirse a tareas de clasificación de imágenes con un número relativamente alto de categorías muy similares. Estas tareas tienden a ser muy complejas, en especial por su alta variabilidad

intra-clase y su baja variabilidad iter-clase. En otras palabras, las pequeñas variaciones que contienen la información necesaria para diferenciar las distintas clases pueden ser fácilmente obviadas por la influencia de factores no informativos como la pose, orientación o iluminación de los elementos que aparecen en una imagen. Uno de los enfoques más populares para resolver este tipo de problemas es el uso de Redes Neuronales Convoluionales (CNNs) bilineales.

Originalmente propuestas por Lin *et al.* [71], las CNNs bilineales generan un descriptor global de las imágenes mediante la aplicación de dos CNNs como extractores de características locales. Después, los descriptores extraídos en cada localización de la imagen por las dos CNNs se combinan haciendo uso de producto de Kronecker. Finalmente, los descriptores resultantes se agrupan usando la media para obtener un descriptor global de la imagen, sobre el que se aplica un clasificador lineal tradicional. De esta forma, las CNNs bilineales son capaces de capturar las interacciones entre pares de características de una forma invariante a la localización, lo que propicia un aumento considerable en la tasa de acierto en problemas de clasificación de imagen de grano fino.

Sin embargo, el uso del producto de Kronecker por parte de las CNNs bilineales acarrea un problema asociado. En particular, el descriptor resultante tiene una dimensionalidad muy elevada, lo que dificulta cualquier paso posterior. Para mitigar este problema, Gao *et al.* [43] propusieron un enfoque conocido como *bilinear pooling* compacto, que se basa en el uso de técnicas de aproximación de kernels para reducir la dimensionalidad del descriptor de *bilinear pooling*. Además, Gao *et al.* sugirieron la posibilidad de usar *Random Projection* para compactar el descriptor bilineal, descartando este enfoque tras concluir que implicaría generar el descriptor de forma explícita en un primer lugar.

Este capítulo desarrolla le idea de usar *Random Projection* para reducir la dimensionalidad del descriptor de *bilinear pooling*. En particular, se propone adaptar las ideas presentadas en el Capítulo 4 para realizar esta proyección sin necesidad de generar el descriptor de forma explícita, aprovechando la relación existente entre el método *bilinear pooling* y los kernels polinómicos. Intuitivamente, nuestro método aprovecha que cuando las dos CNNs usadas para extraer los descriptores locales en *bilinear pooling* son iguales, la combinación de los descriptores locales extraídos por ellas mediante el producto de Kronecker equivale a su mapeo al *feature space* del kernel polinómico de grado dos. Por tanto, con algunas modificaciones es posible aplicar el método descrito en el Capítulo 4 para realizar una proyección aleatoria de estos descriptores, sin necesidad de usar en ningún momento el producto de Kronecker de forma explícita, evitando así las ineficiencias de *bilinear pooling*.

De nuevo, se propone usar un conjunto reducido de vectores aleatorios que se muestréa para obtener los vectores aletorios usados para el cálculo de cada componente de salida. Adicionalmente, se presenta una estrategia avanzada de muestreo que permite minimizar la reutilización de los vectores aleatorios, resultando en una ligera mejora de los resultados sin ningún coste adicional. Otro aspecto clave del

método propuesto es su compatibilidad con esquemas de entrenamiento *end-to-end*. En otras palabras, el algoritmo propuesto puede ser incluido como una capa más de un modelo de red neuronal profunda, y la red resultante podrá ser entrenada con los algoritmos tradicionales basados en el descenso de gradiente. Esto se logra mediante la derivación de las reglas de *back-propagation* para nuestro algoritmo, que permiten calcular la derivada parcial de la función de error con respecto a las entradas del algoritmo, lo que a su vez permite propagar los gradientes hacia las primeras capas de la red. Esta compatibilidad con esquemas de entrenamiento *end-to-end* permite un proceso de ajuste fino de los pesos de las redes que incorporan nuestro algoritmo.

Los experimentos presentados en este capítulo se centran en la tasa de acierto obtenida por los diferentes modelos comparados en problemas de clasificación de imágenes de grano fino. En particular, se eligieron los datasets públicos Caltech UCSD Birds-200-2011 [99] Stanford Cars Dataset [61] y 102 Category Flower Dataset [78]. Así mismo, se eligieron dos populares arquitecturas de CNN, SqueezeNet v1.1 [51] y GoogLeNet [92]. Además de las tasas de acierto obtenidas en estos datasets por los distintos algoritmos evaluados y las dos CNNs elegidas, se midió el tiempo de ejecución o inferencia de los distintos modelos en dos dispositivos hardware de la popular plataforma Raspberry. Estos dispositivos fueron la Raspberry Pi 3 Model B+ y la Raspberry Pi Zero W.

Los resultados presentados en este capítulo evidencian que el enfoque propuesto basado en la proyección aleatoria del descriptor de *bilinear pooling* supera o iguala la tasa de acierto de los métodos existentes para el *bilinear pooling* compacto, al tiempo que produce la mejor aproximación de la tasa de acierto obtenida con el uso del descriptor original. Además, el uso de nuestro método resulta en los menores tiempos de inferencia de entre todos los métodos evaluados, lo que le convierte en la opción idónea para su aplicación en escenarios donde se disponga de una baja potencia de cómputo y el tiempo de inferencia sea crítico. Además, los experimentos demuestran que la compatibilidad del método propuesto con esquemas de entrenamiento *end-to-end* permite un ajuste fino de los pesos de las CNNs utilizadas, mejorando la tasa de acierto cuando se dispone de un conjunto de datos de entrenamiento lo suficientemente grande.

En definitiva, el método propuesto permite aproximar la tasa de acierto de los modelos de tipo *bilinear pooling* tradicionales, evitando al mismo tiempo sus problemas de eficiencia. Los resultados experimentales sugieren que nuestro método es la mejor alternativa cuando se consideran los tiempos de inferencia en dispositivos de baja potencia computacional. Como linea de trabajo futuro, se propone explorar la aplicabilidad de las ideas presentadas en este capítulo a problemas de aprendizaje multi-modal, donde las dos CNNs usadas por *bilinear pooling* para la extracción de características locales son distintas, ya que procesan datos de diferente naturaleza. Además, esperamos estudiar la aplicabilidad del algoritmo propuesto en áreas como el Internet de las Cosas o los sistemas Embebidos, donde los métodos eficientes para el análisis de imagen de grano fino podrían ser de gran utilidad.

Capítulo 6

Conclusiones

A medida que la cantidad de datos producida por los humanos y dispositivos sigue creciendo, el aprendizaje automático se ha convertido en una herramienta esencial para analizar datos, hacer predicciones y automatizar tareas. Gran parte del éxito de los modelos de aprendizaje automático modernos se debe a la disponibilidad de grandes colecciones de datos de las que aprender y a los avances en el hardware que han permitido entrenar modelos con cientos de millones de parámetros. Sin embargo, existe un interés creciente en la creación de soluciones eficientes basadas en el aprendizaje automático, capaces de ejecutarse en entornos con recursos computacionales limitados, manteniendo los estándares de precisión y fiabilidad de los métodos existentes.

Un ejemplo notable de la importancia del diseño de métodos de aprendizaje automático eficientes y escalables es el caso de las técnicas de aproximación para funciones de kernel. Tal y como se vio en el Capítulo 2, en los últimos años ha existido un interés creciente en el diseño de métodos para aproximar de forma eficiente las propiedades de los datos en los espacios de características de diversas funciones de kernel, evitando los problemas de escalabilidad de los algoritmos kernelizados tradicionales.

Esta tesis ha explorado la aplicabilidad del algoritmo de Proyección Aleatoria para aproximar la estructura de los datos en el espacio de características de los kernels polinómicos. En esencia, los métodos presentados en los Capítulos 3 y 4 permiten aproximar una Proyección Aleatoria desde el espacio de características de kernels polinómicos, sin necesidad de trabajar en ningún caso en este espacio de características de forma explícita. Al contrario que los métodos existentes, los algoritmos propuestos en esta tesis son independientes de los datos, lo que significa que no requieren ningún conocimiento acerca de la distribución de los datos que van a transformar para poder operar. Como consecuencia, su fase de entrenamiento o inicialización se reduce a la inicialización de algunas matrices aleatorias, lo que los hace más eficientes y escalables que los enfoques alternativos. Esta independencia de datos se ha conseguido principalmente centrándonos en una familia de funciones de kernel específica, la de los kernel polinómicos, lo que nos ha permitido analizar la forma explícita de los mapas de características asociados a esta familia de kernels

y desarrollar métodos alternativos para aproximar una Proyección Aleatoria en el espacio de características.

Los resultados experimentales presentados se centraron en la preservación de las distancias entre las muestras de datos en el espacio de características del kernel cuando estas eran transformadas a una representación de baja dimensionalidad, y en las tasas de acierto de clasificación obtenidas sobre las representaciones generadas, demostrando que la estructura de los datos en el espacio de características es aproximadamente preservada por los métodos propuestos. Esto abrió la puerta a la utilización de estos métodos en aplicaciones donde la interacción polinómica entre características juega un papel importante. En particular, en el Capítulo 5 exploramos la aplicabilidad de los métodos presentados en los anteriores capítulos para mejorar la eficiencia de los modelos bilineales del paradigma del aprendizaje profundo. Los resultados obtenidos evidenciaron que los modelos del tipo *Bilinear Convolutional Neural Network* pueden ser acelerados mediante la aplicación de las ideas presentadas en el Capítulo 4, resultando en importantes ahorros en términos de tiempo de cómputo y memoria cuando estos modelos se ejecutan en entornos de bajas prestaciones.

En el futuro, se tratará de explorar la aplicabilidad de los métodos propuestos en problemas más allá de la categorización. Si bien los experimentos presentados en esta tesis se han centrado en el problema de la clasificación de muestras, la información presente en el espacio de características de los kernels polinómicos puede ser útil para otras tareas de aprendizaje automático tales como la recuperación de información, la regresión of el clustering. De hecho, las ideas presentadas en esta tesis podrían ser utilizadas para acelerar modelos de aprendizaje automático existentes cuando estos se basen en los kernels polinómicos o en general en la interacción polinómica entre características.

Como conclusión final, las proyecciones aleatorias desde los espacios de características de los kernels polinómicos han demostrado ser un enfoque excelente para generar de forma eficiente una representación compacta de los datos que captura la información útil proporcionada por el kernel, dando lugar a una poderosa herramienta para crear modelos de aprendizaje automático eficientes y escalables.

Bibliography

[1] Martin Abadi et al. "TensorFlow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[2] Dimitris Achlioptas. "Database-friendly random projections". In: *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2001, pp. 274–281.

[3] Dimitris Achlioptas. "Database-friendly random projections: Johnson-Lindenstrauss with binary coins". In: *Journal of computer and System Sciences* 66.4 (2003), pp. 671–687.

[4] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. "Internet of things: A survey on enabling technologies, protocols, and applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376.

[5] Azadeh Alavi, Arnold Wiliem, Kun Zhao, Brian C Lovell, and Conrad Sanderson. "Random projections on manifolds of symmetric positive definite matrices for image classification". In: *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*. IEEE. 2014, pp. 301–308.

[6] Ahmad Alzu'bi, Abbes Amira, and Naeem Ramzan. "Content-based image retrieval with compact deep convolutional features". In: *Neurocomputing* 249 (2017), pp. 95–105.

[7] Andreas Argyriou, Charles A Micchelli, and Massimiliano Pontil. "When is there a representer theorem? Vector versus matrix regularizers". In: *Journal of Machine Learning Research* 10.Nov (2009), pp. 2507–2529.

[8] Rosa I Arriaga and Santosh Vempala. "An algorithmic theory of learning: Robust concepts and random projection". In: *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE. 1999, pp. 616–623.

[9] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. "Kernels as features: On kernels, margins, and low-dimensional mappings". In: *Machine Learning* 65.1 (2006), pp. 79–94.

[10] Ella Bingham and Heikki Mannila. "Random projection in dimensionality reduction: applications to image and text data". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 245–250.

[11] Avrim Blum. "Random projection, margins, kernels, and feature-selection". In: *Subspace, Latent Structure and Feature Selection*. Springer, 2006, pp. 52–68.

[12] Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. "Fast kernel classifiers with online and active learning". In: *Journal of Machine Learning Research* 6.Sep (2005), pp. 1579–1619.

[13] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 144–152.

[14] Léon Bottou and Chih-Jen Lin. "Support vector machine solvers". In: *Large scale kernel machines* 3.1 (2007), pp. 301–320.

[15] Christos Boutsidis, Anastasios Zouzias, and Petros Drineas. "Random projections for k-means clustering". In: *Advances in Neural Information Processing Systems*. 2010, pp. 298–306.

[16] Steve Branson, Grant Van Horn, Serge Belongie, and Pietro Perona. "Bird species categorization using pose normalized deep convolutional nets". In: *arXiv preprint arXiv:1406.2952* (2014).

[17] Leo Breiman. In: *Probability*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM), 1992, pp. 237–238. ISBN: 978-0-89871-296-4.

[18] Theophilos Cacoullos. In: *Exercises in probability*. Springer, 1989, pp. 89–90. DOI: 10.1007/978-1-4612-4526-1.

[19] Ângelo Cardoso and Andreas Wichert. "Iterative random projections for high-dimensional data clustering". In: *Pattern Recognition Letters* 33.13 (2012), pp. 1749–1755.

[20] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM Data: Classification (Binary Class)*. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html. 2017.

[21] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. "Training and testing low-degree polynomial data mappings via linear SVM". In: *Journal of Machine Learning Research* 11.Apr (2010), pp. 1471–1490.

[22] Moses Charikar, Kevin Chen, and Martin Farach-Colton. "Finding frequent items in data streams". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2002, pp. 693–703.

[23] Dapeng Chen, Zejian Yuan, Gang Hua, Nanning Zheng, and Jingdong Wang. "Similarity learning on an explicit polynomial kernel feature map for person re-identification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1565–1573.

[24] Dapeng Chen, Zejian Yuan, Jingdong Wang, Badong Chen, Gang Hua, and Nanning Zheng. "Exemplar-guided similarity learning on polynomial kernel feature map for person re-identification". In: *International Journal of Computer Vision* 123.3 (2017), pp. 392–414.

[25] François Chollet et al. *Keras*. https://keras.io. 2015.

[26] Aruni Roy Chowdhury, Tsung-Yu Lin, Subhransu Maji, and Erik Learned-Miller. "One-to-many face recognition with bilinear cnns". In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2016, pp. 1–9.

[27] Dejun Chu, Changshui Zhang, and Qing Tao. "A faster cutting plane algorithm with accelerated line search for linear SVM". In: *Pattern Recognition* 67 (2017), pp. 127–138.

[28] Adam Coates, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 215–223.

[29] Ronan Collobert, Samy Bengio, and Yoshua Bengio. "A parallel mixture of SVMs for very large scale problems". In: *Advances in Neural Information Processing Systems*. 2002, pp. 633–640.

[30] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[31] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[32] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. "Large-scale malware classification using random projections and neural networks". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 3422–3426.

[33] Sanjoy Dasgupta and Anupam Gupta. "An elementary proof of a theorem of Johnson and Lindenstrauss". In: *Random Structures & Algorithms* 22.1 (2003), pp. 60–65.

[34] *DATA NEVER SLEEPS 6.0. How much data is generated every minute?* Technical Report. DOMO, 2018. URL: https://www.domo.com/learn/data-never-sleeps-6?.

[35] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. "LIBLINEAR: A Library for Large Linear Classification". In: *Journal of Machine Learning Research* 9 (2008), pp. 1871–1874.

[36] Mark Fanty and Ronald Cole. "Spoken letter recognition". In: *Advances in Neural Information Processing Systems*. 1991, pp. 220–226.

[37] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. "Convolutional two-stream network fusion for video action recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1933–1941.

[38] O Fontenla-Romero, Bertha Guijarro-Berdiñas, David Martinez-Rego, Beatriz Pérez-Sánchez, and Diego Peteiro-Barral. "Online machine learning". In: *Efficiency and Scalability Methods for Computational Intellect* 27 (2013), pp. 27–55.

[39] Homa Foroughi, Nilanjan Ray, and Hong Zhang. "Robust people counting using sparse representation and random projection". In: *Pattern Recognition* 48.10 (2015), pp. 3038–3052.

[40] Dmitriy Fradkin and David Madigan. "Experiments with random projections for machine learning". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2003, pp. 517–522.

[41] Peter Frankl and Hiroshi Maehara. "The Johnson-Lindenstrauss lemma and the sphericity of some graphs". In: *Journal of Combinatorial Theory, Series B* 44.3 (1988), pp. 355–362.

[42] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. "Multimodal compact bilinear pooling for visual question answering and visual grounding". In: *arXiv preprint arXiv:1606.01847* (2016).

[43] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. "Compact bilinear pooling". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 317–326.

[44] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. "A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability". In: *Soft Computing* 13.10 (2009), pp. 959–977.

[45] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power". In: *Information Sciences* 180.10 (2010), pp. 2044–2064.

[46] Marc G Genton. "Classes of kernels for machine learning: a statistics perspective". In: *Journal of machine learning research* 2.Dec (2001), pp. 299–312.

[47] Navin Goel, George Bebis, and Ara Nefian. "Face recognition experiments with random projection". In: *Biometric Technology for Human Identification II*. Vol. 5779. International Society for Optics and Photonics. 2005, pp. 426–438.

[48] Arthur Gretton. *Introduction to RKHS, and some simple kernel algorithms*. Advanced Topics in Machine Learning, UCL CS MSc on Machine Learning. 2018.

[49] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. "Kernel methods in machine learning". In: *The annals of statistics* (2008), pp. 1171–1220.

[50] Robert V. Hogg, Joseph W. McKean, and Allen T. Craig. In: *Introduction to Mathematical Statistics, 7th Edition*. Pearson, 2013, pp. 182–183. ISBN: 978-0-321-84943-4.

[51] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[52] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM. 1998, pp. 604–613.

[53] Thorsten Joachims. "Training linear SVMs in linear time". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 217–226.

[54] William B Johnson and Joram Lindenstrauss. "Extensions of Lipschitz mappings into a Hilbert space". In: *Contemporary mathematics* 26.189-206 (1984), p. 1.

[55] Antti Juvonen and Timo Hamalainen. "An efficient network log anomaly detection system using random projection dimensionality reduction". In: *New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on*. IEEE. 2014, pp. 1–5.

[56] Ata Kabán. "Improved bounds on the dot product under random projection and random sign projection". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 487–496.

[57] Olav Kallenberg. *Foundations of modern probability*. Springer Science & Business Media, 2006.

[58] Purushottam Kar and Harish Karnick. "Random feature maps for dot product kernels". In: *Artificial Intelligence and Statistics*. 2012, pp. 583–591.

[59] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. "Novel Dataset for Fine-Grained Image Categorization". In: *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*. Colorado Springs, CO, 2011.

[60] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[61] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. "3D Object Representations for Fine-Grained Categorization". In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.

[62] Jonathan Krause et al. "The unreasonable effectiveness of noisy data for fine-grained recognition". In: *European Conference on Computer Vision*. Springer. 2016, pp. 301–320.

[63] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: *Technical Report, University of Toronto* (2009).

[64] Taku Kudo and Yuji Matsumoto. "Fast methods for kernel-based text analysis". In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*. Association for Computational Linguistics. 2003, pp. 24–31.

[65] Brian Kulis and Kristen Grauman. "Kernelized locality-sensitive hashing". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.6 (2012), pp. 1092–1104.

[66] Brian Kulis and Kristen Grauman. "Kernelized locality-sensitive hashing for scalable image search". In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 2130–2137.

[67] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. "Sampling methods for the Nyström method". In: *Journal of Machine Learning Research* 13.Apr (2012), pp. 981–1006.

[68] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: A LLVM-based python JIT compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. ACM. 2015, p. 7.

[69] Quoc Le, Tamás Sarlós, and Alex Smola. "Fastfood-approximating kernel expansions in loglinear time". In: *Proceedings of the international conference on machine learning*. Vol. 85. 2013.

[70] Ping Li, Trevor J Hastie, and Kenneth W Church. "Very sparse random projections". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 287–296.

[71] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. "Bilinear cnn models for fine-grained visual recognition". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1449–1457.

[72] Daniel López-Sánchez, Angélica González Arrieta, and Juan M Corchado. "Data-independent Random Projections from the feature-space of the Homogeneous Polynomial Kernel". In: *Pattern Recognition* (2018).

[73] Daniel López-Sánchez, Juan Manuel Corchado, and Angélica González Arrieta. "Data-independent Random Projections from the feature-map of the Homogeneous Polynomial Kernel of degree two". In: *Information Sciences* 436-437C (2018), pp. 214–226.

[74] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. "Fine-grained visual classification of aircraft". In: *arXiv preprint arXiv:1306.5151* (2013).

[75] Jiří Matoušek. "On variants of the Johnson–Lindenstrauss lemma". In: *Random Structures & Algorithms* 33.2 (2008), pp. 142–156.

[76] James Mercer. "Xvi. functions of positive and negative type, and their connection the theory of integral equations". In: *Philosophical transactions of the royal society of London. Series A*. 209.441-458 (1909), pp. 415–446.

[77] Alexander McFarlane Mood. In: *Introduction to the Theory of Statistics, 3rd Edition*. McGraw-Hill, 1950, pp. 160–161. ISBN: 0-07-042864-6.

[78] M-E. Nilsback and A. Zisserman. "Automated Flower Classification over a Large Number of Classes". In: *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*. 2008.

[79] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[80] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[81] Jian-Xun Peng, Stuart Ferguson, Karen Rafferty, and Victoria Stewart. "A sequential algorithm for sparse support vector classifiers". In: *Pattern Recognition* 46.4 (2013), pp. 1195–1208.

[82] Ninh Pham and Rasmus Pagh. "Fast and scalable polynomial kernels via explicit feature maps". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2013, pp. 239–247.

[83] John C Platt. "Fast training of support vector machines using sequential minimal optimization". In: *Advances in kernel methods* (1999), pp. 185–208.

[84] Ali Rahimi and Benjamin Recht. "Random features for large-scale kernel machines". In: *Advances in neural information processing systems*. 2008, pp. 1177–1184.

[85] David Reinsel, John Gantz, and John Rydning. *The Digitization of the World. From Edge to Core*. Technical Report. IDC, 2018. URL: https://www.seagate.com/our-story/data-age-2025/.

[86] Bernhard Schölkopf. "The kernel trick for distances". In: *Advances in neural information processing systems*. 2001, pp. 301–307.

[87] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. "A generalized representer theorem". In: *International conference on computational learning theory*. Springer. 2001, pp. 416–426.

[88] Amnon Shashua. "Introduction to machine learning: Class notes 67577". In: *arXiv preprint arXiv:0904.3664* (2009).

[89] Qinfeng Shi, Chunhua Shen, Rhys Hill, and Anton van den Hengel. "Is margin preserved after random projection?" In: *arXiv preprint arXiv:1206.4651* (2012).

[90] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[91] Qiule Sun, Qilong Wang, Jianxin Zhang, and Peihua Li. "Hyperlayer Bilinear Pooling with application to fine-grained categorization and image retrieval". In: *Neurocomputing* 282 (2018), pp. 174–183.

[92] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[93] Gábor J Székely, Maria L Rizzo, Nail K Bakirov, et al. "Measuring and testing dependence by correlation of distances". In: *The Annals of Statistics* 35.6 (2007), pp. 2769–2794.

[94] Joshua B Tenenbaum and William T Freeman. "Separating style and content with bilinear models". In: *Neural computation* 12.6 (2000), pp. 1247–1283.

[95] Evgeniya Ustinova, Yaroslav Ganin, and Victor Lempitsky. "Multi-region bilinear convolutional neural networks for person re-identification". In: *Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on*. IEEE. 2017, pp. 1–6.

[96] Charles F Van Loan. "The ubiquitous Kronecker product". In: *Journal of computational and applied mathematics* 123.1-2 (2000), pp. 85–100.

[97] Andrea Vedaldi and Andrew Zisserman. "Efficient additive kernels via explicit feature maps". In: *IEEE transactions on pattern analysis and machine intelligence* 34.3 (2012), pp. 480–492.

[98] Santosh S Vempala. *The random projection method*. Vol. 65. American Mathematical Soc., 2005.

[99] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. *The Caltech-UCSD Birds-200-2011 Dataset*. Tech. rep. CNS-TR-2011-001. California Institute of Technology, 2011.

[100] Robert Ware and Frank Lad. "Approximating the distribution for sums of products of normal variables". In: *Technical Report, University of Canterbury* 15 (2003), pp. 1–50.

[101] Steve Webb, James Caverlee, and Calton Pu. "Introducing the Webb Spam Corpus: Using Email Spam to Identify Web Spam Automatically". In: *Proceedings of the 3rd Conference on Email and AntiSpam (CEAS)*. 2006.

[102] Eric W. Weisstein. *Normal Product Distribution. From MathWorld, A Wolfram Web Resource.* mathworld.wolfram.com. 2017.

[103] Christopher KI Williams and Matthias Seeger. "Using the Nyström method to speed up kernel machines". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. MIT press. 2000, pp. 661–667.

[104] Qi Xu, Ming Zhang, Zonghua Gu, and Gang Pan. "Overfitting remedy by sparsifying regularization on fully-connected layers of CNNs". In: *Neurocomputing* 328 (2019), pp. 69–74.

[105] Zhe Xu, Shaoli Huang, Ya Zhang, and Dacheng Tao. "Augmenting strong supervision using web data for fine-grained categorization". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2524–2532.

[106] Sibel Yaman and Jason Pelecanos. "Using polynomial kernel support vector machines for speaker verification". In: *IEEE Signal Processing Letters* 20.9 (2013), pp. 901–904.

[107] Sibel Yaman, Jason Pelecanos, and Mohamed Kamal Omar. "On the use of non-linear polynomial kernel svms in language recognition". In: *Thirteenth Annual Conference of the International Speech Communication Association*. 2012.

[108] Christopher JC Burges Yann LeCun Corinna Cortes. *The MNIST database of handwritten digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[109] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. "Recent advances of large-scale linear classification". In: *Proceedings of the IEEE* 100.9 (2012), pp. 2584–2603.

[110] Kun Zhao, Azadeh Alavi, Arnold Wiliem, and Brian C Lovell. "Efficient clustering on Riemannian manifolds: A kernelised random projection approach". In: *Pattern Recognition* 51 (2016), pp. 333–345.

[111] Kun Zhao, Arnold Wiliem, and Brian Lovell. "Kernelised orthonormal random projection on grassmann manifolds with applications to action and gait-based gender recognition". In: *Identity, Security and Behavior Analysis (ISBA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 1–6.

[112] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. "Learning deep features for scene recognition using places database". In: *Advances in neural information processing systems*. 2014, pp. 487–495.