Structured Data Modeling with Deep Kernel Machines and Applications in Computational Biology

Dexiong Chen

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques Appliquées

Arrêté ministériel : 25 mai 2016

Présentée par

Dexiong CHEN

Thèse dirigée par **Julien MAIRAL**, chercheur, Université Grenoble Alpes
et codirigée par **Laurent JACOB**, CNRS

préparée au sein du **Laboratoire Laboratoire Jean Kuntzmann** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Modélisation de données structurées avec des machines profondes à noyaux et des applications en biologie computationnelle

Structured Data Modeling with Deep Kernel Machines and Applications in Computational Biology

Thèse soutenue publiquement le **15 décembre 2020**,
devant le jury composé de :

Monsieur JULIEN MAIRAL
CHARGE DE RECHERCHE HDR, INRIA CENTRE DE GRENOBLE RHÔNE-ALPES, Directeur de thèse
Madame CHLOE-AGATHE AZENCOT
MAITRE DE CONFERENCES HDR, MINES PARISTECH, Rapporteure
Monsieur KARSTEN BORGWARDT
PROFESSEUR, ETH ZURICH - SUISSE, Rapporteur
Monsieur ARTHUR GRETTON
PROFESSEUR, UNIV. COLLEGE DE LONDRES - ROYAUME-UN, Examinateur
Madame FLORENCE D'ALCHE-BUC
PROFESSEUR, TELECOM PARISTECH, Présidente

Structured Data Modeling with Deep Kernel Machines and Applications in Computational Biology

Dexiong Chen

March 10, 2021

Abstract

Developing efficient algorithms to learn appropriate representations of structured data, including sequences or graphs, is a major and central challenge in machine learning. To this end, deep learning has become popular in structured data modeling. Deep neural networks have drawn particular attention in various scientific fields such as computer vision, natural language understanding or biology. For instance, they provide computational tools for biologists to possibly understand and uncover biological properties or relationships among macromolecules within living organisms. However, most of the success of deep learning methods in these fields essentially relies on the guidance of empirical insights as well as huge amounts of annotated data. Exploiting more data-efficient models is necessary as labeled data is often scarce.

Another line of research is kernel methods, which provide a systematic and principled approach for learning non-linear models from data of arbitrary structure. In addition to their simplicity, they exhibit a natural way to control regularization and thus to avoid overfitting. However, the data representations provided by traditional kernel methods are only defined by simply designed hand-crafted features, which makes them perform worse than neural networks when enough labeled data are available. More complex kernels inspired by prior knowledge used in neural networks have thus been developed to build richer representations and thus bridge this gap. Yet, they are less scalable. By contrast, neural networks are able to learn a compact representation for a specific learning task, which allows them to retain the expressivity of the representation while scaling to large sample size. Incorporating complementary views of kernel methods and deep neural networks to build new frameworks is therefore useful to benefit from both worlds. In this thesis, we build a general kernel-based framework for modeling structured data by leveraging prior knowledge from classical kernel methods and deep networks. Our framework provides efficient algorithmic tools for learning representations without annotations as well as for learning more compact representations in a task-driven way.

The first contribution consists of introducing a new convolutional kernel for biological sequences. The proposed kernel is aimed at tackling motif discovery, an important problem in sequence analysis, by assuming genetic determinants to be contiguous. Our method allows efficient prediction from relatively short sequences while providing simple interpretation, through the lens of kernel approximation methods. It is shown to be effective on transcription factor binding prediction and protein homology recognition tasks.

The next contribution presents a natural extension of the above kernel to further modeling sequence gaps, another type of prior knowledge in biological sequences, based on substring kernels. In this work, the above contiguous assumption on motifs is relaxed to allowing gaps, which can be helpful for many bioinformatics tasks such as protein homology recognition. Indeed, our model achieves better performance in this task, especially for predicting remote homologies. Furthermore, our model can be viewed as a new type of recurrent neural networks (RNNs), which uncovers links between many existing deep models and kernel methods. Consequently, it opens the door to better regularization and architecture design of RNNs.

We then consider graph-structured data and give a general view of many existing graph kernels based on substructure counting. We introduce a new multilayer kernel

based on fixed-length paths in the graphs. Rather than neighbor feature aggregation used in graph neural networks, our approach relies on aggregating path features, which makes it more expressive. By leveraging kernel approximation techniques, the resulting graph representations can be learned without supervision, or in a task-driven way as graph neural networks. Moreover, controlling the length of paths in our model allows compromising computational complexity and expressiveness. Our work gives a novel view to designing more expressive kernels or deep networks for graph-structured data.

Most of the kernels or deep networks so far for structured data can be decoupled into two components: local feature extraction and feature aggregation. While the above works were focused on the former component (generally combined with simple feature aggregation such as average pooling), our last contribution is devoted to feature aggregation. We introduce a kernel for sets based on Wasserstein distance, along with an explicit parametrized embedding function. Our approach addresses the problem of feature aggregation for sets with positional information that exhibit long-range dependencies between their members, which is of high importance for modeling long genetic sequences. We propose both supervised and unsupervised methods to learn the parameters in the embedding function. Empirically, our embedding combined with the above proposed kernels brings further improvement on protein homology recognition. We also provide an implementation of our embedding that can be used as a module in deep networks, which is shown to be very effective in the detection of chromatin profiles for genetic sequences.

All these contributions have led to the development of an open-source software package.

Keywords: machine learning, kernel methods, bioinformatics, deep learning, convolutional neural networks, kernels for structured data, string kernels, graph kernels, feature aggregation.

Résumé

Le développement d'algorithmes efficaces pour apprendre des représentations appropriées des données structurées, telles des sequences ou des graphes, est un défi majeur et central de l'apprentissage automatique. Pour atteindre cet objectif, l'apprentissage profond est devenu populaire pour modéliser des données structurées. Les réseaux de neurones profonds ont attiré une attention particulière dans divers domaines scientifiques tels que la vision par ordinateur, la compréhension du langage naturel ou la biologie. Par exemple, ils fournissent aux biologistes des outils de calcul qui leur permettent de comprendre et de découvrir les propriétés biologiques ou les relations entre les macromolécules des organismes vivants. Toutefois, leur succès dans ces domaines repose essentiellement sur des connaissances empiriques ainsi que d'énormes quantités de données annotées. Exploiter des modèles plus efficaces est nécessaire car les données annotées sont souvent rares.

Un autre axe de recherche est celui des méthodes à noyaux, qui fournissent une approche systématique et fondée sur des principes théoriquement solides pour l'apprentissage de modèles non linéaires à partir de données de structure arbitraire. Outre leur simplicité, elles présentent une manière naturelle de contrôler la régularisation et ainsi d'éviter le surapprentissage. Cependant, les représentations de données fournies par les méthodes à noyaux ne sont définies que par des caractéristiques artisanales simplement conçues, ce qui les rend moins performantes que les réseaux de neurones lorsque suffisamment de données étiquetées sont disponibles. Des noyaux plus complexes, inspirés des connaissances préalables utilisées dans les réseaux de neurones, ont ainsi été développés pour construire des représentations plus riches et ainsi combler cette lacune. Pourtant, ils sont moins adaptatifs. Par comparaison, les réseaux de neurones sont capables d'apprendre une représentation compacte pour une tâche d'apprentissage spécifique, ce qui leur permet de conserver l'expressivité de la représentation tout en s'adaptant à une grande taille d'échantillon. Il est donc utile d'intégrer les vues complémentaires des méthodes à noyaux et des réseaux de neurones profonds pour construire de nouveaux cadres afin de bénéficier du meilleur des deux mondes. Dans cette thèse, nous construisons un cadre général basé sur les noyaux pour la modélisation des données structurées en tirant parti des connaissances préalables des méthodes à noyaux classiques et des réseaux profonds. Notre cadre fournit des outils algorithmiques efficaces pour l'apprentissage de représentations sans annotations ainsi que pour l'apprentissage de représentations plus compactes de manière supervisée par les tâches.

La première contribution consiste à introduire un nouveau noyau convolutif pour les séquences biologiques. Le noyau proposé vise à aborder la découverte de motifs, un problème important dans l'analyse de séquence, en supposant que les déterminants génétiques sont contigus. Notre méthode permet une prédiction efficace à partir de séquences relativement courtes tout en fournissant une interprétation simple, à travers le prisme des méthodes d'approximation du noyau. Elle s'est avérée efficace pour la prédiction de liaison de facteurs de transcription et les tâches de reconnaissance de l'homologie des protéines.

La contribution suivante présente une extension naturelle du noyau ci-dessus pour modéliser davantage les sauts génétiques, un autre type de connaissances a priori sur les séquences biologiques, basé sur des noyaux de sous-chaînes. Dans ce travail, l'hypothèse contiguë ci-dessus sur les motifs est assouplie pour permettre des sauts, ce qui peut

être utile pour de nombreuses tâches bio-informatiques telles que la reconnaissance de l'homologie des protéines. En effet, notre modèle atteint de meilleures performances dans cette tâche, notamment pour la prédiction d'homologies à distance. De plus, notre modèle peut être considéré comme un nouveau type de réseaux de neurones récurrents (RNN), qui permet de découvrir des liens entre de nombreux modèles profonds existants et des méthodes à noyaux. Par conséquent, il ouvre la porte à une meilleure régularisation et à une meilleure conception de l'architecture des RNN.

Nous considérons ensuite les données structurées en graphes et donnons une vue générale de nombreux noyaux de graphes existants basés sur le comptage de sous-structures. Nous introduisons un nouveau noyau à multicouche basé sur des chemins de longueur fixe dans les graphes. Plutôt que l'agrégation des caractéristiques dans un voisinage utilisée dans les réseaux de neurones de graphes, notre approche repose sur l'agrégation des caractéristiques des chemins, ce qui la rend plus expressive. En exploitant des techniques d'approximation du noyau, les représentations de graphes résultantes peuvent être apprises sans supervision, ou de manière supervisée par les tâches comme dans les réseaux de neurones de graphes. De plus, contrôler la longueur des chemins dans notre modèle permet de compromettre la complexité de calcul et l'expressivité. Notre travail donne une nouvelle vision de la conception de noyaux plus expressifs ou de réseaux profonds pour des données structurées en graphes.

Jusqu'à présent, la plupart des noyaux ou réseaux profonds pour les données structurées peuvent être découplés en deux composants : l'extraction de caractéristiques locales et l'agrégation de caractéristiques. Alors que les travaux ci-dessus se sont concentrés sur la première composante (généralement combinée avec une simple agrégation de caractéristiques), notre dernière contribution est consacrée à l'agrégation de caractéristiques. Nous introduisons un noyau pour les ensembles basé sur la distance de Wasserstein, ainsi qu'une fonction de description explicitement paramétrée. Notre approche aborde le problème de l'agrégation de caractéristiques pour les ensembles avec des informations de position qui présentent des dépendances à longue distance entre leurs membres, ce qui est d'une grande importance pour la modélisation de longues séquences génétiques. Nous proposons des méthodes à la fois supervisées et non supervisées pour l'apprentissage des paramètres de la fonction de description. Empiriquement, notre fonction de description combinée avec les noyaux proposés ci-dessus apporte une amélioration supplémentaire sur la reconnaissance de l'homologie des protéines. Nous fournissons également une implémentation de notre intégration qui peut être utilisée comme module dans des réseaux profonds, ce qui s'avère très efficace dans la détection des profils de chromatine pour les séquences génétiques.

Chacun de ces travaux a fait l'objet du développement d'un logiciel open source.

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my PhD advisors Julien Mairal and Laurent Jacob. I really feel fortunate to have had Julien and Laurent to work with during my PhD study and research, and I believe that the accomplishment of the thesis would not have been possible without them. To Julien, his dynamism, patience and immense knowledge have deeply inspired me. I appreciated every remark he made during our discussions that was crucial but easy to overlook. To Laurent, his rigorousness, patience, enthusiasm have also greatly influenced me. Though I did not visit his lab that often, I appreciated every discussion with him, detailed and insightful. Both of them have provided me with invaluable guidance and taught me research methodology that will be useful for my whole life.

Besides my supervisors, I would like to thank the rest of the thesis committee: Florence d'Alché-Buc, Arthur Gretton and particularly the manuscript reviewers Chloé-Agathe Azencott and Karsten Borgwardt, for their encouragement, insightful comments and interesting discussions.

My sincere thanks also goes to the researchers and administrative assistants at Thoth team in Grenoble and LBBE team in Lyon, including Cordelia Schmid, Karteek Alahari, Jakob Verbeek, Jocelyn Chanussot, Pierre Gaillard, Nathalie Gillot and Franck Picard, for providing excellent working conditions and organizing lots of team activities. I would also like to thank a couple of my colleagues that I am fortunate to have collaborated with: Grégoire Mialon, Alberto Bietti, Ghislain Durif and François Gindraud. I am grateful to Eric Moulines for introducing me to Julien in 2016, so that I could work with Julien and Laurent for my master internship and pursue my PhD with them. I would like to thank all my fellow labmates at Inria and at LBBE. To my officemates Daan Wynen who taught me a few very useful programming and project organization tricks, and Vladyslav Sydorov for entertaining topics during tea breaks. To Mathilde Caron, Thomas Lucas and Valentin Gabeur for wonderful memories in Grenoble and Italy. To Nikita Dvornik, Mariia Vladimirova, Minttu Alakuijala, Alexandre Zouaoui, Ricardo Garcia, Xavier Martin, Andrei Kulunchakov, Magali Jaillard, Antoine Villié, Bruno Lecouat, Alexander Pashevich, Lina Mezghani, Ekaterina Iakovleva, Florent Bartoccioni, Adria Ruiz, Hongzhou Lin, Guosheng Hu, Xiaojie Guo, Jiajie Wang, Jiabin Chen and all my friends in Grenoble, Lyon and Paris, that I cannot possibly name all of them, for making my PhD years some of the most memorable years of my life.

Last but not least, I would like to thank my family for giving me continuous spiritual support and encouragement throughout my PhD.

Contents

1

Introduction

Contents

With the advancements in technology, today's real-world data comes from various fields and with diverse modalities. Examples include images and videos in computer vision or biological sequences and molecules in bioinformatics. One of the major problems in machine learning is then to build a general framework for finding good representations of data with arbitrary complex structure. Such good representations are expected to be predictive for a specific task or more generally for multiple tasks at the same time, which is challenging.

To that effect, deep learning models have become the most popular methods for learning representations of high-dimensional structured data in computer vision, natural language understanding, and have also drawn increasing attention in bioinformatics. For instance, they have shown good performance for making predictions from genomic sequences about their chromatin profiles (Alipanahi et al., 2015; Zhou and Troyanskaya, 2015), splicing outcome (Jha et al., 2017), gene expression profiling (Chen et al., 2016) or from protein sequences about their homology (Hochreiter et al., 2007; Asgari and Mofrad, 2015), secondary structure (Wang et al., 2016) or more recently from molecular graphs about their chemical properties (Duvenaud et al., 2015; Kearnes et al., 2016). Nevertheless, most of these successes essentially rely on the guidance of empirical insights (possibly from other fields) and huge amounts of annotated data. The links between network architectures and prior knowledge implemented in classical statistical and machine learning methods have not been well established yet.

A priori smoothness assumptions are generally required to make learning models achieve good generalization performance (ability to predict on unseen data), especially in the small-data regime which is often the case for many biological applications. In deep learning models, these assumptions have led to various heuristics, such as weight decay (Hanson and Pratt, 1989), early stopping (Prechelt, 1998) or Dropout (Srivastava et al., 2014). While these algorithmic techniques for deep networks were mostly designed with practical insights, more natural and systematical assumptions were formalized in statistics and classical machine learning methods, often referred to as the term *regularization*. Among them, Tikhonov (Tikhonov and Arsenin, 1977) and Lasso (Tibshirani, 1996) regularizations are the most typical ones. With the widespread use of deep neural networks in many scientific fields, bridging the gap between the traditional

regularization formalism and smoothness assumptions on deep models to make models more data-efficient is of growing importance. Several contributions in the thesis are thus aimed at this.

Interpretability is another important concept for learning models in scientific applications. A good prediction model should not only perform well but also be able to explain its predictions. This concept has been considered as a major limitation in many machine learning models and many efforts have been made to learn more interpretable models. For instance in deep learning, the space of functions described by a network is only characterized by its architecture design. Hence, the subsequent analysis and interpretation is often difficult and possibly only accessible for models with simple architectures (Alipanahi et al., 2015; Shrikumar et al., 2017a; Lanchantin et al., 2017). Learning models that perform well while being interpretable is becoming the central topic of a large amount of work.

Kernel methods were widely used to represent structured data before the deep learning revolution. Their algorithms and statistical properties are typically well founded on convex optimization and statistical theory (Hofmann et al., 2008). They provide a systematic and principled approach for learning non-linear predictors from data with arbitrary structure by simply defining a pairwise similarity function on input data named *kernel*. For instance, many kernels have been proposed to handle data of different modalities in biology, including string kernels (Leslie et al., 2001, 2004) and alignment-based kernels (Saigo et al., 2004) for sequence data, substructure-counting-based kernels (Gärtner et al., 2003; Borgwardt and Kriegel, 2005; Shervashidze et al., 2011) for graph data. Besides their versatility to diverse data structures, kernel methods also exhibit a natural way to control regularization and thus to avoid overfitting. In fact, each kernel defines a Hilbert space of functions (called reproducing kernel Hilbert space) in which the prediction function is learned and its smoothness can be controlled through the penalization of the norm. Despite these fruitful properties of kernel methods, the data representations are only defined by simply designed hand-crafted kernels, which has become the main limitation of many traditional kernels. For this reason, more complex kernels that leverage the multi-scale and hierarchical structure inspired by the operations in deep convolutional networks have been proposed to build richer representations and thus alleviate this limitation, especially for image data (Daniely et al., 2016; Mairal et al., 2014; Mairal, 2016). Yet, computing exact kernels is not scalable. By contrast, neural networks are able to learn a more compact representation for a specific learning task, which allows them to retain the expressivity of the representation while scaling to large sample size. Moreover, the performance gap between kernel methods and deep networks is still important for other data types, such as sequences and graphs. Hence, incorporating their complementary views to develop a general kernel framework for handling various data structures is helpful for benefitting from both worlds.

In this thesis, we present the work following the above lines of research, with a particular focus on biological applications. We build a general kernel-based framework for learning representations of structured data such as sequences and graphs, by leveraging the prior domain knowledge from classical kernel methods. In our framework, efficient algorithmic tools are provided for learning representations without labels as in classical kernel methods. More compact data representations can also be learned in a task-driven way just as deep networks, which pronouncedly reduces the gap between kernel methods

and deep networks. We apply our models to various important applications in computational biology, including transcription factor binding prediction, protein homology detection and several molecular classification tasks. These contributions are presented in more detail in Section 1.1. The other sections of the chapter provide an overview of the key elements involved in the next chapters. Specifically, we review the basic notions of kernel methods in Section 1.2 and present some widely-used kernels for structured data in Section 1.3. Section 1.4 considers deep neural networks for structured data. And finally some background on molecular biology will be presented in Section 1.5.

1.1. Contributions of the Thesis

This thesis brings several contributions to the fields of structured data modeling, especially for sequence- and graph-structured data in computational biology. These contributions are organized as follows, and each of them has led to the development of an open-source software package.

- Chapter 2 introduces a new convolutional kernel, called CKN-seq, for biological sequences, by extending the previous work (Mairal, 2016) for images. Our kernel was built to tackle the important problem in sequence analysis called *motif discovery*, or more generally to model the genotype-phenotype relationship, with the assumption that genetic determinants take the form of contiguous *sequence motifs*, *i.e.*, subsequences without gaps. Our approach allows efficiently making predictions from relatively short sequences while providing simple kernel-based interpretation, through the lens of kernel approximation methods. It is shown to be effective on transcription factor binding prediction and protein homology recognition tasks. In the large-data setting, it substantially reduces the gap between classical string kernels (Leslie et al., 2001, 2004) and deep convolutional networks (Alipanahi et al., 2015). On the other hand, it outperforms convolutional networks in the small-data regime.

  D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. In *Research in Computational Molecular Biology (RECOMB)*, 2019c
  D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. *Bioinformatics*, 35(18):3294–3302, 2019a

- Chapter 3 presents an extension of the above work to modeling sequence gaps in biological sequences, based on substring kernels. In this work, the above contiguous assumption on motifs is relaxed to allow gaps, which can be helpful for many bioinformatics tasks such as protein homology recognition. Indeed, our model achieves better performance in this task, especially for predicting remote homologies. Furthermore, our model can be viewed as a new type of recurrent neural networks (RNNs), which establishes connections between many existing deep models and kernel methods. Consequently, it opens the door to better regularization and architecture design of RNNs.

D. Chen, L. Jacob, and J. Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b

- Chapter 4 is focused on another ubiquitous data modality in biology, which is graph with node attributes. It provides a general view of many existing graph kernels based on substructure counting and introduces a new multilayer kernel based on fixed-length paths in the graphs. By leveraging the above kernel approximation techniques, the resulting graph representations can similarly be learned without supervision, or in a task-driven way. Moreover, controlling the length of paths in our model allows compromising computational complexity and expressiveness. Our work gives a novel view to designing more expressive kernels or deep networks for graphs.

D. Chen, L. Jacob, and J. Mairal. Convolutional kernel networks for graph-structured data. In *International Conference on Machine Learning (ICML)*, 2020

- Chapter 5 considers a more general circumstance where each data example is expressed as an arbitrary set of features, including biological sequences (set of characters), sentences (set of words), images (set of pixels) and graphs (set of nodes). On the one hand, we propose a valid kernel operation that simulates max pooling to aggregate the local features of such data in the RKHS. On the other hand, we introduce a kernel for these kinds of data based on an optimal transport distance, along with a parametrized embedding function. Our approach addresses the problem of feature aggregation for sets with positional information that exhibit long-range dependencies between their members, which is of high importance for long genetic sequences. According to applications, we propose both supervised and unsupervised methods to learn the parameters in the embedding function. Empirically, our embedding combined with the above CKN-seq brings further improvement on protein homology recognition. We also provide an implementation of our embedding that can be used as a module in deep networks, which shows effective in the detection of chromatin profiles.

G. Mialon*, D. Chen*, A. d'Aspremont, and J. Mairal. A trainable optimal transport embedding for feature aggregation. In *International Conference on Learning Representations (ICLR)*

(*equal contributions)

- Another contribution of the thesis consists of the following work on kernel-based regularization methods, with the collaboration of Alberto Bietti and Grégoire Mialon.

A. Bietti, G. Mialon, D. Chen, and J. Mairal. A kernel perspective for regularizing deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019

The paper proposes an algorithmic framework for regularization and robustness of neural networks by leveraging regularization techniques from kernel methods. By assuming neural networks are elements of an RKHS, we provide practical strategies to approximate the RKHS norm of a neural network and explicitly control it. Our methods are shown to be effective in practice when learning on small datasets, including the protein homology detection. Our methods can also learn models that are robust to adversarial perturbation. This work is not included in the thesis, as my contribution was essentially the application of these kernel-based regularization techniques to the problem of protein homology detection.

1.2. Kernel Methods and Large-Scale Learning with Kernels

Kernel methods were among the most popular methods in machine learning two decades ago. They provide a systematic and well-founded approach for learning from structured data. In this section, we review some useful elements of kernel methods. A more thorough introduction to the topic can be found in, *e.g.*, Scholkopf and Smola (2001); Shawe-Taylor et al. (2004); Schölkopf et al. (2004).

1.2.1. Positive Definite Kernels

In machine learning, kernel methods represent a class of algorithms to learn non-linear models from data through a pairwise similarity measure. This similarity measure, called kernel, is defined as a function over the data space $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ can be any non-empty set that includes the input examples. Most of the algorithms and theories in kernel methods are developed by restricting the kernel to be *positive definite* that carries nice properties for analysis. A positive definite kernel is defined as a *symmetric* kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that satisfies for any subset of samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$ and real-valued coefficients $a_1, \ldots, a_n \in \mathbb{R}$ that

$$\sum_{i,j=1}^{n} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$$

Such kernels are also referred to as *valid* kernels. Then any linear models (either supervised or unsupervised) only relying on pairwise dot products between data points, such as linear support vector machine (SVM), can easily be transformed to a non-linear model by replacing the dot product with a kernel. And the resulting algorithm only requires manipulating the *positive semidefinite* Gram matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ with respect to the input examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. This non-linearity is illustrated in the following theorem through the mapping Φ, which was proved in Aronszajn (1950).

Theorem 1.1 (Aronszajn, 1950)**.** *K is a positive definite kernel on the set $\mathcal{X}$ if and only if there exists a Hilbert space $\mathcal{H}$ and a mapping*

$$\Phi : \mathcal{X} \to \mathcal{H},$$

such that for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$,

$$K(\mathbf{x}, \mathbf{x}') = \left\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \right\rangle_{\mathcal{H}}. \tag{1.1}$$

An associated space $\mathcal{H}$ may be high or even infinite-dimensional, in which the transformed representations lend themselves to the learning tasks. In particular, any explicitly defined finite-dimensional feature map $\Phi : \mathcal{X} \to \mathbb{R}^d$ defines a positive definite kernel, which is often the case for most kernels used in bioinformatics such as string kernels. Some simple examples are given in Section 1.3.1.

1.2.2. Reproducing Kernel Hilbert Spaces

Among the Hilbert spaces mentioned in Aronszajn's theorem, the reproducing kernel Hilbert space (RKHS) is of the most interest, which is uniquely defined by a kernel and fully characterizes the functions we learn from. Formally, it is defined as follows.

Definition 1.1. *A Hilbert space $\mathcal{H} \subset \mathbb{R}^{\mathcal{X}}$ is a RKHS if there exists a kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that*

- *$\mathcal{H}$ contains all functions of the form $\Phi(\mathbf{x}) := K(\mathbf{x}, \cdot)$ for any $\mathbf{x} \in \mathcal{X}$.*
- *for any $\mathbf{x} \in \mathcal{X}$ and $f \in \mathcal{H}$, the reproducing property holds*

$$f(\mathbf{x}) = \langle f, \Phi(\mathbf{x}) \rangle_{\mathcal{H}}. \tag{1.2}$$

If such K exists, then it is unique and known as the reproducing kernel of $\mathcal{H}$. And its corresponding feature map Φ is called the canonical feature map of kernel mapping. Conversely, it is also possible to show that a positive definite kernel defines a unique RKHS. As the positive definite kernel and its RKHS entirely characterize each other, the kernel is mostly used for computation while the RKHS is useful for understanding the statistical properties of the functional space described by the kernel.

Interestingly, the reproducing property provides a natural way to control the smoothness of the functions described by the kernel. Specifically, applying Cauchy-Schwarz inequality to (1.2) simply implies that the variation of a function f in $\mathcal{H}$ is upper bounded by its norm in the RKHS

$$|f(\mathbf{x}) - f(\mathbf{x}')| \leq \|f\|_{\mathcal{H}} \|\Phi(\mathbf{x}) - \Phi(\mathbf{x}')\|_{\mathcal{H}}.$$

Another important remark is that the choice of feature map does not affect the geometry of the kernel since the distance between $\Phi(\mathbf{x})$ and $\Phi(\mathbf{x}')$ remains constant for any feature map Φ. However, the norm of f is directly affected by the choice of the feature map and one can reconstruct the RKHS from any explicit mapping to a Hilbert space. This is shown in the following theorem (see, *e.g.*, Saitoh, 1997, §2.1).

Theorem 1.2. *Let $\phi : \mathcal{X} \to \mathcal{F}$ be a mapping from a data space $\mathcal{X}$ to a Hilbert space $\mathcal{F}$, and let $K(\mathbf{x}, \mathbf{x}') := \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{F}}$ for $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$. Consider the Hilbert space*

$$\mathcal{H} := \{ f_{\mathbf{z}} \ : \ \mathbf{z} \in \mathcal{F} \} \quad s.t. \quad f_{\mathbf{z}} : \mathbf{x} \mapsto \langle \mathbf{z}, \phi(\mathbf{x}) \rangle_{\mathcal{F}},$$

endowed with the norm

$$\|f\|_{\mathcal{H}}^2 := \inf_{\mathbf{z} \in \mathcal{F}} \left\{ \|\mathbf{z}\|_{\mathcal{F}}^2 \quad s.t. \quad f = f_{\mathbf{z}} \right\}.$$

Then, $\mathcal{H}$ is the reproducing kernel Hilbert space associated to kernel K.

1.2.3. Learning Algorithms

As mentioned above, a linear model (either classifier or regressor) can be easily transformed to a non-linear model by replacing the inner product with a kernel function. In this way, the linear model implicitly operates on the high or even infinite-dimensional feature vectors in the RKHS associated to the kernel. This process is known as the *kernel trick*. In this section, we will review the general supervised learning problem and see how it works in the context of kernel methods.

Supervised learning problem with empirical risk minimization. In supervised learning setting, we observe some measurement (random variable) $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ following some unknown distribution $(\mathbf{x}, y) \sim \mathbb{P}$. The goal is to find a predictive function (often called *hypothesis*) $f : \mathcal{X} \to \mathcal{Y}$ minimizing the *risk*

$$\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}}[\ell(f(\mathbf{x}), y)],$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a (convex) loss function that measures the goodness of the prediction. For instance, $\ell(\hat{y}, y) = \mathbb{1}_{\hat{y}\neq y}$ for classification and $\ell(\hat{y}, y) = (\hat{y} - y)^2$ for regression.

In practice, we do not know the data distribution $\mathbb{P}$ but only have access to its approximation on a known set of training data. Specifically, we observe n independent and identically distributed training pairs $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$ following the distribution $\mathbb{P}$, and the *empirical risk* is computed by averaging the loss function on the training set

$$\mathcal{R}_{\text{emp}}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i).$$

However, there are some issues concerning this formulation. On the one hand, searching the predictive function in the entire functional space $\mathcal{Y}^{\mathcal{X}}$ is generally infeasible, restricting the search space to an easier-to-characterize one is thus needed. On the other hand, replacing the risk with the empirical risk could cause *overfitting* (Tsybakov, 2008; Wainwright, 2019), which motivates us to impose some smoothness constraint on the predictive function to control the model complexity. Taking both considerations into account leads to the following optimization problem, known as the *regularized empirical risk minimization* (ERM)

$$\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) + \lambda\Omega(f), \tag{1.3}$$

where $\mathcal{F} \subset \mathcal{Y}^{\mathcal{X}}$ is a predefined functional space based on some prior knowledge about the problem, and Ω denotes a regularization function to control the smoothness of predictive function f and λ is a regularization parameter, which can be selected for example by cross validation (Stone, 1974).

Supervised learning with kernels. In the context of kernel methods, the hypothesis space is an RKHS associated with a kernel K and the above regularized ERM problem becomes

$$\min_{f\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) + \lambda\|f\|_{\mathcal{H}}^2. \tag{1.4}$$

At the first glance, this problem seems still difficult to solve as $\mathcal{H}$ can potentially be infinite-dimensional and is not explicitly expressed. Here we will see that this problem can indeed be transformed to a problem in $\mathbb{R}^n$ only dependent of the pairwise kernel values thanks to the *representer theorem* (Kimeldorf and Wahba, 1971). The problem can then be solved using convex optimization techniques when the loss function is convex. The representer theorem states that the solution lives in a finite-dimensional space spanned by the embeddings of the input examples:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}), \quad \text{for some } \alpha_1, \dots, \alpha_n \in \mathbb{R},$$

by using a simple geometric argument. Now, if we denote by $\mathbf{K}$ the Gram matrix with entries $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ and $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^\top \in \mathbb{R}^n$, the problem is equivalent to

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} \ell([\mathbf{K}\boldsymbol{\alpha}]_i, y_i) + \lambda \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}.$$

In the case of regression, the loss function is the squared loss $\ell(\hat{y}, y) = (\hat{y} - y)^2$ and the problem is known as the *kernel ridge regression*, which admits an explicit solution that equals to

$$\boldsymbol{\alpha} = (\mathbf{K} + n\lambda \mathbf{I})^{-1} \mathbf{y} \quad \text{with } \mathbf{y} = (y_1, \dots, y_n) \in \mathbb{R}^n.$$

Despite its nice formulation, the main bottleneck of this approach is scalability to large-scale datasets. Indeed, the complexity of computing the kernel matrix is quadratic of the number of examples and solving the ERM is even cubic. This drives us to perform approximations to reduce time complexity while preserving prediction performance, as presented in the following section.

1.2.4. Large-Scale Learning with Kernels

Recent advances in stochastic convex optimization enables us to deal with large-scale linear problems (see, *e.g.*, Bottou, 2010; Mairal, 2019). This success provides us the basic elements for boosting the computation of kernel methods. In this section, we will revisit two well-known approximation techniques for kernels, namely Nyström method and random features. These techniques can make kernel methods scale well to very large datasets, while retaining their prediction performance.

Nyström method. The Nyström method essentially performs low rank approximations of the kernel matrix to accelerate its computation. The emergence of this idea dates back to Scholkopf and Smola (2001); Williams and Seeger (2001). Let us consider here a kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and its associated kernel mapping $\Phi : \mathcal{X} \to \mathcal{H}$ and RKHS $\mathcal{H}$. For any $\mathbf{x} \in \mathcal{X}$, the basic idea of the Nyström method is to replace $\Phi(\mathbf{x})$ in $\mathcal{H}$ by its orthogonal projection onto a finite-dimensional subspace of $\mathcal{H}$. In most cases, this subspace is chosen as

$$\mathcal{E} = \operatorname{span}(\Phi(\mathbf{z}_1), \dots, \Phi(\mathbf{z}_p)),$$

with some pre-selected set of *anchor points* $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_p\}$. And we define the p-dimensional vector

$$\Psi(\mathbf{x}) = K_{ZZ}^{-\frac{1}{2}} K_Z(\mathbf{x}) \in \mathbb{R}^p, \quad \text{for any } \mathbf{x} \in \mathcal{X},$$

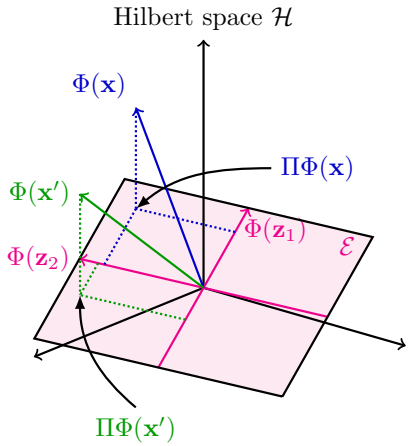


Figure 1.1.: Nyström approximation method. The feature map $\Phi(\mathbf{x})$ is replaced by its orthogonal projection onto the finite-dimensional subspace of $\mathcal{H}$ defined by $\mathcal{E} = \text{span}(\Phi(\mathbf{z}_1), \ldots, \Phi(\mathbf{z}_p))$. Then this projection $\Pi\Phi(\mathbf{x}) \in \mathcal{E}$ can be re-parametrized by $\Psi(\mathbf{x}) \in \mathbb{R}^p$ such that $\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathcal{H}} \approx \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle$ for any $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$.

where K_{ZZ} denotes the $p \times p$ Gram matrix of K restricted to the anchor points $\mathbf{z}_1, \ldots, \mathbf{z}_p$ and $K_Z(\mathbf{x}) \in \mathbb{R}^p$ carries the kernel values $K(\mathbf{x}, \mathbf{z}_j)$ for $j = 1, \ldots, p$. Then the kernel evaluation between two data points $\mathbf{x}$ and $\mathbf{x}'$ in $\mathcal{X}$ can be approximated by the inner-product between $\Psi(\mathbf{x})$ and $\Psi(\mathbf{x}')$ in $\mathbb{R}^p$:

$$K(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathcal{H}} \approx \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle,$$

since the inner-product $\langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle$ is exactly the inner-product between the orthogonal projections of $\Phi(\mathbf{x})$ and $\Phi(\mathbf{x}')$ onto $\mathcal{E}$. This is illustrated in Figure 1.1. A useful remark is that these projections onto $\mathcal{E}$ still remain in the original RKHS $\mathcal{H}$ so that any linear operations on the mapping still transform them in the RKHS. Finally, the Gram matrix computed with $\Psi(\mathbf{x})$ with respect to the full data examples is supposed to be a low-rank approximation of the exact Gram matrix for proper choice of anchor points. Then $\Psi(\mathbf{x})$ can be fed to a linear model to handle supervised learning problems or to a clustering model for unsupervised learning.

In practice, the set of anchor points can be chosen in various ways. Random sampling or greedy selection from training examples are the most typical ones (Williams and Seeger, 2001; Smola and Bartlett, 2001; Fine and Scheinberg, 2001). More recent data-dependent approach via clustering (Zhang et al., 2008; Mairal, 2016) provides significantly better performance. On the other hand, the anchor points can also be learned, but not necessarily at the aim of approximating the kernel, end-to-end with a gradient descent method when the kernel is differentiable and combined with a linear model such as SVM, presented usually in the context of *reduced-set* selection (Burges et al., 1996; Scholkopf and Smola, 2001; Wu et al., 2005) or more recently in deep kernels (Mairal, 2016).

Throughout the thesis, we use the (spherical) K-means algorithm to learn the set of anchors (Zhang et al., 2008). The approximation error of such algorithm is guaranteed by the following theorem, adapted from Zhang et al. (2008).

Theorem 1.3 (adapted from Zhang et al. (2008)). *Given a set of training data* $\mathbf{x}_1, \ldots, \mathbf{x}_n$, *the approximation error (in terms of Frobenius norm) between the exact kernel matrix and the approximate kernel matrix given by the Nyström method is defined by*

$$E^2 := \sum_{i,j=1}^{n} [K(\mathbf{x}_i, \mathbf{x}_j) - \langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}_j) \rangle]^2 = \|\mathbf{K} - K_{ZX}^\top K_{ZZ}^{-1} K_{ZX}\|_F^2,$$

where $K_{ZX} \in \mathbb{R}^{p \times n}$ *denotes the matrix with columns* $K_Z(\mathbf{x}_i)$ *for* $i = 1, \ldots, p$. *If* K *is bounded such that* $K(\mathbf{x}, \mathbf{x}) \leq C$ *for any* $\mathbf{x} \in \mathcal{X}$, *this approximation error is then bounded by*

$$E \leq 4T\sqrt{pCeT} + pCeT\|\mathbf{K}_{ZZ}^{-1}\|_F,$$

where $T \leq n$ *denotes the number of samples of the largest cluster of the data,* e *describes the quantization error of the data in* $\mathcal{H}$ *given by*

$$e := \sum_{i=1}^{n} \|\Phi(\mathbf{x}_i) - \Phi(\mathbf{z}_{s(i)})\|_{\mathcal{H}}^2,$$

with $s(i) \in \{1, \ldots, p\}$ *the index function that maps each sample* $\mathbf{x}_i$ *to the closest anchor point* $\mathbf{z}_{s(i)}$ *in* $\mathcal{H}$.

In particular, if the kernel mapping Φ is L-Lipschitz such that

$$\|\Phi(\mathbf{x}) - \Phi(\mathbf{x}')\|_{\mathcal{H}} \leq L\|\mathbf{x} - \mathbf{x}'\| \quad \text{for any } \mathbf{x}, \mathbf{x}' \in \mathcal{X},$$

which includes Gaussian kernels, polynomial kernels, dot-product kernels an so on, then e is bounded by the quantization error in the input space. Therefore, controlling the quantization error in the input space allows to control the approximation error, which justifies the validity of K-means algorithm that minimizes the quantization error.

Random features. Another class of approximations are based on random features, which are relevant for a large family of kernel functions admitting the following decomposition

$$K(\mathbf{x}, \mathbf{x}') = \mathbb{E}_{\mathbf{w} \sim p(\mathbf{w})}[\varphi(\mathbf{w}, \mathbf{x})\varphi(\mathbf{w}, \mathbf{x}')], \tag{1.5}$$

where φ is a continuous and bounded function and each $\varphi(\mathbf{x}, \mathbf{w})$ is known as a random feature (Bach, 2017), and $p(\mathbf{w})$ is a probability distribution. This family includes many commonly used kernels such as shift-invariant and rotation-invariant kernels. These kernels are of the respective form $\kappa(\mathbf{x} - \mathbf{x}')$ and $\kappa(\langle \mathbf{x}, \mathbf{x}' \rangle)$ with $\kappa : \mathbb{R} \to \mathbb{R}$ some appropriate function and can be expressed as the above form respectively using the *Fourier transform* (Rahimi and Recht, 2008) and *spherical harmonics* (Schoenberg, 1942). A natural approximation for the kernels of the form 1.5 makes use of the standard Monte Carlo sampling scheme to construct an inner-product between p-dimensional vectors:

$$K(\mathbf{x}, \mathbf{x}') \approx \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle,$$

where $\psi(\mathbf{x}) = [\varphi(\mathbf{w}_1, \mathbf{x}), \ldots, \varphi(\mathbf{w}_p, \mathbf{x})]^\top \in \mathbb{R}^p$ and $\mathbf{w}_1, \ldots, \mathbf{w}_p$ are i.i.d. sampled from $p(\mathbf{w})$. Unlike the anchor points (also called basis functions) used by the Nyström method that are sampled in a data dependent fashion, basis functions used here are sampled from a distribution independent from the data. Thus, approaches based on the Nyström method can generally yield better generalization error bound than random features based approaches (Yang et al., 2012). A more detailed survey on this topic can be found in Liu et al. (2020).

1.3. Kernels for Structured Data

There are various kernels introduced with different prior knowledge and for different data modalities. Good kernels should be appropriate for the learning task while being positive definite. In this section, we will review some of them, especially those for structured data like sequences and graphs.

1.3.1. Kernels for Real-Valued Vectors

We begin with some simple yet widely used kernels for vectors. Here all the kernels are defined over $\mathcal{X} = \mathbb{R}^d$.

Linear kernel. The simplest one should be linear kernel. This kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle = \mathbf{x}^\top \mathbf{x}', \quad \text{for any } \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d.$$

It is not hard to show the corresponding RKHS is

$$\mathcal{H} = \{ f_{\mathbf{w}} : \mathbf{x} \to \mathbf{x}^\top \mathbf{w} \,|\, \mathbf{w} \in \mathbb{R}^d \},$$

endowed with the the inner-product

$$\langle f_{\mathbf{w}}, f_{\mathbf{w}'} \rangle_{\mathcal{H}} = \mathbf{w}^\top \mathbf{w}', \quad \text{for any } f_{\mathbf{w}}, f_{\mathbf{w}'} \in \mathcal{H}.$$

Thus using this kernel with the regularized ERM problem 1.4 amounts to solving the linear problem

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{x}_i^\top \mathbf{w}, y_i) + \lambda \|\mathbf{w}\|^2.$$

This reparametrization can be quite useful for approximated kernels provided by Nyström or random features in Section 1.2.4.

Polynomial kernels. The polynomial kernel with degree k is defined as

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + c)^k,$$

where $c \geq 0$ is a free parameter. In particular when $c = 0$, the kernel is called homogeneous. Intuitively, this kernel not only takes into accounts the input features but also their higher order polynomial combinations. The advantage is that it does not require explicitly adding these higher-order features thus exploding the number of parameters to be learned. This kernel is quite popular in natural language processing (Chang et al., 2010).

Histogram intersection kernel. In contrast to the distance induced by the linear kernel which is an ℓ_2-norm in $\mathbb{R}^d$, here we present the histogram intersection kernel which induces an ℓ_1-norm in the input space. Specifically, the kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} \min(\mathbf{x}_i, \mathbf{x}'_i) \quad \text{for any } \mathbf{x}, \mathbf{x}' \in \mathbb{R}_+^d.$$

It is not hard to show the positive definiteness of the kernel by noticing

$$\min(u, v) = \int_0^\infty \mathbb{1}_{t \le u}(t)\mathbb{1}_{t \le v}(t)\,\mathrm{d}t \quad \text{for any } u, v \in \mathbb{R},$$

and using the definition of the p.d. kernel. This kernel is mostly used with histograms and show better performance than linear or RBF kernels in computer vision (Barla et al., 2003). The induced distance defined by $d_K^2(\mathbf{x}, \mathbf{x}') := K(\mathbf{x}, \mathbf{x}) - 2K(\mathbf{x}, \mathbf{x}') + K(\mathbf{x}', \mathbf{x}')$ is given by

$$d_K^2(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_1 := \sum_{i=1}^d |\mathbf{x}_i - \mathbf{x}'_i|.$$

Gaussian kernel. The most popular kernel must be the Gaussian kernel, or radial basis function (RBF) kernel. It is defined as

$$K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{\sigma^2}} \quad \text{for any } \mathbf{x}, \mathbf{x}' \in \mathbb{R},$$

where $\sigma > 0$ is called the *bandwidth* parameter. This kernel is transition-invariant or shift-invariant such that it depends only on the difference between its arguments

$$K(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x} - \mathbf{x}') \quad \text{with } \varphi(\mathbf{u}) = e^{-\|\mathbf{u}\|^2/\sigma^2}.$$

Thus the random Fourier features presented in Section 1.2.4 can be applied to this kernel to speed up the kernel computation. We can also characterize its corresponding RKHS for each $\sigma > 0$:

$$\mathcal{H}_\sigma = \Big\{ f \in L_2(\mathbb{R}^d) \,\Big|\, \int_{\mathbb{R}^d} |\hat{f}(\mathbf{w})|^2 e^{\frac{\sigma^2\|\mathbf{w}\|_2^2}{2}}\,\mathrm{d}\mathbf{w} < \infty \Big\},$$

endowed with the right term as its norm up to a constant factor. In particular, all the functions in $\mathcal{H}_\sigma$ are infinitely differentiable with all derivatives in $L_2(\mathbb{R}^2)$ owing to the exponential decay. Note that the RKHS $\mathcal{H}_\sigma$ increases when decreasing σ, and contains less smooth functions.

Homogenous dot-product kernels. A large class of commonly used kernels are homogenous dot-product kernels, which have been found useful particularly for understanding deep networks (Cho and Saul, 2009). These kernels, which can be efficiently described by the dot-product, are of the form

$$K(\mathbf{x}, \mathbf{x}') = \|\mathbf{x}\|\|\mathbf{x}'\|\kappa\left(\left\langle \frac{\mathbf{x}}{\|\mathbf{x}\|}, \frac{\mathbf{x}'}{\|\mathbf{x}'\|} \right\rangle\right),$$

with $\kappa : [-1; 1] \to \mathbb{R}$ some arbitrary function. In order to ensure the positive definiteness, The Taylor series expansion of κ admits only non-negative coefficients (Schoenberg, 1942; Scholkopf and Smola, 2001). There are various of kernels satisfying this condition, more discussions on these examples and also on the eigenvector decomposition can be found in Scholkopf and Smola (2001). In particular, when $\kappa(u) = e^{\alpha(u-1)}$, we recover the Gaussian kernel on the sphere $\mathbb{S}^{d-1}$

$$K(\mathbf{x}, \mathbf{x}') = e^{\alpha(\langle \mathbf{x}, \mathbf{x}' \rangle - 1)} = e^{-\frac{\alpha}{2}\|\mathbf{x}-\mathbf{x}'\|_2^2} \quad \text{for any } \mathbf{x}, \mathbf{x}' \in \mathbb{S}^{d-1}.$$

As mentioned before, this class also includes kernels derived from deep learning such as the arc-cosine kernel. We briefly review here this connection with two-layer networks. The connection between kernels and neural networks can be dated back to late 1990s in the limit of infinitely wide networks (Neal, 1996; Williams, 1998). Let us consider a two-layer network of the form

$$f(\mathbf{x}) = \frac{1}{\sqrt{p}} \sum_{j=1}^{p} v_j \varphi(\mathbf{x}^\top \mathbf{w}_j),$$

where $v_j \in \mathbb{R}$ denotes the neurons (parameters) at second layer and $\varphi(\mathbf{x}^\top \mathbf{w}_j)$ are hidden units that depends on the hidden weights $\mathbf{w}_j \in \mathbb{R}^d$ and an activation function φ. Now let v_j's have independent Gaussian distribution $\mathcal{N}(0,1)$ and $\mathbf{w}_j$ for each hidden unit be i.i.d. following some distribution, then we have the kernel for any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ in the finite-width limit

$$K_{\mathrm{NN}}(\mathbf{x}, \mathbf{x}') := \lim_{p \to \infty} f(\mathbf{x}) f(\mathbf{x}') = \mathbb{E}_{\mathbf{w}}[\varphi(\mathbf{x}^\top \mathbf{w}) \varphi(\mathbf{x}'^\top \mathbf{w})],$$

which takes the form of random features in Section 1.2.4. In particular when φ is the rectified linear unit (ReLU) function such that $\varphi(u) = \max(u, 0)$ and $\mathbf{w} \sim \mathcal{N}(0, 2\mathbf{I})$, then K_{NN} is a dot-product kernel (Cho and Saul, 2009) with

$$\kappa(u) = \frac{1}{\pi} \left(u(\pi - \arccos(u)) + \sqrt{1 - u^2} \right).$$

1.3.2. Combining Kernels

In order to build novel and more expressive kernels, a natural strategy is to combine existing kernels. This is possible thanks to the closure properties of the class of kernel functions by a few commonly used operations. In fact, the class of kernel functions on the data space $\mathcal{X}$ is a convex cone, *i.e.* for any two valid kernels K_1 and K_2, their linear combination

$$\lambda_1 K_1 + \lambda_2 K_2$$

is still a valid kernel for any $\lambda_1, \lambda_2 \geq 0$. Another useful property consists of closure under the pointwise multiplication, also referred to as *Hadamard product* or *Schur product*. It has been proven in Schur (1911) that for any two valid kernels K_1 and K_2, the kernel defined as

$$K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}') K_2(\mathbf{x}, \mathbf{x}') \quad \text{for any } \mathbf{x}, \mathbf{x}' \in \mathcal{X}$$

is also a valid kernel. Moreover, the pointwise limit of a sequence of valid kernels is also a valid kernel. Concretely, if $(K_n)_{n \geq 1}$ is a sequence of valid kernels that converges pointwisely to a function

$$\forall (\mathbf{x}, \mathbf{x}') \in \mathcal{X}^2, \quad K(\mathbf{x}, \mathbf{x}') = \lim_{n \to \infty} K_n(\mathbf{x}, \mathbf{x}'),$$

then K is also a valid kernel. These closure properties are useful to show the positive definiteness of most of the above kernels, such as the Gaussian kernel. However, other operations are generally not guaranteed to generate valid kernels.

In practice, when a family of valid kernels are available, taking the sum of them to build a new kernel provides a simple way to combine the heterogeneous information

and thus improve performance. The RKHS of the resulting kernel can be interpreted as the direct sum of the RKHS of the basic kernels and its feature map can be seen as the concatenation of the basic kernels' feature maps. This approach has been shown useful in many contexts such as in functional genomics (Pavlidis et al., 2002) and protein network (Yamanishi et al., 2004). A generalization of this approach is to learn to weight each kernel in the summation such that the resulting kernel takes the form of the convex combination of n basic kernels

$$K = \sum_{i=1}^{n} \mu_i K_i \quad \text{with } \boldsymbol{\mu} \in \Big\{ \mu_i \geq 0, \sum_{i=1}^{n} \mu_i = 1 \Big\},$$

and the weights μ_i can be jointly optimized with the predictor parameters. This approach is known as multiple kernel learning (MKL) (Lanckriet et al., 2004a). Its relationships to *group Lasso* have been further explored in Bach et al. (2004) and an efficient optimization algorithm has been subsequently proposed in Rakotomamonjy et al. (2008). This approach has been shown to boost performance in protein annotation (Lanckriet et al., 2004b) and image recognition (Harchaoui and Bach, 2007).

1.3.3. Kernels for Biological Sequences

As kernels can be flexibly defined on arbitrary data structures, computational biology has thus been one of the first and major application fields for them due to the diversity of its data modalities. One of the most common data types in computational biology is sequence data, such as protein and DNA sequences. A biological sequence generally is a string of characters defined over a relatively small-size alphabet $\mathcal{A}$. For instance, the alphabet of DNA sequences consists of four characters $\{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ while it has 20 characters for protein sequences, corresponding to different amino acids. Despite the simple structure of the sequence, many efforts have been made to design and engineer valid, expressive and computationally efficient kernels for sequences since two decades ago. Though these kernels may be out-of-date to deal with today's genome-scale data, it is still worth revisiting them to understand the development history and their potential hidden connections. Chapter 2 and Chapter 3 essentially are devoted to finding the relationships between these classical kernels and eventually to proposing new kernels that are more flexible and scale well to large datasets.

Relying on the construction strategy of the kernel, kernels for biological sequences can be divided into three classes: kernels derived from large feature descriptors, from a similarity score or from a generative model. However, most of the kernels have tight connections, despite the different construction strategies.

Kernels from large feature descriptors. The first class of kernels are constructed from large feature descriptors. In this case, the mapping Φ in (1.1) embeds each data point $\mathbf{x}$ to a real-valued vector $\Phi(\mathbf{x}) \in \mathbb{R}^p$, with possibly high dimensions. A typical subclass includes the string kernels, given in the following form

$$\Phi(\mathbf{x}) := (\Phi_u(\mathbf{x}))_{u \in \mathcal{A}^k},$$

where the right term represents the vector with entries $\Phi_u(\mathbf{x})$ and $\Phi_u(\mathbf{x})$ denotes some statistics of u in $\mathbf{x}$. For instance, Leslie et al. (2001) describes $\Phi_u(\mathbf{x})$ by the number

of occurrences of each contiguous subsequence of k characters u, which is called a k-mer, in sequence $\mathbf{x}$. This kernel, called spectrum kernel, has been shown to be relevant in protein homology detection or transcription factor binding prediction (Elmas et al., 2017), where short motifs that could determine the biological properties of interest occur recurrently in the sequences.

Computing this kernel by naively enumerating all the k-mers is intractable since its number (which equals to $|\mathcal{A}|^k$) grows exponentially with k. Fortunately, one only needs to count the k-mers present in the sequences thanks to the kernel trick, which only depends on the sum of the sequence lengths instead of the number of all k-mers. Since then, many extensions and variants have been proposed, making the use of such kernels more flexible in various circumstances. A natural extension is to allow a few mismatches when counting u in the sequences, which is achieved by the mismatch kernel (Leslie et al., 2004). This prior information is fairly relevant as it makes the kernel invariant to the substitution mutation, which is a common phenomenon in genetics. Interestingly, Kuksa et al. (2009) have shown that the mismatch kernel can be rewritten in the following form

$$K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{m-k+1} \sum_{j=1}^{m'-k+1} \delta_a(\mathbf{x}_{i:i+k-1}, \mathbf{x}'_{j:j+k-1}), \tag{1.6}$$

where $m = |\mathbf{x}|$ and $m' = |\mathbf{x}'|$ denote the sequence lengths and δ_a denotes the Dirac kernel up to $a \in \mathbb{N}$ mismatches, defined on k-mers as

$$\delta_a(u, v) = \begin{cases} 1 & \text{if Hamming}(u, v) \leq a, \\ 0 & \text{otherwise,} \end{cases}$$

where Hamming denotes the *Hamming distance* that is the number of distinct characters. These kernels can be interpreted as the sum of the similarities between each pair of k-mers. Consequently, they can also be seen as cases in the second class which are derived from a similarity measure.

Other relevant biological priors than substitutions can also be incorporated into the kernel by properly defining the descriptors, such as gaps (Lodhi et al., 2002; Leslie and Kuang, 2004) or even protein profiles (Kuang et al., 2005; Rangwala and Karypis, 2005). An alternative approach is to leverage supplementary informations about the amino acids or nucleotides by using numerical features instead of characters that represent their physico-chemical properties or secondary structure information. Then, general signal processing techniques or kernels for sequences of vectors can be performed on the resulting numerical time series to extract meaningful features (Zhang et al., 2003; Wang et al., 2004). These kernels are shown to be useful to predict certain properties such as classifying membrane proteins. However, they are of less interest when these supplementary informations are expensive to obtain, such as when dealing with large-scale raw sequences.

The final subclass consists in "projecting" the sequences onto a predefined dictionary, based on a similarity function. This idea is very similar to the Nyström method, though the similarity function is not required to be a valid kernel. Examples include the motif kernel (Ben-Hur and Brutlag, 2003) using as dictionary a set of motifs and the pairwise kernel (Liao and Noble, 2003) using as dictionary a fixed set of sequences. As an important consequence, these choices of the dictionary shed light on how to choose and interpret the anchor points in the Nyström method, which will be detailed in Chapter 2.

Kernels from a similarity score. Another construction strategy consists in building kernels from a similarity score, by *e.g.* using existing similarity scores between biological sequences. This concept was first explored by Haussler (1999), which introduced a class of kernels for a wide range of data that can be represented as sets whose elements are discrete structures, named *convolution kernels*. By combining some kernels adapted to the comparison of the local elements, convolution kernels provide a general framework for defining valid kernels. In particular, the above presented mismatch kernel is also a convolution kernel thanks to the form of (1.6), though its original construction was based on large feature descriptors. Since then, it has led to a couple of applications in natural language processing such as Lodhi et al. (2002); Collins and Duffy (2002); Suzuki et al. (2004) and also in bioinformatics. A typical example of convolution kernels in bioinformatics is the local alignment kernel (Saigo et al., 2004), introduced in the context of protein homology detection. This kernel is built upon the well-known Smith-Waterman score (Smith et al., 1981) that performs local sequence alignment. Since both the above presented substring kernel (Lodhi et al., 2002) and the local alignment kernel are convolution kernels and encode prior knowledge about gaps, they have indeed a close relationship as shown in Chapter 3.

The local alignment kernel was later extended to further making use of protein sequence profiles (Kuang et al., 2005), which shows substantial improvement in performance yet requires much more computations to obtain the sequence profiles. Though this kernel has been shown very effective in many sequence classification tasks such as the recognition of protein remote homologies, the reason for its success over other kernels such as the mismatch or substring kernels has hardly been understood. This will be demystified by using our general kernel framework introduced in Chapter 3. We will see later that many other kernels have been proposed for sets based on the alignment of their elements or more generally on the *optimal transport* between two sets seen as point clouds, in the context of other application fields such as computer vision.

Kernels from a generative model. The third class of kernels for biological sequences is less related to the previous ones. The construction of these kernels is based on probabilistic models, which have even longer history than kernel methods. Before the surge of various string kernels, several probabilistic models had been put forward and proven successful for characterizing families of biological sequences. Typical examples include hidden Markov models (HMM) for DNA or protein sequences and later stochastic context-free grammars for RNA sequences (Bishop and Thompson, 1986; Durbin et al., 1998), though both of them were first introduced in computational linguistics. In order to benefit from the adaptivity of these models to the data, many attempts have been made to build valid kernels upon such models. The pioneering work must be the *Fisher kernel* introduced by Jaakkola et al. (2000). This kernel makes use of a parametric probabilistic model to define the feature vector of each sequence. The feature vector is built upon the Fisher score vector that describes the local contribution of each parameter in the probabilistic model, and the Fisher information matrix that can is related to the covariance matrix. In Tsuda et al. (2002a), a variant of the Fisher kernel was proposed derived from tangent vectors of posterior log-odds. These kernels have also offered the possibility to the aggregation of local visual descriptors in image recognition, when used with a probabilistic Gaussian mixture model (Perronnin and Dance, 2007).

A parallel line of research was *mutual information kernels* introduced by Seeger (2002). In contrast to the Fisher kernel, the mutual information kernels do not explicitly provide a finite-dimensional feature vector but involve an integration over all the parameters. Thus probabilistic models that allow tractable computation of this integration is required in practice, which was achieved for instance by Cuturi and Vert (2005). These kernels essentially quantify how much information shared by two sequences, from a information theory viewpoint.

A third line of research consists in building kernels from probabilistic models with latent variables such as HMMs, named *marginalized kernels* (Tsuda et al., 2002b). Latent variables are often biologically interpretable and may encode meaningful prior knowledge such as the local structure of a sequence. The construction of such kernels is to marginalize another kernel defined over the complete data (*i.e.* latent and observed variables), weighted by the conditional distribution. Later a similar idea was put forward by Jebara et al. (2004), involving a joint distribution instead of the conditional distribution. The marginalized kernels have led to many well performing kernels from various probabilistic models, including Kin et al. (2002) for RNA sequences, Vert et al. (2006) for multiple alignments and so on. Though this class of kernels will be little involved in this thesis, the beautiful concept behind them and their data-adaptive representations still make them worthwhile to discuss.

1.3.4. Kernels for Graphs

Graph is another ubiquitous data type occurring in many scientific fields. Most information in the world is connected and their relations cannot simply be described by vectors or fixed grids. Graphs provide a natural and generic data structure for representating such relational information. Specific examples of graph-structured data include molecules, social networks, chemical pathways, gene regulatory networks and so on. Learning from graph-structured data is a challenging task as good algorithms should exploit the rich information inherent to the graphs' structure while being computationally fast. Thus developing effective and fast learning techniques for graph-structured data has still been an active research area. A classical but powerful approach to representing such kind of data is graph kernels. By properly defining a valid kernel on the space of graphs, we can apply any models that can be expressed in terms of pairwise dot products to dealing with graphs. In contrast to the flat 1D structure of biological sequences above as a special case of graph data, graphs have a more complex structure generally defined as a pair $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of vertices or nodes and $\mathcal{E}$ denotes the set of edges. Optional informations can also be offered including node attributes represented as a function $a : \mathcal{V} \to \mathbb{R}^d$ associating each node to a real-valued vector, and edge attributes represented as a function $b : \mathcal{E} \to \mathbb{R}^p$.

Similar to kernels for sequences, a large class of successful graph kernels are constructed from large feature descriptors, by representing graphs as high-dimensional feature vectors that enumerate and count the occurrences of particular graph substructures. However, detecting the presence of such substructures can be computationally intractable (*i.e.* NP-hard), even for simple substructures such as all *paths* (Gärtner et al., 2003; Borgwardt and Kriegel, 2005) defined as a sequence of edges without nodes repeated. Hence, a good kernel should be polynomial-time computable while still being expressive, *i.e.* able to distinguish graphs with different topological properties (Kriege et al.,

2018). The first work that has formally discussed this trade-off between computation and expressiveness is Gärtner et al. (2003), who showed the computational intractability of several natural kernels including *complete* graph kernels (that is able to distinguish non-isomorphic graphs), subgraph kernels and path kernels. They strongly expressed the necessity of restricting the set of substructures or alternatively the feature space and thus proposed to consider *walks* or *random walks* instead of paths that allow nodes to be repeated in the feature vector, which can be enumerated efficiently in a recursive way. Since then, a couple of graph kernels have been put forward by considering different sets of substructures. These substructures include shortest paths (Borgwardt and Kriegel, 2005), non-tottering walks (Mahé et al., 2005), all frequent subgraphs in a database (Helma et al., 2004), subtree patterns (Ramon and Gärtner, 2003; Mahé et al., 2005) and graphlets (Shervashidze et al., 2009). Later, a node label (*i.e.* discrete node attribute) enrichment algorithm has been exploited in the computation of the Weisfeiler-Lehman (WL) subtree kernel (Shervashidze et al., 2011), which provides a very efficient way to evaluate a subtree kernel up to a fixed height. Moreover, this label enrichment technique has also been shown effective when combined with other kernels like the shortest path kernel (Borgwardt and Kriegel, 2005). As this technique can only be applied to graphs with discrete node attributes, it has recently been extended to graphs with continuous attributes (Orsini et al., 2015; Togninalli et al., 2019). Other kernels based on comparisons of shortest paths (Feragen et al., 2013) or hashing (Morris et al., 2016) have also been proposed to handle continuous node attributes. In Chapter 4, we will introduce a new hierarchical kernel based on paths that compromise the expressiveness and computation, which may shed light on new architectures of graph neural networks that will be presented below.

While this line of work is focused on the exploitation of the local features from graphs, an orthogonal line of research is devoted to the aggregation of these local features, or the best matching of the local features making up the entire graphs from a kernel point of view. This problem is not limited to graphs but related to any objects made up of local features, which will be discussed in more detail in the next section. Another class of kernels (Du et al., 2019) derived from deep neural networks have recently drawn much attention and will be discussed in Section 1.3.6.

1.3.5. Kernels for Feature Aggregation

Most of the kernels seen previously for structured data, including string kernels, convolution kernels and many graph kernels, can be decoupled into two components: local feature extraction and feature aggregation. Specifically, a *local kernel* is first performed on some local patterns of the data example such as k-mers of a sequence or node neighbors of a graph, which embeds the data example to an unordered set of features in the RKHS associated to the local kernel. Then, a *global kernel* for set of features is applied to aggregate these local features in order to summarize local information and potentially capture dependencies between the local features. In this section, we will concentrate on the latter that performs feature aggregation on sets of features. Let us consider the space composed of sets of vectors drawn from $\mathcal{F} \subseteq \mathbb{R}^d$:

$$\mathcal{X} = \Big\{ \mathbf{x} \,|\, \mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \text{ with } \mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{F} \text{ and } n \geq 1 \Big\},$$

whose elements are typically vectorial representations of local data structures.

Summation kernel. The most natural kernel for aggregation that arises in string kernels (Leslie et al., 2001, 2004) and histogram comparison based kernels in computer vision (Lyu, 2005) should be the summation kernel, which can be simply written as

$$K(\mathbf{x}, \mathbf{x}') = \frac{1}{m}\frac{1}{m'}\sum_{i=1}^{m}\sum_{j=1}^{m'}\kappa(\mathbf{x}_i, \mathbf{x}'_j) = \left\langle \frac{1}{m}\sum_{i=1}^{m}\varphi(\mathbf{x}_i), \frac{1}{m'}\sum_{j=1}^{m'}\varphi(\mathbf{x}'_j)\right\rangle_{\mathcal{H}},$$

where m, m' denotes the respective size of $\mathbf{x}$ and $\mathbf{x}'$, κ is a local kernel and $\varphi : \mathbb{R}^d \to \mathcal{H}$ denotes the corresponding kernel mapping. Interestingly, when κ is a linear kernel, K amounts to performing a global average pooling across local features from a neural network viewpoint. This kernel also exhibits a close connection to the dot-product between histograms. When κ is the Dirac kernel as defined in Section 1.3.3, which is suitable to compare elements in a (finite) discrete space $\mathcal{F}$, then K is equal to the spectrum kernel, expressed as the dot-product between the normalized histograms of the local features:

$$K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{H}(\mathbf{x}), \mathbf{H}(\mathbf{x}')\rangle,$$

where $\mathbf{H}(\mathbf{x}) = (\mathbf{H}_u(\mathbf{x})/m)_{u\in\mathcal{F}}$ denotes the vector representing the normalized histogram of $\mathbf{x}$ and $\mathbf{H}_u(\mathbf{x})$ denotes the number of occurrences of u in $\mathbf{x}$. In particular, the induced distance in the RKHS is the ℓ_2-norm of the difference between the normalized histograms given by

$$d_K(\mathbf{x}, \mathbf{x}') = \sqrt{K(\mathbf{x}, \mathbf{x}) + K(\mathbf{x}', \mathbf{x}') - 2K(\mathbf{x}, \mathbf{x}')} = \|\mathbf{H}(\mathbf{x}) - \mathbf{H}(\mathbf{x}')\|_2.$$

This connection can be further extended to the mismatch kernel presented in Section 1.3.3 by considering the Dirac kernel up to $2a$ mismatches for κ and histograms with bin sizes a for some $a \in \mathbb{N}$, as detailed in Kuksa et al. (2009).

Match and optimal assignment kernel. While the summation kernel takes a simple average of the local features, another line of work focused on matching and re-weighting local features before averaging. Several attempts have been made to define a kernel based on the optimal matching between components of pairs of examples, in the context of images (Wallraven et al., 2003), graphs (Fröhlich et al., 2005; Kriege et al., 2016; Togninalli et al., 2019) and so on. However, these kernels have later been proven to be non positive definite (Lyu, 2005; Vert, 2008). Despite these failure cases, there are also some valid kernels including the exponent kernel (Lyu, 2005) by introducing an exponent in the summation kernel to control the matching level, or the intermediate kernels (Boughorbel et al., 2005; Johansson and Dubhashi, 2015) by introducing some *prototypes* to be compared with, or the Pyramid match kernel (Grauman and Darrell, 2007) by creating and comparing multi-resolution histograms etc. Among them, the most typical one should be the histogram intersection kernel (Barla et al., 2003), which is tightly related to the summation kernel. Specifically, the histogram intersection kernel is defined as the sum of the minima between histograms' bins:

$$K(\mathbf{x}, \mathbf{x}') = \frac{1}{m}\frac{1}{m'}\sum_{u\in\mathcal{F}}\min(\mathbf{H}_u(\mathbf{x}), \mathbf{H}_u(\mathbf{x}')).$$

We have shown in Section 1.3.1 that its induced distance in the RKHS is the square root of ℓ_1-norm of the difference between the normalized histograms in contrast to the

ℓ_2-norm in the summation kernel. At the first glance, this kernel is hardly related to matching of optimal assignment. It can indeed be interpreted as a kind of match kernels by the following theorem

Theorem 1.4. *The above histogram intersection kernel K can be rewritten as*

$$K(\mathbf{x}, \mathbf{x}') = \max_{\mathbf{P} \in \mathbf{U}(m,m')} \sum_{i=1}^{m} \sum_{j=1}^{m'} \mathbf{P}_{ij} \delta(\mathbf{x}_i, \mathbf{x}'_j),$$

where $\mathbf{P}$ represents a matching matrix obeying the following constraint

$$\mathbf{U}(m, m') := \left\{ \mathbf{P} \in \mathbb{R}_+^{m \times m'} \,\middle|\, \mathbf{P}\mathbf{1}_{m'} \leq \frac{1}{m} \text{ and } \mathbf{P}^\top \mathbf{1}_m \leq \frac{1}{m'} \right\}.$$

The proof can be found in Gardner et al. (2017) or in Chapter 5. This connection between matchings and histogram comparisons offers a way to define a more general p.d. kernel based on optimal transport, which will be explored in Chapter 5.

Fisher kernel. As presented in Section 1.3.3, the Fisher kernel can be used to perform aggregation of local features when combined with a Gaussian mixture model (Perronnin and Dance, 2007; Sánchez et al., 2013). Based on the Fisher vector, some variants have been proposed including VLAD representations (Jégou et al., 2010; Jegou et al., 2011), which can be more efficiently computed as a simplified version of Fisher kernels.

1.3.6. Kernels Derived from Deep Neural Networks

In recent years, some classes of kernels have been derived from deep neural networks and have drawn increasing attention as they brought possibility to theoretically understand deep networks. The correspondence between kernel methods and neural networks was first observed by Neal (1996); Williams (1998), who proved that infinitely wide two-layer networks are equivalent to a dot-product kernel as detailed in Section 1.3.1. More recently, this Gaussian process behavior has been extended to most common over-parametrized (large or infinitely-wide) deep convolutional neural networks (Lee et al., 2018; Matthews et al., 2018; Garriga-Alonso et al., 2019; Novak et al., 2019). These kernels may take advantage of the local stationarity and shift invariance captured by CNNs as presented in Section 1.4.

While these kernels correspond to neural networks where only the last layer is trained and the other layers are kept fixed at random initialization, a different line of work has studied such over-parametrized networks where all layers are trained. In this case, the evolution of the over-parametrized networks during the training by gradient descent has been shown to be fully described by an another kernel known as the *neural tangent kernel* (NTK), which converges in the large width limit to a deterministic and constant kernel only dependent of the network architecture (Jacot et al., 2018; Arora et al., 2019). This kernel admits a close form as the infinite-width limit of the kernel associated to the feature map given by the gradient of the model with respect to its parameters. From a theoretical viewpoint, the NTK provides a rigorous connection between neural networks and kernel methods, and shed light on understanding deep learning through theoretical tools from kernel methods. For example, the positive definitiveness of the NTK guarantees that large-width neural networks converge to a global minimum when trained

with a convex empirical loss (Jacot et al., 2018; Allen-Zhu et al., 2019). Empirically, this approach also offers a practical and systematic way to define kernels from neural networks. They have shown good performance in image classification (Arora et al., 2019) and graph classification (Du et al., 2019). Nevertheless, while these kernels may enjoy the expressiveness provided by the hierarchical structure of the deep networks, they still suffer from the computational bottleneck of the classical kernels.

1.3.7. Hierarchical Kernels and Convolutional Kernel Networks for Images

Another line of research consists in building hierarchical kernels relying on successful architectures of neural networks. As we will see in Section 1.4, most of these successful architectures involve multiple layers, which have motivated extensions of simple kernels to incorporate some concept of hierarchy. A natural approach is to compose kernel mappings to construct richer kernels, which remain positive definite. This approach was first investigated in Cho and Saul (2009) by composing homogenous dot-product kernels, and later extended to more sophisticated architectures borrowed from deep learning (Bo et al., 2011; Mairal et al., 2014; Mairal, 2016; Daniely et al., 2016). While simply composing kernel mappings does not seem to uncover benefits of the hierarchy, hierarchical kernels relying on deep learning architectures such as convolutional architectures have proven to be effective for modeling images. In particular, convolutional kernel networks (CKNs) (Mairal et al., 2014; Mairal, 2016) achieved very good performance for natural image classification, even though using a layer-by-layer Nyström approximation method to speed up the computation. The construction of one layer of CKNs for images principally consists of three steps: patch extraction, kernel mapping and linear pooling, as illustrated in Figure 1.2 where the image is represented on continuous domain for simplicity. More recently, Shankar et al. (2020) have shown that computing the exact kernel built upon a particular convolutional architecture outperforms the approximate kernel as well as the above NTKs, which further reduces the gap between kernel methods and deep CNNs. The theoretical aspects of CKNs such as their invariance to translations or more general groups of transformations, stability to deformations and model complexity have also been comprehensively studied in Bietti and Mairal (2019).

1.4. Deep Neural Networks for Structured Data

Deep learning has brought remarkable advancements in structured data modeling across many fields since its revolution (Krizhevsky et al., 2012). This work introduces a convolutional neural network (CNN) to deal with the image classification task and achieves a major breakthrough in computer vision research. In contrast to the classical kernel methods that decouple the hand-crafted data representation and learning algorithms, this approach merges the two steps into one and jointly optimizes them with respect to the final task which results in a single system able to predict the class directly from a raw image. In this way, the learned representations are learned from the task and may thus be task-adaptive and more compact.

However, the cost for such success in image classification is the necessity of huge amounts of annotated data and large quantities of computational resources. Since then, CNNs have achieved state-of-the-art in almost all vision tasks and also in bioinformatics

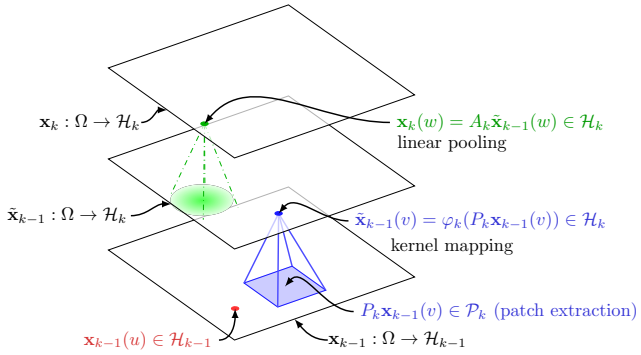


Figure 1.2.: Construction of one layer of a CKN. Figure adapted from (Bietti and Mairal, 2019). From a two-dimensional signal $\mathbf{x}_{k-1} : \Omega \to \mathcal{H}_{k-1}$ are first extracted patches $P_k \mathbf{x}_{k-1}(v)$ at each pixel v. Then the patch features are embedded to a new RKHS $\mathcal{H}_k$ via a kernel mapping φ_k. This step results in an intermediate representation $\tilde{\mathbf{x}}_{k-1} : \Omega \to \mathcal{H}_k$. Finally, a linear pooling A_k transforms $\tilde{\mathbf{x}}_{k-1}$ to the new representation $\mathbf{x}_k : \Omega \to \mathcal{H}_k$.

on biological sequence prediction tasks (Alipanahi et al., 2015). Other deep models such as recurrent neural networks (RNNs) have also been explored for predicting from sequences (Hochreiter et al., 2007). More recently, deep models for other data structures such as graphs have also drawn growing attention, with applications in bioinformatics or chemoinformatics (Kipf and Welling, 2017; Gilmer et al., 2017; Xu et al., 2019). This class of deep models is often referred to as graph neural network (GNN). In this section, we will review some of the typical deep neural networks for modeling sequences and graphs.

1.4.1. Deep Neural Networks for Sequences

Just like kernel methods, neural networks solve the same ERM problem in (1.3). The major difference is the choice of the hypothesis space $\mathcal{F}$. In neural networks (LeCun et al., 1989), the functions in $\mathcal{F}$ perform sequentially a linear transformation followed by a point-wise non-linearity in a multilayer fashion. Such hierarchical structure makes up of functions that can be represented as a composition of simple non-linear functions, where each function usually named a *fully connected layer* is parametrized independently by

$$f_i(\mathbf{x}) = \sigma(W_i \mathbf{x} + b_i) \quad \text{for } i = 1, \dots, n,$$

where $\mathbf{x}$ represents the output vector of the $(i-1)$-th layer or the input vector when $i = 0$, σ denotes a non-linear function that typically is a ReLU, and W_i and b_i are parameters for the linear transformation at i-th layer. Then training a neural network parametrized in this way is not much different from training any other machine learning models with (stochastic) gradient descent, by differentiating the objective function. In particular, the computation of the gradient is efficiently achieved by the *back-propagation* algorithm (Goodfellow et al., 2016). However, the major difference here from learning with kernel methods is the non-convexity of the objective function, caused by the non-linearity and multilayer structure. Despite the fact that there is no convergence guar-

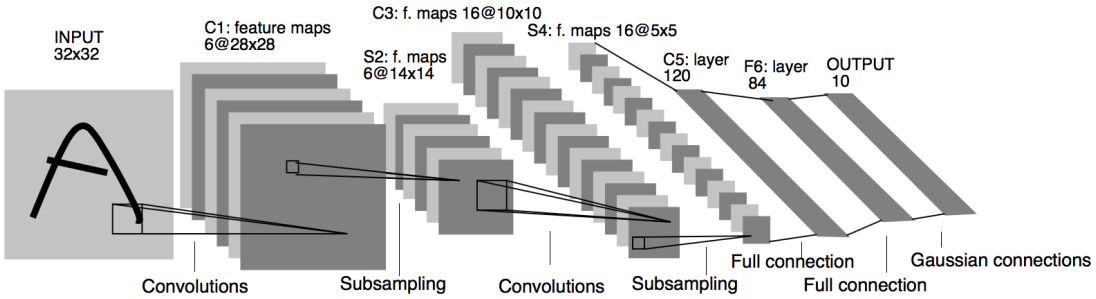


Figure 1.3.: An example of CNNs architecture. Figure from LeCun et al. (1998).

antee of such gradient-based method applied to non-convex objective functions, training neural networks is shown to converge well in practice when parameters are properly initialized.

Convolutional neural networks. Based on the construction of neural networks, a more adapted framework for image data was proposed a few years later by LeCun et al. (1998), named convolutional neural networks. CNNs adopt an idea similar to neural networks to build a multilayer model, but using different operations at each layer. In contrast to the simple linear transformations and non-linearities used in neural networks, CNNs also involve convolutions and subsampling (sometimes also referred to as pooling), as shown in Figure 1.3. These operations have proved to effectively exploit the local stationarity of natural images as well as provide translation invariance (Zeiler and Fergus, 2014). As a result, Krizhevsky et al. (2012) have put forward a novel architecture of CNN, that is deeper than LeCun et al. (1998), and have achieved breakthrough gains over the existing machine learning models in image classification. Its success in such large-scale learning task is also inseparable from the big advance of computing resources, as all the operations in CNNs can be computed in a parallel fashion. However, in contrast to kernel-based methods, one should be very careful to train a CNN model as it involves non-smooth operations and a non-convex loss function. Correctly regularizing the CNNs to avoid overfitting is still an open and central issue, though a few heuristics have been proposed.

Inspired by the significant improvement brought by CNNs in computer vision, researchers have also gradually exploited and applied them to other fields, including computational biology. The pioneering task successfully applied should be the transcription factor binding prediction or more generally the chromatin profile detection, which involves millions of annotated DNA sequences provided by the ENCODE project (Dunham et al., 2012). The seminal work of Alipanahi et al. (2015) proposed a shallow CNN architecture (with only one hidden-layer) called DeepBind that predicts the transcription factor binding from short (101 base pairs) DNA sequences and outperforms all the existing string kernels in terms of both classification accuracy and computational time. Unlike the real-valued vector representation of image data, biological sequences are strings and must be converted to vectors to be fed to CNNs. To achieve this, DeepBind first represents four DNA characters and possibly an unknown character (describing missing or padded character) respectively as the vectors $(1,0,0,0)$, $(0,1,0,0)$, $(0,0,1,0)$, $(0,0,1,0)$, $(0.25,0.25,0.25,0.25)$, such that each sequence $\mathbf{x}$ of length m is represented as a $4 \times m$

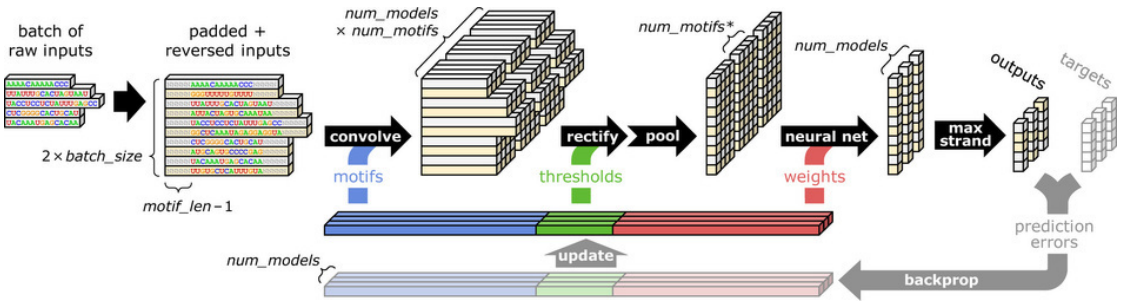


Figure 1.4.: Architecture of the DeepBind model. Figure from Alipanahi et al. (2015).

vector. It then performs a standard one-dimensional convolution of the full sequence $\mathbf{x} \in \mathbb{R}^{4\times m}$ with p convolution filters (of some predefined filter size), followed by a ReLU and a max pooling operation along the sequence. T step produce an intermediate representation $\psi(\mathbf{x})$ in $\mathbb{R}^q$, whose dimension is invariant to the sequence length. A linear prediction layer is applied to $\psi(\mathbf{x})$ to send the final outcome and the loss is computed based on the outcomes and the true labels.

The optimization of the parameters follows the same as the standard CNNs for image classification, by jointly learning the intermediate representations $\psi(\mathbf{x})$ and the linear classifier over the representations. All the steps are illustrated in Figure 1.4, where multiple models are trained in parallel for time efficiency reason. However, DNA sequences are double stranded and it is generally unknown whether the input strand or its opposite strand is associated with the label. DeepBind thus slightly modifies the objective function in (1.3) to enforce an invariance to reverse complement of the sequence. If we denote by $\bar{\mathbf{x}}$ the reverse complement of $\mathbf{x}$, the ERM (1.3) is replaced by

$$\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \ell(\max(f(\mathbf{x}_i), f(\bar{\mathbf{x}}_i)), y_i) + \lambda\Omega(f).$$

Since then, several variants have been proposed to work on the same dataset in Zeng et al. (2016), such as by using more hidden layers. However, they did not really perform better than the shallow counterpart as for image classification. When predicting from longer sequences (*e.g.* 600-bp), multilayer models (Zhou and Troyanskaya, 2015; Kelley et al., 2016) seem to improve the performance as they take into account the positional relationships between sequence signals. More recently, dilated convolutions have been found useful when dealing very long sequences (131-kb) (Kelley et al., 2018). In contrast, positional information generally is missing in string kernels or becomes computationally intractable when imposing such information into the kernel. Understanding and devising such positional relationships between sequence signals into a kernel is becoming a central problem.

Interpretation of convolutional neural networks. In order to understand what CNNs have learned from sequence classification tasks, several methods have been developed to interpret and visualize CNNs for transcription factor binding prediction. Alipanahi et al. (2015) interpret each convolutional filter as a sequence motif and visualize them by extracting and merging k-mers sufficiently close to each filter from a validation set of sequences. Later, in contrast to the separate visualization of each filter in CNN

models, Shrikumar et al. (2017a) proposed to simultaneously score the contribution of all nucleotides in an input sequence to a prediction, with the help of back-propagation based techniques. These approaches are critical as the resulting visualizations rely on some specific sequences. For this reason, Lanchantin et al. (2017) proposed a class-specific approach to generate a motif by maximizing its prediction score, such that it does not depend on any input sequence. This approach generates only one motif for a specific class, which is opposed to the fact that several motifs may be associated to a specific biological property. More interpretation techniques are needed to understand the biologically meaningful patterns captured by deep learning models in shallow or even deeper layers.

Recurrent neural networks for sequences. While convolutions are only able to exploit contiguous correlations but not dependencies between nonadjacent components or tokens in a sequence, recurrent neural networks offer the possibility to capture such dependencies by successively transforming the input tokens and updating the temporal intermediate representations, called *hidden states*. Each state value is produced by feeding the old ones and the current input token to an RNN unit, which is a parametrized function shared by all the positions. Hence, the hidden state at a certain time t provide an intermediate representation of the entire sequence until the t-th token, in contrast to the decoupling of feature extraction (convolution) and aggregation (pooling) in CNN models. The price to pay for such a joint performance of extraction and aggregation is much more computational time as the hidden states have to be computed recursively but not in parallel. Early works that used simple linear transformation as the RNN unit failed to perform well, due to vanishing or exploding gradient problems. Lately, more advanced architectures have been developed such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) and have been shown useful for certain sequence prediction tasks, such as the detection of remote protein homologies (Hochreiter et al., 2007) or the detection of multiple chromatin profiles (Quang and Xie, 2016). Despite these successes, the role of each component in RNNs and their relationship to string kernels have barely been studied and understood (Lei et al., 2017). To this end, Chapter 3 will make an attempt to bridge this gap.

1.4.2. Graph Neural Networks

Similar to kernel methods, deep neural networks also exhibit typical architectures for learning with graph-structured data, namely graph neural networks (GNNs). GNNs borrow central ideas from CNNs to obtain task-adaptive representations for graphs. They provide a simple framework for making use of the graph structure and node attributes to learn a representation of the graph. The latest GNN models (Niepert et al., 2016; Kipf and Welling, 2017; Xu et al., 2019) are built upon a multilayer structure, where each layer updates the representation of a node by aggregating and transforming its neighbor features at the previous layer as illustrated in Figure 1.5 compared to a 2D convolution in CNNs. If we use the same notation as in Section 1.2 for graph kernels, and suppose that we are given a graph with node attributes $G = (\mathcal{V}, \mathcal{E}, h : \mathcal{V} \to \mathbb{R}^d)$. Then the operations of the k-th layer of a GNN can be described as follows (Xu et al., 2019)

$$h_k(u) = \sigma(W_k \cdot f(\{h_{k-1}(v) \,|\, v \in \mathcal{N}(u) \cup \{u\}\}) + b_k) \quad \text{for any } u \in \mathcal{V},$$

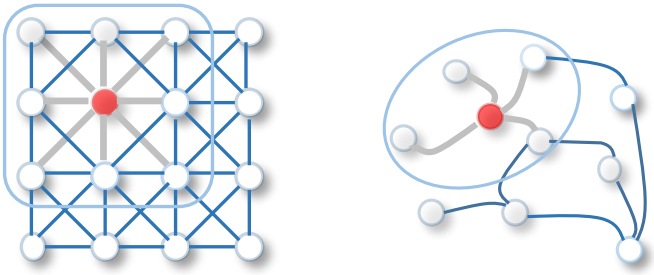


Figure 1.5.: Comparison of 2D convolution and graph convolution. Figure from Wu et al. (2020). 2D convolution in CNNs on the left: each image pixel is taken as a node and its neighbors are determined by the filter size. The neighbors of a node are ordered and have a fixed size. Graph convolution on the right: the neighbors of a node are unordered and variable in size. Thus in contrast to different filter values used at each neighbor position in 2D convolution, the neighbors are "convolved" with the same filter.

where σ is a non-linear function such as ReLU, W_k and b_k represents the parameters for the linear transformation and f denotes an element-wise pooling function operating on a set of features, typically average, sum or max pooling. Finally, a global pooling operation is applied to the node representations to obtain the representation of the graph after n layers:

$$h_n(G) = g(\{h_n(u) \,|\, u \in \mathcal{V}\}),$$

where g denotes some permutation-invariant function such as average or max pooling, or more sophisticated aggregation operators (Ying et al., 2018). Then a classification block is used to operate on this representation to produce the final prediction. The optimization procedure follows the same as CNNs using back-propagation.

In practice, the graph representations usually are concatenated across all layers to improve the performance, analogous to the idea of the WL subtree kernel that accounts for subtree patterns up to a fixed height. With the empirical development of GNNs, a few work on the theoretical aspects of GNNs have been put forward. Xu et al. (2019) have studied the expressiveness of GNNs and proved that GNNs in the above form such as Kipf and Welling (2017); Hamilton et al. (2017) are at most as powerful as WL graph isomorphism test, on which WL kernels (Shervashidze et al., 2011) are based. They also show that GNNs as above are not universal approximators of continuous functions defined on multisets. Maron et al. (2018) have studied the equivariance and invariance to node permutation for GNNs and given a full characterization of such a linear layer. These works suggest that other types of architectures than simple neighborhood aggregation are needed to capture more complex topological properties of graphs. In this respect, Chapter 4 introduces a new class of GNNs that rely on paths rather than neighbors for the aggregation step, which are more expressive.

Interpretation of graph neural networks. While the empirical and theoretical aspects of GNNs are rapidly growing, understanding and visualizing GNNs have hardly been tackled and are becoming an important task. Ying et al. (2019) propose a generic approach to provide interpretable explanations for predictions made by any GNNs. By

solving a non-convex mutual information maximization problem, their method allows to identify a compact subgraph from a given instance that plays crucial role in the prediction. Empirical experiments showed that their method managed to extract some known graph structures that are determinant in a couple of tasks. In Chapter 4, we adapt similar ideas to our framework and manage to identify several known graph motifs as well. This line of work could pave the way to efficiently uncover structural design principles of complex networks in bioinformatics or chemoinformatics.

1.5. Background on Molecular Biology

Molecular biology is a branch of biology that studies the molecules which make up and control living organisms. It attempts to explain the phenomena of life, including common biological processes within and between *cells* (the smallest unit of life), through the macromolecular properties that produce them (Alberts et al., 2014). Two categories of macromolecules are of particular importance to molecular biology: nucleic acids including deoxyribonucleic acid (DNA) and ribonucleic acid (RNA), and proteins. The central scope of molecular biology is therefore to characterize the structure, function and relationships between these two types of macromolecules. In this section, we will give a brief overview of these macromolecules. We will see that these macromolecules can be represented as sequences, whose order can be efficiently determined by sequencing technologies. The fast development of these sequencing technologies provides rich datasets of DNA, RNA and proteins, which enables modeling and analyzing these macromolecules through machining learning for sequences. We will particularly focus on two specific supervised learning tasks that are tackled in the thesis.

1.5.1. Macromolecules of the Cell

As mentioned above, nucleic acids and proteins are of particular importance to molecular biology as they form the molecular basis of cells.

Nucleic acids. DNA is a chain molecule composed of linearly linked monomers, called *nucleotides*, carrying the major part of the heritable information of a cell. Each nucleotide is a simple chemical compound essentially consisting of one of four different *nucleobases* namely adenine [A], cytosine [C], guanine [G] and thymine [T]. The nucleotides are joined to one another in a chain by covalent bonds. This chain has a direction as its two ends are chemically different. By consequence, each DNA molecule can be described by a string defined over a four-letter alphabet composed of A, C, G and T.

In cells, the genomic DNA generally is double stranded, where the two strands coil around each other to form a double helix, as illustrated in Figure 1.6. The nucleobases of the two separate strands are bound together, following the base pairing rules, with hydrogen bonds. The rules state that the nucleotides A and T can bind to each other and they are said to be *complementary*, so do G and C. Following such rules, the two strands of a DNA are complementary to each other as well and one strand is the *reverse complement* of another as each of its bases is complementary to the base of another, but read in the reverse direction. Such complementary strands can bind to each tightly by

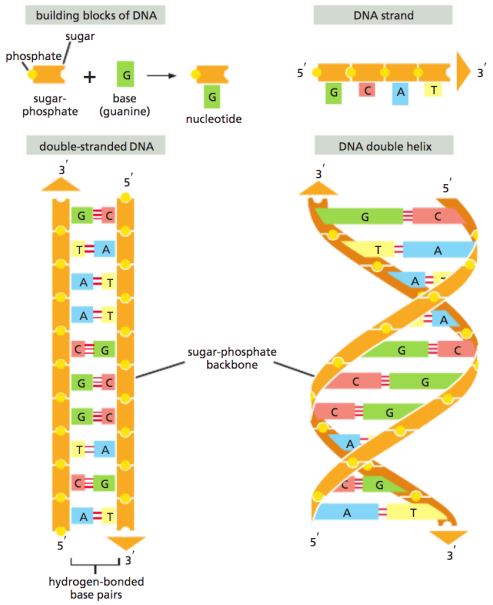


Figure 1.6.: DNA building blocks and its double helix structure. Figure from Alberts et al. (2014). A DNA molecule is composed of two DNA strands held together by hydrogen bonds between the paired bases. The arrowheads at the ends of the DNA strands indicate the polarities of the two strands, which run antiparallel to each other in the DNA molecule. At the bottom left of the figure, the DNA molecule is shown straightened out; in reality, it is twisted into a double helix, as shown on the right.

forming a double helix structure, which makes DNA very stable. Besides, this brings two positive effects as the two complementary strands carry the same biological information. First, erroneous changes, insertions or deletions of a single nucleotide base, known as a *point mutation*, can accordingly be identified and corrected. Second, the information carried by a DNA is replicated as and when the two strands separate, which provides a natural way to duplicate the *genome*, which represents the sum total of an organism's DNA. Then each strand serves as a template for synthesizing its complement and results in two reliable copies of the original DNA. Despite the simple basic units making up genomes, the size of them can be surprisingly huge. For instance, the human genome consists of more than 3 billion nucleotides. Thus super computers are usually required to analyze genomes of such scale.

RNA is another type of nucleic acids composed of nucleotides linked in a chain, very similar to DNA. While there are four different nucleotide bases including A, C, G, T that occur in DNA, the nucleobase T is replaced with the uracil [U] in RNA. This substitution along with some other chemical differences in the nucleotides make RNA molecules generally single-stranded and folded onto itself to form various 3D structures which adapt to performing complex tasks. Such RNAs are called *ribozymes*. Despite the large size of a genome, a large proportion of DNA known as non-coding DNA do not encode proteins. For instance, this proportion is more than 98% for humans (Elgar and Vavouri, 2008). The rest of DNA sequences contain genes, where each gene is defined as

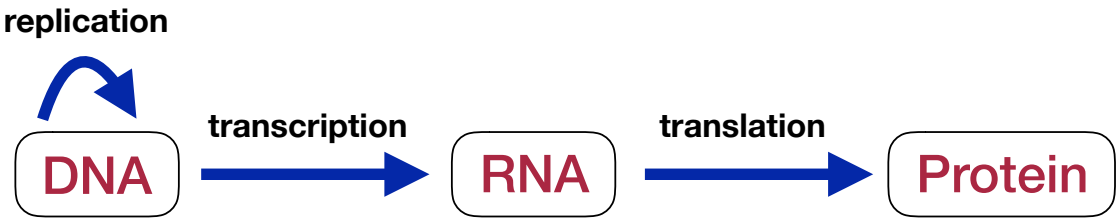


Figure 1.7.: The flow of genetic information from DNA to protein.

a subsequence that encodes the synthesis of one or a couple of RNA or protein molecules. During the process of *gene expression*, the DNA is first copied into RNA, which is the complement of a part of its template DNA. Then, the synthesized RNA can be directly functional as ribozymes or be the intermediate template for a protein that performs a function for the majority of genes. Such RNA molecules are known as messenger RNAs (mRNAs). This process from DNA to protein describes the flow of genetic information, often stated as DNA is *transcribed* into RNA then *translated* into protein, is referred to as the *central dogma of molecular biology* (Crick, 1970) as illustrated in Figure 1.7.

Proteins. Proteins are macromolecules composed of *amino acid residues* or simply residues. A linear chain of amino acid residues is called *polypeptide* which forms a protein. Short polypeptides, consisting of a few amino acids are commonly called *oligopeptides* or simply *peptides*. Two adjacent amino acids in a protein are linked with a *peptide bond*, a special type of covalent bond. As mentioned above, the sequence of amino acids in a protein is determined by the sequence of a gene. In general, there are 20 different types of standard amino acids for protein synthesis.

Proteins differ from one to another primarily in their sequence of amino acids which usually folds into an elaborate 3D structure (called *tertiary structure*) that determines its activity. This spatial structure is mostly assumed to be uniquely determined by the sequence through the chemical properties of their amino acids. However, these are still some other proteins that require the guide of *chaperones* to accomplish the folding process.

The functions of proteins are diverse within organisms to order to perform almost all the tasks occurred within and between cells. These functions include but not limited to catalyzing metabolic reactions, carrying energy, protein synthesis (transcription or translation), transporting molecules, communication and DNA replication.

1.5.2. Transcription Factors

A transcription factor (TF) is a protein that modulates the activation and repression of gene transcription from DNA to mRNA, by binding to regulatory DNA in a sequence-specific manner. Identifying TF-DNA binding specificities is a crucial step for understanding the regulatory processes and has not been resolved. Current knowledge on this problem is mainly based on two complementary lines of research, namely genomics and structural biology (Slattery et al., 2014). Recent studies suggest that the TF-DNA binding specificity is determined by several features, such as the DNA sequence, 3D structure and flexibility of TFs and their binding sites, cooperation and competition between TFs, chromatin accessibility and nucleosome occupancy etc (Slattery et al., 2014). However,

due to the existing interactions between all these factors and to the lack of measurements of relevant features and DNA binding, building models based on these features to predict the TF-DNA binding specificities is a very challenging task.

With the recent development of experimental high-throughput DNA binding assays, it has become more feasible to predict TF-DNA binding specificity through machine learning. The rich datasets provided by these high-throughput technologies have enabled many recent models in large-scale machine learning to perform well and thus have made a step towards a complete understanding of the determinants of binding specificity. In particular, the *in vivo* high-throughput assays such as genome-wide chromatin immunoprecipitation combined with sequencing (ChIP-seq) have been developed to offer the possibility to annotate genomic regulatory regions (Slattery et al., 2014). A large dataset of ChIP-seq data is now available for a large collection of TFs in humans (Dunham et al., 2012). Nevertheless, due to fundamental material and cost constraints, it is infeasible to perform these assays for all TFs in every possible cellular state and species. Building large-scale machine learning models on this dataset provides a natural way to understand and predict binding specificities in various cellular contexts.

In practice, the continuous signals offered by ChIP-seq, that are used to identify locations of TF binding, are usually converted to binary labels through signal peak detection. Then a classification model can be learned to predict one or multiple TF binding specificities. Typical models include k-mer based string kernels (Ghandi et al., 2014) or deep learning models (Alipanahi et al., 2015).

1.5.3. Protein Homology

Protein homology is the biological homology between protein sequences defined in terms of shared ancestry during evolution. Searching to identifying homologous sequences is a fundamental step in any analysis of newly obtained sequences from, *e.g.*, poorly studied species. Based on the homologous sequences of a given protein, it is possible to predict its structure and function and thus perform further analysis of the protein. A larger similarity level should be protein fold. Proteins belong to the same fold if they exhibit the same folding but are not necessarily homologous, which is generally sufficient to determine their structure and function.

The most commonly used dataset for protein homology detection should be the Structural Classification of Proteins (SCOP) database (Andreeva et al., 2014). SCOP database was created in 1994 and has been continuously developed until today. It consists of a manual classification of homologous protein sequences on multiple levels, including for instance class, fold, superfamily, family, protein domain, species and domain in SCOP version 1.75. Several typical models have been evaluated on this dataset, including string kernels (Leslie et al., 2001, 2004) and deep learning models (Hochreiter and Schmidhuber, 1997; Hou et al., 2018).

2

Biological Sequence Modeling with Convolutional Kernel Networks

Contents

Chapter abstract: The growing number of annotated biological sequences available makes it possible to learn genotype-phenotype relationships from data with increasingly high accuracy. When large quantities of labeled samples are available for training a model, convolutional neural networks can be used to predict the phenotype of unannotated sequences with good accuracy. Unfortunately, their performance with medium- or small-scale datasets is mitigated, which requires inventing new data-efficient approaches. In this chapter, we introduce a hybrid approach between convolutional neural networks and kernel methods to model biological sequences. Our method enjoys the ability of convolutional neural networks to learn data representations that are adapted to a specific task, while the kernel point of view yields algorithms that perform significantly better when the amount of training data is small. We illustrate these advantages for transcription factor binding prediction and protein homology detection, and we demonstrate that our model is also simple to interpret, which is crucial for discovering predictive motifs in sequences. The source code is freely available at https://gitlab.inria.fr/dchen/CKN-seq.

The chapter is based on the following publications:

D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. In *Research in Computational Molecular Biology (RECOMB)*, 2019c

D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. *Bioinformatics*, 35(18):3294–3302, 2019a

2.1. Introduction

Understanding the relationship between biological sequences and the associated phenotypes is a fundamental problem in molecular biology. Accordingly, machine learning techniques have been developed to exploit the growing number of phenotypic sequences in automatic annotation tools. Typical applications include classifying protein domains into superfamilies (Leslie et al., 2003; Saigo et al., 2004), predicting whether a DNA or RNA sequence binds to a protein (Alipanahi et al., 2015), its splicing outcome (Jha et al., 2017), or its chromatin accessibility (Kelley et al., 2016), predicting the resistance of a bacterial strain to a drug (Drouin et al., 2016), or denoising a ChIP-seq signal (Koh et al., 2017).

Choosing how to represent biological sequences is a critical part of methods that predict phenotypes from genotypes. Kernel-based methods (Scholkopf and Smola, 2001) have often been used for this task. Biological sequences are represented by a large set of descriptors, constructed for instance by Fisher score (Jaakkola et al., 2000), k-mer spectrum up to some mismatches (Leslie et al., 2003), or local alignment score (Saigo et al., 2004). By using the so-called kernel trick, these huge-dimensional descriptors never need to be explicitly computed as long as the inner-products between pairs of such vectors can be efficiently computed. A major limitation of traditional kernel methods is their use of fixed representations of data, as opposed to optimizing representations for a specific task. Another issue is their poor scalability since they require computing a $n \times n$ Gram matrix where n is the number of data points.

By contrast, methods based on convolutional neural networks (CNN) are more scalable and are able to optimize data representations for a specific prediction problem (LeCun et al., 1989). Even though their predictive performance was first demonstrated for two-dimensional images, they have been recently successfully adopted for DNA sequence modeling (Alipanahi et al., 2015; Zhou and Troyanskaya, 2015). When sufficient annotated data is available, they can lead to good prediction accuracy, though they still suffer from some known limitations. An important one is their lack of interpretability: the set of functions described by the network is only characterized by its algorithmic construction, which makes both the subsequent analysis and interpretation difficult. CNNs for DNA sequences typically involve much fewer layers than CNNs for images, and lend themselves to some level of interpretation (Alipanahi et al., 2015; Lanchantin et al., 2017; Shrikumar et al., 2017a). However, a systematic approach is still lacking as existing methods rely on specific sequences to interpret trained filters (Alipanahi et al., 2015; Shrikumar et al., 2017a) or output a single feature per class (Lanchantin et al., 2017, (3.3)). Correctly regularizing neural networks to avoid overfitting is another open issue and involves various heuristics such as dropout (Srivastava et al., 2014), weight decay (Hanson and Pratt, 1989), and early stopping. Finally, training neural networks generally requires large amounts of labeled data. When few training samples are available, training CNNs is challenging, motivating us for proposing a more data-efficient approach.

In this chapter we introduce CKN-seq, a strategy combining kernel methods and deep neural networks for sequence modeling, by adapting the convolutional kernel network (CKN) model originally developed for image data (Mairal, 2016). CKN-seq relies on a continuous relaxation of the mismatch kernel (Leslie and Kuang, 2004). The relaxation

makes it possible to learn the kernel from data, and we provide an unsupervised and a supervised algorithm to do so – the latter being a special case of CNNs. On the datasets we consider, both approaches show better performance than DeepBind, another existing CNN (Alipanahi et al., 2015), especially when the amount of training data is small. On the other hand, the supervised algorithm produces task-specific and small-dimensional sequence representations while the unsupervised version dominates all other methods on small-scale problems but leads to higher dimensional representations. Consequently, we introduce a *hybrid* approach which enjoys the benefits of both supervised and unsupervised variants, namely the ability of learning low-dimensional models with good prediction performance in all data size regimes. Finally, the kernel point of view of our method provides us simple ways to visualize and interpret our models, and obtain sequence logos.

We investigate the performance of CKN-seq on a transcription factor binding prediction task as well as on a protein remote homology detection. We provide a free implementation of CKN-seq for learning from biological sequences, which can easily be adapted to other sequence prediction tasks.

2.2. Method

In this section, we introduce our approach to learning sequence representations. We first review CNNs and kernel methods over which our convolutional kernel network is built. Then, we present the construction of CKN followed by the learning method. We finish the section with discussions on the interpretation and visualization of a trained CKN.

2.2.1. Supervised Learning Problem

Let us consider n sequence samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ in a set $\mathcal{X}$ of variable-length biological sequences. The sequences are assumed to be over an alphabet $\mathcal{A}$. Each sequence $\mathbf{x}_i$ is associated to a measurement y_i in $\mathcal{Y}$ denoting some biological property of the sequence. For instance, $\mathcal{Y}$ may be binary labels $\{-1, 1\}$ (*e.g.*, whether the sequence is bound by a particular transcription factor or not) or $\mathbb{R}$ for continuous traits (*e.g.*, the expression of a gene). The goal of supervised learning is to use these n examples $\{\mathbf{x}_i, y_i\}_{i=1,\ldots,n}$ to learn a function $f : \mathcal{X} \mapsto \mathcal{Y}$ which accurately predicts the label of a new, unobserved sequence. Learning is typically achieved by minimizing the following objective:

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \lambda \Omega(f), \tag{2.1}$$

where L is a loss function measuring how well the prediction $f(\mathbf{x}_i)$ fits the true label y_i, and Ω measures the smoothness of f. $\mathcal{F}$ is a set of candidate functions over which the optimization is performed. Both CNNs and kernel methods can be thought of as manners to design this set.

Convolutional neural networks. In neural networks, the functions in $\mathcal{F}$ perform a sequence of linear and nonlinear operations that are interleaved in a multilayer fashion. Specifically, the CNN DeepBind (Alipanahi et al., 2015) represents the four DNA characters respectively as the vectors $(1, 0, 0, 0)$, $(0, 1, 0, 0)$, $(0, 0, 1, 0)$, $(0, 0, 0, 1)$, such that

an input sequence $\mathbf{x}$ of length m is represented as a $4 \times m$ matrix. DeepBind then produces an intermediate representation obtained by one-dimensional convolution of the full sequence $\mathbf{x}$ with p convolution filters, followed by a pointwise non-linear function and a max pooling operation along each sequence, yielding a representation $\tilde{\mathbf{x}}$ in $\mathbb{R}^p$ of the sequence. A final linear prediction layer is applied to $\tilde{\mathbf{x}}$. The optimization in (2.1) acts on both the weights of this linear function and the convolution filters. Therefore, DeepBind simultaneously learns a representation $\tilde{\mathbf{x}}$ and a linear prediction function over this representation.

DeepBind additionally modifies the objective function (2.1) to enforce an invariance to reverse complementation of $\mathbf{x}$. The loss term is replaced with $L\left(y_i, \max\left(f(\mathbf{x}_i), f(\bar{\mathbf{x}}_i)\right)\right)$ where $\bar{\mathbf{x}}$ denotes the reverse complement of $\mathbf{x}$. Using this formulation is reported by Alipanahi et al. (2015) to improve the prediction performance. Other versions have been then considered, by using a fully connected layer that allows mixing information from the two DNA strands (Shrikumar et al., 2017b), or by considering several hidden layers instead of a single one (Zeng et al., 2016). Overall, across several versions, the performance of DeepBind with a single hidden layer turned out to be the best on average on ChIP-seq experiments from ENCODE (Zeng et al., 2016).

Kernel methods. Like in CNNs, the main principle of kernel methods is to implicitly map each training point $\mathbf{x}_i$ to a feature space in which simpler predictive functions are applied. For kernel methods, these feature spaces are generally high- (or even infinite-) dimensional vector spaces. This is achieved indirectly, by defining a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which acts as a similarity measure between input data. When the kernel function is symmetric and positive definite, a classical result (see Scholkopf and Smola, 2001) states that there exists a Hilbert space $\mathcal{F}$ of functions from $\mathcal{X}$ to $\mathbb{R}$, called reproducing kernel Hilbert space (RKHS), along with a mapping $\varphi : \mathcal{X} \to \mathcal{F}$, such that $\langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{F}} = K(\mathbf{x}, \mathbf{x}')$ for all $(\mathbf{x}, \mathbf{x}')$ in $\mathcal{X}^2$, where $\langle ., . \rangle_{\mathcal{F}}$ is the Hilbertian inner-product associated with $\mathcal{F}$. In other words, there exists a mapping of sequences into a Hilbert space, such that the kernel value between any sequence pairs is equal to the inner-product between their maps in the Hilbert space. Besides, any function f in $\mathcal{F}$ may be interpreted as a linear form $f(\mathbf{x}) = \langle \varphi(\mathbf{x}), f \rangle_{\mathcal{F}}$ for all $\mathbf{x}$ in $\mathcal{X}$. A large number of kernels have been specifically designed for biological sequences (see Ben-Hur et al., 2008, and references therein).

In the context of supervised learning (2.1), training points $\mathbf{x}_i$ can be mapped into $\varphi(\mathbf{x}_i)$ in $\mathcal{F}$, and we look for a prediction function f in $\mathcal{F}$. Interestingly, regularization is also convenient in the context of kernel methods, which is crucial for learning when few labeled samples are available. By choosing the regularization function $\Omega(f) = \|f\|_{\mathcal{F}}^2$, it is indeed possible to control the regularity of the prediction function f: for any two points $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$, the variations of the predictions are bounded by $|f(\mathbf{x}) - f(\mathbf{x}')| \leq \|f\|_{\mathcal{F}} \|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\|_{\mathcal{F}}$. Hence, a small norm $\|f\|_{\mathcal{F}}$ implies that $f(\mathbf{x})$ will be close to $f(\mathbf{x}')$ whenever $\mathbf{x}$ and $\mathbf{x}'$ are close to each other according to the geometry induced by the kernel.

Kernel methods have several assets: (i) they are generic and can be directly applied to any type of data – *e.g.*, sequences or graphs – as long as a relevant positive definite kernel is available; (ii) they are easy to regularize. However, as alluded earlier, naive implementations lack scalability. A typical workaround is the Nyström approximation

(Williams and Seeger, 2001), which builds an explicit q-dimensional mapping $\psi : \mathcal{X} \to \mathbb{R}^q$ for a reasonably small q approximating the kernel, *i.e.*, such that $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle_{\mathbb{R}^q} \simeq K(\mathbf{x}, \mathbf{x}')$. Then, solving the regularized problem (2.1) under this approximation amounts to learning a linear model with q dimensions. We will discuss how CKNs circumvent the scalability problem, while being capable to produce task-adapted data representations.

2.2.2. Convolutional Kernel Networks for Sequences

We introduce convolutional kernel networks for sequences, and show their link with mismatch kernels (Leslie and Kuang, 2004).

Convolutional kernel for sequences

Given two sequences $\mathbf{x}$ and $\mathbf{x}'$ of respective lengths m and m', we consider a window size k, and we define the following kernel, which compares pairwise subsequences of length k (k-mers) within $\mathbf{x}$ and $\mathbf{x}'$:

$$K(\mathbf{x}, \mathbf{x}') = \frac{1}{mm'} \sum_{i=1}^{m} \sum_{j=1}^{m'} K_0(P_i(\mathbf{x}), P_j(\mathbf{x}')), \tag{2.2}$$

where $P_i(\mathbf{x})$ is a k-mer of $\mathbf{x}$ centered at position i, represented as a one-hot encoded vector of size $p = |\mathcal{A}|k$ and K_0 is a positive definite kernel used to compare k-mers.[1] We follow Mairal (2016) and use a homogeneous dot-product kernel such that for two vectors $\mathbf{z}$ and $\mathbf{z}'$ in $\mathbb{R}^p$,

$$K_0(\mathbf{z}, \mathbf{z}') = \|\mathbf{z}\| \|\mathbf{z}'\| \kappa \left(\left\langle \frac{\mathbf{z}}{\|\mathbf{z}\|}, \frac{\mathbf{z}'}{\|\mathbf{z}'\|} \right\rangle \right), \tag{2.3}$$

and $\kappa : u \to e^{\frac{1}{\sigma^2}(u-1)}$. Note that when $\mathbf{z}$ and $\mathbf{z}'$ are one-hot encoded vectors of subsequences, $K_0(\mathbf{z}, \mathbf{z}') = ke^{-\frac{1}{2\sigma^2 k}\|\mathbf{z}-\mathbf{z}'\|^2}$ (more details can be found in Section 2.A), and we recover a Gaussian kernel that involves the Hamming distance $\|\mathbf{z} - \mathbf{z}'\|^2/2$ between the two subsequences. Up to the normalization factors, this choice leads to the same kernel used by Morrow et al. (2017). Yet, the algorithms we will present next are significantly different. While Morrow et al. (2017) use random features (Rahimi and Recht, 2008) to find a finite-dimensional mapping $\psi : \mathcal{X} \to \mathbb{R}^q$ that approximates the kernel map, our approach relies on the Nyström approximation (Williams and Seeger, 2001). A major advantage of the Nyström method is that it may be extended to produce lower-dimensional *task-dependent* mappings (Mairal, 2016) and it admits a model interpretation in terms of sequence logos (see Section 2.3).

Learning sequence representation

The positive definite kernel K_0 defined in (2.3) implicitly defines a reproducing kernel Hilbert space $\mathcal{F}$ over k-mers, along with a mapping $\varphi_0 : \mathcal{X} \to \mathcal{F}$. The convolutional

[1] It is also possible to introduce a concept of zero-padding for sequences, such that $P_i(\mathbf{x})$ may contain characters outside of the original sequence, when i is close to the sequence boundary, see Section 2.3.

kernel network model uses the Nyström method to approximate any point in $\mathcal{F}$ onto its projection on a finite-dimensional subspace $\mathcal{E}$ defined as the span of some anchor points

$$\mathcal{E} = \text{Span}\left(\varphi_0(\mathbf{z}_1), \ldots, \varphi_0(\mathbf{z}_p)\right),$$

where the $\mathbf{z}_i$'s are the anchor points in $\mathbb{R}^{|\mathcal{A}|k}$. Subsequently, it is possible to define a coordinate system in $\mathcal{E}$ such that the orthogonal projection of $\varphi_0(\mathbf{z})$ onto $\mathcal{E}$ may be represented by a p-dimensional vector $\psi_0(\mathbf{z})$. Assume for now that the anchor points $\mathbf{z}_i$ are given. Then, a finite-dimensional embedding (see Mairal, 2016, for details) is given by

$$\psi_0(\mathbf{z}) := \mathbf{K}_{ZZ}^{-\frac{1}{2}} \mathbf{K}_Z(\mathbf{z}),$$

where $\mathbf{K}_{ZZ}^{-\frac{1}{2}}$ is the inverse (or pseudo inverse) square root of the $p \times p$ Gram matrix $[K_0(\mathbf{z}_i, \mathbf{z}_j)]_{ij}$ and $\mathbf{K}_Z(\mathbf{z}) = (K_0(\mathbf{z}_1, \mathbf{z}), \ldots, K_0(\mathbf{z}_p, \mathbf{z}))^\top$. It is indeed possible to show that this vector preserves the Hilbertian inner-product in $\mathcal{F}$ after projection: $\langle \Pi\varphi_0(\mathbf{z}), \Pi\varphi_0(\mathbf{z}')\rangle_{\mathcal{F}} = \langle \psi_0(\mathbf{z}), \psi_0(\mathbf{z}')\rangle_{\mathbb{R}^p}$ for any $\mathbf{z}, \mathbf{z}'$ in $\mathbb{R}^{|\mathcal{A}|k}$, where Π denotes the orthogonal projection onto $\mathcal{E}$. Assuming $P_i(\mathbf{x})$ and $P_j(\mathbf{x}')$ map close enough to $\mathcal{E}$, a reasonable approximation is therefore $K_0(P_i(\mathbf{x}), P_j(\mathbf{x}')) \approx \langle \psi_0(P_i(\mathbf{x})), \psi_0(P_j(\mathbf{x}'))\rangle_{\mathbb{R}^p}$ for all i, j in (2.2), and then

$$K(\mathbf{x}, \mathbf{x}') \approx \langle \psi(\mathbf{x}), \psi(\mathbf{x}')\rangle_{\mathbb{R}^p} \quad \text{with} \quad \psi(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} \psi_0(P_i(\mathbf{x})).$$

Finally, the original optimization problem (2.1) can be approximated by

$$\min_{\mathbf{w}\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n} L(y_i, \langle \mathbf{w}, \psi(\mathbf{x}_i)\rangle) + \lambda\|\mathbf{w}\|^2. \tag{2.4}$$

We have assumed so far that the anchor points $\mathbf{z}_i$, $i = 1 \ldots, p$ were given – *i.e.*, that the sequence representation $\psi(\mathbf{x})$ was fixed in advance. We now present two methods to learn this representation. The overall approximation scheme is illustrated in the left panel of Figure 2.1.

Unsupervised learning of the anchor points. The first strategy consists in running a clustering algorithm such as K-means in order to find p centroids $\mathbf{z}_i$ in $\mathbb{R}^{|\mathcal{A}|k}$ that "span" well the data. This is achieved by extracting a large number of k-mers from the training sequences and by clustering them. The method is simple, performs well in practice as shown in Section 2.3, and can also be used to initialize the training of the following supervised variant. However, the main drawback is that it generally requires a large number of anchor points (see Section 2.3) to achieve good prediction, which can be problematic for model interpretation.

Supervised learning of the anchor points. The other strategy consists in jointly optimizing (2.4) with respect to the vector $\mathbf{w}$ in $\mathbb{R}^p$ and to the anchor points that parametrize the representation ψ.

In practice, we adopt an optimization scheme that alternates between two steps: (a) we fix the anchor points $(\mathbf{z}_i)_{i=1,\ldots,p}$, compute the finite-dimensional representations

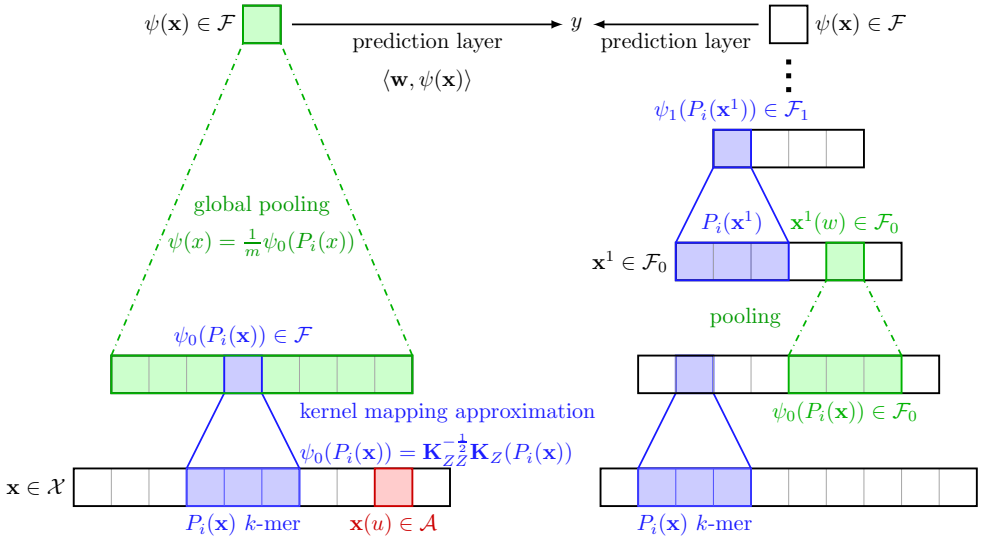


Figure 2.1.: Construction of single-layer (left) and multilayer (right) CKN-seq. For a single-layer model, each k-mer $P_i(\mathbf{x})$ is mapped to $\varphi_0(P_i(\mathbf{x}))$ in $\mathcal{F}$ and projected to $\Pi\varphi_0(P_i(\mathbf{x}))$ parametrized by $\psi_0(P_i(\mathbf{x}))$. Then, the final finite-dimensional sequence is obtained by the global pooling, $\psi(\mathbf{x}) = \frac{1}{m}\sum_{i=0}^{m}\psi_0(P_i(\mathbf{x}))$. The multilayer construction is similar, but relies on intermediate maps, obtained by local pooling, see main text for details.

$\psi(\mathbf{x}_1), \ldots, \psi(\mathbf{x}_n)$ of all data points, and minimize function (2.4) with respect to $\mathbf{w}$, which is convex if L is convex; (b) We fix $\mathbf{w}$ and update all the $(\mathbf{z}_i)_{i=1,\ldots,p}$ using one pass of a projected stochastic gradient descent (SGD) algorithm while fixing $\mathbf{w}$, at a similar computational cost per iteration as a classical CNN. The optimization for the reverse-complement formulation can be done in the same way except that it is no more convex with respect to $\mathbf{w}$, but we can still apply a fast optimization method such as L-BFGS (Liu and Nocedal, 1989). We find this alternating scheme more efficient and more stable than using an SGD algorithm jointly on $\mathbf{w}$ and the anchor points.

Multilayer construction

We have presented CKNs with a single layer for simplicity, but the extension to multiple layers is straightforward. Instead of reducing the intermediate representation in the left panel of Figure 2.1 to a single point, the pooling operation may simply reduce the sequence length by a constant factor (right panel of Figure 2.1), in a similar way as pooling reduces image resolution in CNN. This leads to an intermediate sequence representation $\mathbf{x}^1$ and we can define a valid kernel K_1, the same as K_0 in (2.3), but on subsequences of $\mathbf{x}^1$. Then the same approximation described in Section 2.2.2 can be applied to K_1. In this way, the previous process can be repeated and stacked several times, by defining a sequence of kernels $K_1, K_2, \ldots$ on subsequences from the previous respective layer representations, along with Hilbert spaces $\mathcal{F}_1, \mathcal{F}_2, \ldots$ and mapping functions $\varphi_1, \varphi_2, \ldots$ (see Mairal, 2016). Going up in the hierarchy, each point would carries information from a larger sequence neighborhood with more invariance due to the effect of pooling layers (Bietti and Mairal, 2017). The training strategy is the same as for

single-layer models.

Multilayer networks can potentially model larger motifs, with larger receptive fields, and possibly discover more interesting nonlinear relations between input variables than single-layer models. However, for the transcription factor binding prediction task under the setting of DeepBind or Zeng et al. (2016), we have observed that increasing the number of convolutional layers for CKN-seq did not improve the predictive performance (Appendix Figure 2.C.4), as also observed by Zeng et al. (2016) for CNNs. The use of multiple layers may be however important when processing very long sequences, as observed for instance by Kelley et al. (2018), who also use dilated convolutions to model even larger receptive fields than what regular CNNs can achieve.

Difference between supervised CKNs and CNNs

The main differences between CKN and CNN models are the choice of activation function (we used an exponential function in our experiments: $\kappa(x) = e^{\alpha(x-1)}$) and the transformation by the inverse square root of the Gram matrix. From a kernel point of view, the inverse square root of the Gram matrix allows us to interpret the operation as a projection onto a finite-dimensional subspace of an RKHS. From a neural network point of view, this operation decorrelates the channel entries. This can be observed when using a linear activation function $\kappa(u) = u$. In such a case, the approximated mapping is then $\psi_0(\mathbf{x}) = (\mathbf{Z}^\top\mathbf{Z})^{-\frac{1}{2}}\mathbf{Z}^\top\mathbf{x} = \tilde{\mathbf{Z}}^\top\mathbf{x}$, where $\tilde{\mathbf{Z}}^\top = (\mathbf{Z}^\top\mathbf{Z})^{-\frac{1}{2}}\mathbf{Z}^\top$ is an orthogonal matrix. Encouraging orthogonality of the filters has been shown useful to regularize deep networks (Cisse et al., 2017), and may provide intuition why our models perform better when small amounts of labeled data are available.

2.2.3. Data-Augmented and Hybrid CKN

As shown in our experiments, the unsupervised variant is sometimes more effective than the supervised one when there are only few training samples. In this section, we present a hybrid approach that can achieve similar performance as the unsupervised variant, while keeping a low-dimensional sequence representation that is easier to interpret. Before introducing this approach, we first present a classical data augmentation method for sequences, which consists in artificially generating additional training data, by perturbing the existing training samples. Formally, we consider random perturbations δ, such that given a sequence represented by a one-hot encoded vector $\mathbf{x}$, we denote by $\mathbf{x} + \delta$ the one-hot encoding vector of a perturbed sequence obtained by randomly changing some characters. Each character is switched to a different one, randomly chosen from the alphabet, with some probability p. With such a data augmentation strategy, the objective (2.1) then becomes

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\delta \sim \Delta}[L(y_i, f(\mathbf{x}_i + \delta))] + \lambda\Omega(f), \tag{2.5}$$

where Δ is a probability distribution of the variables δ corresponding to the perturbation process described above. The main assumption is that a perturbed sequence $\mathbf{x}_i + \delta$ should have the same phenotype y_i when the perturbation δ is small enough. Whereas such an assumption may not be justified in general from a biological point of view, it led to significant improvements in terms of predictive accuracy. One possible explanation

may be that for the tasks we consider, determining sequences may be short compared to the entire sequence: changing a few uniformly sample positions is therefore unlikely to perturb key bases.

As we show in Section 2.3, data-augmented CKN performs significantly better than its unaugmented counterpart when the amount of data is small. Yet, the unsupervised variant of CKN appears to be easier to regularize, and sometimes outperform all other approaches in such a low-data regime. This observation motivates us to introduce the following hybrid variant. In a first step, we learn a prediction function f_u based on the unsupervised variant of CKN, which leads to a high-dimensional sequence representation with good predictive performance. Then, we learn a low-dimensional model f_s, whose purpose is to mimic the prediction of f_u, by minimizing the cost function

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\delta \sim \Delta}[L(\hat{y}_i(\mathbf{x}_i + \delta), f(\mathbf{x}_i + \delta))] + \lambda \Omega(f), \tag{2.6}$$

where $\hat{y}_i(\mathbf{x}_i + \delta) = y_i$ if $\delta = 0$ and $f_u(\mathbf{x}_i + \delta)$ otherwise. Typically, the amount of perturbation that formulation (2.6) can afford is much larger than (2.5), as shown in our experiments, since it does not require to make the assumption that the sequence $\mathbf{x}_i + \delta$ should have exactly label y_i, which is a wrong assumption when δ is large.

2.2.4. Model Interpretation and Visualization

As observed by Morrow et al. (2017), the mismatch kernel (Leslie and Kuang, 2004) for modeling sequences may be written as Eq. (2.2) when replacing K_0 with a discrete function I_0 that assesses whether the two k-mers are identical up to some mismatches. Thus, the convolutional kernel (2.2) can be viewed as a continuous relaxation of the mismatch kernel. Such a relaxation allows us to characterize the approximated convolutional kernel by the learned anchor points (the variables $\mathbf{z}_1, \dots, \mathbf{z}_p$ in Section 2.2.2) that can be written as matrices in $\mathbb{R}^{|\mathcal{A}| \times k}$.

To transform these optimized anchor points $\mathbf{z}_i$ into position weight matrices (PWMs) which can then be visualized as sequence logos, we identify the closest PWM to each $\mathbf{z}_i$: the kernel K_0 implicitly defines a distance between one-hot-encoded sequences of length k, which is approximated by the Euclidean norm after mapping with ψ_0. Given an anchor point $\mathbf{z}_i$, the closest PWM $\boldsymbol{\mu}$ according to the geometry induced by the kernel is therefore obtained by solving

$$\min_{\boldsymbol{\mu} \in \mathcal{M}} \|\psi_0(\boldsymbol{\mu}) - \psi_0(\mathbf{z}_i)\|^2,$$

where $\mathcal{M}$ is the set of matrices in $\mathbb{R}^{\mathcal{A} \times k}$ whose columns sum to one. This projection problem can be solved using a projected gradient descent algorithm. The simplicial constraints induce some sparsity to the resulting PWM, yielding more informative logos. As opposed to the approach of Alipanahi et al. (2015) which has relied on extracting k-mers sufficiently close to the filters in a validation set of sequences, the results obtained by our method do not depend on a particular dataset.

2.3. Application

We now study the effectiveness of CKN-seq on a transcription factor (TF) binding prediction and a protein homology detection problem.

2.3.1. Prediction of Transcription Factor Binding Sites

The problem of predicting TF binding sites has been extensively studied in the recent years with the continuously growing number of TF-binding datasets. This problem can be modeled as a classification task where the input is some short DNA sequence, and the label indicates whether the sequence can be bound by a TF of interest. It has recently been pointed out that incorporating non-sequence-based data modalities such as chromatin state can improve TF binding prediction (Karimzadeh and Hoffman, 2018). However, since our method is focused on the modeling of biological sequences, our experiments are limited to sequence data only.

Datasets and evaluation metric

In our experiments, we consider the datasets used by Alipanahi et al. (2015), consisting of fragment peaks in 506 different ENCODE ChIP-seq experiments. While negative sequences are originally generated by random dinucleotide shuffling, we also train our models with real negative sequences not bound by the TF, a task called motif occupancy by Zeng et al. (2016). Both datasets have a balanced number of positive and negative samples, and we therefore measure performances by the area under the ROC curve (auROC). As noted by Karimzadeh and Hoffman (2018), even though classical, this setting may lead to overoptimistic performance: the real detection problem is more difficult as it involves a few binding sites and a huge number of non-binding sites.

Hyperparameter tuning

We discuss here the choice of different hyperparameters used in CKN and DeepBind-based CNN models.

Hyperparameter tuning for CNNs. In DeepBind (Alipanahi et al., 2015), the search for hyperparameters (learning rate, momentum, initialization, weight decay, DropOut) is computationally expensive. We observe that training with the initialization mechanism proposed by Glorot and Bengio (2010) and the Adam optimization algorithm (Kingma and Ba, 2015) leads to a set of canonical hyper-parameters that perform well across datasets, and to get rid of such an expensive dataset-specific calibration step. The results we obtain in such a setting are consistent with those reported by Alipanahi et al. (2015) (and produced by their software package) and by Zeng et al. (2016) (see Appendix Figure 2.D.1 and 2.D.2). Overall, this simplified strategy comes with great practical benefits in terms of speed.

Specifically, to choose the remaining parameters such as weight decay, we randomly select 100 datasets from DeepBind's datasets, and we use one quarter of the training samples as validation set, on which the error is used as a proxy of the generalization error. We observe that neither DropOut (Srivastava et al., 2014), nor fully connected layers bring significant improvements, which leads to an even simpler model.

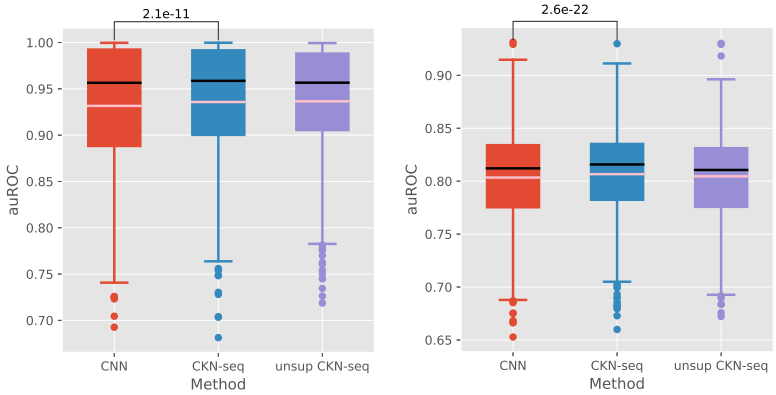


Figure 2.2.: Performance comparison of CNN and CKN-seq on the DeepBind (left) and Zeng et al. (2016) (right) datasets. Number of filters for CNN and CKN-seq was set to 128 while it was 4096 for unsupervised CKN-seq. The average auROCs for CNN, CKN-seq and unsupervised CKN-seq are 0.931, 0.936, 0.937 on the DeepBind datasets and 0.803, 0.807, 0.804 on the Zeng et al. (2016) datasets. The pink and black lines respectively represent mean and median. P-values are from one-sided Wilcoxon signed-rank test. All the following figures are obtained in the same way.

Hyperparameter tuning for CKNs. The hyperparameters of CKNs are also fixed across datasets, and we select them using the same methodology described above for CNNs. Specifically, this strategy is used to select the bandwidth parameter σ and the regularization parameter λ (see Appendix Figure 2.B.2 and 2.B.3), which is then fixed for all the versions of CKN and on either the DeepBind's or Zeng et al. (2016) datasets. For unsupervised CKN, the regularization parameter is dataset-specific and is obtained by a five-fold cross validation. To train CKN-seq, we initialize the supervised CKN-seq with the unsupervised method (which is parameter-free) and use the alternating optimization update presented in section 2.2.2. We use the Adam algorithm (Kingma and Ba, 2015) to update the filters and the L-BFGS algorithm (Zhu et al., 1997) to optimize the prediction layer. The learning rate is fixed to 0.01 for both CNN and CKN. The logistic loss is chosen to be the loss function for both this and the next protein homology detection task. All the models only use one layer. The choice of filter size, number of filters, and number of layers are also discussed in Section 2.3.3.

Performance benchmark

We compare here the auROC scores on test datasets between different CKN and DeepBind-based CNN models.

Performance on entire datasets. Both supervised and unsupervised versions of CKN-seq show performance similar to DeepBind-based CNN models (Figure 2.2), on either the DeepBind or Zeng et al. (2016) datasets.

Performance on small-scale datasets. When few labeled samples are available, unsupervised CKNs achieve better predictive performance than fully supervised approaches that are hard to regularize. Specifically, we have selected all the datasets with less than 5000 training samples and reevaluated the above models. The results are presented in the top part of Figure 2.3. As expected, we observe that the data-augmented version outperform the corresponding unaugmented version for all the models, while the supervised CKN is still dominated by the unsupervised CKN. Finally, the hybrid version of CKN-seq presented in section 2.2.3 performs nearly as well as the unsupervised one while only using 32 times fewer (only 128) filters. It is also more robust to the perturbation intensity used in augmentation than the data-augmented version (detailed choice and study of perturbation intensity can be found in Appendix Figure 2.B.4 and 2.B.5).

We obtain similar results on the Zeng et al. (2016) datasets as shown in the middle part of Figure 2.3, except that the data-augmented unsupervised CKN-seq does not improve performance over its unaugmented counterpart.

2.3.2. Protein Homology Detection

Protein homology detection is a fundamental problem in computational biology to understand and analyze the structure and function similarity between protein sequences. String kernels, see, *e.g.*, Leslie et al. (2002b, 2004); Saigo et al. (2004); Rangwala and Karypis (2005), have shown state-of-the-art prediction performance but are computationally expensive, which restricts their use to small-scale datasets. In comparison, CKN-seq and CNN models are much more computationally efficient and also turn out to achieve better performance, which we show in the rest of this section. Specifically, we consider the remote homology detection problem and benchmark different methods on the widely-used SCOP 1.67 dataset from Hochreiter et al. (2007), including 102 superfamily recognition tasks and extending the positive training samples with Uniref50. The number of training protein samples for each task is around 5000, whose length varies from tens to thousands of amino acids. Under our formulation, positive protein sequences are taken from one superfamily from which one family is withheld to serve as test samples, while negative sequences are chosen from outside the target family's fold.

Regarding the training of CNN and CKN-seq, we adopt the same setting as for the TF binding prediction task and the same methodology for the selection of hyper-parameters. A larger bandwidth parameter $\sigma = 0.6$ is selected (in contrast to $\sigma = 0.3$ in Section 2.3.1) due to the larger number of (20) characters in protein sequences. Further details about the validation scores obtained for various parameters are presented in Appendix Figure 2.B.1-2.B.3. We also use max pooling in CKN-seq to aggregate feature vectors instead of mean pooling, which shows better performance in this problem. We fix the filter size to be 10 which seems computationally intractable for the exact algorithms, such as trie-based algorithm, for computing mismatch kernels (Kuksa et al., 2009).

Profile-based methods (Kuang et al., 2005; Rangwala and Karypis, 2005) have shown very good performance on this task but suffer a few limitations as pointed out by (Hochreiter et al., 2007), including computation time and interpretability. Nevertheless, we propose an approach which integrates profiles with CKN models. Specifically, we compute the position-specific probability matrix (PSPM) using PSI-BLAST for all the sequences in SCOP 1.67 dataset, following the same protocols as Rangwala and Karypis (2005).

PSI-BLAST is performed against Uniparc[2] filtering out all the sequences after 2015, which leads to a database similar to the NCBI non-redundant database used by Rangwala and Karypis (2005). We encode the sequences in Uniref50 using the BLOSUM62 position-independent probability matrix (Henikoff and Henikoff, 1992) by replacing each character with its corresponding substitution probability in BLOSUM62. Finally, we train CKN models by replacing each sequence in our kernel (2.2) with the square root of its corresponding PSPM (or BLOSUM62). The training and hyperparameters remain unchanged.

Performance on entire datasets. Besides auROC, we also use auROC50 (area under the ROC up to 50 false positives) as evaluation metric, which is extensively used in the literature (Leslie et al., 2004; Saigo et al., 2004). Table 2.1 shows that unsupervised CKN-seq and CNN achieve similar performance and supervised CKN-seq achieves even better performance while they outperform all typical string kernels including local alignment kernel. They also outperform the LSTM model proposed by Hochreiter et al. (2007). Finally, training CKN-seq is much faster than using string kernel-based methods. While training string kernel-based models requires hours or days (Hochreiter et al., 2007), training CNN or CKN-seq are done in a few minutes. In our experiments, the average training time for CNN and supervised CKN-seq is less than 3 minutes on a single cluster with a GTX1080_TI GPU and 8 CPU cores of 2.4 GHz, while training an unsupervised CKN-seq with 16384 filters (which seems to be the maximal size that can be fit to GPU memory and gives 0.956 and 0.792 respectively for auROC and auROC50) needs 30 minutes in average. We also notice that using a random sampling instead of K-means in unsupervised CKN-seq reduces the training time to 6 minutes without loss of performance. By contrast, the training time for a local alignment kernel is about 4 hours.

Profile-based CKN-seq models show substantial improvements over their non-profile counterparts, including the BLOSUM62-based CKN-seq which uses the position-independent BLOSUM62 probability matrix instead of one-hot encoding to encode sequence characters. Supervised CKN-seq shows comparable results to the best performing methods. The performance may be further improved by computing the profiles for the extended sequences in Uniref50.

Performance on subsampled datasets. We simulate situations where few training samples are available by subsampling only 500 class-balanced training samples for each dataset. We reevaluate the above CNN and CKN models, the data-augmented versions and also the hybrid method. The results (bottom part of Figure 2.3) are similar to the ones obtained for the TF binding prediction problem except that supervised version of CKN-seq performs remarkably well in this task. We also notice that CKN-seq versions trained with only 500 samples outperform the best string kernel trained with all training samples.

[2] https://www.uniprot.org

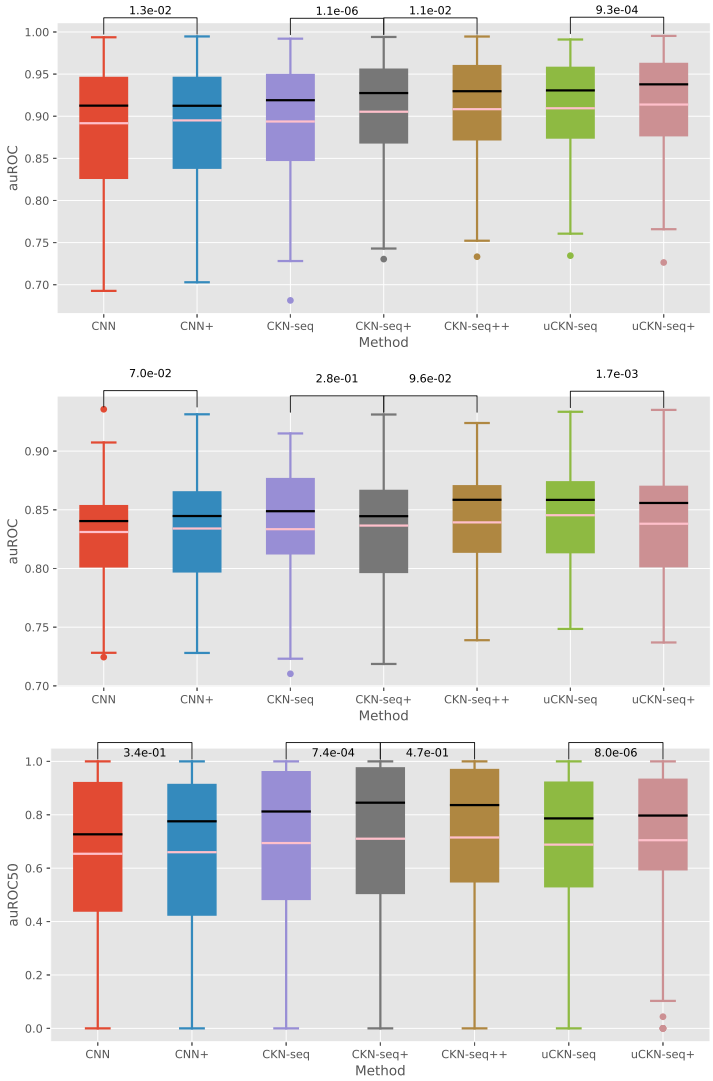


Figure 2.3.: Performance comparison on small-scale datasets (top: DeepBind datasets, middle: Zeng et al. (2016) datasets, bottom: SCOP 1.67); CKN-seq+ (respectively uCKN-seq+) represents training CKN-seq (respectively uCKN-seq) with perturbation while CKN-seq++ means the hybrid model introduced in section 2.2.3 that combines supervised and unsupervised versions: all the models use 128 convolutional filters except that unsupervised CKN-seq (uCKN-seq) and uCKN-seq+ use 4096 filters for DeepBind's dataset and 8192 for SCOP 1.67. The perturbation amount used in CKN-seq+, uCKN-seq+ and CKN-seq++ are respectively 0.2, 0.1 and 0.2 (0.3 for SCOP 1.67) for both tasks. The average auROC(50) for CNN+, CKN-seq++ and uCKN-seq(+) are 0.873, 0.908, 0.914 on the DeepBind datasets and 0.834, 0.839, 0.845 on the Zeng et al. (2016) datasets and 0.663, 0.715, 0.705 on SCOP 1.67.

Table 2.1.: Average auROC and auROC50 for SCOP 1.67 benchmark.

Method	auROC	auROC50
GPkernel (Håndstad et al., 2007)	0.902	0.591
SVM-pairwise (Liao and Noble, 2003)	0.849	0.555
Mismatch (Leslie et al., 2004)	0.878	0.543
LA-kernel (Saigo et al., 2004)	0.919	0.686
LSTM (Hochreiter et al., 2007)	0.942	0.773
CNN (128 filters)	0.960	0.799
CKN-seq (128 filters)	0.965	0.819
CKN-seq (128 filters) + BLOSUM62	0.973	0.835
unsup CKN-seq (32768 filters)	0.958	0.806
Profile-based methods		
Mismatch-profile on SCOP 1.53 (Kuang et al., 2005)	0.980	0.794
SW-PSSM on SCOP 1.53 (Rangwala and Karypis, 2005)	0.982	0.904
CKN-seq (128 filters) + profile	0.986	0.906
unsup CKN-seq (4096 filters) + profile	0.968	0.863

2.3.3. Hyperparameter Study

We now study the effect of hyperparameters and focus on the supervised version of CKN, which is more interpretable than the unsupervised one.

Both CNN and CKN-seq with one layer achieve better performance with a filter size of 12 for every fixed number of filters (Appendix Figure 2.C.2). Since this optimal value is only slightly larger than the typical length of the motifs for TFs (Stewart et al., 2012), we deduce that the prediction mainly relies on a canonical motif while the nearby content has little contribution.

Increasing the number of filters improves the auROCs for both models regardless of the filter size, in line with the observation in Zeng et al. (2016) for CNNs. This improvement saturates when more than 128 filters are deployed, sometimes leading to overfitting (Appendix Figure 2.C.1). We observe the same behavior for the unsupervised version of CKN-seq (Appendix Figure 2.C.1), but usually with much larger saturation bar (larger than 4096 for TF binding prediction and 32768 for protein homology detection). When using only 16 filters, CKN-seq shows better performance than DeepBind-based CNNs. This is an advantage as large numbers of filters make the model redundant and harder to interpret.

2.3.4. Model Interpretation and Visualization

In this section, we study the ability of a trained CKN-seq model to capture motifs and generate accurate and informative sequence logos. We use here simulated data since the true motifs are generally not known in practice. To simulate sequences containing some given motifs represented by a PWM, we follow the methodology adopted by Shrikumar et al. (2017a) and generate 500 training and 100 test samples. We train a 1-layer CKN-seq and CNN on two tasks of the respective motif FOXA1 and GATA1 (Kheradpour and

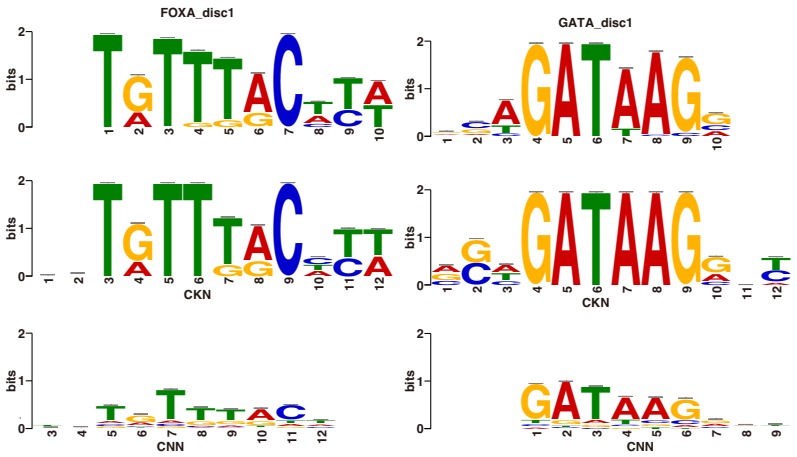


Figure 2.4.: Motifs recovered by CKN-seq (middle row) and by CNN (bottom row) compared to the true motifs (top row)

Table 2.2.: Tomtom motif p-value comparison of CKN-seq and CNN for different distance functions, see Gupta et al. (2007).

	FOXA1		GATA1	
Distance	CKN-seq	CNN	CKN-seq	CNN
KL	8.79e-13	3.22e-03	9.94e-10	2.43e-03
Euclidean	1.90e-12	3.12e-04	6.25e-09	4.35e-04
SW	1.48e-12	3.83e-04	1.77e-09	4.66e-04
Pearson	1.29e-08	6.02e-05	1.37e-09	2.88e-04

Kellis, 2013), using the same hyperparameter settings as previously. We fix the filter size and number of filters to 12 and 16 to avoid capturing too many redundant features. Both models achieve about 0.99 for the auROC on test set. The trained CNN is visualized by using the approach introduced by Alipanahi et al. (2015). Specifically, all sequences from the test set are fed through the convolutional and rectification stages of the CNN, and only the k-mers that passed the activation threshold (which is 0 by default) were aligned to generate a PWM and the trained CKN is visualized by using the approach presented in section 2.2.4, *i.e.*, solving $\min_{\boldsymbol{\mu} \in \mathcal{M}} \|\psi_0(\boldsymbol{\mu}) - \psi_0(\mathbf{z}_i)\|^2$ with a projected gradient descent method. The best recovered motifs (in the sense of information content) are compared to the true motifs using Tomtom (Gupta et al., 2007).

Motifs recovered by CKN-seq and CNN are both aligned to the true motifs (Figure 4.1). The logos given by CKN-seq are more informative and match better with the ground truth in terms of any distance measures (Table 2.2). This suggests that CKN-seq may be able to find more accurate motifs. We also perform the same experiments with more training samples (see Appendix Figure 4.B.2). We observe that CKN-seq achieves small p-values in both data regimes while p-values for CNN are larger when few training samples are available.

2.4. Discussion and Conclusion

We have introduced a convolutional kernel for sequences which combines advantages of CNNs and string kernels. The resulting CKN-seq is a special case of CNN which generalizes the mismatch kernel to motifs – instead of discrete k-mers – and makes it *task-adaptive* and scalable.

CKN-seq retains the ability of CNNs to learn sequence representations from large datasets, leading to slightly better performance than classical CNNs on a TF binding site prediction task and on a protein homology detection task. The unsupervised version of CKN-seq keeps the kernel formalism, which makes it easier to regularize and thus leads to good performance on small-scale datasets despite the use of a huge number of convolutional filters. A hybrid version of CKN-seq performs equally well as its unsupervised version but with much fewer filters. Finally, the kernel interpretation also makes the learned model more interpretable and thus recovers more accurate motifs.

The fact that CKNs retain the ability of CNNs to learn feature spaces from large training sets of data while enjoying a RKHS structure has other uncharted applications which we would like to explore in future work. First, it will allow us to leverage the existing literature on kernels for biological sequences to define the bottom kernel K_0, possibly capturing other aspects than contiguous sequence motifs, such as genetic gaps as presented in the next chapter. More generally, it provides a straightforward way to build models for non-vector objects such as graphs, taking as input molecules or protein structures, which will be presented in Chapter 4. Finally, it paves the way for making deep networks amenable to statistical analysis, in particular to hypothesis testing. This important step would be complementary to the interpretability aspect, and necessary to make deep networks a powerful tool for molecular biology beyond prediction.

Appendix

In the Appendix, we present details and additional experiments mentioned in the chapter.

2.A. Details about the Convolutional Kernel

In the definition of convolutional kernel, a bottom kernel K_0 was defined, for any $\mathbf{z}, \mathbf{z}'$ in $\mathbb{R}^d$, as

$$K_0(\mathbf{z}, \mathbf{z}') = \|\mathbf{z}\|\|\mathbf{z}'\|\kappa\left(\left\langle \frac{\mathbf{z}}{\|\mathbf{z}\|}, \frac{\mathbf{z}'}{\|\mathbf{z}'\|} \right\rangle\right),$$

where $\kappa : u \to e^{\frac{1}{\sigma^2}(u-1)}$. When $\mathbf{z}$ and $\mathbf{z}'$ are one-hot encoded vectors of k-mers, we have $\|\mathbf{z}\| = \|\mathbf{z}'\| = \sqrt{k}$ and thus

$$\begin{aligned} K_0(\mathbf{z}, \mathbf{z}') &= k e^{\frac{1}{\sigma^2}\left(\frac{1}{k}\langle \mathbf{z}, \mathbf{z}'\rangle - 1\right)} \\ &= k e^{-\frac{1}{2\sigma^2 k}(2k - 2\langle \mathbf{z}, \mathbf{z}'\rangle)} \\ &= k e^{-\frac{1}{2\sigma^2 k}(\|\mathbf{z}\|^2 + \|\mathbf{z}'\|^2 - 2\langle \mathbf{z}, \mathbf{z}'\rangle)} \\ &= k e^{-\frac{1}{2\sigma^2 k}\|\mathbf{z} - \mathbf{z}'\|^2}, \end{aligned}$$

which recovers a Gaussian kernel.

2.B. Choice of Model Hyperparameters

We justify here the choice of the hyperparameters used in our experiments, including weight decay for CNNs, regularization parameter, bandwidth parameter in exponential kernel and perturbation intensity used in data-augmented CNN, CKN and hybrid model. We denote respectively by k the filter size and p the number of filters.

The scores for the following experiments are computed on a validation set, which is taken from one quarter of the training samples for each dataset and the models are trained on the rest of the training samples. For DeepBind's datasets, we only perform validation on 100 randomly sampled datasets, which save a lot of computation time and should give similar results when using all datasets.

Weight decay for CNN. The choice of weight decay is validated on the validation set as shown in Figure 2.B.1.

Bandwidth parameter in exponential kernel. The choice of the bandwidth parameter is only validated for supervised CKN-seq and the same value is used for the unsupervised variant. Figure 2.B.2 shows the scores on the validation set when the other hyperparameters are fixed. The same choice as DeepBind's dataset is applied to Zeng's dataset.

Regularization parameter. The choice of the regularization parameter is validated following the same protocol as the bandwidth parameter. Figure 2.B.3 shows the scores on the validation set.

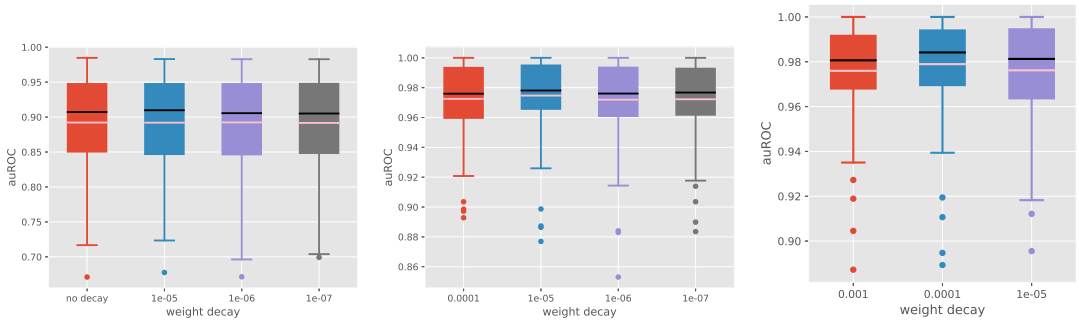


Figure 2.B.1.: Validation of weight decay in CNNs for DeepBind's datasets (left) and SCOP 1.67 and its subsampled datasets (middle and right); $k = 12$ and 10 respectively for each task; $p = 128$ for both tasks.

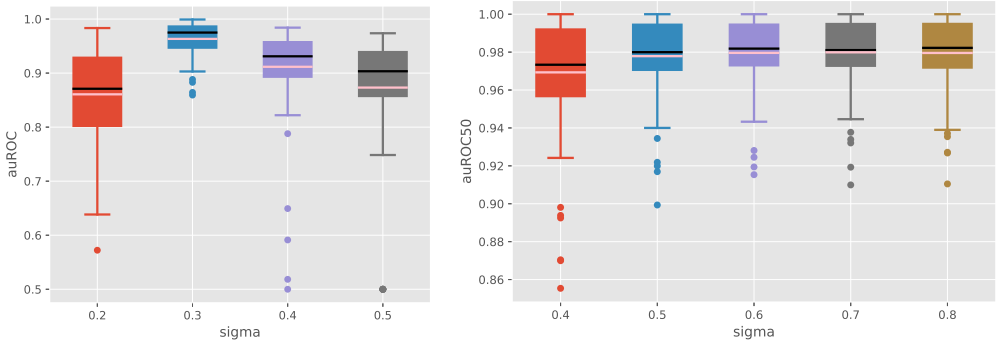


Figure 2.B.2.: Validation of the bandwidth parameter σ for DeepBind's datasets (left) and SCOP 1.67 (right). The regularization parameter is fixed to 1e-6 and 1.0 and $k = 12$ and 10 respectively for each task; $p = 128$ for both tasks.

Perturbation intensity in data-augmented and hybrid model. The perturbation amount used in the data-augmented CNN, CKN and the hybrid variant of CKN are also validated on the corresponding validation set. The scores are shown in Figure 2.B.5.

2.C. Hyperparameter Study

We discuss here in more detail the effect of the number and size of convolutional filters and number of layers on CNN and CKN performances. We also present the discussions on the perturbation intensity in data-augmented and hybrid variants of CKN-seq.

For some of the following comparisons, we also include the oracle model, which represents the best performance achievable by choosing the optimal parameter in comparison for each dataset (whereas parameters used in our experiments are fixed across datasets). The experiment shows that a dataset-dependent parameter calibration step could possibly improve the performance, but that the potential gain would be relatively small.

Number of filters, filter size and number of layers. We show in Figure 2.C.1 that increasing the number of filters improved the performance for both supervised and un-

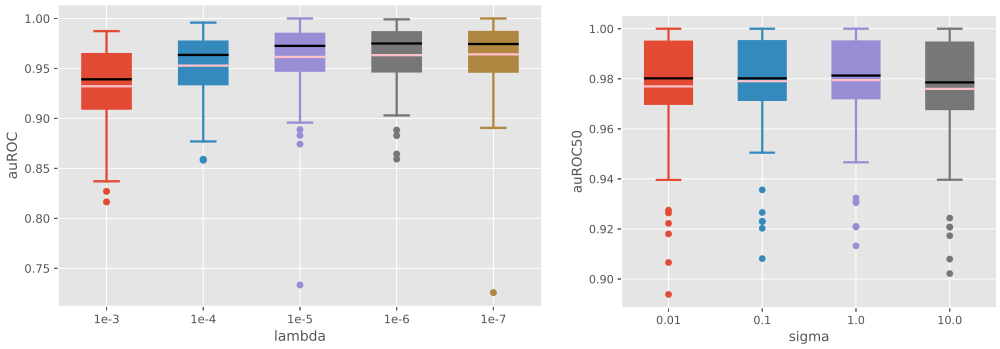


Figure 2.B.3.: Validation of the regularization λ for DeepBind's datasets (left) and SCOP 1.67 (right). The bandwith parameter is fixed to 0.3 and 0.6 and $k = 12$ and 10 respectively for each task; $p = 128$ for both tasks.

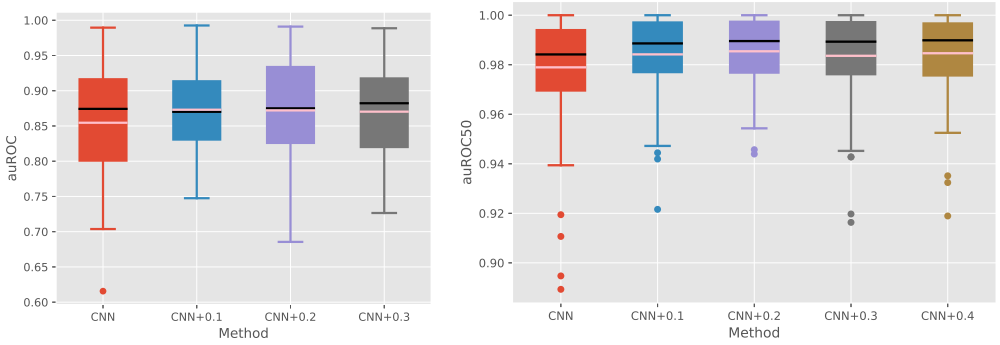


Figure 2.B.4.: Validation of the perturbation intensity for data-augmented CNN on DeepBind's small-scale datasets and (left) and subsampled SCOP 1.67 (right); $k = 12$ and 10 respectively for each task and $p = 128$ for both tasks.

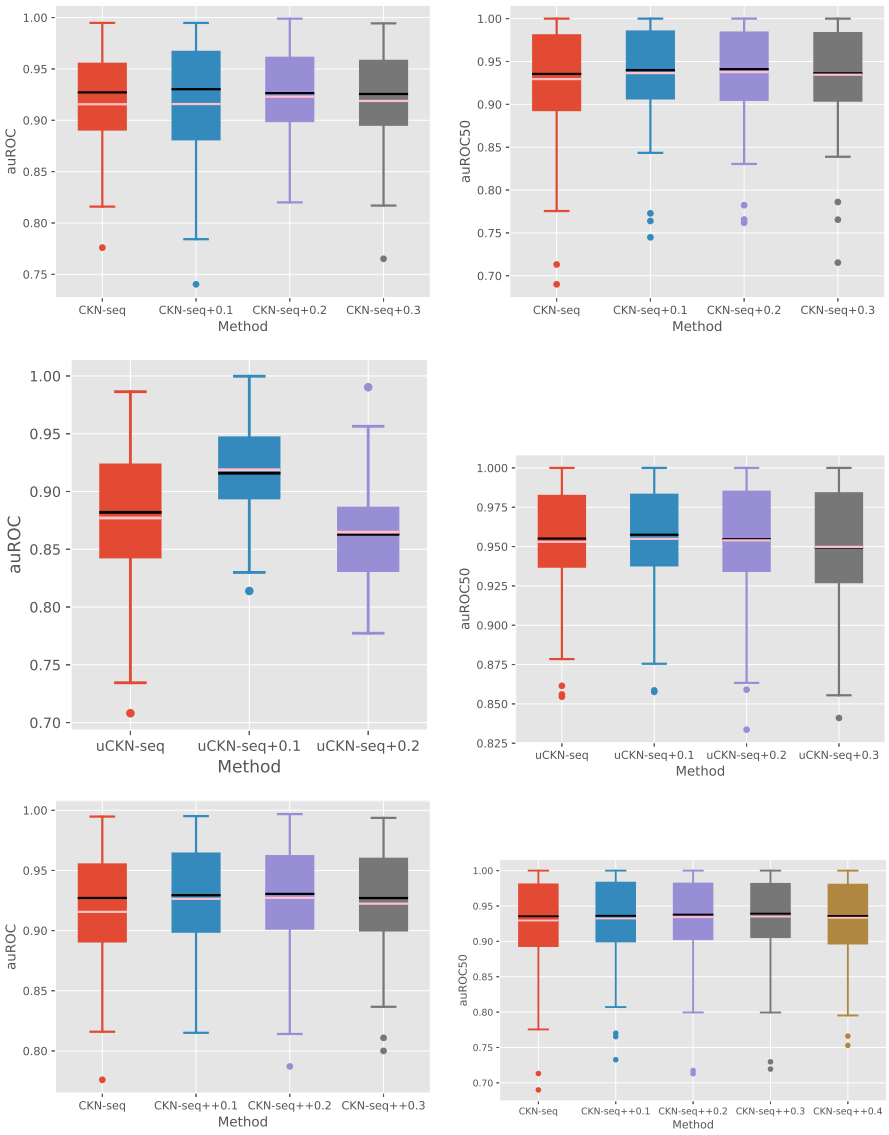


Figure 2.B.5.: Validation of the perturbation intensity for CKN on DeepBind's small-scale datasets (left) and subsampled SCOP 1.67 (right); each line corresponds to data-augmented supervised (top), data-augmented unsupervised (middle) and hybrid (bottom) variants of CKN-seq. The bandwith parameter is fixed to 0.3 and 0.6, the regularization parameter is fixed to 1e-6 and 1.0, and $k = 12$ and 10 respectively for each task; $p = 128$ for both tasks.

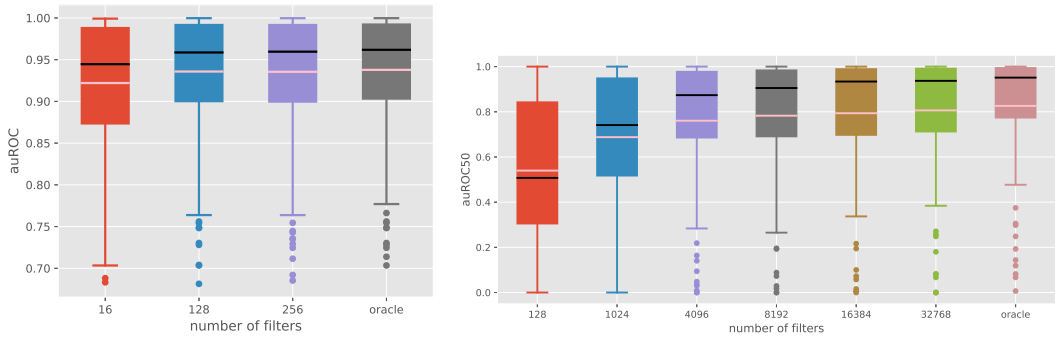


Figure 2.C.1.: Influence of the number of filters for supervised and unsupervised CKN-seq: left supervised variant with $k = 12$ on DeepBind's datasets; right unsupervised variant with $k = 10$ on SCOP 1.67 datasets.

supervised variants of CKN-seq. Furthermore, the improvement of prediction performance of the supervised one was saturated when more than 128 convolutional filters were deployed.

Both CNN and CKN-seq with one layer achieve better performance with a filter size of 12 for every fixed number of filters (Figure 2.C.2). Since this optimal value is only slightly larger than the typical length of the motifs for TFs, we deduce that the prediction mainly relies on a canonical motif while the nearby content has little contribution. However if one is interested in motif discovery only, running the algorithm with larger filter size may be of interest whenever one believes that some TF binding sites are explained by larger motifs.

Increasing the number of convolutional layers in CNNs has been shown to decrease its performance. By contrast, it does not affect the performance of CKN-seq when using a sufficient number of convolutional filters (Figure 2.C.3). Multilayer architectures allow to learn richer or more complex descriptors such as co-motifs, but may require a larger amount of data. They would also make the interpretation of the trained models more difficult. When training with 2-layer CKN models, we also notice that increasing the number of filters from 64 to 128 at the first layer or that from 16 to 64 at the second layer does not improve performance (Figure 2.C.4).

Perturbation intensity in data-augmented and hybrid CKN. We have shown that data augmentation improves both supervised and unsupervised CKN-seq. The hybrid approach has further improved data-augmented CKN-seq. We study here how the amount of perturbation used in augmenting training samples impacts performance. Specifically, we characterize the perturbation intensity by the percentage of changed characters in a sequence and show in Figure 2.C.6 the behavior of CKN-seq when increasing the amount of perturbation. By leveraging the best data-augmented unsupervised model on validation set, we train our hybrid variant and show its performance when increasing the amount of perturbation (Figure 2.C.5). We observe that the hybrid variant is more robust to larger amount of perturbation applied in the training samples than simply data-augmented one. Note that the results are consistent to those obtained on validation set (Section 2.B).

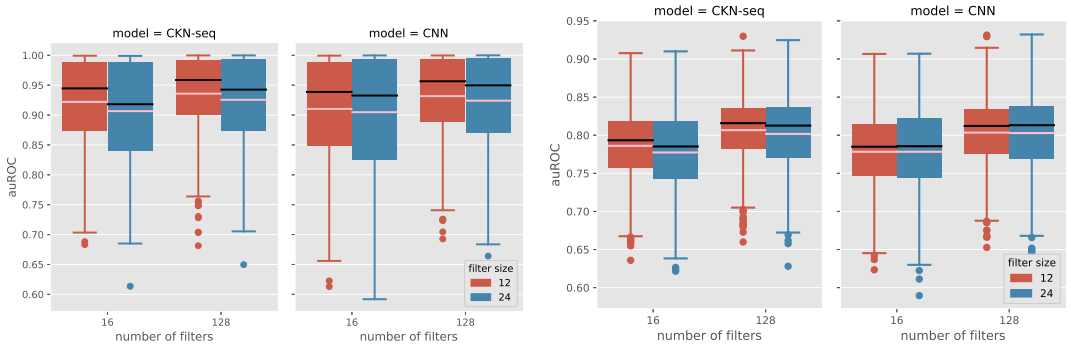


Figure 2.C.2.: auROC scores on test datasets of DeepBind (left) and Zeng *et al.* (2016) (right) for single-layer CKN-seq and DeepBind-based CNNs with number of filters varying between 16, 64, 128 and filter size between 12, 18, 24; The pink and black line respectively represent mean and median.

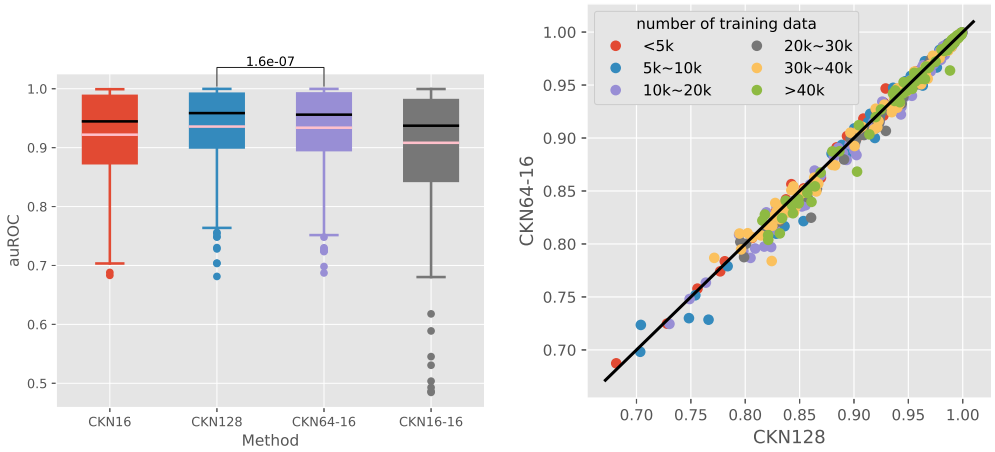


Figure 2.C.3.: Comparison between single-layer and 2-layer CKN-seq models; note that CKN64-16 has nearly the same number of parameters as CKN128.

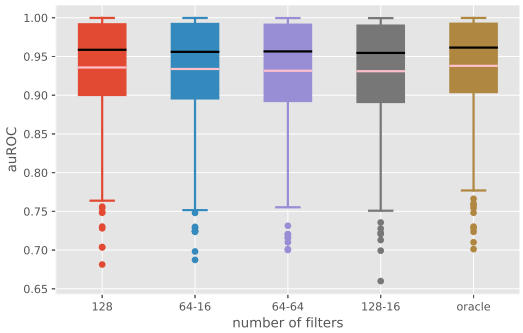


Figure 2.C.4.: Influence of the number of filters for 2-layer supervised CKN-seq on DeepBind's datasets.

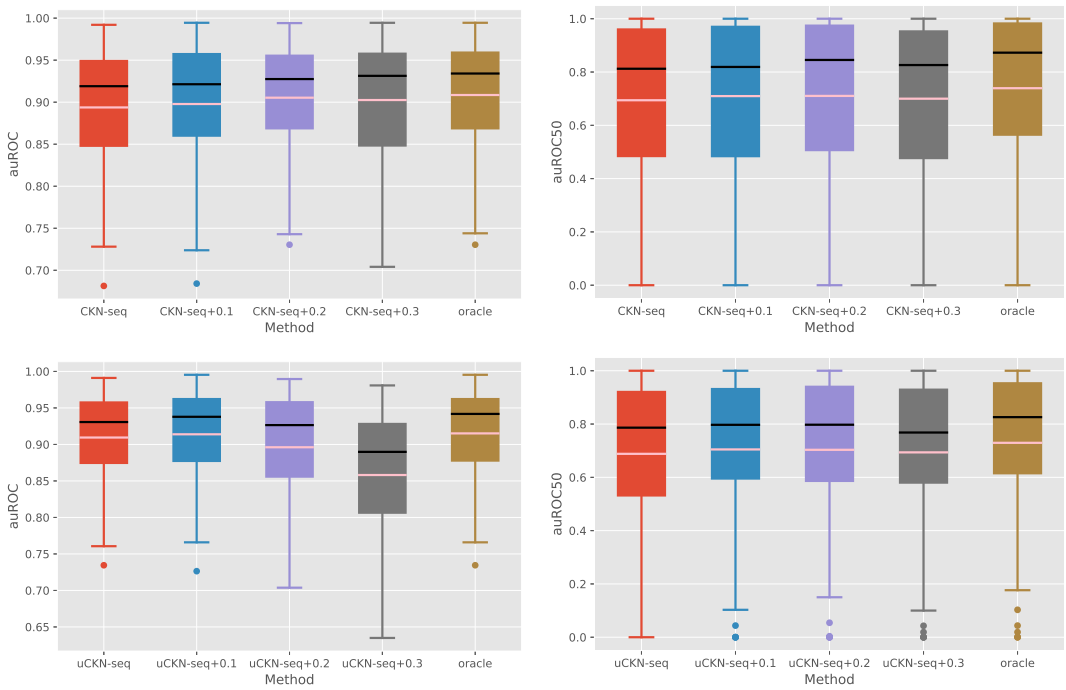


Figure 2.C.5.: Effect of perturbation intensity on supervised and unsupervised CKN-seq: top: data-augmented supervised CKN-seq; bottom: data-augmented unsupervised CKN-seq; left: on DeepBind's datasets; right: on SCOP 1.67. The number after + indicates the percentage of perturbation amount applied to the training samples.

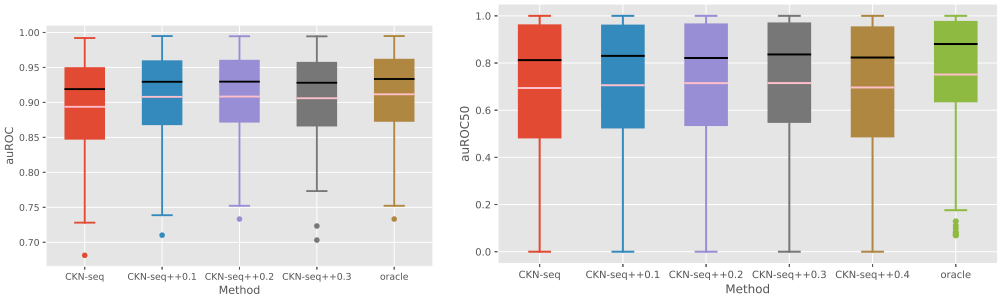


Figure 2.C.6.: Effect of perturbation intensity on hybrid CKN-seq: left: on DeepBind's datasets; right: on SCOP 1.67. All the hybrid models are trained using uCKN-seq+0.1. The number after ++ indicates the percentage of perturbation amount applied to the training samples.

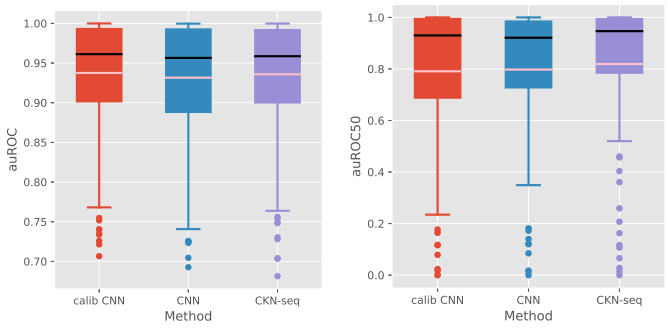


Figure 2.D.1.: Comparison of calibrated CNN and universal models; left: DeepBind's dataset and right: SCOP 1.67 dataset

2.D. Effect of Hyperparameter Calibration in CNN

We study here how hyperparameter calibration as used in DeepBind could affect performance and training time for CNNs. For the calibrated variant of CNN, we used the same hyperparameter search scheme used in DeepBind for the CNN, with 30 randomly chosen calibration settings and 6 training trials across the data sets.

The calibrated variant slightly outperformed hyperparameter-fixed CNN and showed similar performance to CKN-seq in the TF binding prediction task while it didn't achieve better performance in the protein homology detection task (Figure 2.D.1).

On the other hand, training a calibrated CNN is much slower compared to hyperparameter-fixed CNN or CKN-seq. To make a fair comparison, we reimplemented and evaluated both DeepBind and CKN-seq in Pytorch. Our reimplemented model achieved almost identical performance to the original DeepBind (left panel of Figure 2.D.2) in DeepBind's Datasets. In order to quantify the gain in training time for hyperparameter-fixed models, we measured the average training time on 50 different datasets for original DeepBind, our reimplemented DeepBind and CKN-seq on a Geforce GTX Titan Black GPU. The right panel of Figure 2.D.2 shows that training a CKN-seq model is about 25 times faster than training the original DeepBind model and 5 times faster than our reimplemented version.

2.E. Influence of Fully Connected Layer in CNN

The authors of DeepBind have used a fully connected layer in their model. However, we found that there was no significant gain with this supplementary layer in our experiments, as shown in Figure 2.E.1.

2.F. Pairwise Comparison of CKN and CNN

We include here some scatter plots to illustrate the pairwise comparison on each individual dataset of DeepBind and Zeng. The results are shown in 2.F.1.

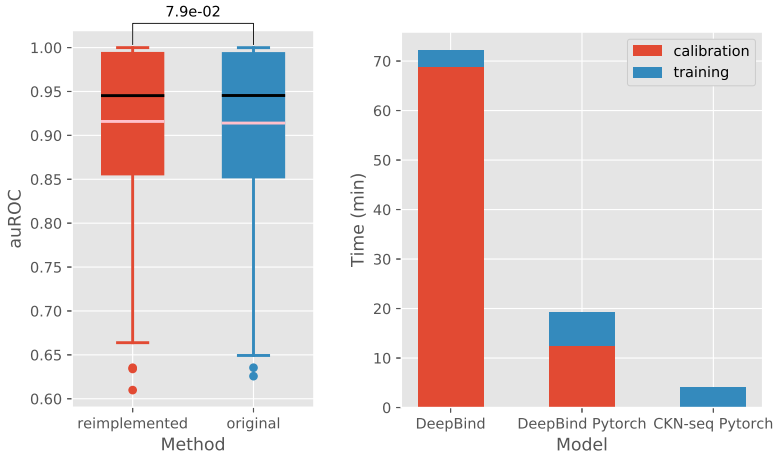


Figure 2.D.2.: left: Comparison of reimplemented and original DeepBind, with the p-value of Wilcoxon unsigned-rank test; right: Average training time for DeepBind and CKN-seq on 50 datasets

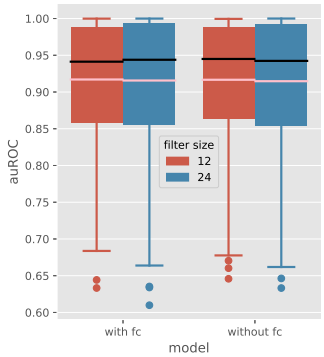


Figure 2.E.1.: Influence of the fully connected layer in CNN on DeepBind's datasets: all models were trained with $p = 16$.

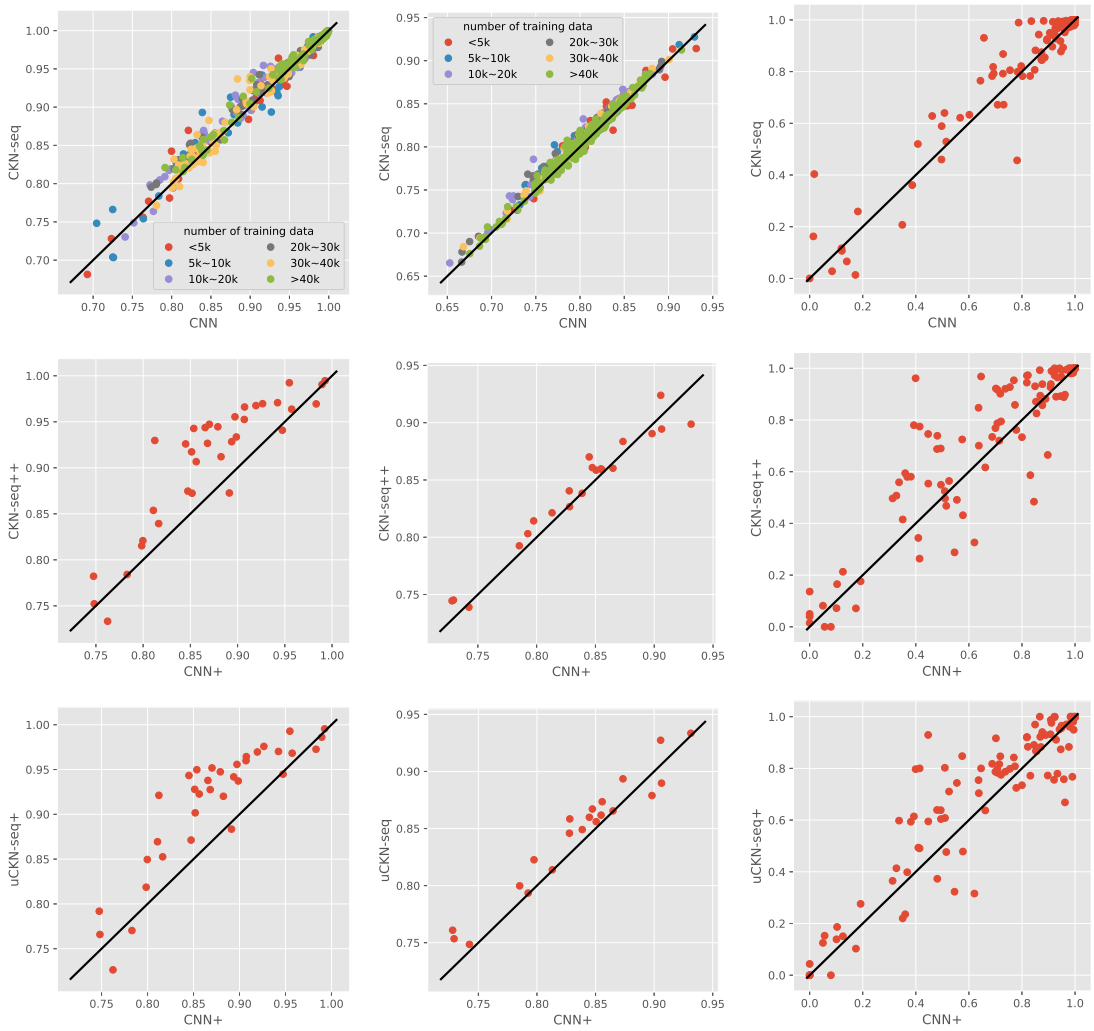


Figure 2.F.1.: Pairwise comparison of CKN-seq and CNN on DeepBind, Zeng and SCOP 1.67 datasets. The metric is auROC for the two earlier datasets and auROC50 for the latter. The middle and bottom lines show performance of models trained on small-scale datasets.

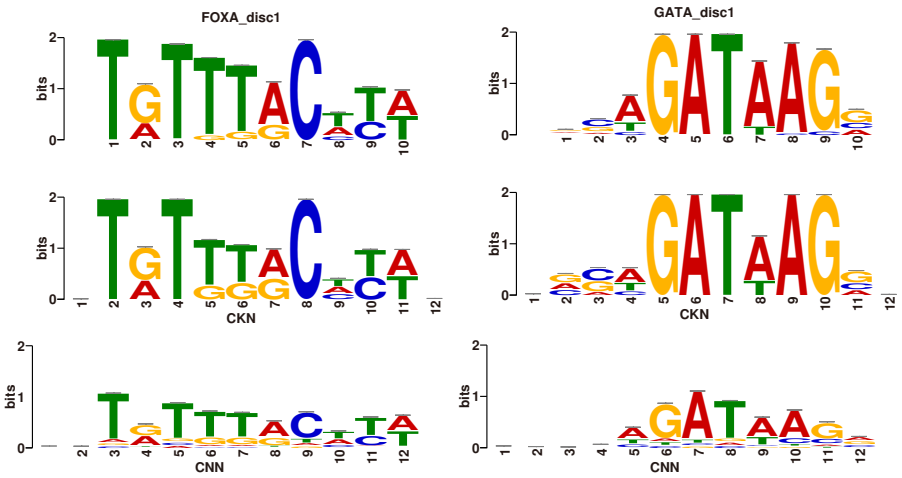


Figure 2.G.1.: Motifs recovered by CKN-seq (middle row) and by CNN (bottom row) compared to the true motifs (top row)

Table 2.G.1.: Tomtom motif p-value comparison of CKN-seq and CNN for different distance functions.

	FOXA1		GATA1	
Distance	CKN-seq	CNN	CKN-seq	CNN
KL	9.77e-14	1.78e-08	3.61e-11	3.73e-08
Euclidean	1.10e-12	6.62e-10	6.49e-12	1.07e-07
SW	6.75e-11	4.65e-10	1.93e-11	2.71e-08
Pearson	2.63e-07	3.59e-09	1.72e-08	5.32e-07

2.G. Model Interpretation and Visualization

We perform the same experiments as in section 2.3.4 of the chapter but on a larger datasets, with 9000 training samples and 1000 test samples. Motifs recovered by CKN-seq and CNN were aligned to the true motifs (Figure 4.B.2) while the logos given by CKN-seq are more informative and match better with the ground truth in terms of any distance measures (Table 2.G.1). The same conclusions can be drawn as in the small-scale case.

3

Recurrent Kernel Networks

Contents

Chapter abstract: Substring kernels are classical tools for representing biological sequences or text. However, when large amounts of annotated data are available, models that allow end-to-end training such as neural networks are often preferred. Links between recurrent neural networks (RNNs) and substring kernels have recently been drawn, by formally showing that RNNs with specific activation functions were points in a reproducing kernel Hilbert space (RKHS). In this chapter, we revisit this link by generalizing convolutional kernel networks—originally related to a relaxation of the mismatch kernel—to model gaps in sequences. It results in a new type of recurrent neural network which can be trained end-to-end with back-propagation, or without supervision by using kernel approximation techniques. We experimentally show that our approach is well suited to biological sequences, where it outperforms existing methods for protein classification tasks.

The chapter is based on the following publication. The work was also presented at Machine Learning in Computational Biology (MLCB) conference 2019.

D. Chen, L. Jacob, and J. Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b

3.1. Introduction

Learning from biological sequences is important for a variety of scientific fields such as evolution (Flagel et al., 2019) or human health (J. Topol, 2019). In order to use classical statistical models, a first step is often to map sequences to vectors of fixed size, while retaining relevant features for the considered learning task. For a long time, such features have been extracted from sequence alignment, either against a reference or between each others (Auton et al., 2015). The resulting features are appropriate for sequences that are similar enough, but they become ill-defined when sequences are not

suited to alignment. This includes important cases such as microbial genomes, distant species, or human diseases, and calls for alternative representations (Consortium, 2016).

String kernels provide generic representations for biological sequences, most of which do not require global alignment (Schölkopf et al., 2004). In particular, a classical approach maps sequences to a huge-dimensional feature space by enumerating statistics about all occuring subsequences. These subsequences may be simple classical k-mers leading to the spectrum kernel (Leslie et al., 2001), k-mers up to mismatches (Leslie et al., 2004), or gap-allowing subsequences (Lodhi et al., 2002). Other approaches involve kernels based on a generative model (Jaakkola et al., 1999; Tsuda et al., 2002b), or based on local alignments between sequences (Vert et al., 2004) inspired by convolution kernels (Haussler, 1999; Watkins, 1999).

The goal of kernel design is then to encode prior knowledge in the learning process. For instance, modeling gaps in biological sequences is important since it allows taking into account short insertion and deletion events, a common source of genetic variation. However, even though kernel methods are good at encoding prior knowledge, they provide fixed task-independent representations. When large amounts of data are available, approaches that optimize the data representation for the prediction task are now often preferred. For instance, convolutional neural networks (LeCun et al., 1989) are commonly used for DNA sequence modeling (Alipanahi et al., 2015; Angermueller et al., 2016; Zhou and Troyanskaya, 2015), and have been successful for natural language processing (Kalchbrenner et al., 2014). While convolution filters learned over images are interpreted as image patches, those learned over sequences are viewed as sequence motifs. RNNs such as long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) are also commonly used in both biological (Hochreiter et al., 2007) and natural language processing contexts (Cho et al., 2014; Merity et al., 2018).

Motivated by the regularization mechanisms of kernel methods, which are useful when the amount of data is small and are yet imperfect in neural networks, hybrid approaches have been developed between the kernel and neural networks paradigms (Cho and Saul, 2009; Morrow et al., 2017; Zhang et al., 2017). Closely related to our work, the convolutional kernel network (CKN) model originally developed for images (Mairal, 2016) was successfully adapted to biological sequences in Chen et al. (2019a). CKNs for sequences consist in a continuous relaxation of the mismatch kernel: while the latter represents a sequence by its content in k-mers up to a few discrete errors, the former considers a continuous relaxation, leading to an infinite-dimensional sequence representation. Finally, a kernel approximation relying on the Nyström method (Williams and Seeger, 2001) projects the mapped sequences to a linear subspace of the RKHS, spanned by a finite number of motifs. When these motifs are learned end-to-end with back-propagation, learning with CKNs can also be thought of as performing feature selection in the—infinite dimensional—RKHS.

In this chapter, we generalize CKNs for sequences by allowing gaps in motifs, motivated by genomics applications. The kernel map retains the convolutional structure of CKNs but the kernel approximation that we introduce can be computed using a recurrent network, which we call recurrent kernel network (RKN). This RNN arises from the dynamic programming structure used to compute efficiently the substring kernel of Lodhi et al. (2002), a link already exploited by Lei et al. (2017) to derive their sequence neural network, which was a source of inspiration for our work. Both our kernels

rely on a RNN to build a representation of an input sequence by computing a string kernel between this sequence and a set of learnable filters. Yet, our model exhibits several differences with Lei et al. (2017), who use the regular substring kernel of Lodhi et al. (2002) and compose this representation with another non-linear map—by applying an activation function to the output of the RNN. By contrast, we obtain a different RKHS directly by relaxing the substring kernel to allow for inexact matching at the compared positions, and embed the Nyström approximation within the RNN. The resulting feature space can be interpreted as a continuous neighborhood around all substrings (with gaps) of the described sequence. Furthermore, our RNN provides a finite-dimensional approximation of the relaxed kernel, relying on the Nyström approximation method (Williams and Seeger, 2001). As a consequence, RKNs may be learned in an unsupervised manner (in such a case, the goal is to approximate the kernel map), and with supervision with back-propagation, which may be interpreted as performing feature selection in the RKHS.

Contributions. In this chapter, we make the following contributions:
• We generalize convolutional kernel networks for sequences (Chen et al., 2019a) to allow gaps, an important option for biological data. As in Chen et al. (2019a), we observe that the kernel formulation brings practical benefits over traditional CNNs or RNNs (Hochreiter et al., 2007) when the amount of labeled data is small or moderate.
• We provide a kernel point of view on recurrent neural networks with new unsupervised and supervised learning algorithms. The resulting feature map can be interpreted in terms of gappy motifs, and end-to-end learning amounts to performing feature selection.

3.2. Background on Kernel Methods and String Kernels

Kernel methods consist in mapping data points living in a set $\mathcal{X}$ to a possibly infinite-dimensional Hilbert space $\mathcal{H}$, through a mapping function $\Phi : \mathcal{X} \to \mathcal{H}$, before learning a simple predictive model in $\mathcal{H}$ (Scholkopf and Smola, 2001). The so-called kernel trick allows to perform learning without explicitly computing this mapping, as long as the inner-product $K(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathcal{H}}$ between two points $\mathbf{x}, \mathbf{x}'$ can be efficiently computed. Whereas kernel methods traditionally lack scalability since they require computing an $n \times n$ Gram matrix, where n is the amount of training data, recent approaches based on approximations have managed to make kernel methods work at large scale in many cases (Rahimi and Recht, 2008; Williams and Seeger, 2001).

For sequences in $\mathcal{X} = \mathcal{A}^*$, which is the set of sequences of any possible length over an alphabet $\mathcal{A}$, the mapping Φ often enumerates subsequence content. For instance, the spectrum kernel maps sequences to a fixed-length vector $\Phi(\mathbf{x}) = (\phi_u(\mathbf{x}))_{u \in \mathcal{A}^k}$, where $\mathcal{A}^k$ is the set of k-mers—length-k sequence of characters in $\mathcal{A}$ for some k in $\mathbb{N}$, and $\phi_u(\mathbf{x})$ counts the number of occurrences of u in $\mathbf{x}$ (Leslie et al., 2001). The mismatch kernel (Leslie et al., 2004) operates similarly, but $\phi_u(\mathbf{x})$ counts the occurrences of u up to a few mismatched letters, which is useful when k is large and exact occurrences are rare.

3.2.1. Substring Kernels

As Lei et al. (2017), we consider the substring kernel introduced in Lodhi et al. (2002), which allows to model the presence of gaps when trying to match a substring u to a sequence $\mathbf{x}$. Modeling gaps requires introducing the following notation: $\mathcal{I}_{\mathbf{x},k}$ denotes the set of indices of sequence $\mathbf{x}$ with k elements $(i_1, \dots, i_k)$ satisfying $1 \leq i_1 < \cdots < i_k \leq |\mathbf{x}|$, where $|\mathbf{x}|$ is the length of $\mathbf{x}$. For an index set $\mathbf{i}$ in $\mathcal{I}_{\mathbf{x},k}$, we may now consider the subsequence $\mathbf{x}_\mathbf{i} = (\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_k})$ of $\mathbf{x}$ indexed by $\mathbf{i}$. Then, the substring kernel takes the same form as the mismatch and spectrum kernels, but $\phi_u(\mathbf{x})$ counts all—consecutive or not—subsequences of $\mathbf{x}$ equal to u, and weights them by the number of gaps. Formally, we consider a parameter λ in $[0, 1]$, and $\phi_u(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x},k}} \lambda^{\text{gaps}(\mathbf{i})} \delta(u, \mathbf{x}_\mathbf{i})$, where $\delta(u, v) = 1$ if and only if $u = v$, and 0 otherwise, and $\text{gaps}(\mathbf{i}) := i_k - i_1 - k + 1$ is the number of gaps in the index set $\mathbf{i}$. When λ is small, gaps are heavily penalized, whereas a value close to 1 gives similar weights to all occurrences. Ultimately, the resulting kernel between two sequences $\mathbf{x}$ and $\mathbf{x}'$ is

$$\mathcal{K}^s(\mathbf{x}, \mathbf{x}') := \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x},k}} \sum_{\mathbf{j} \in \mathcal{I}_{\mathbf{x}',k}} \lambda^{\text{gaps}(\mathbf{i})} \lambda^{\text{gaps}(\mathbf{j})} \delta\left(\mathbf{x}_\mathbf{i}, \mathbf{x}'_\mathbf{j}\right). \tag{3.1}$$

As we will see in Section 3.3, our RKN model relies on (3.1), but unlike Lei et al. (2017), we replace the quantity $\delta(\mathbf{x}_\mathbf{i}, \mathbf{x}'_\mathbf{j})$ that matches exact occurrences by a relaxation, allowing more subtle comparisons. Then, we will show that the model can be interpreted as a gap-allowed extension of CKNs for sequences. We also note that even though $\mathcal{K}^s$ seems computationally expensive at first sight, it was shown in Lodhi et al. (2002) that (3.1) admits a dynamic programming structure leading to efficient computations.

3.2.2. The Nyström Method

When computing the Gram matrix is infeasible, it is typical to use kernel approximations (Rahimi and Recht, 2008; Williams and Seeger, 2001), consisting in finding a q-dimensional mapping $\psi : \mathcal{X} \to \mathbb{R}^q$ such that the kernel $K(\mathbf{x}, \mathbf{x}')$ can be approximated by a Euclidean inner-product $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle_{\mathbb{R}^q}$. Then, kernel methods can be simulated by a linear model operating on $\psi(\mathbf{x})$, which does not raise scalability issues if q is reasonably small. Among kernel approximations, the Nyström method consists in projecting points of the RKHS onto a q-dimensional subspace, allowing to represent points into a q-dimensional coordinate system.

Specifically, consider a collection of $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_q\}$ points in $\mathcal{X}$ and consider the subspace

$$\mathcal{E} = \text{Span}(\Phi(\mathbf{z}_1), \dots, \Phi(\mathbf{z}_q)) \quad \text{and define} \quad \psi(\mathbf{x}) = K_{ZZ}^{-\frac{1}{2}} K_Z(\mathbf{x}),$$

where K_{ZZ} is the $q \times q$ Gram matrix of K restricted to the samples $\mathbf{z}_1, \dots, \mathbf{z}_q$ and $K_Z(\mathbf{x})$ in $\mathbb{R}^q$ carries the kernel values $K(\mathbf{x}, \mathbf{z}_j), j = 1, \dots, q$. This approximation only requires q kernel evaluations and often retains good performance for learning. Interestingly as noted in Mairal (2016), $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle_{\mathbb{R}^q}$ is exactly the inner-product in $\mathcal{H}$ between the projections of $\Phi(\mathbf{x})$ and $\Phi(\mathbf{x}')$ onto $\mathcal{E}$, which remain in $\mathcal{H}$.

When $\mathcal{X}$ is a Euclidean space—this can be the case for sequences when using a one-hot encoding representation, as discussed later— a good set of anchor points $\mathbf{z}_j$ can be obtained by simply clustering the data and choosing the centroids as anchor points (Zhang

et al., 2008). The goal is then to obtain a subspace $\mathcal{E}$ that spans data as best as possible. Otherwise, previous works on kernel networks (Chen et al., 2019a; Mairal, 2016) have also developed procedures to learn the set of anchor points end-to-end by optimizing over the learning objective. This approach can then be seen as performing feature selection in the RKHS.

3.3. Recurrent Kernel Networks

With the previous tools in hand, we now introduce RKNs. We show that it admits variants of CKNs, substring and local alignment kernels as special cases, and we discuss its relation with RNNs.

3.3.1. A Continuous Relaxation of the Substring Kernel Allowing Mismatches

From now on, and with an abuse of notation, we represent characters in $\mathcal{A}$ as vectors in $\mathbb{R}^d$. For instance, when using one-hot encoding, a DNA sequence $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_m)$ of length m can be seen as a 4-dimensional sequence where each $\mathbf{x}_j$ in $\{0,1\}^4$ has a unique non-zero entry indicating which of $\{A, C, G, T\}$ is present at the j-th position, and we denote by $\mathcal{X}$ the set of such sequences. We now define the single-layer RKN as a generalized substring kernel (3.1) in which the indicator function $\delta(\mathbf{x_i}, \mathbf{x'_j})$ is replaced by a kernel for k-mers:

$$\mathcal{K}_k(\mathbf{x}, \mathbf{x}') := \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x},k}} \sum_{\mathbf{j} \in \mathcal{I}_{\mathbf{x}',k}} \lambda_{\mathbf{x},\mathbf{i}} \lambda_{\mathbf{x},\mathbf{j}} e^{-\frac{\alpha}{2}\|\mathbf{x_i} - \mathbf{x'_j}\|^2}, \tag{3.2}$$

where we assume that the vectors representing characters have unit ℓ_2-norm, such that $e^{-\frac{\alpha}{2}\|\mathbf{x_i} - \mathbf{x'_j}\|^2} = e^{\alpha(\langle \mathbf{x_i}, \mathbf{x'_j}\rangle - k)} = \prod_{t=1}^{k} e^{\alpha\left(\langle \mathbf{x}_{i_t}, \mathbf{x}'_{j_t}\rangle - 1\right)}$ is a dot-product kernel, and $\lambda_{\mathbf{x},\mathbf{i}} = \lambda^{\mathrm{gaps}(\mathbf{i})}$ if we follow (3.1).

For $\lambda = 0$ and using the convention $0^0 = 1$, all the terms in these sums are zero except those for k-mers with no gap, and we recover the kernel of the CKN model of Chen et al. (2019a) with a convolutional structure—up to the normalization, which is done k-mer-wise in CKN instead of position-wise.

Compared to (3.1), the relaxed version (3.2) accommodates inexact k-mer matching. This is important for protein sequences, where it is common to consider different similarities between amino acids in terms of substitution frequency along evolution (Henikoff and Henikoff, 1992). This is also reflected in the underlying sequence representation in the RKHS illustrated in Figure 3.1: by considering $\varphi(.)$ the kernel mapping and RKHS $\mathcal{H}$ such that $K(\mathbf{x_i}, \mathbf{x'_j}) = e^{-\frac{\alpha}{2}\|\mathbf{x_i} - \mathbf{x'_j}\|^2} = \langle \varphi(\mathbf{x_i}), \varphi(\mathbf{x'_j})\rangle_{\mathcal{H}}$, we have

$$\mathcal{K}_k(\mathbf{x}, \mathbf{x}') = \left\langle \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x},k}} \lambda_{\mathbf{x},\mathbf{i}} \varphi(\mathbf{x_i}), \sum_{\mathbf{j} \in \mathcal{I}_{\mathbf{x}',k}} \lambda_{\mathbf{x},\mathbf{j}} \varphi(\mathbf{x'_j}) \right\rangle_{\mathcal{H}}. \tag{3.3}$$

A natural feature map for a sequence $\mathbf{x}$ is therefore $\Phi_k(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x},k}} \lambda_{\mathbf{x},\mathbf{i}} \varphi(\mathbf{x_i})$: using the RKN amounts to representing $\mathbf{x}$ by a mixture of continuous neighborhoods $\varphi(\mathbf{x_i}) : \mathbf{z} \mapsto e^{-\frac{\alpha}{2}\|\mathbf{x_i} - \mathbf{z}\|^2}$ centered on all its k-subsequences $\mathbf{x_i}$, each weighted by the corresponding $\lambda_{\mathbf{x},\mathbf{i}}$ (*e.g.*, $\lambda_{\mathbf{x},\mathbf{i}} = \lambda^{\mathrm{gaps}(\mathbf{i})}$). As a particular case, a feature map of CKN (Chen et al., 2019a) is the sum of the kernel mapping of all the k-mers without gap.

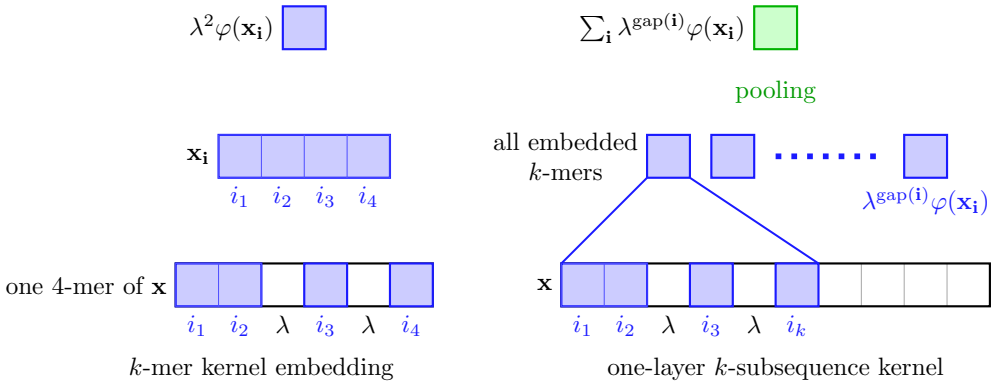


Figure 3.1.: Representation of sequences in a RKHS based on $\mathcal{K}_k$ with $k = 4$ and $\lambda_{\mathbf{x},\mathbf{i}} = \lambda^{\text{gaps(i)}}$.

3.3.2. Extension to All k-mers and Relation to the Local Alignment Kernel

Dependency in the hyperparameter k can be removed by summing $\mathcal{K}_k$ over all possible values:

$$\mathcal{K}_{\text{sum}}(\mathbf{x}, \mathbf{x}') := \sum_{k=1}^{\infty} \mathcal{K}_k(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{\max(|\mathbf{x}|,|\mathbf{x}'|)} \mathcal{K}_k(\mathbf{x}, \mathbf{x}').$$

Interestingly, we note that $\mathcal{K}_{\text{sum}}$ admits the local alignment kernel of Vert et al. (2004) as a special case. More precisely, local alignments are defined via the tensor product set $\mathcal{A}_k(\mathbf{x}, \mathbf{x}') := \mathcal{I}_{\mathbf{x},k} \times \mathcal{I}_{\mathbf{x}',k}$, which contains all possible alignments of k positions between a pair of sequences $(\mathbf{x}, \mathbf{x}')$. The local alignment score of each such alignment $\pi = (\mathbf{i}, \mathbf{j})$ in $\mathcal{A}_k(\mathbf{x}, \mathbf{x}')$ is defined, by Vert et al. (2004), as $S(\mathbf{x}, \mathbf{x}', \pi) := \sum_{t=1}^{k} s(\mathbf{x}_{i_t}, \mathbf{x}'_{j_t}) - \sum_{t=1}^{k-1}[g(i_{t+1} - i_t - 1) + g(j_{t+1} - j_t - 1)]$, where s is a symmetric substitution function and g is a gap penalty function. The local alignment kernel in Vert et al. (2004) can then be expressed in terms of the above local alignment scores (Thrm. 1.7 in Vert et al. (2004)):

$$K_{LA}(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{\infty} K^k_{LA}(\mathbf{x}, \mathbf{x}') := \sum_{k=1}^{\infty} \sum_{\pi \in \mathcal{A}_k(\mathbf{x},\mathbf{x}')} \exp(\beta S(\mathbf{x}, \mathbf{x}', \pi)) \quad \text{for some } \beta > 0. \quad (3.4)$$

When the gap penalty function is linear—that is, $g(x) = cx$ with $c > 0$, K^k_{LA} becomes

$$\begin{aligned} K^k_{LA}(\mathbf{x}, \mathbf{x}') &= \sum_{\pi \in \mathcal{A}_k(\mathbf{x},\mathbf{x}')} \exp(\beta S(\mathbf{x}, \mathbf{x}', \pi)) \\ &= \sum_{(\mathbf{i},\mathbf{j}) \in \mathcal{A}_k(\mathbf{x},\mathbf{x}')} e^{-c\beta\text{gaps(i)}} e^{-c\beta\text{gaps(j)}} \prod_{t=1}^{k} e^{\beta s(\mathbf{x}_{i_t}, \mathbf{x}'_{j_t})}. \end{aligned}$$

When $s(\mathbf{x}_{i_t}, \mathbf{x}'_{j_t})$ can be written as an inner-product $\langle \psi_s(\mathbf{x}_{i_t}), \psi_s(\mathbf{x}'_{j_t}) \rangle$ between normalized vectors, we see that K_{LA} becomes a special case of (3.2)—up to a constant factor—with $\lambda_{\mathbf{x},\mathbf{i}} = e^{-c\beta\text{gaps(i)}}$, $\alpha = \beta$.

This observation sheds new lights on the relation between the substring and local alignment kernels, which will inspire new algorithms in the sequel. To the best of our knowledge, the link we will provide between RNNs and local alignment kernels is also new.

3.3.3. Nyström Approximation and Recurrent Neural Networks

As in CKNs, we now use the Nyström approximation method as a building block to make the above kernels tractable. According to (3.3), we may first use the Nyström method described in Section 3.2.2 to find an approximate embedding for the quantities $\varphi(\mathbf{x_i})$, where $\mathbf{x_i}$ is one of the k-mers represented as a matrix in $\mathbb{R}^{k\times d}$. This is achieved by choosing a set $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_q\}$ of anchor points in $\mathbb{R}^{k\times d}$, and by encoding $\varphi(\mathbf{x_i})$ as $K_{ZZ}^{-1/2} K_Z(\mathbf{x_i})$—where K is the kernel of $\mathcal{H}$. Such an approximation for k-mers yields the q-dimensional embedding for the sequence $\mathbf{x}$:

$$\psi_k(\mathbf{x}) = \sum_{\mathbf{i}\in\mathcal{I}_{\mathbf{x},k}} \lambda_{\mathbf{x},\mathbf{i}} K_{ZZ}^{-\frac{1}{2}} K_Z(\mathbf{x_i}) = K_{ZZ}^{-\frac{1}{2}} \sum_{\mathbf{i}\in\mathcal{I}_{\mathbf{x},k}} \lambda_{\mathbf{x},\mathbf{i}} K_Z(\mathbf{x_i}). \tag{3.5}$$

Then, an approximate feature map $\psi_{\text{sum}}(\mathbf{x})$ for the kernel $\mathcal{K}_{\text{sum}}$ can be obtained by concatenating the embeddings $\psi_1(\mathbf{x}), \dots, \psi_k(\mathbf{x})$ for k large enough.

The anchor points as motifs. The continuous relaxation of the substring kernel presented in (3.2) allows us to learn anchor points that can be interpreted as sequence motifs, where each position can encode a mixture of letters. This can lead to more relevant representations than k-mers for learning on biological sequences. For example, the fact that a DNA sequence is bound by a particular transcription factor can be associated with the presence of a T followed by either a G or an A, followed by another T, would require two k-mers but a single motif (Chen et al., 2019a). Our kernel is able to perform such a comparison.

Efficient computations of $\mathcal{K}_k$ and $\mathcal{K}_{\text{sum}}$ approximation via RNNs. A naive computation of $\psi_k(\mathbf{x})$ would require enumerating all substrings present in the sequence, which may be exponentially large when allowing gaps. For this reason, we use the classical dynamic programming approach of substring kernels (Lei et al., 2017; Lodhi et al., 2002). Consider then the computation of $\psi_j(\mathbf{x})$ defined in (3.5) for $j = 1, \dots, k$ as well as a set of anchor points $Z_k = \{\mathbf{z}_1, \dots, \mathbf{z}_q\}$ with the $\mathbf{z}_i$'s in $\mathbb{R}^{d\times k}$. We also denote by Z_j the set obtained when keeping only j-th first positions (columns) of the $\mathbf{z}_j$'s, leading to $Z_j = \{[\mathbf{z}_1]_{1:j}, \dots, [\mathbf{z}_q]_{1:j}\}$, which will serve as anchor points for the kernel $\mathcal{K}_j$ to compute $\psi_j(\mathbf{x})$. Finally, we denote by $\mathbf{z}_i^j$ in $\mathbb{R}^d$ the j-th column of $\mathbf{z}_i$ such that $\mathbf{z}_i = [\mathbf{z}_i^1, \dots, \mathbf{z}_i^k]$. Then, the embeddings $\psi_1(\mathbf{x}), \dots, \psi_k(\mathbf{x})$ can be computed recursively by using the following theorem:

Theorem 3.1. *For any $j \in \{1, \dots, k\}$ and $t \in \{1, \dots, |\mathbf{x}|\}$,*

$$\psi_j(\mathbf{x}_{1:t}) = K_{Z_jZ_j}^{-\frac{1}{2}} \begin{cases} \mathbf{c}_j[t] & \text{if } \lambda_{\mathbf{x},\mathbf{i}} = \lambda^{|\mathbf{x}|-i_1-j+1}, \\ \mathbf{h}_j[t] & \text{if } \lambda_{\mathbf{x},\mathbf{i}} = \lambda^{gaps(\mathbf{i})}, \end{cases} \tag{3.6}$$

where $c_j[t]$ and $h_j[t]$ form a sequence of vectors in $\mathbb{R}^q$ indexed by t such that $c_j[1] = h_j[1] = 0$, and $c_0[t]$ is a vector that contains only ones, while the sequence obeys the recursion

$$\begin{aligned} \mathbf{c}_j[t] &= \lambda \mathbf{c}_j[t-1] + \mathbf{c}_{j-1}[t-1] \odot \mathbf{b}_j[t] \quad 1 \le j \le k, \\ \mathbf{h}_j[t] &= \mathbf{h}_j[t-1] + \mathbf{c}_{j-1}[t-1] \odot \mathbf{b}_j[t] \quad 1 \le j \le k, \end{aligned} \tag{3.7}$$

where $\odot$ *is the elementwise multiplication operator and* $\mathbf{b}_j[t]$ *is a vector in* $\mathbb{R}^q$ *whose entry* i *in* $\{1, \dots, q\}$ *is* $e^{-\frac{\alpha}{2}\|\mathbf{x}_t - \mathbf{z}_j^i\|^2} = e^{\alpha(\langle \mathbf{x}_t, \mathbf{z}_j^i \rangle - 1)}$ *and* $\mathbf{x}_t$ *is the* t*-th character of* $\mathbf{x}$*.*

A proof is provided in Appendix 3.A and is based on classical recursions for computing the substring kernel, which were interpreted as RNNs by Lei et al. (2017). The main difference in the RNN structure we obtain is that their non-linearity is applied over the outcome of the network, leading to a feature map formed by composing the feature map of the substring kernel of Lodhi et al. (2002) and another one from a RKHS that contains their non-linearity. By contrast, our non-linearities are built explicitly in the substring kernel, by relaxing the indicator function used to compare characters. The resulting feature map is a continuous neighborhood around all substrings of the described sequence. In addition, the Nyström method yields an orthogonalization factor $K_{ZZ}^{-1/2}$ to the output $K_Z(\mathbf{x})$ of the network to compute our approximation, which is perhaps the only non-standard component of our RNN. This factor provides an interpretation of $\psi(\mathbf{x})$ as a kernel approximation. As discussed next, it makes it possible to learn the anchor points by k-means, see Chen et al. (2019a), which also makes the initialization of the supervised learning procedure simple without having to deal with the scaling of the initial motifs/filters $\mathbf{z}_j$.

Learning the anchor points Z. We now turn to the application of RKNs to supervised learning. Given n sequences $\mathbf{x}^1, \dots, \mathbf{x}^n$ in $\mathcal{X}$ and their associated labels $y^1, \dots, y^n$ in $\mathcal{Y}$, *e.g.*, $\mathcal{Y} = \{-1, 1\}$ for binary classification or $\mathcal{Y} = \mathbb{R}$ for regression, our objective is to learn a function in the RKHS $\mathcal{H}$ of $\mathcal{K}_k$ by minimizing

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}^i), y^i) + \frac{\mu}{2} \|f\|_{\mathcal{H}}^2,$$

where $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a convex loss function that measures the fitness of a prediction $f(\mathbf{x}^i)$ to the true label y^i and μ controls the smoothness of the predictive function. After injecting our kernel approximation $\mathcal{K}_k(\mathbf{x}, \mathbf{x}') \simeq \langle \psi_k(\mathbf{x}), \psi_k(\mathbf{x}') \rangle_{\mathbb{R}^q}$, the problem becomes

$$\min_{\mathbf{w} \in \mathbb{R}^q} \frac{1}{n} \sum_{i=1}^{n} L\left(\langle \psi_k(\mathbf{x}^i), \mathbf{w} \rangle, y^i\right) + \frac{\mu}{2} \|\mathbf{w}\|^2. \tag{3.8}$$

Following Chen et al. (2019a); Mairal (2016), we can learn the anchor points Z without exploiting training labels, by applying a k-means algorithm to all (or a subset of) the k-mers extracted from the database and using the obtained centroids as anchor points. Importantly, once Z has been obtained, the linear function parametrized by $\mathbf{w}$ is still optimized with respect to the supervised objective (3.8). This procedure can be thought of as learning a general representation of the sequences disregarding the supervised task, which can lead to a relevant description while limiting overfitting.

Another strategy consists in optimizing (3.8) jointly over $(Z, \mathbf{w})$, after observing that $\psi_k(\mathbf{x}) = K_{ZZ}^{-1/2} \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x},k}} \lambda_{\mathbf{x},\mathbf{i}} K_Z(\mathbf{x}_\mathbf{i})$ is a smooth function of Z. Learning can be achieved by using back-propagation over $(Z, \mathbf{w})$, or by using an alternating minimization strategy between Z and $\mathbf{w}$. It leads to an end-to-end scheme where both the representation and the function defined over this representation are learned with respect to the supervised objective (3.8). Back-propagation rules for most operations are classical, except for the matrix inverse square root function, which is detailed in Appendix 3.B. Initialization is also parameter-free since the unsupervised learning approach may be used for that.

3.3.4. Extensions

Multilayer construction. In order to account for long-range dependencies, it is possible to construct a multilayer model based on kernel compositions similar to Lei et al. (2017). Assume that $\mathcal{K}_k^{(n)}$ is the n-th layer kernel and $\Phi_k^{(n)}$ its mapping function. The corresponding $(n+1)$-th layer kernel is defined as

$$\mathcal{K}_k^{(n+1)}(\mathbf{x},\mathbf{x}') = \sum_{\mathbf{i}\in\mathcal{I}_{\mathbf{x},k},\mathbf{j}\in\mathcal{I}_{\mathbf{x}',k}} \lambda_{\mathbf{x},\mathbf{i}}^{(n+1)}\lambda_{\mathbf{x}',\mathbf{j}}^{(n+1)} \prod_{t=1}^{k} K_{n+1}(\Phi_k^{(n)}(\mathbf{x}_{1:i_t}), \Phi_k^{(n)}(\mathbf{x}'_{1:j_t})), \tag{3.9}$$

where K_{n+1} will be defined in the sequel and the choice of weights $\lambda_{\mathbf{x},\mathbf{i}}^{(n)}$ slightly differs from the single-layer model. We choose indeed $\lambda_{\mathbf{x},\mathbf{i}}^{(N)} = \lambda^{\text{gaps}(\mathbf{i})}$ only for the last layer N of the kernel, which depends on the number of gaps in the index set $\mathbf{i}$ but not on the index positions. Since (3.9) involves a kernel K_{n+1} operating on the representation of prefix sequences $\Phi_k^{(n)}(\mathbf{x}_{1:t})$ from layer n, the representation makes sense only if $\Phi_k^{(n)}(\mathbf{x}_{1:t})$ carries mostly local information close to position t. Otherwise, information from the beginning of the sequence would be overrepresented. Ideally, we would like the range-dependency of $\Phi_k^{(n)}(\mathbf{x}_{1:t})$ (the size of the window of indices before t that influences the representation, akin to receptive fields in CNNs) to grow with the number of layers in a controllable manner. This can be achieved by choosing $\lambda_{\mathbf{x},\mathbf{i}}^{(n)} = \lambda^{|\mathbf{x}|-i_1-k+1}$ for $n < N$, which assigns exponentially more weights to the k-mers close to the end of the sequence.

For the first layer, we recover the single-layer network $\mathcal{K}_k$ defined in (3.2) by defining $\Phi_k^{(0)}(\mathbf{x}_{1:i_k}) = \mathbf{x}_{i_k}$ and $K_1(\mathbf{x}_{i_k}, \mathbf{x}'_{j_k}) = e^{\alpha(\langle \mathbf{x}_{i_k}, \mathbf{x}'_{j_k}\rangle - 1)}$. For $n > 1$, it remains to define K_{n+1} to be a homogeneous dot-product kernel, as used for instance in CKNs (Mairal, 2016):

$$K_{n+1}(\mathbf{u},\mathbf{u}') = \|\mathbf{u}\|_{\mathcal{H}_n}\|\mathbf{u}\|_{\mathcal{H}_n}\kappa_n\left(\left\langle \frac{\mathbf{u}}{\|\mathbf{u}\|_{\mathcal{H}_n}}, \frac{\mathbf{u}'}{\|\mathbf{u}'\|_{\mathcal{H}_n}}\right\rangle_{\mathcal{H}_n}\right) \quad \text{with} \quad \kappa_n(t) = e^{\alpha_n(t-1)}. \tag{3.10}$$

Note that the Gaussian kernel K_1 used for 1st layer may also be written as (3.10) since characters are normalized. As for CKNs, the goal of homogenization is to prevent norms to grow/vanish exponentially fast with n, while dot-product kernels lend themselves well to neural network interpretations.

As detailed in Appendix 3.C, extending the Nyström approximation scheme for the multilayer construction may be achieved in the same manner as with CKNs—that is, we learn one approximate embedding $\psi_k^{(n)}$ at each layer, allowing to replace the inner-products $\langle \Phi_k^{(n)}(\mathbf{x}_{1:i_t}), \Phi_k^{(n)}(\mathbf{x}'_{1:j_t})\rangle$ by their approximations $\langle \psi_k^{(n)}(\mathbf{x}_{1:i_t}), \psi_k^{(n)}(\mathbf{x}'_{1:j_t})\rangle$, and it is easy to show that the interpretation in terms of RNNs is still valid since $\mathcal{K}_k^{(n)}$ has the same sum structure as (3.2).

3.4. Experiments

We evaluate RKN and compare it to typical string kernels and RNN for protein fold recognition. Pytorch code is provided with the submission and additional details given in Appendix 3.D.

3.4.1. Protein Fold Recognition on SCOP 1.67

Sequencing technologies provide access to gene and, indirectly, protein sequences for yet poorly studied species. In order to predict the 3D structure and function from the linear sequence of these proteins, it is common to search for evolutionary related ones, a problem known as homology detection. When no evolutionary related protein with known structure is available, a—more difficult—alternative is to resort to protein fold recognition. We evaluate our RKN on such a task, where the objective is to predict which proteins share a 3D structure with the query (Rangwala and Karypis, 2005).

Here we consider the Structural Classification Of Proteins (SCOP) version 1.67 (Murzin et al., 1995). We follow the preprocessing procedures of Håndstad et al. (2007) and remove the sequences that are more than 95% similar, yielding 85 fold recognition tasks. Each positive training set is then extended with Uniref50 to make the dataset more balanced, as proposed in Hochreiter et al. (2007). The resulting dataset can be downloaded from `http://www.bioinf.jku.at/software/LSTM_protein`. The number of training samples for each task is typically around 9,000 proteins, whose length varies from tens to thousands of amino-acids. In all our experiments we use logistic loss. We measure classification performances using auROC and auROC50 scores (area under the ROC curve and up to 50% false positives).

For CKN and RKN, we evaluate both one-hot encoding of amino-acids by 20-dimensional binary vectors and an alternative representation relying on the BLOSUM62 substitution matrix (Henikoff and Henikoff, 1992). Specifically in the latter case, we represent each amino-acid by the centered and normalized vector of its corresponding substitution probabilities with other amino-acids. The local alignment kernel (3.4), which we include in our comparison, natively uses BLOSUM62.

Hyperparameters. We follow the training procedure of CKN presented in Chen et al. (2019a). Specifically, for each of the 85 tasks, we hold out one quarter of the training samples as a validation set, use it to tune α, gap penalty λ and the regularization parameter μ in the prediction layer. These parameters are then fixed across datasets. RKN training also relies on the alternating strategy used for CKN: we use an Adam algorithm to update anchor points, and the L-BFGS algorithm to optimize the prediction layer. We train 100 epochs for each dataset: the initial learning rate for Adam is fixed to 0.05 and is halved as long as there is no decrease of the validation loss for 5 successive epochs. We fix k to 10, the number of anchor points q to 128 and use single layer CKN and RKN throughout the experiments.

Implementation details for unsupervised models. The anchor points for CKN and RKN are learned by k-means on 30,000 extracted k-mers from each dataset. The resulting sequence representations are standardized by removing mean and dividing by standard deviation and are used within a logistic regression classifier. α in Gaussian kernel and the parameter λ are chosen based on validation loss and are fixed across the datasets. μ for regularization is chosen by a 5-fold cross validation on each dataset. As before, we fix k to 10 and the number of anchor points q to 1024. Note that the performance could be improved with larger q as observed in Chen et al. (2019a), at a higher computational cost.

Table 3.1.: Average auROC and auROC50 for SCOP fold recognition benchmark. LA-kernel uses BLOSUM62 to compare amino acids which is a little different from our encoding approach. Details about pairwise statistical tests between methods can be found in Appendix 3.D.

Method	pooling	one-hot		BLOSUM62	
		auROC	auROC50	auROC	auROC50
GPkernel (Håndstad et al., 2007)		0.844	0.514		
SVM-pairwise (Liao and Noble, 2003)		0.724	0.359	–	–
Mismatch (Leslie et al., 2004)		0.814	0.467		
LA-kernel (Saigo et al., 2004)		–	–	0.834	0.504
LSTM (Hochreiter et al., 2007)		0.830	0.566	–	–
CKN-seq (Chen et al., 2019a)	mean	0.827	0.536	0.843	0.563
CKN-seq (Chen et al., 2019a)	max	0.837	0.572	0.866	0.621
CKN-seq (unsup)(Chen et al., 2019a)	mean	0.804	0.493	0.827	0.548
RKN ($\lambda = 0$)	mean	0.829	0.542	0.838	0.563
RKN	mean	0.829	0.541	0.840	0.571
RKN ($\lambda = 0$)	max	0.840	0.575	0.862	0.618
RKN	max	0.844	**0.587**	**0.871**	**0.629**
RKN (unsup)	mean	0.805	0.504	0.833	0.570

Comparisons and results. The results are shown in Table 3.1. The blosum62 version of CKN and RKN outperform all other methods. Improvement against the mismatch and LA kernels is likely caused by end-to-end trained kernel networks learning a task-specific representation in the form of a sparse set of motifs, whereas data-independent kernels lead to learning a dense function over the set of descriptors. This difference can have a regularizing effect akin to the ℓ_1-norm in the parametric world, by reducing the dimension of the learned linear function w while retaining relevant features for the prediction task. GPkernel also learns motifs, but relies on the exact presence of discrete motifs. Finally, both LSTM and Lei et al. (2017) are based on RNNs but are outperformed by kernel networks. The latter was designed and optimized for NLP tasks and yields a 0.4 auROC50 on this task.

RKNs outperform CKNs, albeit not by a large margin. Interestingly, as the two kernels only differ by their allowing gaps when comparing sequences, this results suggests that this aspect is not the most important for identifying common foldings in a one versus all setting: as the learned function discriminates on fold from all others, it may rely on coarser features and not exploit more subtle ones such as gappy motifs. In particular, the advantage of the LA-kernel against its mismatch counterpart is more likely caused by other differences than gap modeling, namely using a max rather than a mean pooling of k-mer similarities across the sequence, and a general substitution matrix rather than a Dirac function to quantify mismatches.

Additional details and results, scatter plots, and pairwise tests between methods to assess the statistical significance of our conclusions are provided in Appendix 3.D. Note that when $k = 14$, the auROC and auROC50 further increase to 0.877 and 0.636 respectively.

Table 3.2.: Classification accuracy for SCOP 2.06. The complete table with error bars can be found in Appendix 3.D.

Method	♯Params	Accuracy on SCOP 2.06			Level-stratified accuracy (top1/top5/top10)		
		top 1	top 5	top 10	family	superfamily	fold
PSI-BLAST	-	84.53	86.48	87.34	82.20/84.50/85.30	86.90/88.40/89.30	18.90/35.10/35.10
DeepSF	920k	73.00	90.25	94.51	75.87/91.77/95.14	72.23/90.08/94.70	51.35/67.57/72.97
CKN (128 filters)	211k	76.30	92.17	95.27	83.30/94.22/96.00	74.03/91.83/95.34	43.78/67.03/77.57
CKN (512 filters)	843k	84.11	94.29	96.36	**90.24/95.77/97.21**	82.33/94.20/96.35	45.41/69.19/79.73
RKN (128 filters)	211k	77.82	92.89	95.51	76.91/93.13/95.70	78.56/92.98/95.53	60.54/83.78/**90.54**
RKN (512 filters)	843k	**85.29**	**94.95**	**96.54**	84.31/94.80/96.74	**85.99/95.22/96.60**	**71.35/84.86**/89.73

3.4.2. Protein Fold Classification on SCOP 2.06

We further benchmark RKN in a fold classification task, following the protocols used in Hou et al. (2018). Specifically, the training and validation datasets are composed of 14699 and 2013 sequences from SCOP 1.75, belonging to 1195 different folds. The test set consists of 2533 sequences from SCOP 2.06, after removing the sequences with similarity greater than 40% with SCOP 1.75. The input sequence feature is represented by a vector of 45 dimensions, consisting of a 20-dimensional one-hot encoding of the sequence, a 20-dimensional position-specific scoring matrix (PSSM) representing the profile of amino acids, a 3-class secondary structure represented by a one-hot vector and a 2-class solvent accessibility. We further normalize each type of the feature vectors to have unit ℓ_2-norm, which is done for each sequence position. More dataset details can be found in Hou et al. (2018). We use mean pooling for both CKN and RKN models, as it is more stable during training for multi-class classification. The other hyperparameters are chosen in the same way as previously. More details about hyperparameter search grid can be found in Appendix 3.D.

The accuracy results are obtained by averaging 10 different runs and are shown in Table 3.2, stratified by prediction difficulty (family/superfamily/fold, more details can be found in Hou et al. (2018)). By contrast to what we observed on SCOP 1.67, RKN sometimes yields a large improvement on CKN for fold classification, especially for detecting distant homologies. This suggests that accounting for gaps does help in some fold prediction tasks, at least in a multi-class context where a single function is learned for each fold.

Appendix

3.A. Nyström Approximation for Single-Layer RKN

We detail here the Nyström approximation presented in Section 3.3.3, which we recall here for a sequence $\mathbf{x}$:

$$\psi_k(\mathbf{x}) = K_{ZZ}^{-1/2} \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x},k}} \lambda_{\mathbf{x},\mathbf{i}} K_Z(\mathbf{x}_\mathbf{i}). \tag{3.11}$$

Consider then the computation of $\psi_j(\mathbf{x})$ defined in (3.11) for $j = 1, \dots, k$ given a set of anchor points $Z_k = \{\mathbf{z}_1, \dots, \mathbf{z}_q\}$ with the $\mathbf{z}_i$'s in $\mathbb{R}^{d \times k}$. Given the notations introduced in Section 3.3.3, we are now in shape to prove Theorem 3.1.

Proof. The proof is based on Theorem 1 of Lei et al. (2017) and definition 2 of Lodhi et al. (2002). For $\mathbf{i} \in \mathcal{I}_{\mathbf{x},j}$, let us denote by $\mathbf{i}' = (i_1, \dots, i_{j-1})$ the $j-1$ first entries of $\mathbf{i}$. We first notice that for the Gaussian kernel K, we have the following factorization relation for $i = 1, \dots, q$

$$\begin{aligned} K(\mathbf{x}_\mathbf{i}, [\mathbf{z}_i]_{1:j}) &= e^{\alpha(\langle \mathbf{x}_\mathbf{i}, [\mathbf{z}_i]_{1:j} \rangle - j)} \\ &= e^{\alpha(\langle \mathbf{x}_{\mathbf{i}'}, [\mathbf{z}_i]_{1:j-1} \rangle - (j-1))} e^{\alpha(\langle \mathbf{x}_{i_j}, \mathbf{z}_j \rangle - 1)} \\ &= K(\mathbf{x}_{\mathbf{i}'}, [\mathbf{z}_i]_{1:j-1}) e^{\alpha(\langle \mathbf{x}_{i_j}, \mathbf{z}_j \rangle - 1)}. \end{aligned}$$

Thus

$$K_{Z_j}(\mathbf{x}_\mathbf{i}) = K_{Z_{j-1}}(\mathbf{x}_{\mathbf{i}'}) \odot \mathbf{b}_j[i_j],$$

with $\mathbf{b}_j[t]$ defined as in the theorem.

Let us denote $\sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x}_{1:t},j}} \lambda_{\mathbf{x}_{1:t},\mathbf{i}} K_{Z_j}(\mathbf{x}_\mathbf{i})$ by $\tilde{\mathbf{c}}_j[t]$ if $\lambda_{\mathbf{x},\mathbf{i}} = \lambda^{|\mathbf{x}| - i_1 - j + 1}$ and by $\tilde{\mathbf{h}}_j[t]$ if $\lambda_{\mathbf{x},\mathbf{i}} = \lambda^{\text{gaps}(\mathbf{i})}$. We want to prove that $\tilde{\mathbf{c}}_j[t] = \mathbf{c}_j[t]$ and $\tilde{\mathbf{h}}_j[t] = \mathbf{h}_j[t]$. First, it is clear that $\tilde{\mathbf{c}}_j[0] = 0$ for any j. We show by induction on j that $\tilde{\mathbf{c}}_j[t] = \mathbf{c}_j[t]$. When $j = 1$, we have

$$\begin{aligned} \tilde{\mathbf{c}}_1[t] &= \sum_{1 \le i_1 \le t} \lambda^{t - i_1} K_{Z_1}(\mathbf{x}_{i_1}) \\ &= \sum_{1 \le i_1 \le t-1} \lambda^{t - i_1} K_{Z_1}(\mathbf{x}_{i_1}) + K_{Z_1}(\mathbf{x}_t), \\ &= \lambda \tilde{\mathbf{c}}_1[t-1] + \mathbf{b}_1[t]. \end{aligned}$$

$\tilde{\mathbf{c}}_1[t]$ and $\mathbf{c}_1[t]$ have the same recursion and initial state thus are identical. When $j > 1$ and suppose that $\tilde{\mathbf{c}}_{j-1}[t] = \mathbf{c}_{j-1}[t]$, then we have

$$\begin{aligned} \tilde{\mathbf{c}}_j[t] &= \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x}_{1:t},j}} \lambda^{t - i_1 - j + 1} K_{Z_j}(\mathbf{x}_\mathbf{i}), \\ &= \underbrace{\sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x}_{1:t-1},j}} \lambda^{t - i_1 - j + 1} K_{Z_j}(\mathbf{x}_\mathbf{i})}_{i_j < t} + \underbrace{\sum_{\mathbf{i}' \in \mathcal{I}_{\mathbf{x}_{1:t-1},j-1}} \lambda^{(t-1) - s_1 - (j-1) + 1} K_{Z_{j-1}}(\mathbf{x}_{i'}) \odot \mathbf{b}_j[t]}_{i_j = t}, \\ &= \lambda \tilde{\mathbf{c}}_j[t-1] + \tilde{\mathbf{c}}_{j-1}[t] \odot \mathbf{b}_j[t], \\ &= \lambda \tilde{\mathbf{c}}_j[t-1] + \mathbf{c}_{j-1}[t] \odot \mathbf{b}_j[t]. \end{aligned}$$

$\tilde{\mathbf{c}}_j[t]$ and $\mathbf{c}_j[t]$ have the same recursion and initial state. We have thus proved that $\tilde{\mathbf{c}}_j[t] = c_j[t]$. Let us move on for proving $\tilde{\mathbf{h}}_j[t] = \mathbf{h}_j[t]$ by showing that they have the same initial state and recursion. It is straightforward that $\tilde{\mathbf{h}}_j[0] = 0$, then for $1 \leq j \leq k$ we have

$$\begin{aligned}\tilde{\mathbf{h}}_j[t] &= \sum_{\mathbf{i}\in\mathcal{I}_{\mathbf{x}_{1:t},j}} \lambda^{i_j - i_1 - j + 1} K_{Z_j}(\mathbf{x}_i), \\ &= \sum_{\mathbf{i}\in\mathcal{I}_{\mathbf{x}_{1:t-1},j}} \lambda^{i_j - i_1 - j + 1} K_{Z_j}(\mathbf{x}_\mathbf{i}) + \sum_{\mathbf{i}'\in\mathcal{I}_{\mathbf{x}_{1:t-1},j-1}} \lambda^{(t-1) - s_1 - (j-1) + 1} K_{Z_{j-1}}(\mathbf{x}_{i'}) \odot \mathbf{b}_j[t] \\ &= \tilde{\mathbf{h}}_j[t-1] + \mathbf{c}_{j-1}[t] \odot \mathbf{b}_j[t].\end{aligned}$$

Therefore $\tilde{\mathbf{h}}_j[t] = \mathbf{h}_j[t]$. □

3.B. Back-propagation for Matrix Inverse Square Root

In Section 3.3.3, we have described an end-to-end scheme to jointly optimize Z and $\mathbf{w}$. The back-propagation of Z requires computing that of the matrix inverse square root operation as it is involved in the approximate feature map of $\mathbf{x}$ as shown in (3.11). The back-propagation formula is given by the following proposition, which is based on an errata of (Mairal, 2016) and we include it here for completeness.

Proposition 3.1. *Given* $\mathbf{A}$ *a symmetric positive definite matrix in* $\mathbb{R}^{n\times n}$ *and the eigencomposition of* $\mathbf{A}$ *is written as* $\mathbf{A} = \mathbf{U}\mathbf{\Delta}\mathbf{U}^\top$ *where* $\mathbf{U}$ *is orthogonal and* $\mathbf{\Delta}$ *is diagonal with eigenvalues* $\delta_1, \ldots, \delta_n$*. Then*

$$d(\mathbf{A}^{-\frac{1}{2}}) = -\mathbf{U}(\mathbf{F} \circ (\mathbf{U}^\top d\mathbf{A}\mathbf{U}))\mathbf{U}^\top. \tag{3.12}$$

Proof. First, let us differentiate with respect to the inverse matrix $\mathbf{A}^{-1}$:

$$\mathbf{A}^{-1}\mathbf{A} = \mathcal{I} \quad \Longrightarrow \quad \mathbf{A}^{-1}d\mathbf{A} + d(\mathbf{A}^{-1})\mathbf{A} = 0 \quad \Longrightarrow \quad d(\mathbf{A}^{-1}) = -\mathbf{A}^{-1}d\mathbf{A}\mathbf{A}^{-1}.$$

Then, by applying the same (classical) trick,

$$\mathbf{A}^{-\frac{1}{2}}\mathbf{A}^{-\frac{1}{2}} = \mathbf{A}^{-1} \quad \Longrightarrow \quad d(\mathbf{A}^{-\frac{1}{2}})\mathbf{A}^{-\frac{1}{2}} + \mathbf{A}^{-\frac{1}{2}}d(\mathbf{A}^{-\frac{1}{2}}) = d(\mathbf{A}^{-1}) = -\mathbf{A}^{-1}d\mathbf{A}\mathbf{A}^{-1}.$$

By multiplying the last relation by $\mathbf{U}^\top$ on the left and by $\mathbf{U}$ on the right.

$$\mathbf{U}^\top d(\mathbf{A}^{-\frac{1}{2}})\mathbf{U}\mathbf{\Delta}^{-\frac{1}{2}} + \mathbf{\Delta}^{-\frac{1}{2}}\mathbf{U}^\top d(\mathbf{A}^{-\frac{1}{2}})\mathbf{U} = -\mathbf{\Delta}^{-1}\mathbf{U}^\top d\mathbf{A}\mathbf{U}\mathbf{\Delta}^{-1}.$$

Note that $\mathbf{\Delta}$ is diagonal. By introducing the matrix $\mathbf{F}$ such that $\mathbf{F}_{kl} = \frac{1}{\sqrt{\delta_k}\sqrt{\delta_l}(\sqrt{\delta_k}+\sqrt{\delta_l})}$, it is then easy to show that

$$\mathbf{U}^\top d(\mathbf{A}^{-\frac{1}{2}})\mathbf{U} = -\mathbf{F} \circ (\mathbf{U}^\top d\mathbf{A}\mathbf{U}),$$

where $\circ$ is the Hadamard product between matrices. Then, we are left with

$$d(\mathbf{A}^{-\frac{1}{2}}) = -\mathbf{U}(\mathbf{F} \circ (\mathbf{U}^\top d\mathbf{A}\mathbf{U}))\mathbf{U}^\top.$$

□

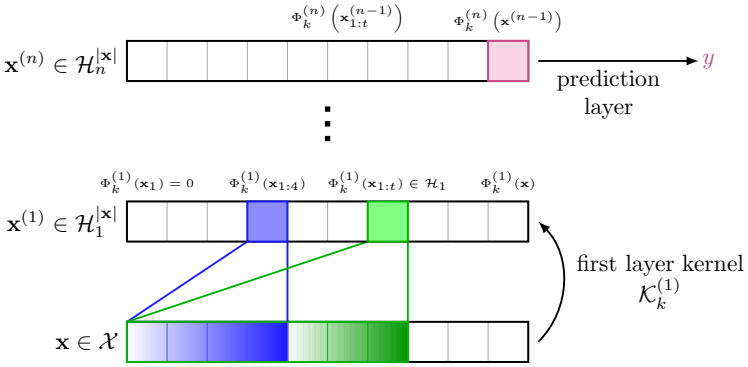


Figure 3.C.1.: Multilayer construction of RKN: an example with $k = 4$.

When doing back-propagation, one is usually interested in computing a quantity $\bar{\mathbf{A}}$ such that given $\bar{\mathbf{B}}$ (with appropriate dimensions), we have

$$\langle \bar{\mathbf{B}}, d(\mathbf{A}^{-\frac{1}{2}}) \rangle_F = \langle \bar{\mathbf{A}}, d\mathbf{A} \rangle_F,$$

see Giles (2008), for instance. Here, $\langle , \rangle_F$ denotes the Frobenius inner product. Then, it is easy to show that

$$\bar{\mathbf{A}} = -\mathbf{U}(\mathbf{F} \circ (\mathbf{U}^\top \bar{\mathbf{B}} \mathbf{U}))\mathbf{U}^\top.$$

3.C. Multilayer Construction of RKN

For multilayer RKN, assume that we have defined $\mathcal{K}^{(n)}$ the n-th layer kernel. To simplify the notation below, we consider that an input sequence $\mathbf{x}$ is encoded at layer n as $\mathbf{x}^{(n)} := (\Phi_k^{(n)}(\mathbf{x}_1), \Phi_k^{(n)}(\mathbf{x}_{1:2}), \ldots, \Phi_k^{(n)}(\mathbf{x}))$ where the feature map at position t is $\mathbf{x}_t^{(n)} = \Phi_k^{(n)}(\mathbf{x}_{1:t})$. The $(n+1)$-layer kernel is defined by induction by

$$\mathcal{K}_k^{(n+1)}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{i} \in \mathcal{I}_{\mathbf{x},k}, \mathbf{j} \in \mathcal{I}_{\mathbf{x}',k}} \lambda_{\mathbf{x},\mathbf{i}}^{(n)} \lambda_{\mathbf{x}',\mathbf{j}}^{(n)} \prod_{t=1}^{k} K_{n+1}(\mathbf{x}_{i_t}^{(n)}, \mathbf{x}_{j_t}'^{(n)}), \tag{3.13}$$

where K_{n+1} is defined in (3.10. With the choice of weights described in Section 3.3.4, the construction scheme for an n-layer RKN is illustrated in Figure 3.C.1 The Nyström approximation scheme for multilayer RKN is straightforward by inductively applying the Nyström method to the kernels $\mathcal{K}^{(1)}, \ldots, \mathcal{K}^{(n)}$ from bottom to top layers. Specifically, assume that $\mathcal{K}^{(n)}(\mathbf{x}, \mathbf{x}')$ is approximated by $\langle \psi_k^{(n)}(\mathbf{x}), \psi_k^{(n)}(\mathbf{x}') \rangle_{\mathbb{R}^{q_n}}$ such that the approximate feature map of $\mathbf{x}^{(n)}$ at position t is $\psi_k^{(n)}(\mathbf{x}_{1:t})$. Now Consider a set of anchor points $Z_k = \{\mathbf{z}_1, \ldots, \mathbf{z}_{q_{n+1}}\}$ with the $\mathbf{z}_i$'s in $\mathbb{R}^{q_n \times k}$ which have unit norm at each column. We use the same notations as in single-layer construction. Then very similar to the single-layer RKN, the embeddings $(\psi_j^{(n+1)}(\mathbf{x}_{1:t}^{(n)}))_{1=1,\ldots,k,t=1,\ldots,|\mathbf{x}^{(n)}|}$ are given by the following recursion

Theorem 3.2. *For any $j \in \{1, \ldots, k\}$ and $t \in \{1, \ldots, |\mathbf{x}^{(n)}|\}$,*

$$\psi_j^{(n+1)}(\mathbf{x}_{1:t}^{(n)}) = K_{Z_j Z_j}^{-1/2} \begin{cases} \mathbf{c}_j[t] & \text{if } \lambda_{\mathbf{x},\mathbf{i}}^{(n)} = \lambda^{|\mathbf{x}^{(n)}| - i_1 - j + 1}, \\ \mathbf{h}_j[t] & \text{if } \lambda_{\mathbf{x},\mathbf{i}}^{(n)} = \lambda^{gaps(\mathbf{i})}, \end{cases}$$

where $c_j[t]$ and $h_j[t]$ form a sequence of vectors in $\mathbb{R}^{q_{n+1}}$ indexed by t such that $c_j[0] = h_j[0] = 0$, and $c_0[t]$ is a vector that contains only ones, while the sequence obeys the recursion

$$\begin{aligned} \mathbf{c}_j[t] &= \lambda \mathbf{c}_j[t-1] + \mathbf{c}_{j-1}[t-1] \odot \mathbf{b}_j[t] \quad 1 \leq j \leq k, \\ \mathbf{h}_j[t] &= \mathbf{h}_j[t-1] + \mathbf{c}_{j-1}[t-1] \odot \mathbf{b}_j[t] \quad 1 \leq j \leq k, \end{aligned} \tag{3.14}$$

where $\odot$ is the elementwise multiplication operator and $\mathbf{b}_j[t]$ whose entry i is $K_{n+1}(\mathbf{z}_j^i, \mathbf{x}_t^{(n)}) = \|\mathbf{x}_t^{(n)}\| \kappa_n \left(\left\langle \mathbf{z}_j^i, \frac{\mathbf{x}_t^{(n)}}{\|\mathbf{x}_t^{(n)}\|} \right\rangle \right)$.

Proof. The proof can be obtained by that of Theorem 3.1 by replacing the Gaussian kernel $e^{\alpha(\langle \mathbf{x}_t, \mathbf{z}_j^i \rangle)}$ with the kernel $K_{n+1}(\mathbf{x}_t^{(n)}, \mathbf{z}_j^i)$. □

3.D. Additional Experimental Material

In this section, we provide additional details about experiments and scatter plots with pairwise statistical tests.

3.D.1. Protein Fold Recognition on SCOP 1.67

Hyperparameter search grids. Here, we first provide the grids used for hyperparameter search. In our experiments, we use σ instead of α such that $\alpha = 1/k\sigma^2$. The search range is specified in Table 3.D.1.

Table 3.D.1.: Hyperparameter search range.

hyperparameter	search range
σ ($\alpha = 1/k\sigma^2$)	[0.3;0.4;0.5;0.6]
μ for mean pooling	[1e-06;1e-05;1e-04]
μ for max pooling	[0.001;0.01;0.1;1.0]
λ	integer multipliers of 0.05 in [0;1]

Comparison of unsupervised CKNs and RKNs. Then, we provide an additional table of results to compare the unsupervised models of CKN and RKN. In this unsupervised regime, mean pooling perform better than max pooling, which is different than what we have observed in the supervised case. RKN tend to work better than CKN, while RKN-sum—that is, using the kernel $\mathcal{K}_{\text{sum}}$ instead of $\mathcal{K}_k$, works better than RKN.

Study of filter number q and size k. Here we use max pooling and fix σ to 0.4 and λ to 0.1. When q varies k is fixed to 10 and q is fixed to 128 when k varies. We show here the performance of RKN with different choices of q and k. The gap hyperparameter λ is chosen optimally for each q and k. The results are shown in Figure 3.D.1.

Table 3.D.2.: Comparison of unsupervised CKN and RKN with 1024 anchor points.

Method	Pooling	one-hot		BLOSUM62	
		auROC	auROC50	auROC	auROC50
CKN	mean	0.804	0.493	0.827	0.548
CKN	max	0.795	0.480	0.821	0.545
RKN ($\lambda = 0$)	mean	0.804	0.500	0.833	0.565
RKN	mean	0.805	0.504	0.833	0.570
RKN ($\lambda = 0$)	max	0.795	0.482	0.824	0.537
RKN	max	0.801	0.492	0.824	0.542
RKN-sum ($\lambda = 0$)	mean	0.820	0.526	0.834	0.567
RKN-sum	mean	0.821	0.527	0.834	0.565
RKN-sum ($\lambda = 0$)	max	0.825	0.526	0.837	0.563
RKN-sum	max	0.825	0.528	0.837	0.564

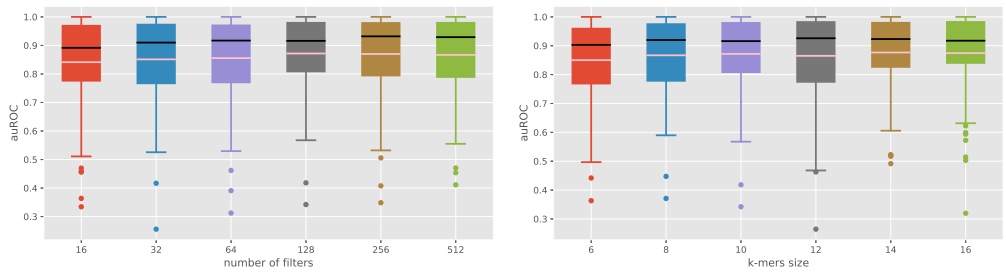


Figure 3.D.1.: Boxplots when varying filter number q (left) and filter size (right).

Discussion about complexity. Performing backpropgation with our RKN model has the same complexity has a performing a similar step within a recurrent neural network, up to the computation of the inverse square root matrix $K_{ZZ}^{-1/2}$, which has complexity $O(q^3)$. When q is reasonably small $q = 128$ in our experiments, such a complexity is negligible. For instance, one forward pass with a minibatch of $b = 128$ sequences of length m yields a complexity $O(k^2mbq)$, which can typically be much greater than q^3.

Computing infrastructures. Experiments were conduced by using a shared GPU cluster, in large parts build with Nvidia gamer cards (Titan X, GTX1080TI). About 10 of these GPUs were used simultaneously to perform the experiments of this chapter.

Scatter plots and statistical testing. Even though each method was run only one time for each task, the 85 tasks allow us to conduct statistical testing when comparing two methods. In Figure 3.D.2, we provide pairwise comparisons allowing us to assess the statistical significance of various conclusions drawn in the chapter. We use a Wilcoxon signed-rank test to provide p-values.

3.D.2. Protein Fold Classification on SCOP 2.06

Hyperparameter search grids. We provide the grids used for hyperparameter search, shown in Table 3.D.3.

Table 3.D.3.: Hyperparameter search range for SCOP 2.06.

hyperparameter	search range
σ ($\alpha = 1/k\sigma^2$)	[0.3;0.4;0.5;0.6]
μ	[0.01;0.03;0.1;0.3;1.0;3.0;10.0]
λ	integer multipliers of 0.05 in [0;1]

Complete results with error bars. The classification accuracy for CKNs and RKNs on protein fold classification on SCOP 2.06 are obtained by averaging on 10 runs with different seeds. The results are shown in Table 3.D.4 with error bars.

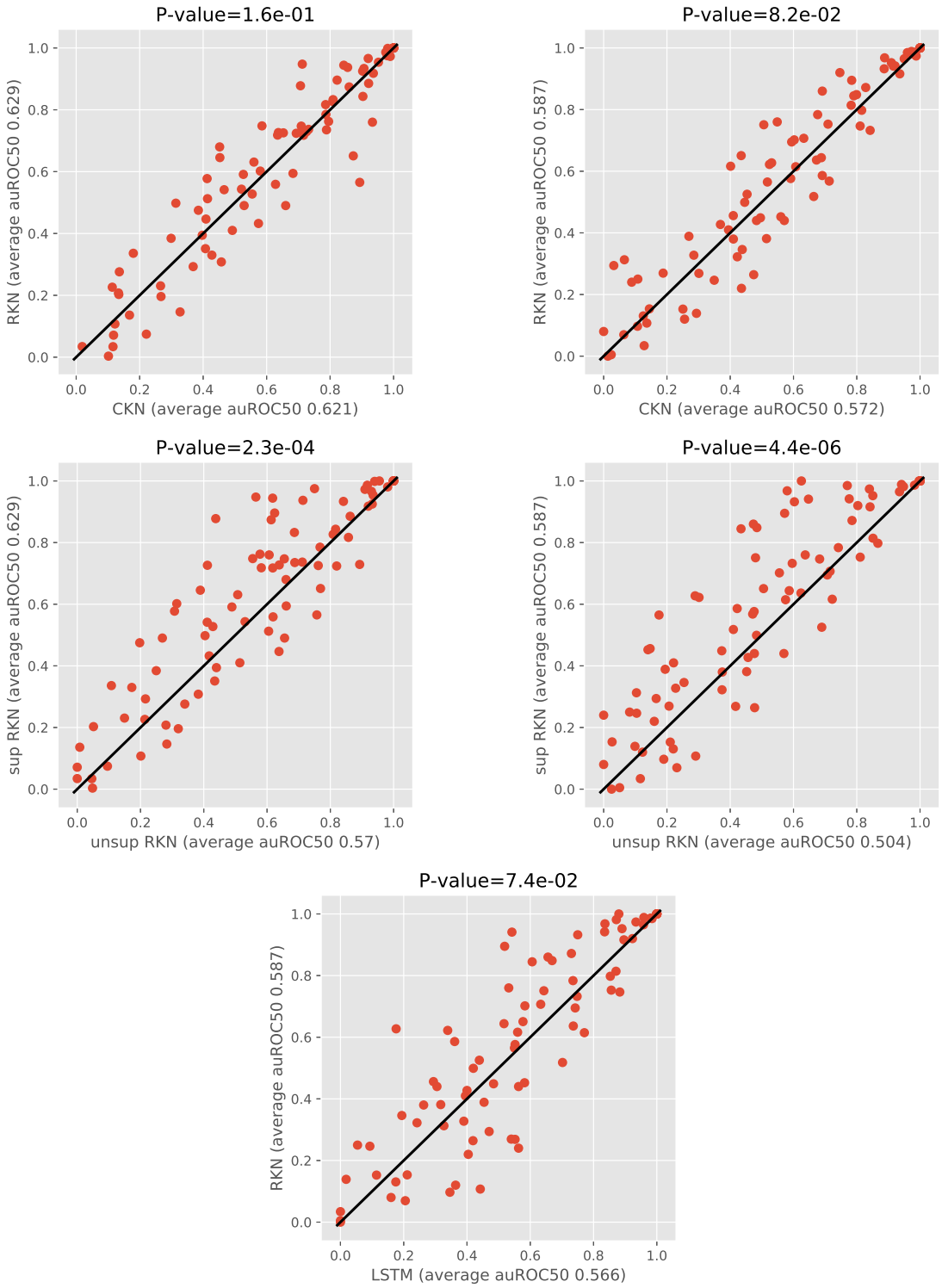


Figure 3.D.2.: Scatterplots when comparing pairs of methods. In particular, we want to compare RKN vs CKN (top); , RKN vs RKN (unsup) (middle); RKN vs. LSTM (bottom).

Table 3.D.4.: Classification accuracy for SCOP 2.06 on all (top) and level-stratified (bottom) test data. For CKNs and RKNs, the results are obtained over 10 different runs.

Method	Params	Accuracy on SCOP 2.06		
		top 1	top 5	top 10
PSI-BLAST	-	84.53	86.48	87.34
DeepSF	920k	73.00	90.25	94.51
CKN (128 filters)	211k	76.30±0.70	92.17±0.16	95.27±0.17
CKN (512 filters)	843k	84.11±0.16	94.29±0.20	96.36±0.13
RKN (128 filters)	211k	77.82±0.35	92.89±0.19	95.51±0.20
RKN (512 filters)	843k	85.29±0.27	94.95±0.15	96.54±0.12

Method	Level-stratified accuracy (top1/top5/top10)		
	family	superfamily	fold
PSI-BLAST	82.20/84.50/85.30	86.90/88.40/89.30	18.90/35.10/35.10
DeepSF	75.87/91.77/95.14	72.23/90.08/94.70	51.35/67.57/72.97
CKN (128 filters)	83.30±0.78/94.22±0.25/96.00±0.26	74.03±0.87/91.83±0.24/95.34±0.20	43.78±3.59/67.03±3.38/77.57±3.64
CKN (512 filters)	90.24±0.16/95.77±0.21/97.21±0.15	82.33±0.19/94.20±0.21/96.35±0.13	45.41±1.62/69.19±1.79/79.73±3.68
RKN (128 filters)	76.91±0.87/93.13±0.17/95.70±0.37	78.56±0.40/92.98±0.22/95.53±0.18	60.54±2.76/83.78±2.96/90.54±1.35
RKN (512 filters)	84.31±0.61/94.80±0.21/96.74±0.29	85.99±0.30/95.22±0.16/96.60±0.12	71.35±1.32/84.86±2.16/89.73±1.08

4

Convolutional Kernel Networks for Graph-Structured Data

Contents

We introduce a family of multilayer graph kernels and establish new links between graph convolutional neural networks and kernel methods. Our approach generalizes convolutional kernel networks to graph-structured data, by representing graphs as a sequence of kernel feature maps, where each node carries information about local graph substructures. On the one hand, the kernel point of view offers an unsupervised, expressive, and easy-to-regularize data representation, which is useful when limited samples are available. On the other hand, our model can also be trained end-to-end on large-scale data, leading to new types of graph convolutional neural networks. We show that our method achieves competitive performance on several graph classification benchmarks, while offering simple model interpretation. Our code is freely available at https://github.com/claying/GCKN.

The chapter is based on the following publication:

D. Chen, L. Jacob, and J. Mairal. Convolutional kernel networks for graph-structured data. In *International Conference on Machine Learning (ICML)*, 2020

4.1. Introduction

Graph kernels are classical tools for representing graph-structured data (see Kriege et al., 2020, for a survey). Most successful examples represent graphs as very-high-dimensional feature vectors that enumerate and count occurences of local graph sub-structures. In order to perform well, a graph kernel should be as expressive as possible, *i.e.*, able to distinguish graphs with different topological properties (Kriege et al., 2018), while

admitting polynomial-time algorithms for its evaluation. Common sub-structures include walks (Gärtner et al., 2003), shortest paths (Borgwardt and Kriegel, 2005), subtrees (Shervashidze et al., 2011), or graphlets (Shervashidze et al., 2009).

Graph kernels have shown to be expressive enough to yield good empirical results, but decouple data representation and model learning. In order to obtain task-adaptive representations, another line of research based on neural networks has been developed recently (Niepert et al., 2016; Kipf and Welling, 2017; Xu et al., 2019; Verma et al., 2018). The resulting tools, called graph neural networks (GNNs), are conceptually similar to convolutional neural networks (CNNs) for images; they provide graph-structured multilayer models, where each layer operates on the previous layer by aggregating local neighbor information. Even though harder to regularize than kernel methods, these models are trained end-to-end and are able to extract features adapted to a specific task. In a recent work, Xu et al. (2019) have shown that the class of GNNs based on neighborhood aggregation is at most as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test, on which the WL kernel is based (Shervashidze et al., 2011), and other types of network architectures than simple neighborhood aggregation are needed for more powerful features.

Since GNNs and kernel methods seem to benefit from different characteristics, several links have been drawn between both worlds in the context of graph modeling. For instance, Lei et al. (2017) introduce a class of GNNs whose output lives in the reproducing kernel Hilbert space (RKHS) of a WL kernel. In this line of research, the kernel framework is essentially used to design the architecture of the GNN since the final model is trained as a classical neural network. This is also the approach used by Zhang et al. (2018a) and Morris et al. (2019). By contrast, Du et al. (2019) adopt an opposite strategy and leverage a GNN architecture to design new graph kernels, which are equivalent to infinitely-wide GNNs initialized with random weights and trained with gradient descent. Other attempts to merge neural networks and graph kernels involve using the metric induced by graph kernels to initialize a GNN (Navarin et al., 2018), or using graph kernels to obtain continuous embeddings that are plugged to neural networks (Nikolentzos et al., 2018).

In this chapter, we go a step further in bridging graph neural networks and kernel methods by proposing an explicit multilayer kernel representation, which can be used either as a traditional kernel method, or trained end-to-end as a GNN when enough labeled data are available. The multilayer construction allows to compute a series of maps which account for local sub-structures ("receptive fields") of increasing size. The graph representation is obtained by pooling the final representations of its nodes. The resulting kernel extends to graph-structured data the concept of convolutional kernel networks (CKNs), which was originally designed for images and sequences (Mairal, 2016; Chen et al., 2019a). As our representation of nodes is built by iteratively aggregating representations of their outgoing paths, our model can also be seen as a multilayer extension of path kernels. Relying on paths rather than neighbors for the aggregation step makes our approach more expressive than the GNNs considered in Xu et al. (2019), which implicitly rely on walks and whose power cannot exceed the Weisfeiler-Lehman (WL) graph isomorphism test. Even with medium/small path lengths (which leads to reasonable computational complexity in practice), we show that the resulting representation outperforms walk or WL kernels.

Our model called graph convolutional kernel network (GCKN) relies on the successive uses of the Nyström method (Williams and Seeger, 2001) to approximate the feature map at each layer, which makes our approach scalable. GCKNs can then be interpreted as a new type of graph neural network whose filters may be learned without supervision, by following kernel approximation principles. Such unsupervised graph representation is known to be particularly effective when small amounts of labeled data are available. Similar to CKNs, our model can also be trained end-to-end, as a GNN, leading to task-adaptive representations, with a computational complexity similar to that of a GNN when the path lengths are small enough.

Notation. A graph G is defined as a triplet $(\mathcal{V}, \mathcal{E}, a)$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is the set of edges, and $a : \mathcal{V} \to \Sigma$ is a function that assigns attributes, either discrete or continous, from a set Σ to nodes in the graph. A path is a sequence of distinct vertices linked by edges and we denote by $\mathcal{P}(G)$ and $\mathcal{P}_k(G)$ the set of paths and paths of length k in G, respectively. In particular, $\mathcal{P}_0(G)$ is reduced to $\mathcal{V}$. We also denote by $\mathcal{P}_k(G, u) \subset \mathcal{P}_k(G)$ the set of paths of length k starting from u in $\mathcal{V}$. For any path p in $\mathcal{P}(G)$, we denote by $a(p)$ in $\Sigma^{|p|+1}$ the concatenation of node attributes in this path. We replace $\mathcal{P}$ with $\mathcal{W}$ to denote the corresponding sets of walks by allowing repeated nodes.

4.2. Related Work on Graph Kernels

Graph kernels were originally introduced by Gärtner et al. (2003) and Kashima et al. (2003), and have been the subject of intense research during the last twenty years (see the reviews of Vishwanathan et al., 2010; Kriege et al., 2020).

In this chapter, we consider graph kernels that represent a graph as a feature vector counting the number of occurrences of some local connected sub-structure. Enumerating common local sub-structures between two graphs is unfortunately often intractable; for instance, enumerating common subgraphs or common paths is known to be NP-hard (Gärtner et al., 2003). For this reason, the literature on graph kernels has focused on alternative structures allowing for polynomial-time algorithms, *e.g.*, walks.

More specifically, we consider graph kernels that perform pairwise comparisons between local sub-structures centered at every node. Given two graphs $G = (\mathcal{V}, \mathcal{E}, a)$ and $G' = (\mathcal{V}', \mathcal{E}', a')$, we consider the kernel

$$K(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \kappa_{\text{base}}(l_G(u), l_{G'}(u')), \tag{4.1}$$

where the base kernel κ_{base} compares a set of local patterns centered at nodes u and u', denoted by $l_G(u)$ and $l_{G'}(u')$, respectively. For simplicity, we will omit the notation $l_G(u)$ in the rest of the chapter, and the base kernel will be simply written $\kappa_{\text{base}}(u, u')$ with an abuse of notation. As noted by Lei et al. (2017); Kriege et al. (2020), this class of kernels covers most of the examples mentioned in the introduction.

Walks and path kernels. Since computing all path co-occurences between graphs is NP-hard, it is possible instead to consider paths of length k, which can be reasonably enumerated if k is small enough, or the graphs are sparse. Then, we may define the

kernel $K_{\text{path}}^{(k)}$ as (4.1) with

$$\kappa_{\text{base}}(u, u') = \sum_{p \in \mathcal{P}_k(G,u)} \sum_{p' \in \mathcal{P}_k(G',u')} \delta(a(p), a'(p')), \tag{4.2}$$

where $a(p)$ represents the attributes for path p in G, and δ is the Dirac kernel such that $\delta(a(p), a'(p')) = 1$ if $a(p) = a'(p')$ and 0 otherwise.

It is also possible to define a variant that enumerates all paths up to length k, by simply adding the kernels $K_{\text{path}}^{(i)}$:

$$K_{\text{path}}(G, G') = \sum_{i=0}^{k} K_{\text{path}}^{(i)}(G, G'). \tag{4.3}$$

Similarly, one may also consider using walks by simply replacing the notation $\mathcal{P}$ by $\mathcal{W}$ in the previous definitions.

Weisfeiler-Lehman subtree kernels. A subtree is a subgraph with a tree structure. It can be extended to subtree patterns (Shervashidze et al., 2011; Bach, 2008) by allowing nodes to be repeated, just as the notion of walks extends that of paths. All previous subtree kernels compare subtree patterns instead of subtrees. Among them, the Weisfeiler-Lehman (WL) subtree kernel is one of the most widely used graph kernels to capture such patterns. It is essentially based on a mechanism to augment node attributes by iteratively aggregating and hashing the attributes of each node's neighborhoods. After i iterations, we denote by a_i the new node attributes for graph $G = (\mathcal{V}, \mathcal{E}, a)$, which is defined in Algorithm 1 of Shervashidze et al. (2011) and then the WL subtree kernel after k iterations is defined, for two graphs $G = (\mathcal{V}, \mathcal{E}, a)$ and $G' = (\mathcal{V}', \mathcal{E}', a')$, as

$$K_{WL}(G, G') = \sum_{i=0}^{k} K_{\text{subtree}}^{(i)}(G, G'), \tag{4.4}$$

where

$$K_{\text{subtree}}^{(i)}(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \kappa_{\text{subtree}}^{(i)}(u, u'), \tag{4.5}$$

with $\kappa_{\text{subtree}}^{(i)}(u, u') = \delta(a_i(u), a_i'(u'))$ and the attributes $a_i(u)$ capture subtree patterns of depth i rooted at node u.

4.3. Graph Convolutional Kernel Networks

In this section, we introduce our model, which builds upon the concept of graph-structured feature maps, following the terminology of convolutional neural networks.

Definition 4.1 (Graph feature map)**.** *Given a graph $G = (\mathcal{V}, \mathcal{E}, a)$ and a RKHS $\mathcal{H}$, a graph feature map is a mapping $\varphi : \mathcal{V} \to \mathcal{H}$, which associates to every node a point in $\mathcal{H}$ representing information about local graph substructures.*

We note that the definition matches that of convolutional kernel networks (Mairal, 2016) when the graph is a two-dimensional grid. Generally, the map φ depends on the graph G, and can be seen as a collection of $|\mathcal{V}|$ elements of $\mathcal{H}$ describing its nodes. The kernel associated to the feature maps φ, φ' for two graphs G, G', is defined as

$$K(G,G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \langle \varphi(u), \varphi'(u') \rangle_{\mathcal{H}} = \langle \Phi(G), \Phi(G') \rangle_{\mathcal{H}}, \tag{4.6}$$

with

$$\Phi(G) = \sum_{u \in \mathcal{V}} \varphi(u) \quad \text{and} \quad \Phi(G') = \sum_{u \in \mathcal{V}'} \varphi'(u). \tag{4.7}$$

The RKHS of K can be characterized by using Theorem 4.2 in Appendix 4.A. It is the space of functions $f_z : G \mapsto \langle z, \Phi(G) \rangle_{\mathcal{H}}$ for all z in $\mathcal{H}$ endowed with a particular norm.

Note that even though graph feature maps φ, φ' are graph-dependent, learning with K is possible as long as they all map nodes to the same RKHS $\mathcal{H}$—as Φ will then also map all graphs to the same space $\mathcal{H}$. We now detail the full construction of the kernel, starting with a single layer.

4.3.1. Single-Layer Construction of the Feature Map

We propose a single-layer model corresponding to a continuous relaxation of the path kernel. We assume that the input attributes $a(u)$ live in $\mathbb{R}^{q_0}$, such that a graph $G = (\mathcal{V}, \mathcal{E}, a)$ admits a graph feature map $\varphi_0 : \mathcal{V} \to \mathcal{H}_0$ with $\mathcal{H}_0 = \mathbb{R}^{q_0}$ and $\varphi_0(u) = a(u)$. Note that this assumption also allows us to handle discrete labels by using a one-hot encoding strategy—that is *e.g.*, four labels $\{A, B, C, D\}$ are represented by four-dimensional vectors $(1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)$, respectively.

Continuous relaxation of the path kernel. We rely on paths of length k, and introduce the kernel K_1 for graphs G, G' with feature maps φ_0, φ_0' of the form (4.1) with

$$\kappa_{\text{base}}(u,u') = \sum_{p \in \mathcal{P}_k(G,u)} \sum_{p' \in \mathcal{P}_k(G',u')} \kappa_1(\varphi_0(p), \varphi_0'(p')), \tag{4.8}$$

where $\varphi_0(p) = [\varphi_0(p_i)]_{i=0}^k$ denotes the concatenation of $k+1$ attributes along path p, which is an element of $\mathcal{H}_0^{k+1}$, p_i is the i-th node on path p starting from index 0, and κ_1 is a Gaussian kernel comparing such attributes:

$$\kappa_1(\varphi_0(p), \varphi_0'(p')) = e^{-\frac{\alpha_1}{2} \sum_{i=0}^k \|\varphi_0(p_i) - \varphi_0'(p_i')\|_{\mathcal{H}_0}^2}. \tag{4.9}$$

This is an extension of the path kernel, obtained by replacing the hard matching function δ in (4.2) by κ_1, as done for instance by Togninalli et al. (2019) for the WL kernel. This replacement not only allows us to use continuous attributes, but also has important consequences in the discrete case since it allows to perform inexact matching between paths. For instance, when the graph is a chain with discrete attributes—in other words, a string—then, paths are simply k-mers, and the path kernel (with matching function δ) becomes the spectrum kernel for sequences (Leslie et al., 2001). By using κ_1 instead, we obtain the single-layer CKN kernel of Chen et al. (2019a), which performs inexact matching, as the mismatch kernel does (Leslie et al., 2004), and leads to better performances in many tasks involving biological sequences.

From graph feature map φ_0 to graph feature map φ_1. The kernel κ_1 acts on pairs of paths in potentially different graphs, but only through their mappings to the same space $\mathcal{H}_0^{k+1}$. Since κ_1 is positive definite, we denote by $\mathcal{H}_1$ its RKHS and consider its mapping $\phi_1^{\text{path}} : \mathcal{H}_0^{k+1} \to \mathcal{H}_1$ such that

$$\kappa_1(\varphi_0(p), \varphi_0'(p')) = \langle \phi_1^{\text{path}}(\varphi_0(p)), \phi_1^{\text{path}}(\varphi_0'(p')) \rangle_{\mathcal{H}_1}.$$

For any graph G, we can now define a graph feature map $\varphi_1 : \mathcal{V} \to \mathcal{H}_1$, operating on nodes u in $\mathcal{V}$, as

$$\varphi_1(u) = \sum_{p \in \mathcal{P}_k(G,u)} \phi_1^{\text{path}}(\varphi_0(p)). \tag{4.10}$$

Then, the continuous relaxation of the path kernel, denoted by $K_1(G, G')$, can also be written as (4.6) with $\varphi = \varphi_1$, and its underlying kernel representation Φ_1 is given by (4.7). The construction of φ_1 from φ_0 is illustrated in Figure 4.1.

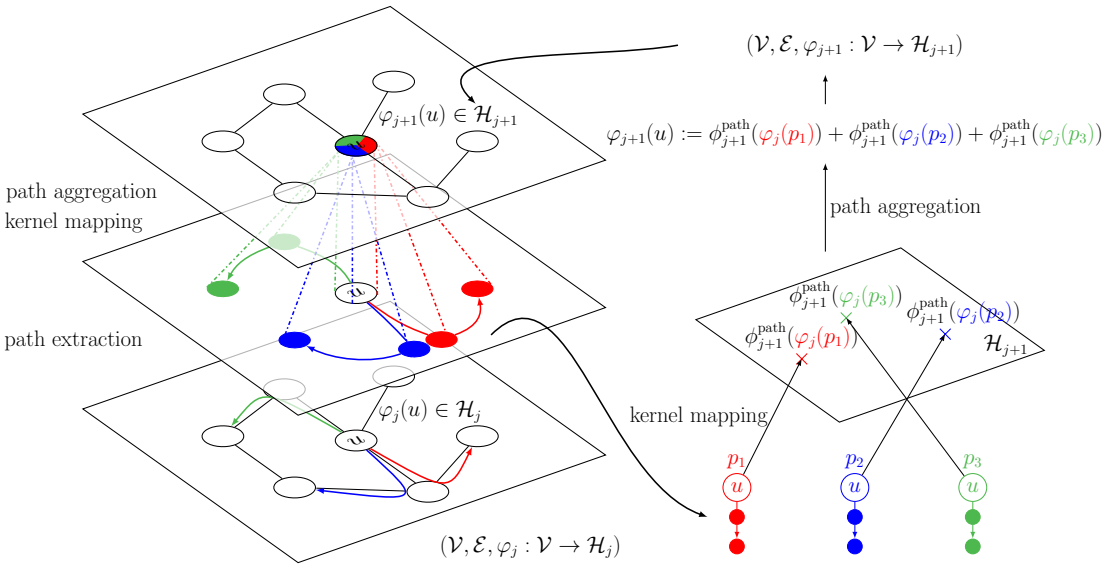


Figure 4.1.: Construction of the graph feature map φ_{j+1} from φ_j given a graph $(\mathcal{V}, \mathcal{E})$. The first step extracts paths of length k (here colored by red, blue and green) from node u, then (on the right panel) maps them to a RKHS $\mathcal{H}_{j+1}$ via the Gaussian kernel mapping. The new map φ_{j+1} at u is obtained by local path aggregation (pooling) of their representations in $\mathcal{H}_{j+1}$. The representations for other nodes can be obtained in the same way. In practice, such a model is implemented by using finite-dimensional embeddings approximating the feature maps, see Section 4.3.2.

The graph feature map φ_0 maps a node (resp a path) to $\mathcal{H}_0$ (resp $\mathcal{H}_0^{k+1}$) which is typically a Euclidean space describing its attributes. By contrast, ϕ_1^{path} is the kernel mapping of the Gaussian kernel κ_1, and maps each path p to a Gaussian function centered at $\varphi_0(p)$—remember indeed that for kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with RKHS $\mathcal{H}$, the kernel mapping is of a data point x is the function $K(x, .) : \mathcal{X} \to \mathbb{R}$. Finally, φ_1 maps each node u to a mixture of Gaussians, each Gaussian function corresponding to a path starting at u.

4.3.2. Concrete Implementation and GCKNs

We now discuss algorithmic aspects, leading to the graph convolutional kernel network (GCKN) model, which consists in building a finite-dimensional embedding $\Psi(G)$ that may be used in various learning tasks without scalability issues. We start here with the single-layer case.

The Nyström method and the single-layer model. A naive computation of the path kernel K_1 requires comparing all pairs of paths in each pair of graphs. To gain scalability, a key component of the CKN model is the Nyström method (Williams and Seeger, 2001), which computes finite-dimensional approximate kernel embeddings. We discuss here the use of such a technique to define finite-dimensional maps $\psi_1 : \mathcal{V} \to \mathbb{R}^{q_1}$ and $\psi_1' : \mathcal{V}' \to \mathbb{R}^{q_1}$ for graphs G, G' such that for all pairs of nodes u, u' in $\mathcal{V}$, $\mathcal{V}'$, respectively,

$$\langle \varphi_1(u), \varphi_1'(u') \rangle_{\mathcal{H}_1} \approx \langle \psi_1(u), \psi_1'(u') \rangle_{\mathbb{R}^{q_1}}.$$

The consequence of such an approximation is that it provides a finite-dimensional approximation Ψ_1 of Φ_1:

$$K_1(G, G') \approx \langle \Psi_1(G), \Psi_1(G') \rangle_{\mathbb{R}^{q_1}}$$
$$\text{with } \Psi_1(G) = \sum_{u \in \mathcal{V}} \psi_1(u).$$

Then, a supervised learning problem with kernel K_1 on a dataset $(G_i, y_i)_{i=1,\dots,n}$, where y_i are labels in $\mathbb{R}$, can be solved by minimizing the regularized empirical risk

$$\min_{w \in \mathbb{R}^{q_1}} \sum_{i=1}^{n} L(y_i, \langle \Psi_1(G_i), w \rangle) + \lambda \|w\|^2, \tag{4.11}$$

where L is a convex loss function. Next, we show that using the Nyström method to approximate the kernel κ_1 yields a new type of GNN, represented by $\Psi_1(G)$, whose filters can be obtained without supervision, or, as discussed later, with back-propagation in a task-adaptive manner.

Specifically, the Nyström method projects points from a given RKHS onto a finite-dimensional subspace and performs all subsequent operations within that subspace. In the context of κ_1, whose RKHS is $\mathcal{H}_1$ with mapping function ϕ_1^{path}, we consider a collection $Z = \{z_1, \dots, z_{q_1}\}$ of q_1 prototype paths represented by attributes in $\mathcal{H}_0^{k+1}$, and we define the subspace $\mathcal{E}_1 = \text{Span}(\phi_1^{\text{path}}(z_1), \dots, \phi_1^{\text{path}}(z_{q_1}))$. Given a new path with attributes z, it is then possible to show (see Chen et al., 2019a) that the projection of path attributes z onto $\mathcal{E}_1$ leads to the q_1-dimensional mapping

$$\psi_1^{\text{path}}(z) = [\kappa_1(z_i, z_j)]_{ij}^{-\frac{1}{2}} [\kappa_1(z_1, z), \dots, \kappa_1(z_{q_1}, z)]^\top,$$

where $[\kappa_1(z_i, z_j)]_{ij}$ is a $q_1 \times q_1$ Gram matrix. Then, the approximate graph feature map ψ_1 is obtained by pooling

$$\psi_1(u) = \sum_{p \in \mathcal{P}_k(G,u)} \psi_1^{\text{path}}(\psi_0(p)) \quad \text{for all} \quad u \in \mathcal{V},$$

where $\psi_0 = \varphi_0$ and $\psi_0(p) = [\psi_0(p_i)]_{i=0,\dots,k}$ in $\mathbb{R}^{q_0(k+1)}$ represents the attributes of path p, with an abuse of notation.

Interpretation as a GNN. When input attributes $\psi_0(u)$ have unit-norm, which is the case if we use one-hot encoding on discrete attributes, the Gaussian kernel κ_1 between two path attributes z, z' in $\mathbb{R}^{q_0(k+1)}$ may be written

$$\kappa_1(z, z') = e^{-\frac{\alpha_1}{2}\|z-z'\|^2} = e^{\alpha_1(z^\top z' - k - 1)} = \sigma_1(z^\top z'), \tag{4.12}$$

which is a dot-product kernel with a non-linear function σ_1. Then, calling Z in $\mathbb{R}^{q_0(k+1)\times q_1}$ the matrix of prototype path attributes, we have

$$\psi_1(u) = \sum_{p \in \mathcal{P}_k(G,u)} \sigma_1(Z^\top Z)^{-\frac{1}{2}} \sigma_1(Z^\top \psi_0(p)), \tag{4.13}$$

where, with an abuse of notation, the non-linear function σ_1 is applied pointwise. Then, the map ψ_1 is build from ψ_0 with the following steps (i) feature aggregation along the paths, (ii) encoding of the paths with a linear operation followed by point-wise non-linearity, (iii) multiplication by the $q_1 \times q_1$ matrix $\sigma_1(Z^\top Z)^{-\frac{1}{2}}$, and (iv) linear pooling. The major difference with a classical GNN is that the "filtering" operation may be interpreted as an orthogonal projection onto a linear subspace, due to the matrix $\sigma_1(Z^\top Z)^{-\frac{1}{2}}$. Unlike the Dirac function, the exponential function σ_1 is differentiable. A useful consequence is the possibility of optimizing the filters Z with back-propagation as detailed below. Note that in practice we add a small regularization term to the diagonal for stability reason: $(\sigma_1(Z^\top Z) + \varepsilon I)^{-\frac{1}{2}}$ with $\varepsilon = 0.01$.

Learning without supervision. Learning the "filters" Z with Nyström can be achieved by simply running a K-means algorithm on path attributes extracted from training data (Zhang et al., 2008). This is the approach adopted for CKNs by Mairal (2016); Chen et al. (2019a), which proved to be very effective as shown in the experimental section.

End-to-end learning with back-propagation. While the previous unsupervised learning strategy consists in finding a good kernel approximation that is independent of labels, it is also possible to learn the parameters Z end-to-end, by minimizing (4.11) jointly with respect to Z and w. The main observations from Chen et al. (2019a) in the context of biological sequences is that such a supervised learning approach may yield good models with much fewer filters q_1 than with the unsupervised learning strategy. We refer the reader to Chen et al. (2019a,b) for how to perform back-propagation with the inverse square root matrix $\sigma_1(Z^\top Z)^{-\frac{1}{2}}$.

Complexity. The complexity for computing the feature map ψ_1 is dominated by the complexity of finding all the paths of length k from each node. This can be done by simply using a depth first search algorithm, whose worst-case complexity for each graph is $O(|\mathcal{V}|d^k)$, where d is the maximum degree of each node, meaning that large k may be used only for sparse graphs. Then, each path is encoded in $O(q_1 q_0(k+1))$ operations; When learning with back-propagation, each gradient step requires computing the eigenvalue decomposition of $\sigma_1(Z^\top Z)^{-\frac{1}{2}}$ whose complexity is $O(q_1^3)$, which is not a computational bottleneck when using mini-batches of order $O(q_1)$, where typical practical values for q_1 are reasonably small, *e.g.*, less than 128.

Algorithm 1 Forward pass for multilayer GCKN

Input: graph $G = (\mathcal{V}, \mathcal{E}, \psi_0 : \mathcal{V} \to \mathbb{R}^{q_0})$, set of anchor points (filters) $Z_j \in \mathbb{R}^{(k+1)q_{j-1} \times q_j}$ for $j = 1, \dots, J$.
for $j = 1, \dots, J$ **do**
  for u **in** $\mathcal{V}$ **do**
    $\psi_j(u) = \sum_{p \in \mathcal{P}_k(G,u)} \psi_j^{\text{path}}(\psi_{j-1}(p))$;
  end for
end for
Global pooling: $\Psi(G) = \sum_{u \in \mathcal{V}} \psi_J(u)$;

4.3.3. Multilayer Extensions

The mechanism to build the feature map φ_1 from φ_0 can be iterated, as illustrated in Figure 4.1 which shows how to build a feature map φ_{j+1} from a previous one φ_j. As discussed by Mairal (2016) for CKNs, the Nyström method may then be extended to build a sequence of finite-dimensional maps $\psi_0, \dots, \psi_J$, and the final graph representation is given by

$$\Psi_J(G) = \sum_{u \in \mathcal{V}} \psi_J(u). \tag{4.14}$$

The computation of $\Psi_J(G)$ is illustrated in Algorithm 1. Here we discuss two possible uses for these additional layers, either to account for more complex structures than paths, or to extend the receptive field of the representation without resorting to the enumeration of long paths.We will denote by k_j the path length used at layer j.

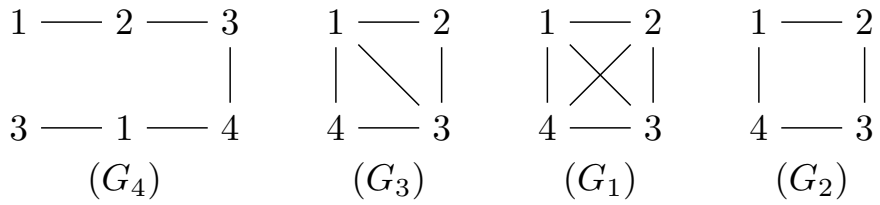
A simple two-layer model to account for subtrees. As emphasized in (4.7), GCKN relies on a representation $\Phi(G)$ of graphs, which is a sum of node-level representations provided by a graph feature map φ. If φ is a sum over paths starting at the represented node, $\Phi(G)$ can simply be written as a sum over all paths in G, consistently with our observation that (4.6) recovers the path kernel when using a Dirac kernel to compare paths in κ_1. The path kernel often leads to good performances, but it is also blind to more complex structures. Figure 4.2 provides a simple example of this phenomenon, using $k = 1$: G_1 and G_3 differ by a single edge, while G_4 has a different set of nodes and a rather different structure. Yet $\mathcal{P}_1(G_3) = \mathcal{P}_1(G_4)$, making $K_1(G_1, G_3) = K_1(G_1, G_4)$ for the path kernel.

Expressing more complex structures requires breaking the succession of linearities introduced in (4.7) and (4.10)—much like pointwise nonlinearities are used in neural networks. Concretely, this effect can simply be obtained by using a second layer with path length $k_2 = 0$—paths are then identified to vertices—which produces the feature map $\varphi_2(u) = \phi_2^{\text{path}}(\varphi_1(u))$, where $\phi_2^{\text{path}} : \mathcal{H}_1 \to \mathcal{H}_2$ is a non-linear kernel mapping. The resulting kernel is then

$$\begin{aligned} K_2(G, G') &= \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \langle \varphi_2(u), \varphi_2'(u') \rangle_{\mathcal{H}_2} \\ &= \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \kappa_2(\varphi_1(u), \varphi_1'(u')). \end{aligned} \tag{4.15}$$

✗ $K_1(G_1, G_3) = K_1(G_1, G_4)$ ✓ $K_1(G_1, G_2) > 0$

✓ $K_2(G_1, G_3) > K_2(G_1, G_4)$ ✗ $K_2(G_1, G_2) = 0$



(G_4) (G_3) (G_1) (G_2)

Figure 4.2.: Example cases using $\kappa_1 = \kappa_2 = \delta$, with path lengths $k_1 = 1$ and $k_2 = 0$; The one-layer kernel K_1 counts the number of common edges while the two-layer K_2 counts the number of nodes with the same set of outgoing edges. The figure suggests using $K_1 + K_2$ to gain expressiveness.

When κ_1 and κ_2 are both Dirac kernels, K_2 counts the number of nodes in G and G' with the exact same set of outgoing paths $\mathcal{P}(G, u)$, as illustrated in Figure 4.2.

Theorem 4.1 further illustrates the effect of using a nonlinear ϕ_2^{path} on the feature map φ_1, by formally linking the walk and WL subtree kernel through our framework.

Theorem 4.1. *Let $G = (\mathcal{V}, \mathcal{E})$, $G' = (\mathcal{V}', \mathcal{E}')$, $\mathcal{M}$ be the set of exact matchings of subsets of the neighborhoods of two nodes, as defined in Shervashidze et al. (2011), and φ defined as in (4.10) with $\kappa_1 = \delta$ and replacing paths by walks. For any $u \in \mathcal{V}$ and $u' \in \mathcal{V}'$ such that $|\mathcal{M}(u, u')| = 1$,*

$$\delta(\varphi_1(u), \varphi_1'(u')) = \kappa_{subtree}^{(k)}(u, u'). \tag{4.16}$$

Recall that when using (4.8) with walks instead of paths and a Dirac kernel for κ_1, the kernel (4.6) with $\varphi = \varphi_1$ is the walk kernel. The condition $|\mathcal{M}(u, u')| = 1$ indicates that u and u' have the same degrees and each of them has distinct neighbors. This can be always ensured by including degree information and adding noise to node attributes. For a large class of graphs, both the walk and WL subtree kernels can therefore be written as (4.6) with the same first layer φ_1 representing nodes by their walk histogram. While walk kernels use a single layer, WL subtree kernels rely on a second layer φ_2 mapping nodes to the indicator function of $\varphi_1(u)$.

Theorem 4.1 also shows that the kernel built in (4.15) is a path-based version of WL subtree kernels, therefore more expressive as it captures subtrees rather than subtree patterns. However, the Dirac kernel lacks flexibility, as it only accounts for pairs of nodes with identical $\mathcal{P}(G, u)$. For example, in Figure 4.2, $K_2(G_1, G_2) = 0$ even though G_1 only differs from G_2 by two edges, because these two edges belong to the set $\mathcal{P}(G, u)$ of all nodes in the graph. In order to retain the stratification by node of (4.15) while allowing for a softer comparison between sets of outgoing paths, we replace δ by the kernel $\kappa_2(\varphi_1(u), \varphi_1'(u')) = e^{-\alpha_2 \|\varphi_1(u) - \varphi_1'(u')\|_{\mathcal{H}_1}^2}$. Large values of α_2 recover the behavior of the Dirac, while smaller values gives non-zero values for similar $\mathcal{P}(G, u)$.

A multilayer model to account for longer paths. In the previous paragraph, we have seen that adding a second layer could bring some benefits in terms of expressiveness,

even when using path lengths $k_2 = 0$. Yet, a major limitation of this model is the exponential complexity of path enumeration, which is required to compute the feature map φ_1, preventing us to use large values of k as soon as the graph is dense. Representing large receptive fields while relying on path enumerations with small k, *e.g.*, $k \leq 3$, is nevertheless possible with a multilayer model. To account for a receptive field of size k, the previous model requires a path enumeration with complexity $O(|\mathcal{V}|d^k)$, whereas the complexity of a multilayer model is linear in k.

4.3.4. Practical Variants

Summing the kernels for different k and different scales. As noted in Section 4.2, summing the kernels corresponding to different values of k provides a richer representation. We also adopt such a strategy, which corresponds to concatenating the feature vectors $\Psi(G)$ obtained for various path lengths k. When considering a multilayer model, it is also possible to concatenate the feature representations obtained at every layer j, allowing to obtain a multi-scale feature representation of the graph and gain expressiveness.

Use of homogeneous dot-product kernel. Instead of the Gaussian kernel (4.9), it is possible to use a homogeneous dot-product kernel, as suggested by Mairal (2016) for CKNs:

$$\kappa_1(z, z') = \|z\|\|z'\|\sigma_1\left(\frac{\langle z, z'\rangle}{\|z\|\|z'\|}\right),$$

where σ_1 is defined in (4.12). Note that when z, z' have unit-norm, we recover the Gaussian kernel (4.9). In this chapter, we use such a kernel for upper layers, or for continuous input attributes when they do not have unit norm. For multilayer models, this homogenization is useful for preventing vanishing or exponentially growing representations. Note that ReLU is also a homogeneous non-linear mapping.

Other types of pooling operations. Another variant consists in replacing the sum pooling operation in (4.13) and (4.14) by a mean or a max pooling. While using max pooling as a heuristic seems to be effective on some datasets, it is hard to justify from a RKHS point of view since max operations typically do not yield positive definite kernels. Yet, such a heuristic is widely adopted in the kernel literature, *e.g.*, for string alignment kernels (Saigo et al., 2004). In order to solve such a discrepancy between theory and practice, Chen et al. (2019b) propose to use the generalized max pooling operator of Murray and Perronnin (2014), which is compatible with the RKHS point of view. Applying the same ideas to GCKNs is straightforward.

Using walk kernel instead of path kernel. One can use a relaxed walk kernel instead of the path kernel in (4.8), at the cost of losing some expressiveness but gaining some time complexity. Indeed, there exists a very efficient recursive way to enumerate walks and thus to compute the resulting approximate feature map in (4.13) for the walk kernel. Specifically, if we denote the k-walk kernel by $\kappa_{\text{walk}}^{(k)}$, then its value between two nodes can be decomposed as the product of the 0-walk kernel between the nodes and the sum

of the $(k-1)$-walk kernel between their neighbors

$$\kappa_{\text{walk}}^{(k)}(u,u') = \kappa_{\text{walk}}^{(0)}(u,u') \sum_{v\in\mathcal{N}(u)} \sum_{v'\in\mathcal{N}(u')} \kappa_{\text{walk}}^{(k-1)}(v,v'),$$

where $\kappa_{\text{walk}}^{(0)}(u,u') = \kappa_1(\varphi_0(u), \varphi_0'(u'))$. After applying the Nyström method, the approximate feature map of the walk kernel is written, similar to (4.13), as

$$\psi_1(u) = \sigma_1(Z^\top Z)^{-\frac{1}{2}} \underbrace{\sum_{p\in\mathcal{W}_k(G,u)} \sigma_1(Z^\top \psi_0(p))}_{c_k(u):=}.$$

Based on the above observation and following similar induction arguments as Chen et al. (2019b), it is not hard to show that $(c_j(u))_{j=1,\dots,k}$ obeys the following recursion

$$c_j(u) = b_j(u) \odot \sum_{v\in\mathcal{N}(u)} c_{j-1}(v),\ 1 \le j \le k,$$

where $\odot$ denotes the element-wise product and $b_j(u)$ is a vector in $\mathbb{R}^{q_1}$ whose entry i in $\{1,\dots,q_1\}$ is $\kappa_1(u, z_i^{(k+1-j)})$ and $z_i^{(k+1-j)}$ denotes the $k+1-j$-th column vector of z_i in $\mathbb{R}^{q_0}$. More details can be found in Appendix 4.C.

4.4. Model Interpretation

Ying et al. (2019) introduced an approach to interpret trained GNN models, by finding a subgraph of an input graph G maximizing the mutual information with its predicted label (note that this approach depends on a specific input graph). We show here how to adapt similar ideas to our framework.

Interpreting GCKN-path and GCKN-subtree. We call GCKN-path our model Ψ_1 with a single layer, and GCKN-subtree our model Ψ_2 with two layers but with $k_2 = 0$, which is the first model presented in Section 4.3.3 that accounts for subtree structures. As these models are built upon path enumeration, we extend the method of Ying et al. (2019) by identifying a small subset of paths in an input graph G preserving the prediction. We then reconstruct a subgraph by merging the selected paths. For simplicity, let us consider a one-layer model. As $\Psi_1(G)$ only depends on G through its set of paths $\mathcal{P}_k(G)$, we note $\Psi_1(\mathcal{P})$ with an abuse of notation for any subset of $\mathcal{P}$ of paths in G, to emphasize the dependency in this set of paths. For a trained model (Ψ_1, w) and a graph G, our objective is to solve

$$\min_{\mathcal{P}'\subseteq\mathcal{P}_k(G)} L(\hat{y}, \langle \Psi_1(\mathcal{P}'), w\rangle) + \mu|\mathcal{P}'|, \tag{4.17}$$

where $\hat{y}$ is the predicted label of G and μ a regularization parameter controlling the number of paths to select. This problem is combinatorial and can be computationally intractable when $\mathcal{P}(G)$ is large. Following Ying et al. (2019), we relax it by using a mask M with values in $[0;1]$ over the set of paths, and replace the number of paths $|\mathcal{P}'|$ by the ℓ_1-norm of M, which is known to have a sparsity-inducing effect (Tibshirani, 1996). The problem then becomes

$$\min_{M\in[0;1]^{|\mathcal{P}_k(G)|}} L(\hat{y}, \langle \Psi_1(\mathcal{P}_k(G)\odot M), w\rangle) + \mu\|M\|_1, \tag{4.18}$$

where $\mathcal{P}_k(G) \odot M$ denotes the use of $M(p)a(p)$ instead of $a(p)$ in the computation of Ψ_1 for all p in $\mathcal{P}_k(G)$. Even though the problem is non-convex due to the non-linear mapping Ψ_1, it may still be solved approximately by using projected gradient-based optimization techniques.

Interpreting multilayer models. By noting that $\Psi_j(G)$ only depends on the union of the set of paths $\mathcal{P}_{k_l}(G)$, for all layers $l \leq j$, we introduce a collection of masks M_l at each layer, and then optimize the same objective as (4.18) over all masks $(M_l)_{l=1,\dots,j}$, with the regularization $\sum_{l=1}^{j} \|M_l\|_1$.

4.5. Experiments

Table 4.1.: Classification accuracies on graphs with discrete node attributes. The accuracies of other models are taken from Du et al. (2019) except LDP, which we evaluate on our splits and for which we tune bin size, the regularization parameter in the SVM and Gaussian kernel bandwidth. Note that RetGK uses a different protocol, performing 10-fold cross-validation 10 times and reporting the average accuracy.

Dataset	MUTAG	PROTEINS	PTC	NCI1	IMDB-B	IMDB-M	COLLAB
size	188	1113	344	4110	1000	1500	5000
classes	2	2	2	2	2	3	3
avg ♯nodes	18	39	26	30	20	13	74
avg ♯edges	20	73	51	32	97	66	2458
LDP	88.9 ± 9.6	73.3 ± 5.7	63.8 ± 6.6	72.0 ± 2.0	68.5 ± 4.0	42.9 ± 3.7	76.1 ± 1.4
WL subtree	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	$\mathbf{86.0 \pm 1.8}$	73.8 ± 3.9	50.9 ± 3.8	78.9 ± 1.9
AWL	87.9 ± 9.8	-	-	-	74.5 ± 5.9	51.5 ± 3.6	73.9 ± 1.9
RetGK	90.3 ± 1.1	75.8 ± 0.6	62.5 ± 1.6	84.5 ± 0.2	71.9 ± 1.0	47.7 ± 0.3	81.0 ± 0.3
GNTK	90.0 ± 8.5	75.6 ± 4.2	67.9 ± 6.9	84.2 ± 1.5	76.9 ± 3.6	52.8 ± 4.6	$\mathbf{83.6 \pm 1.0}$
GCN	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0	74.0 ± 3.4	51.9 ± 3.8	79.0 ± 1.8
PatchySAN	92.6 ± 4.2	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9	71.0 ± 2.2	45.2 ± 2.8	72.6 ± 2.2
GIN	89.4 ± 5.6	76.2 ± 2.8	64.6 ± 7.0	82.7 ± 1.7	75.1 ± 5.1	52.3 ± 2.8	80.2 ± 1.9
GCKN-walk-unsup	92.8 ± 6.1	75.7 ± 4.0	65.9 ± 2.0	80.1 ± 1.8	75.9 ± 3.7	53.4 ± 4.7	81.7 ± 1.4
GCKN-path-unsup	92.8 ± 6.1	76.0 ± 3.4	67.3 ± 5.0	81.4 ± 1.6	75.9 ± 3.7	53.0 ± 3.1	82.3 ± 1.1
GCKN-subtree-unsup	95.0 ± 5.2	$\mathbf{76.4 \pm 3.9}$	$\mathbf{70.8 \pm 4.6}$	83.9 ± 1.6	$\mathbf{77.8 \pm 2.6}$	$\mathbf{53.5 \pm 4.1}$	83.2 ± 1.1
GCKN-3layer-unsup	$\mathbf{97.2 \pm 2.8}$	75.9 ± 3.2	69.4 ± 3.5	83.9 ± 1.2	77.2 ± 3.8	53.4 ± 3.6	83.4 ± 1.5
GCKN-subtree-sup	91.6 ± 6.7	76.2 ± 2.5	68.4 ± 7.4	82.0 ± 1.2	76.5 ± 5.7	53.3 ± 3.9	82.9 ± 1.6

We evaluate GCKN and compare its variants to state-of-the-art methods, including GNNs and graph kernels, on several real-world graph classification datasets, involving either discrete or continuous attributes.

4.5.1. Implementation Details

We follow the same protocols as (Du et al., 2019; Xu et al., 2019), and report the average accuracy and standard deviation over a 10-fold cross validation on each dataset. We use the same data splits as Xu et al. (2019), using their code. Note that performing nested 10-fold cross validation would have provided better estimates of test accuracy for all

models, but it would have unfortunately required 10 times more computation, which we could not afford for many of the baselines we considered.

Considered models. We consider two single-layer models called GCKN-walk and GCKN-path, corresponding to the continuous relaxation of the walk and path kernels respectively. We also consider the two-layer model GCKN-subtree introduced in Section 4.3.3 with path length $k_2 = 0$, which accounts for subtrees. Finally, we consider a 3-layer model GCKN-3layers with path length $k_2 = 2$ (which enumerates paths with three vertices for the second layer), and $k_3 = 0$, which introduces a non-linear mapping before global pooling, as in GCKN-subtree. We use the same parameters α_j and q_j (number of filters) across layers. Our comparisons include state-of-the-art graph kernels such as WL kernel (Shervashidze et al., 2011), AWL (Ivanov and Burnaev, 2018), RetGK (Zhang et al., 2018b), GNTK (Du et al., 2019), WWL (Togninalli et al., 2019) and recent GNNs including GCN (Kipf and Welling, 2017), PatchySAN (Niepert et al., 2016) and GIN (Xu et al., 2019). We also include a simple baseline method LDP (Cai and Wang, 2018) based on node degree information and a Gaussian SVM.

Learning unsupervised models. Following Mairal (2016), we learn the anchor points Z_j for each layer by K-means over 300000 extracted paths from each training fold. The resulting graph representations are then mean-centered, standardized, and used within a linear SVM classifier (4.11) with squared hinge loss. In practice, we use the SVM implementation of the Cyanure toolbox (Mairal, 2019).[1] For each 10-fold cross validation, we tune the bandwidth of the Gaussian kernel (identical for all layers), pooling operation (local (4.13) or global (4.14)), path size k_1 at the first layer, number of filters (identical for all layers) and regularization parameter λ in (4.11). More details are provided in Appendix 4.B, as well as a study of the model robustness to hyperparameters.

Learning supervised models. Following Xu et al. (2019), we use an Adam optimizer (Kingma and Ba, 2015) with the initial learning rate equal to 0.01 and halved every 50 epochs, and fix the batch size to 32. We use the unsupervised model based described above for initialization. We select the best model based on the same hyperparameters as for unsupervised models, with the number of epochs as an additional hyperparameter as used in Xu et al. (2019). Note that we do not use DropOut or batch normalization, which are typically used in GNNs such as Xu et al. (2019). Importantly, the number of filters needed for supervised models is always much smaller (*e.g.*, 32 vs 512) than that for unsupervised models to achieve comparable performance.

4.5.2. Results

Graphs with categorical node labels We use the same benchmark datasets as in Du et al. (2019), including 4 biochemical datasets MUTAG, PROTEINS, PTC and NCI1 and 3 social network datasets IMDB-B, IMDB-MULTI and COLLAB. All the biochemical datasets have categorical node labels while none of the social network datasets has node features. We use degrees as node labels for these datasets, following the protocols of previous works (Du et al., 2019; Xu et al., 2019; Togninalli et al., 2019). Similarly, we

[1] http://julien.mairal.org/cyanure/

also transform all the categorical node labels to one-hot representations. The results are reported in Table 4.1. With a few exceptions, GCKN-walk has a small edge on graph kernels and GNNs—both implicitly relying on walks too—probably because of the soft structure comparison allowed by the Gaussian kernel. GCKN-path often brings some further improvement, which can be explained by its increasing the expressivity. Both multilayer GCKNs bring a stronger increase, whereas supervising the filter learning of GCKN-subtree does not help. Yet, the number of filters selected by GCKN-subtree-sup is smaller than GCKN-subtree-unsup (see Appendix 4.B), allowing for faster classification at test time. GCKN-3layers-unsup performs in the same ballpark as GCKN-subtree-unsup, but benefits from lower complexity due to smaller path length k_1.

Graphs with continuous node attributes We use 4 real-world graph classification datasets with continuous node attributes: ENZYMES, PROTEINS_full, BZR, COX2. All datasets and size information about the graphs can be found in Kersting et al. (2016). The node attributes are preprocessed with standardization as in Togninalli et al. (2019). To make a fair comparison, we follow the same protocol as used in Togninalli et al. (2019). Specifically, we perform 10 different 10-fold cross validations, using the same hyperparameters that give the best average validation accuracy. The hyperparameter search grids remain the same as for training graphs with categorical node labels. The results are shown in Table 4.2. They are comparable to the ones obtained with categorical attributes, except that in 2/4 datasets, the multilayer versions of GCKN underperform compared to GCKN-path, but they achieve lower computational complexity. Paths were indeed presumably predictive enough for these datasets. Besides, the supervised version of GCKN-subtree outperforms its unsupervised counterpart in 2/4 datasets.

4.5.3. Model Interpretation

We train a supervised GCKN-subtree model on the Mutagenicity dataset (Kersting et al., 2016), and use our method described in Section 4.4 to identify important subgraphs. Figure 4.1 shows examples of detected subgraphs. Our method is able to identify chemical groups known for their mutagenicity such as Polycyclic aromatic hydrocarbon (top row left), Diphenyl ether (top row middle) or NO_2 (top row right), thus admitting simple model interpretation. We also find some groups whose mutagenicity is not known, such as polyphenylene sulfide (bottom row middle) and 2-chloroethyl- (bottom row right). More details and additional results are provided in Appendix 4.B.

Table 4.2.: Classification accuracies on graphs with continuous attributes. The accuracies of other models except GNTK are taken from Togninalli et al. (2019). The accuracies of GNTK are obtained by running the code of Du et al. (2019) on a similar setting.

Dataset	ENZYMES	PROTEINS	BZR	COX2
size	600	1113	405	467
classes	6	2	2	2
attr. dim.	18	29	3	3
avg ♯nodes	32.6	39.0	35.8	41.2
avg ♯edges	62.1	72.8	38.3	43.5
RBF-WL	68.4 ± 1.5	75.4 ± 0.3	81.0 ± 1.7	75.5 ± 1.5
HGK-WL	63.0 ± 0.7	75.9 ± 0.2	78.6 ± 0.6	78.1 ± 0.5
HGK-SP	66.4 ± 0.4	75.8 ± 0.2	76.4 ± 0.7	72.6 ± 1.2
WWL	73.3 ± 0.9	$\mathbf{77.9 \pm 0.8}$	84.4 ± 2.0	78.3 ± 0.5
GNTK	69.6 ± 0.9	75.7 ± 0.2	85.5 ± 0.8	79.6 ± 0.4
GCKN-walk-unsup	73.5 ± 0.5	76.5 ± 0.3	85.3 ± 0.5	80.6 ± 1.2
GCKN-path-unsup	$\mathbf{75.7 \pm 1.1}$	76.3 ± 0.5	85.9 ± 0.5	81.2 ± 0.8
GCKN-subtree-unsup	74.8 ± 0.7	77.5 ± 0.3	85.8 ± 0.9	81.8 ± 0.8
GCKN-3layer-unsup	74.6 ± 0.8	77.5 ± 0.4	84.7 ± 1.0	$\mathbf{82.0 \pm 0.6}$
GCKN-subtree-sup	72.8 ± 1.0	77.6 ± 0.4	$\mathbf{86.4 \pm 0.5}$	81.7 ± 0.7

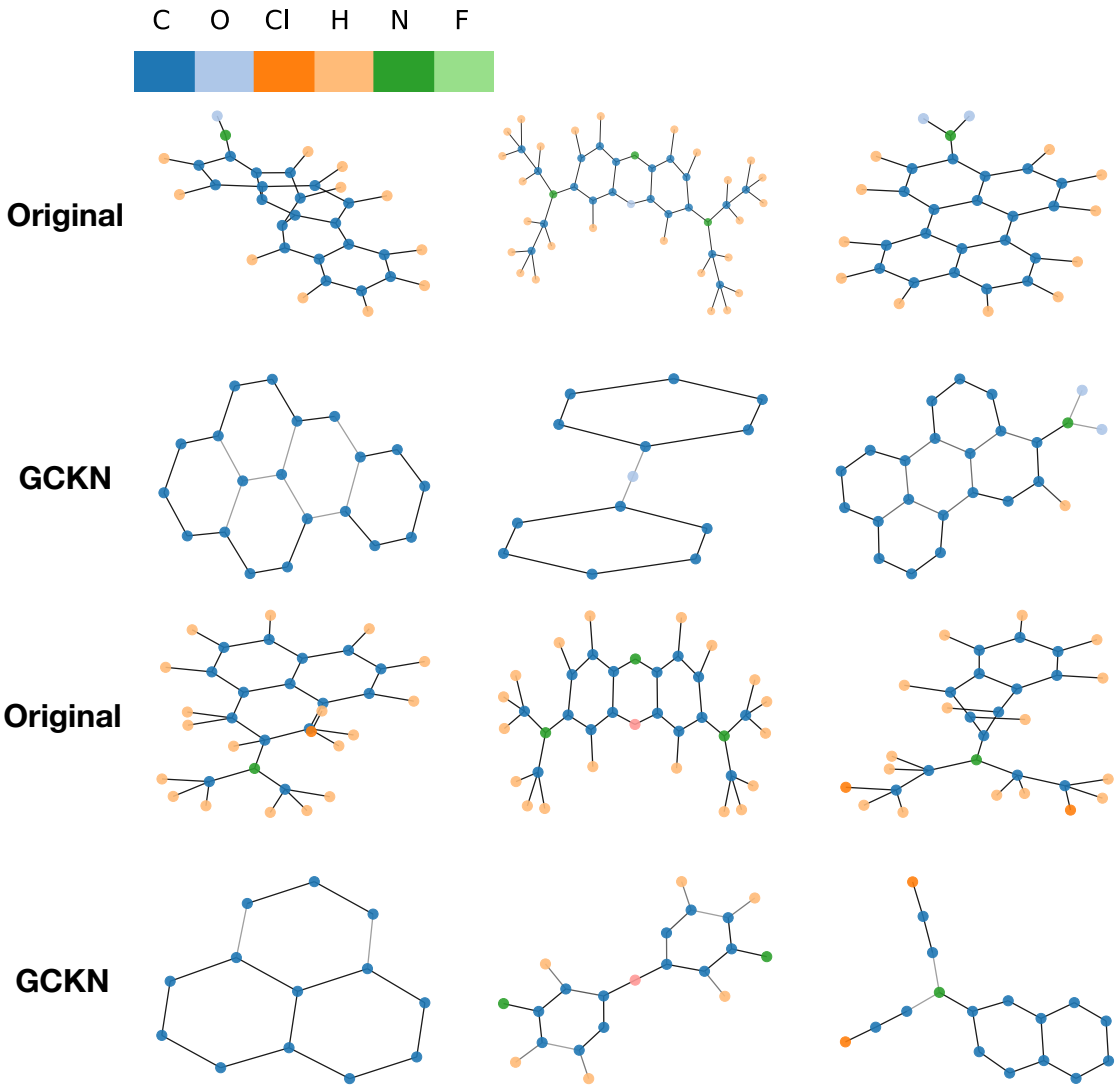


Figure 4.1.: Motifs extracted by GCKN on the Mutagenicity dataset.

Appendix

The appendix provides both theoretical and experimental material and is organized as follows: Appendix 4.A presents a classical result, allowing us to characterize the RKHS of the graph kernels we introduce. Appendix 4.B provides additional experimental details that are useful to reproduce our results and additional experimental results. Then, Appendix 4.C explains how to accelerate the computation of GCKN when using walks instead of paths (at the cost of lower expressiveness), and Appendix 4.D presents a proof of Theorem 4.1 on the expressiveness of WL and walk kernels.

4.A. Useful Result about RKHSs

The following result characterizes the RKHS of a kernel function when an explicit mapping to a Hilbert space is available. It may be found in classical textbooks (see, *e.g.*, Saitoh, 1997, §2.1).

Theorem 4.2. *Let $\Phi : \mathcal{X} \to \mathcal{F}$ be a mapping from a data space $\mathcal{X}$ to a Hilbert space $\mathcal{F}$, and let $K(x, x') := \langle \Phi(x), \psi(x') \rangle_{\mathcal{F}}$ for x, x' in $\mathcal{X}$. Consider the Hilbert space*

$$\mathcal{H} := \{ f_z \ ; \ z \in \mathcal{F} \} \quad s.t. \quad f_z : x \mapsto \langle z, \Phi(x) \rangle_{\mathcal{F}},$$

endowed with the norm

$$\|f\|_{\mathcal{H}}^2 := \inf_{z \in \mathcal{F}} \left\{ \|z\|_{\mathcal{F}}^2 \quad s.t. \quad f = f_z \right\}.$$

Then, $\mathcal{H}$ is the reproducing kernel Hilbert space associated to kernel K.

4.B. Details on Experimental Setup and Additional Experiments

In this section, we provide additional details and more experimental results. In Section 4.B.1, we provide additional experimental details; in Section 4.B.2, we perform a hyperparameter study for unsupervised GCKN on three datasets, showing that our approach is relatively robust to the choice of hyperparameters. In particular, the number of filters controls the quality of Nyström's kernel approximation: more filters means a better approximation and better results, at the cost of more computation. This is in contrast with a traditional (supervised) GNN, where more filters may lead to overfitting. Finally, Section 4.B.3 provides motif discovery results.

4.B.1. Experimental Setup and Reproducibility

Hyperparameter search grids. In our experiments for supervised models, we use an Adam optimizer (Kingma and Ba, 2015) for at most 350 epochs with an initial learning rate equal to 0.01 and halved every 50 epochs with a batch size fixed to 32 throughout all datasets; the number of epochs is selected using cross validation following Xu et al. (2019). The full hyperparameter search range is given in Table 4.B.1 for both unsupervised and supervised models on all tasks. Note that we include some large values (1.5

and 2.0) for σ to simulate the linear kernel as we discussed in Section 4.3.3. In fact, the function $\sigma_1(x) = e^{\alpha(x-1)}$ defined in (4.12) is upper bounded by $e^{-\alpha} + (1 - e^{-\alpha})x$ and lower bounded by $1 + \alpha(x - 1)$ by its convexity at 0 and 1. Their difference is increasing with α and converges to zero when α tends to 0. Hence, when α is small, σ_1 behaves as an affine kernel with a small slope.

Table 4.B.1.: Hyperparameter search range

Hyperparameter	Search range
σ ($\alpha = 1/\sigma^2$)	$[0.3; 0.4; 0.5; 0.6; 1.0; 1.5; 2.0]$
local/global pooling	[sum, mean, max]
path size k_1	integers between 2 and 12
number of filters (unsup)	[32; 128; 512; 1024]
number of filters (sup)	[32; 64] and 256 for ENZYMES
λ (unsup)	$1/n \times$ np.logspace(-3, 4, 60)
λ (sup)	[0.01; 0.001; 0.0001; 1e-05; 1e-06; 1e-07]

Computing infrastructure. Experiments for unsupervised models were conducted by using a shared CPU cluster composed of 2 Intel Xeon E5-2470v2 @2.4GHz CPUs with 16 cores and 192GB of RAM. Supervised models were trained by using a shared GPU cluster, in large parts built with Nvidia gamer cards (Titan X, GTX1080TI). About 20 of these CPUs and 10 of these GPUs were used simultaneously to perform the experiments of this chapter.

4.B.2. Hyperparameter Study

We show here that both unsupervised and supervised models are generally robust to different hyperparameters, including path size k_1, bandwidth parameter σ, regularization parameter λ and their performance grows increasingly with the number of filters q. The accuracies for NCI1, PROTEINS and IMDBMULTI are given in Figure 4.B.1, by varying respectively the number of filters, the path size, the bandwidth parameter and regularization parameter when fixing other parameters which give the best accuracy. Supervised models generally require fewer number of filters to achieve similar performance to its unsupervised counterpart. In particular on the NCI1 dataset, the supervised GCKN outperforms its unsupervised counterpart by a significant margin when using a small number of filters.

4.B.3. Model Interpretation

Implementation details. We use a similar experimental setting as Ying et al. (2019) to train a supervised GCKN-subtree model on Mutagenicity dataset, consisting of 4337 molecule graphs labeled according to their mutagenic effect. Specifically, we use the same split for train and validation set and train a GCKN-subtree model with $k_1 = 3$, which is similar to a 3-layer GNN model. The number of filters is fixed to 20, the same as Ying et al. (2019). The bandwidth parameter σ is fixed to 0.4, local and global pooling are fixed to mean pooling, the regularization parameter λ is fixed to 1e-05. We use an

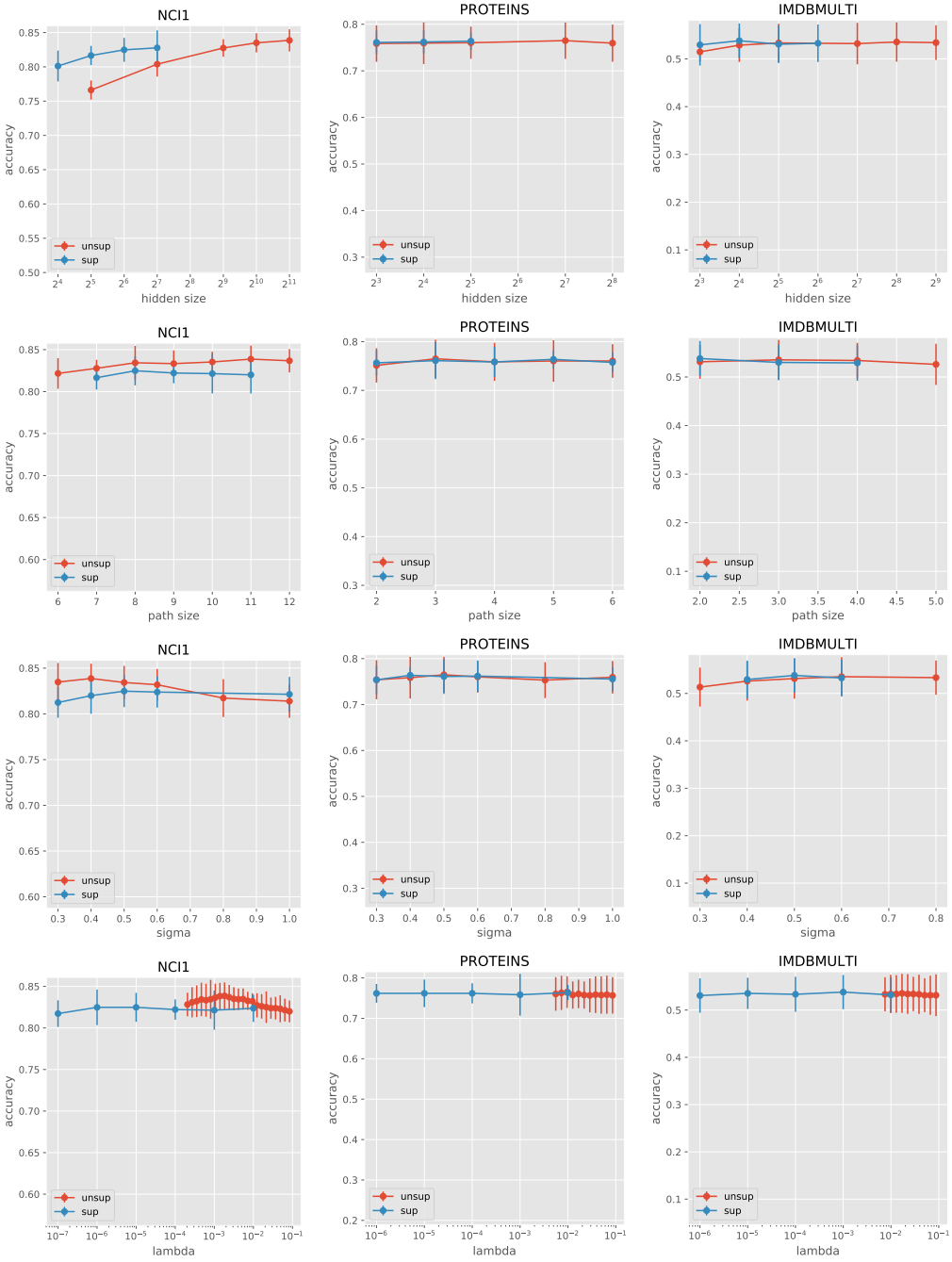


Figure 4.B.1.: Hyperparamter study: sensibility to different hyperparameters for unsupervised and supervised GCKN-subtree models. The row from top to bottom respectively corresponds to number of filters q_1, path size k_1, bandwidth parameter σ and regularization parameter λ. The column from left to right corresponds to different datasets: NC11, PROTEINS and IMDBMULTI.

Adam optimizer with initial learning equal to 0.01 and halved every 50 epochs, the same as previously. The accuracy of the trained model is assured to be more than 80% on the test set as Ying et al. (2019). Then we use the procedure described in Section 4.4 to interpret our trained model. We use an LBFGS optimizer and fixed μ to 0.01. The final subgraph for each given graph is obtained by extracting the maximal connected component formed by the selected paths. A contribution score for each edge can also be obtained by gathering the weights M of all the selected paths that pass through this edge.

More results. More motifs extracted by GCKN are shown in Figure 4.B.2 for the Mutagenicity dataset. We recovered some benzene ring or polycyclic aromatic groups which are known to be mutagenic. We also found some groups whose mutagenicity is not known, such as polyphenylene sulfide in the fourth subgraph and 2-chloroethyl- in the last subgraph.

4.C. Fast Computation of GCKN with Walks

Here we discuss an efficient computational variant using walk kernel instead of path kernel, at the cost of losing some expressive power. Let us consider a relaxed walk kernel by analogy to (4.8) with

$$\kappa_{\text{base}}^{(k)}(u,u') = \sum_{p\in\mathcal{W}_k(G,u)} \sum_{p'\in\mathcal{W}_k(G',u')} \kappa_1(\varphi_0(p),\varphi_0'(p')), \tag{4.19}$$

using walks instead of paths and with κ_1 the Gaussian kernel defined in (4.9). As Gaussian kernel can be decomposed as a product of the Gaussian kernel on pair of nodes at each position

$$\kappa_1(\varphi_0(p),\varphi_0'(p')) = \prod_{j=1}^{k} \kappa_1(\varphi_0(p_j),\varphi_0'(p_j')),$$

We can obtain similar recursive relation as for the original walk kernel in Lemma 2

$$\kappa_{\text{base}}^{(k)}(u,u') = \kappa_1(\varphi_0(u),\varphi_0'(u')) \sum_{v\in\mathcal{N}(u)} \sum_{v'\in\mathcal{N}(u')} \kappa_{\text{base}}^{(k-1)}(v,v'). \tag{4.20}$$

After applying the Nyström method, the approximate feature map in (4.13) becomes

$$\psi_1(u) = \sigma_1(Z^\top Z)^{-\frac{1}{2}} c_k(u),$$

where for any $0 \le j \le k$, $c_j(u) := \sum_{p\in\mathcal{W}_j(G,u)} \sigma_1(Z_j^\top \psi_0(p))$ and Z_j in $\mathbb{R}^{q_0(j+1)\times q_1}$ denotes the matrix consisting of the $j+1$ last columns of q_1 anchor points. Using the above recursive relation (4.20) and similar arguments in *e.g.* Chen et al. (2019b), we can show c_j obeys the following recursive relation

$$c_j(u) = b_j(u) \odot \sum_{v\in\mathcal{N}(u)} c_{j-1}(v), \ 1 \le j \le k, \tag{4.21}$$

where $\odot$ denotes the element-wise product and $b_j(u)$ is a vector in $\mathbb{R}^{q_1}$ whose entry i in $\{1,\dots,q_1\}$ is $\kappa_1(u, z_i^{(k+1-j)})$ and $z_i^{(k+1-j)}$ denotes the $k+1-j$-th column vector of z_i in $\mathbb{R}^{q_0}$. In practice, $\sum_{v\in\mathcal{N}(u)} c_{j-1}(v)$ can be computed efficiently by multiplying the adjacency matrix with the $|\mathcal{V}|$-dimensional vector with entries $c_{j-1}(v)$ for $v \in \mathcal{V}$.

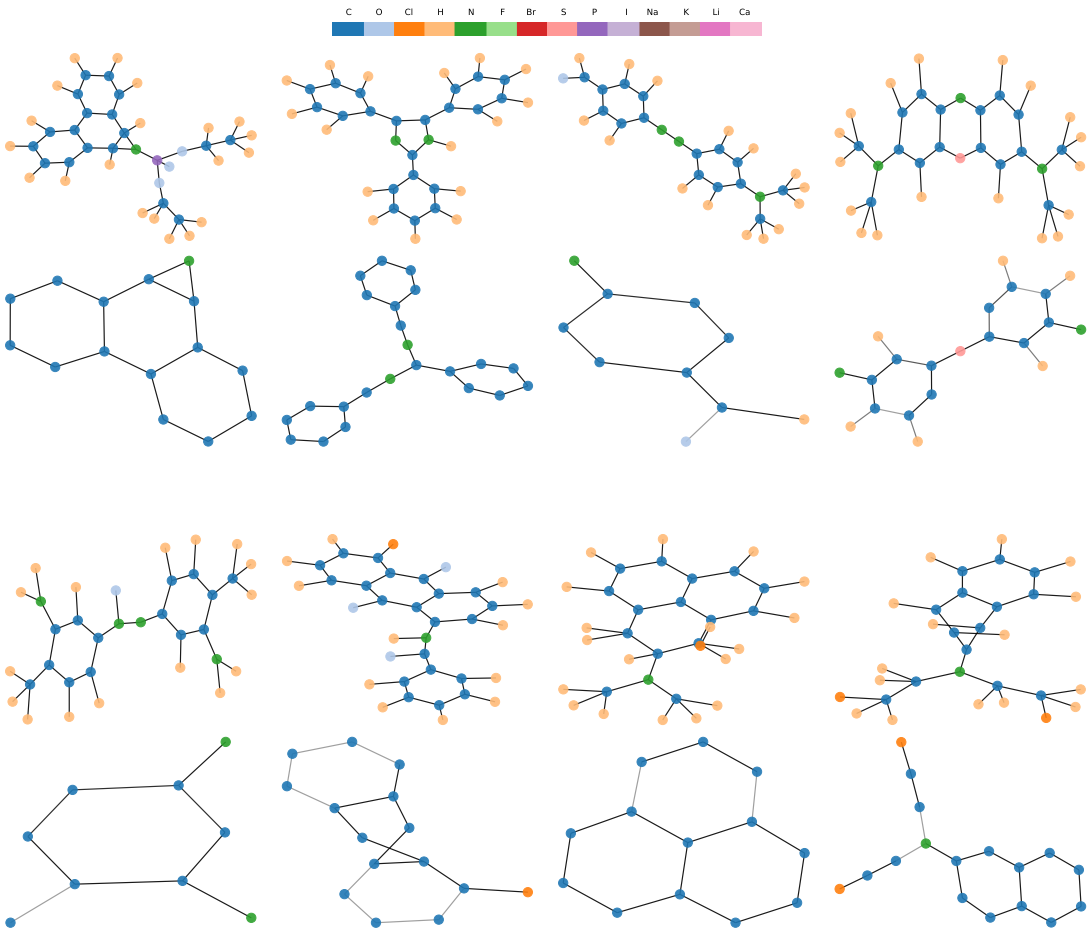


Figure 4.B.2.: More motifs extracted by GCKN on Mutagenicity dataset. First and third rows are original graphs; second and fourth rows are corresponding motifs. Some benzene ring or polycyclic aromatic groups are identified, which are known to be mutagenic. In addition, Some chemical groups whose mutagenicity is not known are also identified, such as polyphenylene sulfide in the fourth subgraph and 2-chloroethyl- in the last subgraph.

4.D. Proof of Theorem 1

Before presenting and proving the link between the WL subtree kernel and the walk kernel, we start by reminding and showing some useful results about the WL subtree kernel and the walk kernel.

4.D.1. Useful Results for the WL Subtree Kernel

We first recall a recursive relation of the WL subtree kernel, given in the Theorem 8 of Shervashidze et al. (2011). Let us denote by $\mathcal{M}(u, u')$ the set of exact matchings of subsets of the neighbors of u and u', formally given by

$$\mathcal{M}(u,u') = \Big\{ R \subseteq \mathcal{N}(u) \times \mathcal{N}(u') \,\Big|\, |R| = |\mathcal{N}(u)| = |\mathcal{N}(u')| \wedge \\ (\forall (v,v'),(w,w') \in R : u = w \Leftrightarrow u' = w') \wedge (\forall (u,u') \in R : a(u) = a'(u')) \Big\}. \quad (4.22)$$

Then we have the following recursive relation for $\kappa^{(k)}_{\text{subtree}}(u,u') := \delta(a_k(u), a'_k(u'))$

$$\kappa^{(k+1)}_{\text{subtree}}(u,u') = \begin{cases} \kappa^{(k)}_{\text{subtree}}(u,u') \max\limits_{R \in \mathcal{M}(u,u')} \prod\limits_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v,v'), & \text{if } \mathcal{M}(u,u') \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases} \quad (4.23)$$

We can further simply the above recursion using the following Lemma

Lemma 1. *If $\mathcal{M}(u,u') \neq \emptyset$, we have*

$$\kappa^{(k+1)}_{subtree}(u,u') = \delta(a(u), a'(u')) \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{subtree}(v,v').$$

Proof. We prove this by induction on $k \geq 0$. For $k = 0$, this is true by the definition of $\kappa^{(0)}_{\text{subtree}}$. For $k \geq 1$, we suppose that

$$\kappa^{(k)}_{\text{subtree}}(u,u') = \delta(a(u), a'(u')) \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k-1)}_{\text{subtree}}(v,v').$$

We have

$$\begin{aligned} \kappa^{(k+1)}_{\text{subtree}}(u,u') &= \kappa^{(k)}_{\text{subtree}}(u,u') \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v,v') \\ &= \delta(a(u), a'(u')) \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k-1)}_{\text{subtree}}(v,v') \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v,v'). \end{aligned}$$

It suffices to show

$$\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k-1)}_{\text{subtree}}(v,v') \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v,v') = \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v,v').$$

Since the only values can take for $\kappa^{(k-1)}_{\text{subtree}}$ is 0 and 1, the only values that

$$\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k-1)}_{\text{subtree}}(v,v')$$

can take is also 0 and 1. Then we can split the proof on these two conditions. It is obvious if this term is equal to 1. If this term is equal to 0, then

$$\max_{R\in\mathcal{M}(u,u')}\prod_{(v,v')\in R}\kappa^{(k)}_{\text{subtree}}(v,v')\leq\max_{R\in\mathcal{M}(u,u')}\prod_{(v,v')\in R}\kappa^{(k-1)}_{\text{subtree}}(v,v')=0,$$

as all terms are not negative and $\kappa^{(k)}_{\text{subtree}}(v,v')$ is not creasing on k. Then

$$\max_{R\in\mathcal{M}(u,u')}\prod_{(v,v')\in R}\kappa^{(k)}_{\text{subtree}}(v,v')=0,$$

and we have 0 for both sides. □

4.D.2. Recursive Relation for the Walk Kernel

We recall that the k-walk kernel is defined as

$$K(G,G')=\sum_{u\in\mathcal{V}}\sum_{u'\in\mathcal{V}'}\kappa^{(k)}_{\text{walk}}(u,u'),$$

where

$$\kappa^{(k)}_{\text{walk}}(u,u')=\sum_{p\in\mathcal{W}_k(G,u)}\sum_{p'\in\mathcal{W}_k(G',u')}\delta(a(p),a'(p')).$$

The feature map of this kernel is given by

$$\varphi^{(k)}_{\text{walk}}(u)=\sum_{p\in\mathcal{W}_k(G,u)}\varphi_\delta(a(p)),$$

where φ_δ is the feature map associated with δ. We give here a recursive relation for the walk kernel on the size of walks, thanks to its allowance of nodes to repeat.

Lemma 2. *For any $k\geq 0$, we have*

$$\kappa^{(k+1)}_{walk}(u,u')=\delta(a(u),a'(u'))\sum_{v\in\mathcal{N}(u)}\sum_{v'\in\mathcal{N}(u')}\kappa^{(k)}_{walk}(v,v'). \tag{4.24}$$

Proof. Noticing that we can always decompose a path $p\in\mathcal{W}_{k+1}(G,u)$, with (u,v) the first edge that it passes and $v\in\mathcal{N}(u)$, into (u,q) with $q\in\mathcal{W}_k(G,v)$, then we have

$$\begin{aligned}
\kappa^{(k+1)}_{\text{walk}}(u,u')&=\sum_{p\in\mathcal{W}_{k+1}(G,u)}\sum_{p'\in\mathcal{W}_{k+1}(G',u')}\delta(a(p),a'(p'))\\
&=\sum_{v\in\mathcal{N}(u)}\sum_{p\in\mathcal{W}_k(G,v)}\sum_{v'\in\mathcal{N}(u')}\sum_{p'\in\mathcal{W}_k(G,v')}\delta(a(u),a'(u'))\delta(a(p),a'(p'))\\
&=\delta(a(u),a'(u'))\sum_{v\in\mathcal{N}(u)}\sum_{v'\in\mathcal{N}(u')}\sum_{p\in\mathcal{W}_k(G,v)}\sum_{p'\in\mathcal{W}_k(G',v')}\delta(a(p),a'(p'))\\
&=\delta(a(u),a'(u'))\sum_{v\in\mathcal{N}(u)}\sum_{v'\in\mathcal{N}(u')}\kappa^{(k)}_{\text{walk}}(v,v').
\end{aligned}$$

□

This relation also provides us a recursive relation for the feature maps of the walk kernel

$$\varphi^{(k+1)}_{\text{walk}}(u)=\varphi_\delta(a(u))\otimes\sum_{v\in\mathcal{N}(u)}\varphi^{(k)}_{\text{walk}}(v),$$

where $\otimes$ denotes the tensor product.

4.D.3. Discriminative Power between Walk Kernel and WL Subtree Kernel

Before proving the Theorem 4.1, let us first show that the WL subtree kernel is always more discriminative than the walk kernel.

Proposition 4.1. *For any node u in graph G and u' in graph G' and any $k \geq 0$, then* $d_{\kappa^{(k)}_{subtree}}(u, u') = 0 \implies d_{\kappa^{(k)}_{walk}}(u, u') = 0$.

This proposition suggests that though both of their feature maps are not injective (see *e.g.* Kriege et al. (2018)), the feature map of $\kappa^{(k)}_{\text{subtree}}$ is more injective in the sense that for a node u, its collision set $\{u' \in \mathcal{V} \,|\, \varphi(u') = \varphi(u)\}$ for $\kappa^{(k)}_{\text{subtree}}$, with φ the corresponding feature map, is included in that for $\kappa^{(k)}_{\text{walk}}$. Furthermore, if we denote by $\hat{\kappa}$ the normalized kernel of κ such that $\hat{\kappa}(u, u') = \kappa(u, u')/\sqrt{\kappa(u, u)\kappa(u', u')}$, then we have

Corollary 1. *For any node u in graph G and u' in graph G' and any $k \geq 0$,* $d_{\kappa^{(k)}_{subtree}}(u, u') \geq d_{\hat{\kappa}^{(k)}_{walk}}(u, u')$.

Proof. We prove by induction on k. It is clear for $k = 0$ as both kernels are equal to the Dirac kernel on the node attributes. Let us suppose this is true for $k \geq 0$, we will show this is also true for $k + 1$. We suppose $d_{\kappa^{(k+1)}_{\text{subtree}}}(u, u') = 0$. Since $\kappa^{(k+1)}_{\text{subtree}}(u, u) = 1$, by equality (4.23) we have

$$1 = \kappa^{(k+1)}_{\text{subtree}}(u, u') = \kappa^{(k)}_{\text{subtree}}(u, u') \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v, v'),$$

which implies that $\kappa^{(k)}_{\text{subtree}}(u, u') = 1$ and $\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v, v') = 1$. Then $\delta(a(u), a'(u')) = 1$ by the non-growth of $\kappa^{(k)}_{\text{subtree}}(u, u')$ on k and it exists an exact matching $R^\star \in \mathcal{M}(u, u')$ such that $|\mathcal{N}(u)| = |\mathcal{N}(u')| = |R^\star|$ and $\forall (v, v') \in R^\star$, $\kappa^{(k)}_{\text{subtree}}(v, v') = 1$. Therefore, we have $d_{\kappa^{(k)}_{\text{walk}}}(v, v') = 0$ for all $(v, v') \in R^\star$ by the induction hypothesis.

On the other hand, by Lemma 2 we have

$$\begin{aligned}\kappa^{(k+1)}_{\text{walk}}(u, u') &= \delta(a(u), a'(u')) \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \kappa^{(k)}_{\text{walk}}(v, v') \\ &= \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \kappa^{(k)}_{\text{walk}}(v, v'),\end{aligned}$$

which suggest that the feature map of $\kappa^{(k+1)}_{\text{walk}}$ can be written as $\varphi^{(k+1)}_{\text{walk}}(u) = \sum_{v \in \mathcal{N}(u)} \varphi^{(k)}_{\text{walk}}(v)$. Then we have

$$\begin{aligned}d_{\kappa^{(k+1)}_{\text{walk}}}(u, u') &= \left\| \sum_{v \in \mathcal{N}(u)} \varphi^{(k)}_{\text{walk}}(v) - \sum_{v' \in \mathcal{N}(u')} \varphi^{(k)}_{\text{walk}}(v') \right\| \\ &= \left\| \sum_{(v,v') \in R^\star} \varphi^{(k)}_{\text{walk}}(v) - \varphi^{(k)}_{\text{walk}}(v') \right\| \\ &\leq \sum_{(v,v') \in R^\star} \|\varphi^{(k)}_{\text{walk}}(v) - \varphi^{(k)}_{\text{walk}}(v')\| \\ &= \sum_{(v,v') \in R^\star} d_{\kappa^{(k)}_{\text{walk}}}(v, v') = 0.\end{aligned}$$

We conclude that $d_{\kappa_{\text{walk}}^{(k+1)}}(u, u') = 0$.

Now let us prove the Corollary 1. The only values that $d_{\kappa_{\text{subtree}}^{(k)}}(u, u')$ can take are 0 and 1. Since $d_{\hat{\kappa}_{\text{walk}}^{(k)}}(u, u')$ is always not larger than 1, we only need to prove $d_{\kappa_{\text{subtree}}^{(k)}}(u, u') = 0 \implies d_{\hat{\kappa}_{\text{walk}}^{(k)}}(u, u') = 0$, which has been shown above. □

4.D.4. Proof of Theorem 4.1

Note that using our notation here, $\varphi_1 = \varphi_{\text{walk}}^{(k)}$

Proof. We prove by induction on k. For $k = 0$, we have for any $u \in \mathcal{V}$ and $u' \in \mathcal{V}'$

$$\kappa_{\text{subtree}}^{(0)}(u, u') = \delta(a(u), a'(u')) = \delta(\varphi_{\text{walk}}^{(0)}(u), \varphi_{\text{walk}}^{(0)}(u')).$$

Assume that (4.16) is true for $k \geq 0$. We want to show this is also true for $k+1$. As the only values that the δ kernel can take is 0 and 1, it suffices to show the equality between $\kappa_{\text{subtree}}^{(k+1)}(u, u')$ and $\delta(\varphi_{\text{walk}}^{(k+1)}(u), \varphi_{\text{walk}}^{(k+1)}(u'))$ in these two situations.

- If $\kappa_{\text{subtree}}^{(k+1)}(u, u') = 1$, by Proposition 4.1 we have $\varphi_{\text{walk}}^{(k+1)}(u) = \varphi_{\text{walk}}^{(k+1)}(u')$, and thus $\delta(\varphi_{\text{walk}}^{(k+1)}(u), \varphi_{\text{walk}}^{(k+1)}(u')) = 1$.
- If $\kappa_{\text{subtree}}^{(k+1)}(u, u') = 0$, by the recursive relation of the feature maps in Lemma 2, we have

  $$\delta(\varphi_{\text{walk}}^{(k+1)}(u), \varphi_{\text{walk}}^{(k+1)}(u')) = \delta(a(u), a'(u'))\delta\left(\sum_{v \in \mathcal{N}(u)} \varphi_{\text{walk}}^{(k)}(v), \sum_{v' \in \mathcal{N}(u')} \varphi_{\text{walk}}^{(k)}(v')\right).$$

  By Lemma 1, it suffices to show that

  $$\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k)}(u, u') = 0 \implies \delta\left(\sum_{v \in \mathcal{N}(u)} \varphi_{\text{walk}}^{(k)}(v), \sum_{v' \in \mathcal{N}(u')} \varphi_{\text{walk}}^{(k)}(v')\right) = 0.$$

  The condition $|\mathcal{M}(u, u')| = 1$ suggests that there exists exactly one matching of the neighbors of u and u'. Let us denote this matching by R. The left equality implies that there exists a non-empty subset of neighbor pairs $S \subseteq R$ such that $\kappa_{\text{subtree}}^{(k)}(v, v') = 0$ for any $(v, v') \in S$ and $\kappa_{\text{subtree}}^{(k)}(v, v') = 1$ for all $(v, v') \notin S$. Then by the induction hypothesis, $\varphi_{\text{walk}}^{(k)}(v) = \varphi_{\text{walk}}^{(k)}(v')$ for all $(v, v') \notin S$ and $\varphi_{\text{walk}}^{(k)}(v) \neq \varphi_{\text{walk}}^{(k)}(v')$ for all $(v, v') \in S$. Consequently, $\sum_{(v,v') \notin S} \varphi_{\text{walk}}^{(k)}(v) - \varphi_{\text{walk}}^{(k)}(v') = 0$. Now we will show $\sum_{(v,v') \in S} \varphi_{\text{walk}}^{(k)}(v) - \varphi_{\text{walk}}^{(k)}(v') \neq 0$ since all neighbors of either u or u' have distinct attributes. Then

  $$\begin{aligned} &\| \sum_{v \in \mathcal{N}(u)} \varphi_{\text{walk}}^{(k)}(v) - \sum_{v' \in \mathcal{N}(u')} \varphi_{\text{walk}}^{(k)}(v') \| \\ =&\| \sum_{(v,v') \in R} \varphi_{\text{walk}}^{(k)}(v) - \varphi_{\text{walk}}^{(k)}(v') \| \\ =&\| \sum_{(v,v') \in S} \varphi_{\text{walk}}^{(k)}(v) - \varphi_{\text{walk}}^{(i)}(v') \| > 0. \end{aligned}$$

  Therefore, $\delta\left(\sum_{v \in \mathcal{N}(u)} \varphi_{\text{walk}}^{(k)}(v), \sum_{v' \in \mathcal{N}(u')} \varphi_{\text{walk}}^{(k)}(v')\right) = 0$.

□

5

Kernel Embeddings for Feature Aggregation

Contents

We introduce two valid kernel embeddings for sets of features based on either max pooling operations or optimal transport distance in RKHS. Our approach addresses the problem of feature aggregation, or pooling, for sets that exhibit long-range dependencies between their members. More precisely, our first embedding enables valid max pooling in RKHS, which was barely studied in previous kernel literature. Our second proposal aggregates the features of a given set according to the transport plan between the set and a reference shared across the data set. Unlike traditional hand-crafted kernels, our embedding can be optimized for a specific task or data set. Our embedding is particularly suited for biological sequence classification tasks and shows promising results for natural language sequences. We provide an implementation of our embedding that can be used alone or as a module in larger learning models. Our code is freely available at https://github.com/claying/OTK.

The chapter is based on the following publications:

D. Chen, L. Jacob, and J. Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b
G. Mialon*, D. Chen*, A. d'Aspremont, and J. Mairal. A trainable optimal transport embedding for feature aggregation. In *International Conference on Learning Representations (ICLR)*
(*equal contributions)

The second work presented in this chapter was achieved with the collaboration of Grégoire Mialon, Alexandre d'Aspremont and Julien Mairal, with equal contribution between Grégoire Mialon and me.

5.1. Introduction

In many applications of machine learning, data objects have complex structures that can be expressed as sets of features with positional information, such as sentences in natural language processing (NLP), images in computer vision or biological sequences in bioinformatics. These objects of varying sizes generally exhibit long-range and complex structural dependencies, which are hard to properly take into account. Kernel methods are widely used to deal with such data. They allow to embed these structured data objects to high or infinite-dimensional vectors simply through a pairwise similarity function, nameed kernel. In general, the construction of most of these kernels can be decoupled into two steps: local feature extraction and feature aggregation. While the former step deploys a local kernel to map all the elements of the set to some reproducing kernel Hilbert space (RKHS), the latter performs aggregation of these local features to summarize local information and potentially to capture dependencies between them. Regarding the feature aggregation, a family of kernels classically used in biology and computer vision that rely on the comparisons of histograms of features (Barla et al., 2003; Leslie et al., 2002b), can be interpreted as performing indifferent comparisons of local features without matching them. The feature map of such kernels amounts to performing a summation of the embeddings of local features. Such a simple aggregation approach is easy to implement but can be restrictive for capturing complex dependencies, which has thus prompted more flexible and sophisticated kernels, through, *e.g.*, the creation of multi-resolution histograms (Grauman and Darrell, 2007), or by computing correspondences with more complex metrics. Such more complex metrics generally explore an optimal matching between components of sets before aggregation, through the earth-mover's distance (also known as the 1-Wasserstein distance) (Rubner et al., 2000), alignment scores (Vert et al., 2004) or more general optimal transport based metrics (Wallraven et al., 2003; Fröhlich et al., 2005) that were introduced in different contexts. However, the associated kernels are generally non positive definite and cannot scale to large datasets. Hence, some variants (Boughorbel et al., 2005; Johansson and Dubhashi, 2015) have been proposed to handle the former issue, but still suffer from the scalability issue. In this respect, methods that directly work with and optimize the representation to the task at hand are now more preferred.

Among them, the concept of attention (Bahdanau et al., 2015) was proposed to cope with sentences in the task of machine translation. This mechanism allows the model to automatically focus on components of a source sentence that are relevant for predicting the next word. Recently, a neural network architecture mostly relying on attention mechanisms, named transformer (Vaswani et al., 2017), has brought striking improvement in machine translation task. Lately, its variants also led to pronounced progress in many other NLP tasks (Wang et al., 2019) and to some extent in other fields such as computer vision (Ramachandran et al., 2019) and bioinformatics (Rives et al., 2019). However, the prediction power of attention-based models in classification of long biological sequences has barely been studied and compared to typical convolution based models, due to their possibly prohibitive number of parameters (Baid, 2018; Raffel et al., 2019). On the other hand, some peculiarities of the transformer architecture, such as the learned dot-product self-attention or the role of the attention heads are now being questioned and investigated (Raganato et al., 2020; Voita et al., 2019; Weiqiu et al., 2020).

In this chapter, we consider proposing valid kernel operations for feature aggregation in RKHS. We fist introduce a valid kernel embedding that simulates the max pooling operation commonly used in convolutional networks. It allows to perform valid pooling in RKHS while retaining the good accuracy of max pooling. We then introduce a new parametrized kernel embedding for feature aggregation based on a by-product of optimal transport (OT). The parameters can be learned in an unsupervised manner, by approximating the Wasserstein distance matrix. It can also be optimized for a given task like the transformer, thus gaining the ability to be task-adaptive while learning more compact representations. OT has recently drawn growing attention in machine learning as it enjoys efficient algorithms (Cuturi, 2013) that are compatible with GPU computation and back-propagation. It has been applied in various contexts, including computer vision, NLP (Kusner et al., 2015), time series (Bock et al., 2019) and graphs (Togninalli et al., 2019). More recently, using the transport plan as an attention score was proposed for network embeddings to align some data modalities (Chen et al., 2019d). Our work goes beyond this idea and uses transport plans as a principle for feature aggregation, or pooling. We demonstrate the effectiveness of our kernel with images, sentences and the transformer architecture. Finally, we provide a simple implementation of our embedding that can be used alone or as a module in large learning models.

Summary of contributions.

- Based on Murray and Perronnin (2014), we propose a new way to simulate max pooling in RKHSs, thus leading to a new kernel embedding for feature aggregation as well as solving a classical discrepancy between theory and practice in the literature of string kernels, where sums are often replaced by a maximum operator that does not ensure positive definiteness (Vert et al., 2004).

- We propose a new parametrized kernel embedding for feature aggregation based on optimal transport. Its parameters can be either learned without supervision through an approximation of Wasserstein distance matrix, or in a task-driven fashion. We demonstrate its scalability and effectiveness on images, biological sequences and sentences using our proposed unsupervised and supervised feature learning algorithms.

- We provide an implementation of our methods that can be used alone or as a module in large learning models.

5.2. Background on Feature Aggregation in RKHS

In this section, we revisit classical kernels for feature aggregation and some basic elements in optimal transport, which will be useful for constructing our kernel embeddings.

5.2.1. Kernel Methods and Summation Kernel

Kernel methods map data living in a space $\mathcal{X}$ to a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, associated with a positive definite (p.d.) kernel K through a mapping

function $\varphi : \mathcal{X} \to \mathcal{H}$, such that $K(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{H}}$. In this chapter, we handle sets of features living in $\mathbb{R}^d$ and we define

$$\mathcal{X} = \left\{ \mathbf{x} \,|\, \mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \text{ such that } \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d \text{ for some } n \geq 1 \right\}.$$

Members of $\mathcal{X}$ are typically vectorial representations of local data structures, such as patches for natural images, or sentences for text. The length of $\mathbf{x}$ denoted by n may vary, which is not a problem since the methods we introduce may take a sequence of any size as input, while providing a fixed-size embedding.

The most natural kernel for aggregation that arose in string kernels (Leslie et al., 2002b, 2004) and histogram comparison based kernels (Lyu, 2004) should be summation kernel, written as

$$K_{\text{match}}(\mathbf{x}, \mathbf{x}') := \frac{1}{n}\frac{1}{n'} \sum_{i=1}^{n} \sum_{j=1}^{n'} \kappa(\mathbf{x}_i, \mathbf{x}'_j) = \left\langle \frac{1}{n} \sum_{i=1}^{n} \varphi(\mathbf{x}_i), \frac{1}{n'} \sum_{j=1}^{n'} \varphi(\mathbf{x}'_j) \right\rangle_{\mathcal{H}}, \tag{5.1}$$

where κ is a p.d. kernel and φ denotes its associated mapping to an RKHS $\mathcal{H}$. A match kernel simply compares all possible pairs of features of $\mathbf{x}$ and $\mathbf{x}'$. The right-hand term exhibits the feature map of K_{match}, which corresponds to a mean pooling in the RKHS. In this process, important information may be averaged (*e.g.*, in biology, rare and relevant patterns may be drowned in useless ones), or artificially strong matches can be made (Jégou et al., 2009). These issues can be addressed by using a max pooling, which may however break the kernel interpretation, or more generally weighting each comparison $\kappa(\mathbf{x}_i, \mathbf{x}'_j)$. The weights are typically independent from the data and may include domain knowledge (Mairal, 2016). In contrast, we will introduce adaptive weights reflecting whether a pair $(\mathbf{x}_i, \mathbf{x}'_j)$ is aligned before comparison or, put differently, whether comparing $\mathbf{x}_i$ and $\mathbf{x}'_j$ is relevant for the task.

Fast computation via finite-dimensional approximation. In some cases, $\varphi(\mathbf{x}_i)$ are already finite-dimensional, which allows to compute the kernel embedding of K_{match} explicitly. This is particularly useful when dealing with large-scale data, as it allows us to use our method for supervised learning tasks without computing the Gram matrix, which grows quadratically with the number of samples. When φ is infinite- or high-dimensional, it is nevertheless possible to use an approximation based on the Nyström method (Williams and Seeger, 2001), which provides an embedding $\psi : \mathbb{R}^d \to \mathbb{R}^k$ such that

$$\langle \psi(\mathbf{x}_i), \psi(\mathbf{x}'_j) \rangle_{\mathbb{R}^k} \approx \kappa(\mathbf{x}_i, \mathbf{x}'_j).$$

Concretely, the Nyström method consists in projecting points from the RKHS $\mathcal{H}$ onto a linear subspace $\mathcal{F}$, which is parametrized by k anchor points $\mathcal{F} = \text{Span}(\varphi(\mathbf{w}_1), \ldots, \varphi(\mathbf{w}_k))$. The corresponding embedding admits an explicit form $\psi(\mathbf{x}_i) = \kappa(\mathbf{w}, \mathbf{w})^{-1/2} \kappa(\mathbf{w}, \mathbf{x}_i)$, where $\kappa(\mathbf{w}, \mathbf{w})$ is the $k \times k$ Gram matrix of κ computed on the set $\mathbf{w} = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ of anchor points and $\kappa(\mathbf{w}, \mathbf{x}_i)$ is in $\mathbb{R}^k$. Then, there are several ways of learning the anchor points: (a) they can be chosen as random points from data; (b) they can be defined as centroids obtained by K-means (Zhang et al., 2008), (c) they can be learned by back-propagation for a supervised task, see Mairal (2016). Once φ is replaced with

ψ, the approximation of the feature map of K_{match} can also be subsequently obtained by

$$K_{\text{match}}(\mathbf{x}, \mathbf{x}') \approx \left\langle \frac{1}{n} \sum_{i=1}^{n} \psi(\mathbf{x}_i), \frac{1}{n'} \sum_{j=1}^{n'} \psi(\mathbf{x}'_j) \right\rangle_{\mathbb{R}^k}, \tag{5.2}$$

5.2.2. Regularized Optimal Transport

Optimal transport provides a natural way to interpretably define the weights between between two sets $\mathbf{x}$ and $\mathbf{x}'$ seen as weighted point clouds or discrete measures, which is a by-product of the optimal transport problem. OT has indeed been widely used in alignment problems. Throughout the chapter, we will refer to the Kantorovich relaxation of OT with entropic regularization, detailed for example in Peyré et al. (2019). Let $\mathbf{a}$ in Δ^n (probability simplex) and $\mathbf{b}$ in $\Delta^{n'}$ be the weights of the discrete measures $\sum_i \mathbf{a}_i \delta_{\mathbf{x}_i}$ and $\sum_j \mathbf{b}_j \delta_{\mathbf{x}'_j}$ with respective locations $\mathbf{x}$ and $\mathbf{x}'$, where $\delta_{\mathbf{x}}$ is the Dirac at position $\mathbf{x}$. Let $\mathbf{C}$ in $\mathbb{R}^{n \times n'}$ be a matrix representing the pairwise costs for aligning the elements of $\mathbf{x}$ and $\mathbf{x}'$. The entropic regularized Kantorovich relaxation of OT from $\mathbf{x}$ to $\mathbf{x}'$ is

$$\min_{\mathbf{P} \in U(\mathbf{a}, \mathbf{b})} \sum_{ij} \mathbf{C}_{ij} \mathbf{P}_{ij} - \varepsilon H(\mathbf{P}), \tag{5.3}$$

where $H(\mathbf{P}) = -\sum_{ij} \mathbf{P}_{ij}(\log(\mathbf{P}_{ij}) - 1)$ is the entropic regularization with parameter ε that controls the non-sparsity of $\mathbf{P}$ as the unregularized OT problem admits sparse solutions, and U is the space of admissible couplings between $\mathbf{a}$ and $\mathbf{b}$:

$$U(\mathbf{a}, \mathbf{b}) = \{\mathbf{P} \in \mathbb{R}_+^{n \times n'} : \mathbf{P}\mathbf{1}_n = \mathbf{a} \text{ and } \mathbf{P}^\top \mathbf{1}_{n'} = \mathbf{b}\}.$$

In practice, $\mathbf{a}$ and $\mathbf{b}$ are uniform measures since we consider the mass to be evenly distributed between the points. $\mathbf{P}$ is called the transport plan, which carries the information on how to distribute the mass of $\mathbf{x}$ in $\mathbf{x}'$ with minimal cost. The objective is ε-strongly convex, such that (5.3) has a unique solution. It is typically solved using a matrix scaling procedure known as the Sinkhorn's algorithm (see, *e.g*, Peyré et al. (2019)).

5.3. Max Pooling in RKHS

Alignment scores (e.g. Smith-Waterman) in molecular biology rely on a max operation—over the scores of all possible alignments—to compute similarities between sequences. However, using maximum instead of summation generally breaks positive definiteness, even though it seems to perform well in practice. To solve such an issue, sum-exponential is used as a proxy in Saigo et al. (2004), but it leads to diagonal dominance issue and makes SVM solvers unable to learn. For the match kernel, the sum in (5.1) can also be replaced by a max

$$\mathbf{K}_k^{\max}(\mathbf{x}, \mathbf{x}') = \left\langle \max_{i=1,\dots,n} \psi(\mathbf{x}_i), \max_{j=1,\dots,n'} \psi(\mathbf{x}'_j) \right\rangle, \tag{5.4}$$

which empirically seems to perform well, but breaks the kernel interpretation, as in Saigo et al. (2004).

An alternative way to aggregate local features is the generalized max pooling (GMP) introduced in Murray and Perronnin (2014) for finite-dimensional vectors, which can be adapted to the context of RKHSs. Assuming that before pooling $\mathbf{x}$ is embedded to a set of n local features $(\varphi_1, \dots, \varphi_n) \in \mathcal{H}^n$, GMP searches for a representation φ^{gmp} whose inner-product with all the local features φ_i is equal to one: $\langle \varphi_i, \varphi^{\text{gmp}} \rangle_{\mathcal{H}} = 1$, for $i = 1, \dots, N$. In particular, φ^{gmp} coincides with the regular max when each φ is an element of the canonical basis of a finite representation, *i.e.*, assuming that at each position, a single feature has value 1 and all others are 0.

Now assuming that each φ_i is represented by a vector ψ_i in $\mathbb{R}^q$, for instance through the Nyström method in Section 5.2.1, the above problem can be approximately solved by search an embedding vector ψ^{gmp} in $\mathbb{R}^q$ such that $\langle \psi_i, \psi^{\text{gmp}} \rangle = 1$ for $i = 1, \dots, n$. In practice, and to prevent ill-conditioned problems, as shown in Murray and Perronnin (2014), it is possible to solve a Ridge regression problem:

$$\psi^{\text{gmp}} := \arg\min_{\psi \in \mathbb{R}^q} \|\Psi^\top \psi - \mathbf{1}\|^2 + \gamma \|\psi\|^2,$$

where $\Psi = [\psi_1, \dots, \psi_n] \in \mathbb{R}^{q \times n}$ and $\mathbf{1}$ denotes the n-dimensional vectors with only 1 as entries. The solution is simply given by $\psi^{\text{gmp}} = (\Psi\Psi^\top + \gamma I)^{-1}\Psi\mathbf{1}$. It requires inverting a $q \times q$ matrix which is usually tractable when the number of anchor points is small. In particular, when $\psi_i = \kappa(\mathbf{w}, \mathbf{w})^{-1/2}\kappa(\mathbf{w}, \mathbf{x}_i)$ the Nyström approximation of a local feature map, we have $\Psi = \kappa(\mathbf{w}, \mathbf{w})^{-1/2}\kappa(\mathbf{w}, \mathbf{x})$ where the matrix $\kappa(\mathbf{w}, \mathbf{x}) \in \mathbb{R}^{q \times n}$ has entries $[\kappa(\mathbf{w}, \mathbf{x})]_{ji} = \kappa(\mathbf{z}_j, \mathbf{x}_i)$ and thus

$$\psi^{\text{gmp}} = \kappa(\mathbf{w}, \mathbf{w})^{1/2} \Big(\kappa(\mathbf{w}, \mathbf{x})\kappa(\mathbf{w}, \mathbf{x})^\top + \gamma\kappa(\mathbf{w}, \mathbf{w})\Big)^{-1} \kappa(\mathbf{w}, \mathbf{x})\mathbf{1}.$$

This computation is compatible with the summation kernel introduced in (5.1) but becomes intractable for RKN presented in Chapter 3 which pools across $|\mathcal{I}_{\mathbf{x},k}|$ positions. Instead, we heuristically apply GMP over the set $\psi_k(\mathbf{x}_{1:t})$ for all t with $\lambda_{\mathbf{x},\mathbf{i}} = \lambda^{|\mathbf{x}|-i_1-k+1}$, which can be obtained from the RNN described in Theorem 3.1. This amounts to composing GMP with mean poolings obtained over each prefix of $\mathbf{x}$. We observe that it performs well in our experiments.

5.4. An Optimal Transport Based Kernel Embedding for Feature Aggregation

In this section, we present an optimal transport based kernel, which is interesting but not positive definite (p.d.). Then, we introduce an alternative parametrized kernel that is p.d., more scalable and can be optimized for a given task.

5.4.1. An Attractive yet non Positive Definite Kernel

In contrast to the simple averaging used in (5.1), we can leverage the correspondence provided by the optimal transport plan to adaptively weight the comparisons of features.

Definition 5.1 (Optimal transport match kernel)**.** *Let $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$ be two sets of respective length n and n'. The Optimal Transport Match Kernel is defined as*

$$K_{OT}(\mathbf{x}, \mathbf{x}') = \langle \mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}'), \kappa(\mathbf{x}, \mathbf{x}') \rangle := \sum_{i,j} \mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}')_{ij} \kappa(\mathbf{x}_i, \mathbf{x}'_j), \tag{5.5}$$

where $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}') \in U\left(1/n, 1/n'\right)$ *is the solution to the regularized optimal transport problem* (5.3) *between* $\mathbf{x}$ *and* $\mathbf{x}'$*, whose cost* $\mathbf{C} \in \mathbb{R}^{n \times n'}$ *has entries* $\mathbf{C}_{ij} = -\kappa(\mathbf{x}_i, \mathbf{x}'_j)$ *since* κ *is a measure of similarity.*

When $\varepsilon = 0$, K_{OT} is equivalent to the 2-Wasserstein distance associated to the distance d_κ induced by κ—defined as $d_\kappa^2(u, v) = \kappa(u, u) - 2\kappa(u, v) + \kappa(v, v)$ for any u, v—in the following sense:

$$\begin{aligned} W_2^2(\mathbf{x}, \mathbf{x}') &:= \min_{\mathbf{P}_\kappa \in U\left(\frac{1}{n}, \frac{1}{n'}\right)} \langle \mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}'), d_\kappa^2(\mathbf{x}, \mathbf{x}') \rangle \\ &= \frac{1}{n} \sum_{i=1}^{n} \kappa(\mathbf{x}_i, \mathbf{x}_i) + \frac{1}{n'} \sum_{j=1}^{n'} \kappa(\mathbf{x}'_j, \mathbf{x}'_j) - 2K_{\mathrm{OT}}(\mathbf{x}, \mathbf{x}'). \end{aligned} \tag{5.6}$$

K_{OT} performs well in practice, as shown in Section 5.5, but principally suffers from three issues: (i) for small values of ε, it is not positive-definite (experiments exhibit negative eigenvalues in the Gram matrix); (ii) computing K_{OT} requires solving the transport problems between all pairs $(\mathbf{x}, \mathbf{x}')$ in the data set, which grows quadratically with the number of samples; (iii) K_{OT} cannot be optimized to the task at hand. We therefore introduce a positive definite kernel addressing these issues, notably reducing the number of the transport problems to solve to be linear of the number of samples.

5.4.2. A Parametrized Optimal Transport Based Kernel Embedding

The issue of positive definiteness of the 2-Wasserstein distance, corresponding to K_{OT} with $\varepsilon = 0$, is well known and has been studied (see Peyré et al. (2019) Section 8.3, Gardner et al. (2017) and Appendix 5.A). Here, we introduce a surrogate of the above kernel to address the three issues above at the same time, which is essentially based on the following observation:

$$\mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}') := p \times \mathbf{P}_\kappa(\mathbf{x}, \mathbf{z}) \mathbf{P}_\kappa(\mathbf{x}', \mathbf{z})^\top,$$

with $\mathbf{x}$, $\mathbf{x}'$ and $\mathbf{z}$ sets of features and $p = |\mathbf{z}|$, is a valid transport plan between $\mathbf{x}'$ and $\mathbf{x}$ thanks to the gluing lemma (see, *e.g*, Peyré et al. (2019)), and empirically, is a rough approximation of $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}')$. Other works explored the idea of computing the transport with respect to a common reference (Mérigot et al., 2020; Wang et al., 2013) yet for the non-regularized transport and with minimal use of the resulting embedding. By replacing the optimal transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}')$ in K_{OT} with $\mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}')$, we obtain a new kernel parametrized by $\mathbf{z}$ that enjoys better properties than K_{OT}.

Definition 5.2 (Optimal Transport Kernel (OTK) and Embedding)**.** *The OTK is defined as*

$$K_{\mathbf{z}}(\mathbf{x}, \mathbf{x}') := \langle \mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}'), \kappa(\mathbf{x}, \mathbf{x}') \rangle, \tag{5.7}$$

and its associated embedding $\Phi_{\mathbf{z}}$ *of* $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ *such that* $K_{\mathbf{z}}(\mathbf{x}, \mathbf{x}') = \langle \Phi_{\mathbf{z}}(\mathbf{x}), \Phi_{\mathbf{z}}(\mathbf{x}') \rangle$ *is*

$$\Phi_{\mathbf{z}}(\mathbf{x}) = \sqrt{p} \times \left(\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})_1^\top \varphi(\mathbf{x}), \ldots, \mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})_p^\top \varphi(\mathbf{x}) \right) = \sqrt{p} \times \mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})^\top \varphi(\mathbf{x}),$$

where $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})_i$ *denotes the i-th column of* $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$*, i.e the couplings between the elements of* $\mathbf{x}$ *and* $\mathbf{z_i}$*, and* $\varphi(\mathbf{x}) := [\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_n)]^\top$*, with* $\varphi : \mathbb{R}^d \to \mathcal{H}$ *the kernel embedding associated to* κ *and its RKHS* $\mathcal{H}$*. We design an element* $\mathbf{z}_i$ *of* $\mathbf{z}$ *as a "support" and* p *as number of supports.*

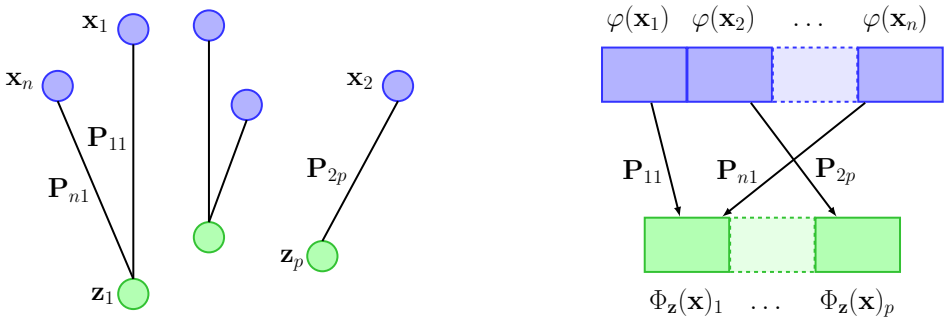


Figure 5.1.: The input point cloud $\mathbf{x}$ is transported onto the reference $\mathbf{z} = (\mathbf{z}_1, \dots, \mathbf{z}_p)$ (left), yielding the optimal transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ used to aggregate the embedded features and form $\Phi_{\mathbf{z}}(\mathbf{x})$ (right).

The OTK $K_{\mathbf{z}}$ solves the issues raised above: (i) it is p.d. since it can be cast as $\langle \Phi_{\mathbf{z}}(\mathbf{x}), \Phi_{\mathbf{z}}(\mathbf{x}') \rangle$; (ii) computing its Gram matrix requires computing only as many transport plans as samples (all the transports from the samples $\mathbf{x}$ to the reference $\mathbf{z}$); (iii) as we will show later, the parameter $\mathbf{z}$ can be adapted to a specific task. Regarding the interpretation of the embedding $\Phi_{\mathbf{z}}(\mathbf{x})$, the notion of pooling in the RKHS $\mathcal{H}$ of κ arises naturally if $p \leq n$. $\Phi_{\mathbf{z}}$ simultaneously embeds $\mathbf{x}$ to $\mathcal{H}^n$ via φ, aligns via $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ and also pools to $\mathcal{H}^p$. In other words, the elements of $\mathbf{x}$ are non-linearly embedded and then aggregated in "buckets", one for each element in the reference $\mathbf{z}$, given the values of $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$. This process is illustrated in Figure 5.1.

Fast computation with Nyström method. The computation of this kernel embedding can be accelerated using the Nyström method presented in Section 5.2.1. Once φ is replaced with ψ, the transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ with $\mathbf{z} \in \mathbb{R}^d$ in (5.7) can be reparametrized as $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z}')$, *i.e.* a transport whose cost is the opposite of the linear kernel and $\mathbf{z}' = \psi(\mathbf{z})$ in $\mathbb{R}^k$. By abuse of notation, we still use $\mathbf{z}$ for the new parametrization. The OTK embedding becomes simply

$$\Phi_{\mathbf{z}}(\mathbf{x}) = \sqrt{p} \times \mathbf{P}(\psi(\mathbf{x}), \mathbf{z})^\top \psi(\mathbf{x}) \in \mathbb{R}^{k \times p}, \tag{5.8}$$

with k the dimension of the Nyström embedding and p the number of support in $\mathbf{z}$. In our experiments, we will often use a Gaussian kernel so that (5.8) is the embedding used in practice. Next, we discuss how to learn the reference set $\mathbf{z}$.

5.4.3. Unsupervised and Supervised Learning of $\mathbf{z}$

Unsupervised learning. Without labels, and in the fashion of the Nyström approximation, the p elements of $\mathbf{z}$ can be defined as the centroids obtained by K-means applied to features from available training sets in $\mathcal{X}$. The next lemma, proved in Appendix 5.C, suggests another algorithm

Lemma 3 (Relation between $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}')$ and $\mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}')$ when $\varepsilon = 0$)**.** *For any* $\mathbf{x}$*,* $\mathbf{x}'$ *and* $\mathbf{z}$ *in* $\mathcal{X}$ *with lengths* n*,* n' *and* p*,*

$$|W_2(\mathbf{x}, \mathbf{x}') - \underbrace{\langle \mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}'), d_\kappa^2(\mathbf{x}, \mathbf{x}') \rangle^{1/2}}_{W_2^{\mathbf{z}}(\mathbf{x},\mathbf{x}')}| \leq 2 \min(W_2(\mathbf{x}, \mathbf{z}), W_2(\mathbf{x}', \mathbf{z})). \tag{5.9}$$

A corollary is a bound on the error term between W_2 *and* $W_2^{\mathbf{z}}$ *for* m *samples* $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$

$$\mathcal{E}^2 := \frac{1}{m^2} \sum_{i,j=1}^{m} |W_2(\mathbf{x}^i, \mathbf{x}^j) - W_2^{\mathbf{z}}(\mathbf{x}^i, \mathbf{x}^j)|^2 \leq \frac{4}{m} \sum_{i=1}^{m} W_2^2(\mathbf{x}^i, \mathbf{z}). \tag{5.10}$$

Equation (5.9) shows that the distance $W_2^{\mathbf{z}}$ resulting from $K_{\mathbf{z}}$ is related to the Wasserstein distance W_2; yet, this relation should not be interpreted as an approximation error, as our goal is not to approximate W_2, but rather to develop a different p.d. kernel with good computational properties. The right-hand term in Equation (5.10) corresponds to the objective to minimize in the Wasserstein Barycenter (Cuturi and Doucet, 2013) problem, which yields the mean of a set of empirical measures (here the $\mathbf{x}$'s) under the OT metric. The Wasserstein barycenter is therefore an attractive candidate for choosing $\mathbf{z}$. Both methods yield similar results as will be shown in Section 5.5 and Appendix 5.E. Wasserstein barycenters are less theoretically grounded for non-linear kernels which further justifies our parametrization (5.8). The anchor points $\mathbf{w}$ and the references $\mathbf{z}$ may be computed using similar algorithms; however, their mathematical interpretation differs as exposed above. The task of representing features (learning $\mathbf{w}$ in $\mathbb{R}^d$ for a specific κ) is decoupled from the task of aggregating (learning the reference $\mathbf{z}$ in $\mathbb{R}^k$), which is similar to the multilayer structure of neural networks.

Supervised learning. As mentioned in Section 5.2, $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z})$ is computed using the Sinkhorn's algorithm, recalled in Appendix 5.A, which can be easily adapted to batches of samples $\mathbf{x}$, with possibly varying lengths, leading to GPU-friendly forward computations of the OTK embedding $\Phi_{\mathbf{z}}$. More important, all the operations in an iteration of the Sinkhorn's algorithm are differentiable, which enables $\mathbf{z}$ to be optimized with stochastic gradient descent through back-propagation in the context of empirical risk minimization when labels are available. In practice, a small number of Sinkhorn iterations is generally sufficient to precisely compute $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z})$. Since the anchor points $\mathbf{w}$ in the embedding layer below can also be learned end-to-end (Mairal, 2016), the OTK is a module that can be injected into any deep network, as demonstrated in our experiments.

5.4.4. Extensions

Integrating positional information into the OTK. The discussed kernels do not take the position of the features into account, which may be problematic when dealing with structured data such as images or sentences. To this end, we borrow the idea of convolutional kernel networks (CKN) (Mairal, 2016; Mairal et al., 2014), *i.e.* to penalize the similarity exponentially with the positional distance between a pair of elements in the sequences. More precisely, we multiply κ by this positional term:

$$\kappa'(\mathbf{x}_i, \mathbf{x}'_j) = \kappa(\mathbf{x}_i, \mathbf{x}'_j) \times e^{-\frac{1}{\sigma_{\text{pos}}^2}(i/n - j/n')^2}.$$

and replace it in the OTK. With similarity weights based *both* on content and position, the OTK can be viewed as a generalization of the CKNs (whose similarity weights are based on position only) with feature alignment based on optimal transport. However, the resulting embedding function of the new kernel is not obvious to see as approximating the position term with a dot-product is not trivial. For this reason, we present here an embedding that works well in practice. After adding the positional term into the OTK, the kernel becomes

$$K_{\mathrm{OT}}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} \sum_{j=1}^{n'} \mathbf{P}_{\kappa'}(\mathbf{x}, \mathbf{x}')_{ij} \kappa'(\mathbf{x}_i, \mathbf{x}'_j).$$

Computing the transport plan against a reference measure $\mathbf{P}_{\kappa'}(\mathbf{x}, \mathbf{z})_{ij}$ is similar to the OTK without position encoding, by simply replacing the input similarity matrix $\mathbf{K}$ in (5.12) with $\mathbf{S} \odot \mathbf{K}$ where $\mathbf{S}$ denotes the matrix with entry

$$\mathbf{S}_{ij} = e^{-\frac{1}{\sigma_{\mathrm{pos}}^2}(i/n - j/p)^2}.$$

Whereas, the kernel mapping of κ' is no more φ (or ψ when φ is infinite dimensional) as there is this additional position term. Yet, we can mimic its effect without adding further dimensions by multiplying elementwisely $\mathbf{P}_{\kappa'}(\mathbf{x}, \mathbf{z})$ with $\mathbf{S}$. This results in the following embedding with position information

$$\Phi_{\mathbf{z}}(\mathbf{x}) = \sqrt{p} \times [\mathbf{P}_{\kappa'}(\mathbf{x}, \mathbf{z}) \odot \mathbf{S}]^{\top} \varphi(\mathbf{x}).$$

Note that when dealing with multi-dimensional objects such as images, we just replace the index scalar i with an index vector of the same spatial dimension as the object, representing the positions of each dimension.

Using multiple references. A naive reconstruction using different references $\mathbf{z}^1, \ldots, \mathbf{z}^q$ in $\mathcal{X}$ may yield a better approximation of the transport plan. In this case, the embedding of $\mathbf{x}$ becomes

$$\Phi_{\mathbf{z}^1,\ldots,\mathbf{z}^q}(\mathbf{x}) = {}^1\!/\!_{\sqrt{q}} \left(\Phi_{\mathbf{z}^1}(\mathbf{x}), \ldots, \Phi_{\mathbf{z}^q}(\mathbf{x})\right), \tag{5.11}$$

with q the number of references (the factor ${}^1\!/\!_{\sqrt{q}}$ comes from the mean). The references do not necessarily have the same number of supports $\mathbf{z_i}$. Using relation (5.9) (see Appendix 5.C for details), we can obtain an error bound similar to (5.10) for a data set of m samples $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$ and q references. To choose multiple references, we tried a K-means algorithm with 2-Wasserstein distance for assigning clusters, and we updated the centroids as in the single-reference case. We observe in Section 5.5 that using multiple references is particularly useful when optimizing $\mathbf{z}$ with supervision.

5.5. Experiments

In this section, we present the experimental results for GMP and OTK respectively.

Table 5.1.: Average auROC and auROC50 for SCOP fold recognition benchmark. Details about pairwise statistical tests between methods can be found in Appendix 5.D.

Method	pooling	one-hot		BLOSUM62	
		auROC	auROC50	auROC	auROC50
CKN-seq	mean	0.827	0.536	0.843	0.563
CKN-seq	max	0.837	0.572	0.866	0.621
CKN-seq	GMP	0.838	0.561	0.856	0.608
RKN	mean	0.829	0.541	0.840	0.571
RKN	max	0.844	**0.587**	**0.871**	**0.629**
RKN	GMP	**0.848**	0.570	0.852	0.609

5.5.1. Results for Generalized Max Pooling

Here we consider the Structural Classification Of Proteins (SCOP) version 1.67 (Murzin et al., 1995). We follow the preprocessing procedures of Håndstad et al. (2007) and remove the sequences that are more than 95% similar, yielding 85 fold recognition tasks. Each positive training set is then extended with Uniref50 to make the dataset more balanced, as proposed in Hochreiter et al. (2007). The resulting dataset can be downloaded from `http://www.bioinf.jku.at/software/LSTM_protein`. The number of training samples for each task is typically around 9,000 proteins, whose length varies from tens to thousands of amino-acids. In all our experiments we use logistic loss. We measure classification performances using auROC and auROC50 scores (area under the ROC curve and up to 50% false positives).

For CKN (Chen et al., 2019a) and RKN (Chen et al., 2019b), we evaluate both one-hot encoding of amino-acids by 20-dimensional binary vectors and an alternative representation relying on the BLOSUM62 substitution matrix (Henikoff and Henikoff, 1992). Specifically in the latter case, we represent each amino-acid by the centered and normalized vector of its corresponding substitution probabilities with other amino-acids. The results for CKN and RKN with different pooling operations are shown in Table 5.1. Consistently, within kernel networks GMP systematically outperforms mean pooling, while being slightly behind max pooling.

5.5.2. Results for Optimal Transport Kernel

We demonstrate the effectiveness of the OTK embedding in biology sequence and image classification tasks in unsupervised and supervised settings. Although K_{OT} in (5.5) and its surrogate $K_{\mathbf{z}}$ in (5.7) are of interest, their lack of scalability – they require to compute the Gram matrix, which is quadratic in the number of samples – makes them less suited to large data sets, unlike our explicit embedding $\Phi_{\mathbf{z}}$. Nevertheless, a brief study of their performance can be found in Appendix 5.E. A brief study of OTK in a natural language processing task can also be found in Appendix 5.E.5 to show the effectiveness of OTK for another data modality.

Influence of z on the OTK embedding. The OTK embedding $\Phi_{\mathbf{z}}$ defined in (5.11) is characterized by the number of references q and the number p of features (supports) in each reference set. As mentioned in Section 5.4, we investigate several algorithms for learning the references without supervision. The discussed results can be found in Appendix 5.E. Using more references and increasing the number of supports generally yields better results at the expense of a larger computational cost and K-means turned out to be a simple and effective approach for learning the references.

Protein fold classification on SCOP 1.75 and 2.06. Our protein fold classification experiments consider the Structural Classification Of Proteins (SCOP) version 1.75 and 2.06. We follow the data preprocessing protocols in Hou et al. (2018), which yields a training and validation set composed of 14699 and 2013 sequences from SCOP 1.75, and a test set of 2533 sequences from SCOP 2.06. The resulting protein sequences belong to 1195 different folds, thus the problem is formulated as a multi-classification task. The input sequence is represented as a 45-dimensional vector at each amino acid. The vector consists of a 20-dimensional one-hot encoding of the sequence, a 20-dimensional position-specific scoring matrix (PSSM) representing the profile of amino acids, a 3-class secondary structure represented by a one-hot vector and a 2-class solvent accessibility. The lengths of the sequences are varying from tens to thousands.

The sequences are encoded with a Gaussian kernel as in CKNs (Chen et al., 2019a). While a global average pooling operation is used to aggregate the kernel embeddings in CKNs, the OTK embedding (5.8) performs an adaptive pooling. Different numbers of anchor points (128, 512 and 1024) are considered in the Nyström approximation. In the unsupervised setting, we benchmark our features against the regular, unsupervised state-of-the-art CKN features (Chen et al., 2019b). As shown in Table 5.2, the OTK embedding clearly outperforms the CKN features. In the supervised setting, we compare our optimized features to three state-of-the-art features for this task, obtained by recurrent kernel networks (RKN) (Chen et al., 2019b) and CKNs, both learned with supervision, and a more traditional model based on CNNs, DeepSF (Hou et al., 2018). Our method outperforms all baselines as shown in Table 5.2. Note how our unsupervised features are as good if not better than the supervised baselines. Complementary results on the effect of $\mathbf{z}$, and comparison with Wasserstein barycenter for learning $\mathbf{z}$ and error bars can be found in Appendix 5.E.3.

Detection of chromatin profiles. Predicting the functional effects of noncoding variants from only genomic sequences is a central task in human genetics. A fundamental step for this task is to simultaneously predict large-scale chromatin features from DNA sequences (Zhou and Troyanskaya, 2015). We consider here the DeepSEA dataset, which consists in simultaneously predicting 919 chromatin profiles including 690 transcription factor (TF) binding profiles for 160 different TFs, 125 DNase I sensitivity profiles and 104 histone-mark profiles. In total, there are 4.4 million, 8000 and 455024 samples for training, validation and test. Each sample consists of a 1000-bp DNA sequence from the human GRCh37 reference. Each sequence is represented as a 1000×4 binary matrix using one-hot encoding on DNA characters. The dataset is available at `http://deepsea.princeton.edu/media/code/deepsea_train_bundle.v0.9.tar.gz`. Note that the labels for each profile are very imbalanced in this task with few positive sam-

Table 5.2.: Classification accuracy (top 1/5/10) on test set for SCOP 1.75 for different combinations of (number of references q × number of supports p). The accuracies are averaged from 10 different runs. DeepSF (Hou et al., 2018) is a CNN with 10 convolutional layers. The OTK outperforms all baselines.

Unsupervised					
Nb filters	CKN			OTK (1×50)	OTK (1×100)
128	64.7/86.3/91.6			77.5/91.7/94.5	**79.4/92.4/94.9**
1024	82.2/92.8/95.2			84.6/95.0/**97.0**	**85.7/95.3**/96.7
Supervised					
Nb filters	DeepSF	CKN	RKN	OTK (1×50)	OTK (5×10)
128	73.0/90.3/94.5	76.3/92.2/95.3	77.8/92.9/95.5	82.8/93.9/96.2	**84.7/94.7/96.5**
512		84.1/94.3/96.4	85.3/95.0/96.5	88.4/95.8/97.1	**88.7/95.9/97.3**

Table 5.3.: Results for prediction of chromatin profiles on the DeepSEA dataset. The metrics are area under ROC (auROC) and area under PR curve (auPRC), averaged over 919 chromatin profiles. The accuracies are averaged from 10 different runs. Armed with the positional encoding (PE) described in Section 5.4, the OTK outperforms the state-of-the-art model and an OTK with the PE proposed in Vaswani et al. (2017).

Method	DeepSEA (3-layer-CNN)	OTK + Sinusoidal PE	OTK + Our PE
auROC	0.933	0.917	**0.936**
auPRC	0.342	0.311	**0.360**

Table 5.4.: Unsupervised classification with CIFAR-10 for various features extracted from 2-layer unsupervised CKNs with different numbers of filters. The accuracies are averaged from 10 different runs. The unsupervised OTK has one reference with 64 supports learned with K-means, with or without our position encoding (PE). Our OTK embedding notably improves the base features.

Dataset	Flatten	Mean pooling	Gaussian pooling (Mairal, 2016)	OTK	OTK (with PE)
$16 \times 16 \times 256$	73.1	64.9	77.5	77.9	**81.4**
$16 \times 16 \times 1024$	76.1	73.4	82.0	80.1	**83.2**

ples. For this reason, learning unsupervised models could be intractable as they may require an extremely large number of parameters if junk or redundant sequences cannot be filtered out. We thus use our supervised OTK in (5.8) as an adaptive pooling layer and inject it into a deep neural network model, between a convolutional layer and a fully connected layer. We compare it to the state-of-the art model (Zhou and Troyanskaya, 2015), a CNN with 3 convolutional layers, in Table 5.3. In contrast to a typical transformer which would have stored a 1000×1000 matrix, our attention score, with a reference of size 64, is only 1000×64. Realizing that position encoding is crucial for this task, we also compare our encoding to the sinusoidal encoding introduced in the transformer (Vaswani et al., 2017) and find that ours is more effective here. More details about model architectures and training can be found in Appendix 5.E.4.

Image classification on CIFAR-10. Here we use CIFAR-10 features, *i.e.* 60000 images with 32×32 pixels and 10 classes encoded using a two-layer CKN (Mairal, 2016), one of the baseline architectures for unsupervised learning of CIFAR-10, and evaluate on the standard test set. The very best configuration of the CKN yields a small number (3×3) of high-dimensional (16384) patches and an accuracy of 85.8%. Because our embedding is designed for larger sets of features, it is more consistent to illustrate it on a configuration which performs slightly less but provides more patches (16×16). While the CKN uses a Gaussian pooling (with pooling size equal to 6) after a 2-layer convolutional kernel, our OTKs (5.8) performs an adaptive pooling. The results are shown on Table 5.4. Again, without supervision, the adaptive pooling of the CKN features by the OTK notably improves their performance. We notice that the position encoding is very important to this task, which substantially improves the performance of its counterpart without it. More details can be found in Appendix 5.E.2.

5.6. Discussion

In this chapter, we proposed a few useful and valid kernel operations for feature aggregation in RKHS, namely GMP and OTK. Rather than using simple mean or sum pooling to aggregate local kernel embeddings, we showed that GMP and OTK can possibly capture complex dependencies between local features and thus perform better in applications, such as tasks of biological sequence classification. The experimental results also suggest the importance of developing efficient algorithms for simultaneously solving a large number of optimal transport problems, which leads to an orthogonal direction to

the large-scale optimal transport problem (Genevay et al., 2016) in the sense of the number of samples from each distribution. A more thorough study on this problem would bring more insight into adaptive weighting techniques and attention-based models.

Appendix

Appendix 5.A provides some background on notions used throughout the chapter; Appendix 5.B adds details on the implementation and foundation of our OTK; Appendix 5.C contains the proofs skipped in the chapter; Appendix 5.E provides details on our experimental protocol for reproducibility and additional results.

5.A. Background

This section provides some background on Sinkhorn's algorithm and on the relationship between optimal transport based kernels and positive definite histogram kernels.

5.A.1. Sinkhorn's Algorithm: Fast Computation of $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$

Without loss of generality, we consider here κ the linear kernel. We recall that $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ is the solution of an optimal transport problem, which can be efficiently solved by Sinkhorn's algorithm (Peyré et al., 2019) involving matrix multiplications only. Specifically, Sinkhorn's algorithm is an iterative matrix scaling method that takes the opposite of the pairwise similarity matrix $\mathbf{K}$ with entry $\mathbf{K}_{ij} := \langle \mathbf{x}_i, \mathbf{z}_j \rangle$ as input $\mathbf{C}$ and outputs the optimal transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z}) = \text{Sinkhorn}(\mathbf{K}, \varepsilon)$. Each iteration step ℓ performs the following updates

$$\mathbf{u}^{(\ell+1)} = \frac{1/n}{\mathbf{E}\mathbf{v}^{(\ell)}} \quad \text{and} \quad \mathbf{v}^{(\ell+1)} = \frac{1/p}{\mathbf{E}^\top \mathbf{u}^{(\ell)}}, \tag{5.12}$$

where $\mathbf{E} = e^{\mathbf{K}/\varepsilon}$. Then the matrix $\text{diag}(\mathbf{u}^{(\ell)})\mathbf{E}\text{diag}(\mathbf{v}^{(\ell)})$ converges to $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ when ℓ tends to ∞. However when ε becomes too small, some of the elements of a matrix product $\mathbf{E}\mathbf{v}$ or $\mathbf{E}^\top \mathbf{u}$ become null and result in a division by 0. To overcome this numerical stability issue, computing the multipliers $\mathbf{u}$ and $\mathbf{v}$ is preferred (see *e.g.* (Peyré et al., 2019, Remark 4.23)). This algorithm can be easily adapted to a batch of data points $\mathbf{x}$, and with possibly varying lengths via a mask vector masking on the padding positions of each data point $\mathbf{x}$, leading to GPU-friendly computation. More importantly, all the operations above at each step are differentiable, which enables $\mathbf{z}$ to be optimized through back-propagation. Consequently, this module can be injected into any deep networks.

5.A.2. On the Relationship Between Optimal Transport Match Kernel and Histogram Kernels

When features are living in a discrete set $\mathcal{F}$, it is classical to represent a set of features $\mathbf{x}$ as a histogram $\mathbf{H}(\mathbf{x}) = (\mathbf{H}_\mathbf{u}(\mathbf{x}))_{\mathbf{u}\in\mathcal{F}}$, with $\mathbf{H}_\mathbf{u}(\mathbf{x})$ the number of occurrences of the pattern $\mathbf{u}$ in $\mathbf{x}$. The spectrum kernel (Leslie et al., 2002a) used in biology computes the dot product between the normalized histograms $\hat{\mathbf{H}}(\mathbf{x}) = \mathbf{H}(\mathbf{x})/|\mathbf{x}|$ and $\hat{\mathbf{H}}(\mathbf{x}')$. Interestingly, this kernel can be rewritten as (Kuksa et al., 2009)

$$K_{\text{spectrum}}(\mathbf{x}, \mathbf{x}') := \langle \hat{\mathbf{H}}(\mathbf{x}), \hat{\mathbf{H}}(\mathbf{x}') \rangle = \frac{1}{n}\frac{1}{n'} \sum_{i=1}^{n} \sum_{j=1}^{n'} \delta(\mathbf{x}_i, \mathbf{x}'_j),$$

where δ denotes the Dirac kernel. As the spectrum kernel performs exact matches between elements, more flexible variants where then proposed, either by allowing mismatches for sequence data (Leslie et al., 2004), or by using a Gaussian kernel for comparing features (Chen et al., 2019a). A related version of K_{spectrum} is the histogram intersection kernel (Barla et al., 2003), proposed in the context of computer vision, which computes the minimum of the counts at each bin of the two histograms instead of the dot product in the spectrum kernel. Interestingly, this kernel exhibits a well-known relation with optimal transport, see Gardner et al. (2017):

Lemma 4 (Histograms intersection as an optimal match kernel)**.** *The histogram intersection can be cast as the optimization of a match kernel:*

$$K_{hist}(\mathbf{x}, \mathbf{x}') := \sum_{u \in \mathcal{F}} \min(\hat{\mathbf{H}}_u(\mathbf{x}), \hat{\mathbf{H}}_u(\mathbf{x}')) = \max_{P \in U_r(\frac{1}{n}, \frac{1}{n'})} \sum_{i=1}^{n} \sum_{j=1}^{n'} \mathbf{P}_{ij} \delta(\mathbf{x}_i, \mathbf{x}'_j),$$

where U_r is a relaxed Kantorovich coupling constraint

$$U_r\left(\frac{1}{n}, \frac{1}{n'}\right) = \{\mathbf{P} \in \mathbb{R}_+^{n \times n'} : \mathbf{P}\mathbf{1}_n \leq \frac{1}{n} \text{ and } \mathbf{P}^\top \mathbf{1}_{n'} \leq \frac{1}{n'}\}.$$

Proof. For completeness, a proof can be found in Appendix 5.C. □

It is possible to show that the distance induced by K_{hist} is the ℓ_1-norm between two histograms in contrast to the ℓ_2-norm in K_{spectrum}. A similar relation between the intersection kernel and a variation of optimal transport was also studied in Gardner et al. (2017). When dealing with discrete feature sets, our kernel K_{OT} differs from K_{hist} in three aspects: (i) the relaxed Kantorovich constraint becomes exact; (ii) we add an entropic penalty term $H(\mathbf{P})$, which brings both flexibility as it interpolates between K_{hist} (when $\varepsilon = 0$) and K_{spectrum} (when ε gets bigger, we maximize the entropy, which is equivalent to summing all the pairs with identical weights as in K_{spectrum}), and computational scalability as well as differentiability thanks to Sinkhorn's algorithm; (iii) we relax δ by a positive definite, differentiable kernel κ allowing mismatches and end-to-end learning (Chen et al., 2019a), *e.g*, a Gaussian kernel.

5.B. Additional details

This section provides additional details on the positional encoding in the OTK, and the reconstruction of the transport plan for Gaussian distributions.

5.B.1. Reconstruction of the Transport for Gaussian Distributions

In the case of Gaussian distributions and Monge formulation of optimal transport, the reconstruction formula given in 5.4.2 is exact. Before we prove this, we introduce the notion of Bures-Wasserstein distance and its well-known relationship to optimal transport. Let x, x' and z be random gaussian vectors in $\mathbb{R}^n$ with zero mean and respective covariance matrices A, B and C. For such distributions, the Bures-Wasserstein distance between A and B, defined as

$$d(A, B) := \left[\text{tr}A + \text{tr}B - 2\text{tr}(A^{1/2}BA^{1/2})^{1/2}\right]^{1/2},$$

coincides with the 2-Wasserstein distance between x and x', $W_2(x, x')$. As a result, Bures-Wasserstein theory (see, *e.g*, Bhatia et al. (2018)) provides a closed form solution to the optimal transport problem cast as

$$W_2(x, x') := \inf_{\gamma \in \Gamma(\mu,\nu)} \left(\int_{\mathbb{R}^n \times \mathbb{R}^n} \|x - y\|^2 d\gamma(x, y) \right)^{1/2},$$

where μ and ν are mass distributions: the minimum is indeed attained in $x' = Tx$, with

$$T = A^{-1} \# B := A^{-1}(AB)^{1/2} = (A^{-1}B^{-1})^{1/2}B.$$

Lemma 5. *Let x, x' and z be gaussian distributions with covariance matrices A, B and C. We assume that we know the Monge optimal transport mapping $T_{A\to C}$ from x to z and $T_{B\to C}$ from x' to z. Then, the mapping from x to x' can be obtained with*

$$\tilde{T}_{A\to B} := T_{B\to C}^{-1} T_{A\to C}.$$

Proof. If we use the regular transport from x (covariance matrix A) to x' (covariance matrix B) the new covariance matrix is

$$\begin{aligned}
\mathbf{E}(T_{A\to B} x x^\top T_{A\to B}^\top) &= T_{A\to B} \mathbf{E}(xx^\top) T_{A\to B}^\top \\
&= T_{A\to B} A T_{A\to B} \\
&= A^{-1}(AB)^{1/2} A A^{-1}(AB)^{1/2} \\
&= B.
\end{aligned}$$

Now, transporting x to z then z to x', the reconstructed covariance matrix is

$$\begin{aligned}
\mathbf{E}(T_{C\to B} T_{A\to C} x x^\top T_{A\to C}^\top T_{C\to B}^\top) &= T_{C\to B} T_{A\to C} \mathbf{E}(xx^\top) T_{A\to C}^\top T_{C\to B}^\top \\
&= T_{C\to B} T_{A\to C} A T_{A\to C} T_{C\to B} \\
&= T_{C\to B} T_{A\to C} A A^{-1}(AC)^{1/2} T_{C\to B} \\
&= T_{C\to B} A^{-1} A C T_{C\to B} \\
&= T_{C\to B} C T_{C\to B} \\
&= B.
\end{aligned}$$

So, transporting x to z before transporting the result to x' is equivalent to directly transporting x to x'. The reconstructed transport yields the same distribution. Since $T_{B\to C}^{-1} = T_{C\to B}$, we can conclude. □

Remark 5.1. *If x and x' have non-zero means, one just needs to use the mapping $\tilde{T}_{A\to B}(x - \mu_x) + \mu_{x'}$.*

5.C. Proofs

5.C.1. Proof of Lemma 4

Proof. For any $\mathbf{P} \in U_r\left(\frac{1}{n}, \frac{1}{n'}\right)$, we have

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n'}\mathbf{P}_{ij}\delta(\mathbf{x}_i, \mathbf{x'}_j) &= \sum_{i=1}^{n}\sum_{j=1}^{n'}\sum_{u\in\mathcal{F}}\mathbf{P}_{ij}\delta(\mathbf{x}_i, u)\delta(\mathbf{x'}_j, u)\\ &= \sum_{u\in\mathcal{F}}\underbrace{\sum_{i=1}^{n}\sum_{j=1}^{n'}\mathbf{P}_{ij}\delta(\mathbf{x}_i, u)\delta(\mathbf{x'}_j, u)}_{:=f_u(\mathbf{x},\mathbf{x}')}.\end{aligned}$$

Then,

$$f_u(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n}\delta(\mathbf{x}_i, u)\sum_{j=1}^{n'}\mathbf{P}_{ij}\delta(\mathbf{x'}_j, u) \leq \sum_{i=1}^{n}\delta(\mathbf{x}_i, u)\underbrace{\sum_{j=1}^{n'}\mathbf{P}_{ij}}_{\leq 1/n} \leq \mathbf{H}_u(\mathbf{x})/n = \hat{\mathbf{H}}_u(\mathbf{x}),$$

and similarly, $f_u(\mathbf{x}, \mathbf{x}') \leq \hat{\mathbf{H}}_u(\mathbf{x}')$. Consequently, $f_u(\mathbf{x}, \mathbf{x}') \leq \min(\hat{\mathbf{H}}_u(\mathbf{x})\hat{\mathbf{H}}_u(\mathbf{x}'))$. Since the index sets $\mathcal{I}_u(\mathbf{x}, \mathbf{x}') = \{(i, j) \in [1, n] \times [1, n'] : \mathbf{x}_i = \mathbf{x'}_j = u\}$ are disjoint for different u, we will show that the equality for any $u \in \mathcal{F}$ can be attained with

$$\mathbf{P}_{ij} = \min\left(\frac{1}{n\mathbf{H}_{\mathbf{x}_i}(\mathbf{x}')}, \frac{1}{n'\mathbf{H}_{\mathbf{x'}_j}(\mathbf{x})}\right)\delta(\mathbf{x}_i, \mathbf{x'}_j).$$

First, $\sum_{j=1}^{n'}\mathbf{P}_{ij} = \min(1/n, \mathbf{H}_{\mathbf{x}_i}(\mathbf{x}')/n'\mathbf{H}_{\mathbf{x}_i}(\mathbf{x})) \leq 1/n$ and, similarly, $\sum_{i=1}^{n}\mathbf{P}_{ij} \leq 1/n'$. As a consequence, $\mathbf{P} \in U_r\left(\frac{1}{n}, \frac{1}{n'}\right)$. Moreover,

$$\begin{aligned}f_u(\mathbf{x}, \mathbf{x}') &= \sum_{(i,j)\in\mathcal{I}_u(\mathbf{x},\mathbf{x}')}\mathbf{P}_{ij}\\ &= \mathbf{H}_u(\mathbf{x})\mathbf{H}_u(\mathbf{x}')\min(1/n\mathbf{H}_u(\mathbf{x}'), 1/n'\mathbf{H}_u(\mathbf{x}))\\ &= \min(\hat{\mathbf{H}}_u(\mathbf{x}), \hat{\mathbf{H}}_u(\mathbf{x}')),\end{aligned}$$

so we have an equality case, which concludes the proof. □

A direct conclusion from this Lemma is that the optimal match problem 4 defines a positive definite kernel since the histogram intersection kernel is known to be positive definite.

5.C.2. Proof of Lemma 3

Proof. First, since $\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk} = 1$ for any k, we have

$$\begin{aligned} W_2(\mathbf{x}, \mathbf{z})^2 &= \sum_{i=1}^{n}\sum_{k=1}^{p} \mathbf{P}(\mathbf{x}, \mathbf{z})_{ik} d_\kappa^2(\mathbf{x}_i, \mathbf{z}_k) \\ &= \sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik} d_\kappa^2(\mathbf{x}_i, \mathbf{z}_k) \\ &= \|\mathbf{u}\|_2^2, \end{aligned}$$

with $\mathbf{u}$ a vector in $\mathbb{R}^{nn'p}$ whose entries are $\sqrt{p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik}}\, d_\kappa(\mathbf{x}_i, \mathbf{z}_k)$ for $i = 1, \dots, n$, $j = 1, \dots, n'$ and $k = 1, \dots, p$. We can also rewrite $W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')$ as an ℓ_2-norm of a vector $\mathbf{v}$ in $\mathbb{R}^{nn'p}$ whose entries are $\sqrt{p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik}}\, d_\kappa(\mathbf{x}_i, \mathbf{x}'_j)$. Then by Minkowski inequality for the ℓ_2-norm, we have

$$\begin{aligned} |W_2(\mathbf{x}, \mathbf{z}) - W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')| &= |\|\mathbf{u}\|_2 - \|\mathbf{v}\|_2| \\ &\leq \|\mathbf{u} - \mathbf{v}\|_2 \\ &= \left(\sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik}(d_\kappa(\mathbf{x}_i, \mathbf{z}_k) - d_\kappa(\mathbf{x}_i, \mathbf{x}'_j))^2\right)^{1/2} \\ &\leq \left(\sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik} d_\kappa^2(\mathbf{x}'_j, \mathbf{z}_k)\right)^{1/2} \\ &= \left(\sum_{k=1}^{p}\sum_{j=1}^{n'} \mathbf{P}(\mathbf{x}', \mathbf{z})_{jk} d_\kappa^2(\mathbf{x}'_j, \mathbf{z}_k)\right)^{1/2} \\ &= W_2(\mathbf{x}', \mathbf{z}), \end{aligned}$$

where the second inequality is the triangle inequality for the distance d_κ. Finally, we have

$$\begin{aligned} &|W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')| \\ \leq &|W_2(\mathbf{x}, \mathbf{x}') - W_2(\mathbf{x}, \mathbf{z})| + |W_2(\mathbf{x}, \mathbf{z}) - W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')| \\ \leq &W_2(\mathbf{x}', \mathbf{z}) + W_2(\mathbf{x}', \mathbf{z}) \\ = &2W_2(\mathbf{x}', \mathbf{z}), \end{aligned}$$

where the second inequality is the triangle inequality for the 2-Wasserstein distance. By symmetry, we also have $|W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')| \leq 2W_2(\mathbf{x}, \mathbf{z})$, which concludes the proof. □

5.C.3. Relationship between W_2 and $W_2^{\mathbf{z}}$ for Multiple References

Using relation (5.9) (see Appendix 5.C for details), we can obtain a bound on the error term between W_2 and $W_2^{\mathbf{z}}$ for a data set of m samples $(\mathbf{x}^1, \dots, \mathbf{x}^m)$ and q references

$(\mathbf{z}^1, \dots, \mathbf{z}^q)$

$$\mathcal{E}^2 := \frac{1}{m^2} \sum_{i,j=1}^{m} |W_2(\mathbf{x}^i, \mathbf{x}^j) - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}^i, \mathbf{x}^j)|^2 \leq \frac{4}{mq} \sum_{i=1}^{m} \sum_{j=1}^{q} W_2^2(\mathbf{x}^i, \mathbf{z}^j). \tag{5.13}$$

When $q = 1$, the right-hand term in the inequality is the objective to minimize in the Wasserstein barycenter problem (Cuturi and Doucet, 2013), which further explains why we considered it: Once $W_2^{\mathbf{z}}$ is close to the Wasserstein distance W_2, $K_{\mathbf{z}}$ will also be close to K_{OT} thanks to relation (5.6). We extend here the bound given in Lemma 3 in the case of one reference to the multiple-reference case. The approximate 2-Wasserstein distance $W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')$ thus becomes

$$W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}, \mathbf{x}') := \left\langle \frac{1}{q} \sum_{j=1}^{q} \mathbf{P}_{\mathbf{z}^j}(\mathbf{x}, \mathbf{x}'), d_\kappa^2(\mathbf{x}, \mathbf{x}') \right\rangle^{1/2} = \left(\frac{1}{q} \sum_{j=1}^{q} W_2^{\mathbf{z}^j}(\mathbf{x}, \mathbf{x}')^2 \right)^{1/2}.$$

Then by Minkowski inequality for the ℓ_2-norm we have

$$\begin{aligned} |W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}, \mathbf{x}')| &= \left| \left(\frac{1}{q} \sum_{j=1}^{q} W_2(\mathbf{x}, \mathbf{x}')^2 \right)^{1/2} - \left(\frac{1}{q} \sum_{j=1}^{q} W_2^{\mathbf{z}^j}(\mathbf{x}, \mathbf{x}')^2 \right)^{1/2} \right| \\ &\leq \left(\frac{1}{q} \sum_{j=1}^{q} (W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}^j}(\mathbf{x}, \mathbf{x}'))^2 \right)^{1/2}, \end{aligned}$$

and by Lemma 3 we have

$$|W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}, \mathbf{x}')| \leq \left(\frac{4}{q} \sum_{j=1}^{q} \min(W_2(\mathbf{x}, \mathbf{z}^j), W_2(\mathbf{x}', \mathbf{z}^j))^2 \right)^{1/2}.$$

Finally the approximation error in terms of Frobenius is bounded by

$$\mathcal{E}^2 := \frac{1}{m^2} \sum_{i,j=1}^{m} |W_2(\mathbf{x}^i, \mathbf{x}^j) - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}^i, \mathbf{x}^j)|^2 \leq \frac{4}{mq} \sum_{i=1}^{m} \sum_{j=1}^{q} W_2^2(\mathbf{x}^i, \mathbf{z}^j).$$

In particular, when $q = 1$ that is the case of single reference, we have

$$\mathcal{E}^2 \leq \frac{4}{m} \sum_{i=1}^{m} W_2^2(\mathbf{x}^i, \mathbf{z}),$$

where the right term equals to the objective of the Wasserstein barycenter problem, which justifies the choice of $\mathbf{z}$ when learning without supervision.

5.D. Additional Experimental Results for Generalized Max Pooling

Even though each method was run only one time for each task, the 85 tasks allow us to conduct statistical testing when comparing two methods. In Figure 5.D.1, we provide pairwise comparisons allowing us to assess the statistical significance of various comparisons of pooling operations drawn in the chapter. We use a Wilcoxon signed-rank test to provide p-values.

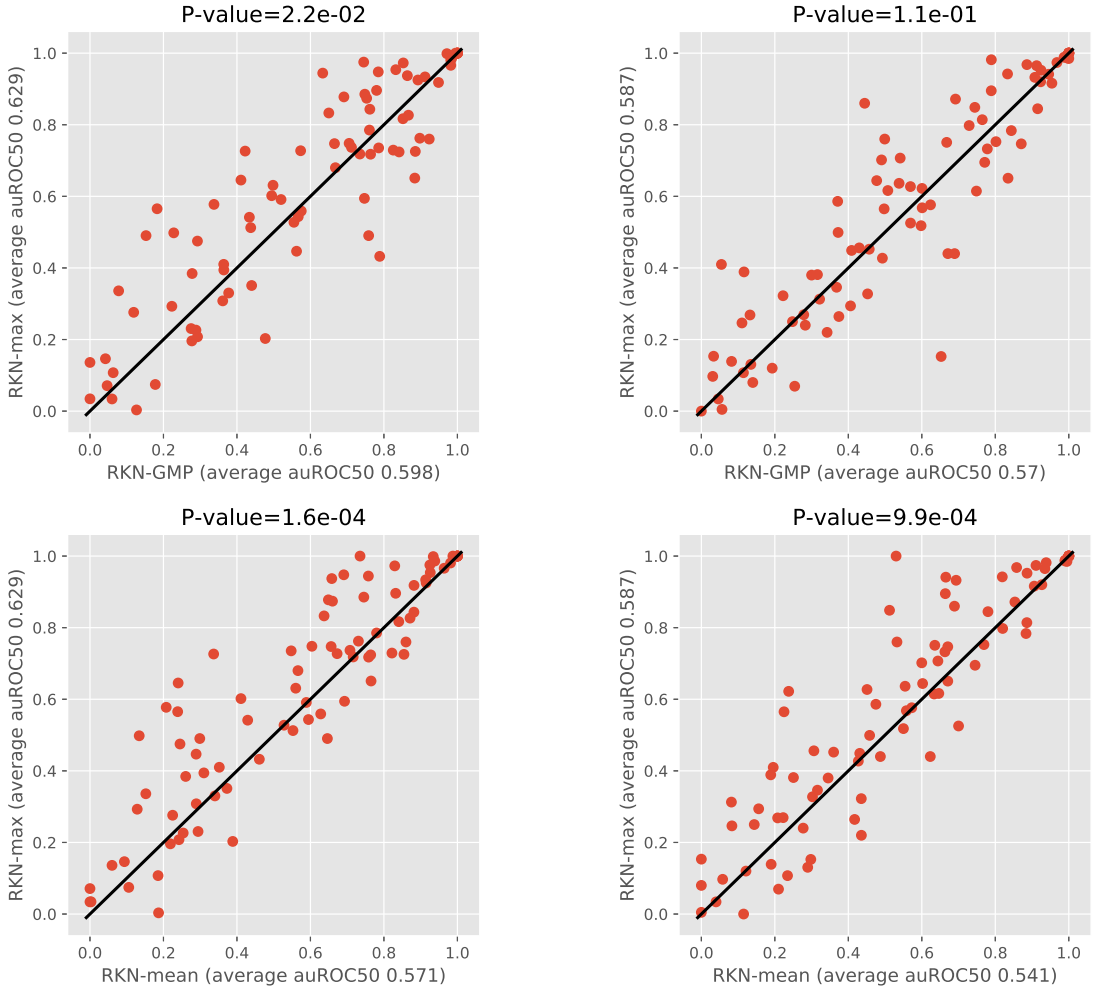


Figure 5.D.1.: Scatterplots when comparing pairs of methods. In particular, we want to compare RKN-gmp vs RKN-max (top); RKN-max vs. RKN-mean (bottom).

Table 5.E.1.: Accuracies obtained by computing the kernel matrices on 5000 samples of CIFAR-10 for various feature size: [(patch × patch), embedding dimension]. Metric: accuracy on validation set.

Dataset	Mean pooling	Linear kernel	K_{OT}	$K_{\mathbf{z}}$	K_{OT}+ Pos. enc.	$K_{\mathbf{z}}$+ Pos. enc.
(3 × 3), 256	0.584	0.649	0.619	0.61	**0.661**	0.652
(3 × 3), 8192	0.636	0.690	0.652	0.65	0.693	**0.694**

5.E. Additional Experimental Results for Optimal Transport Kernel

In this section, we provide details on our experimental protocol for reproducibility, as well as additional experimental results. The results are generally averaged over different runs, and the uncertainty is represented with the standard deviation.

5.E.1. Experiments on Kernel Matrices

Here, we study the optimal transport kernel K_{OT} (5.5) and its surrogate $K_{\mathbf{z}}$ (5.7) which exhibit interesting properties. For $K_{\mathbf{z}}$, the reference $\mathbf{z}$ is learned without supervision. Although our embedding $\Phi_{\mathbf{z}}$ is scalable, the exact kernel require the computation of Gram matrices. Therefore, 5000 samples only of CIFAR-10 (images with 32×32 pixels) are encoded without supervision using a two-layer convolutional kernel network (Mairal, 2016). The resulting features are 3×3 patches living in $\mathbb{R}^d$ with $d = 256$ or 8192. Since the features are already the output of a Gaussian Nyström embedding, the intermediate kernel κ is linear, which means that K_{OT} and $K_{\mathbf{z}}$ aggregate existing features linearly given the computed weight matrix $\mathbf{P}$. In that sense, we can say that our kernels work as an adaptive pooling. We therefore compare it to kernel matrices corresponding to mean pooling and no pooling at all (linear kernel). A linear classifier is trained from this matrices. Although we cannot prove that K_{OT} is positive definite, the classifier trained on the kernel matrix converges when ε is not too small. The results can be seen on Table 5.E.1. Without positional information, our kernels do better than Mean pooling. When the positions are encoded, the Linear kernel is also outperformed. Note that including positions in Mean pooling and Linear kernel means interpolating between these two kernels: in the Linear kernel, only patches with same index are compared while in the Mean pooling, all patches are compared. All interpolations did worse than the Linear kernel.

5.E.2. CIFAR-10

We build our OTK on top of the state-of-the-art unsupervised features for CIFAR-10, extracted from a 2-layer CKN (Mairal et al., 2014; Mairal, 2016) model with kernel sizes equal to 3 and 3, and Gaussian pooling size equal to 2 and 1. We consider the following configurations of the number of filters at each layer, to simulate different input dimensions for OTKs

- 64 filters at first and 256 at second layer, which yields a $16 \times 16 \times 256$ representation for each image;

Table 5.E.2.: Hyperparameter search range for CIFAR-10

Hyperparameter	Search range
Entropic regularization ε	$[1.0; 0.1; 0.01; 0.001]$
Position encoding bandwidth σ_{pos}	$[0.5; 0.6; 0.7; 0.8; 0.9; 1.0]$

Table 5.E.3.: Classification results for CIFAR-10. We consider here OTKs with one reference with different number of supports, learned with K-means. The embeddings are computed with or without position encoding (PE).

Method	Nb. supports	$16 \times 16 \times 256$	$16 \times 16 \times 1024$
Flatten		73.1	76.1
Mean pooling		64.9	73.4
Gaussian pooling (Mairal, 2016)		77.5	82.0
OTK	9	75.6	79.3
OTK (with PE)	9	78.0	82.2
OTK	64	77.9	80.1
OTK (with PE)	64	81.4	83.2
OTK	144	78.4	80.7
OTK (with PE)	144	81.8	83.4

- 256 filters at first and 1024 at second layer, which yields a $16 \times 16 \times 1024$ representation for each image.

We feed to OTKs the output features of a CKN model, which is already a kernel embedding. κ in OTKs will therefore be a linear kernel. The OTK embedding is learned with one reference using K-means method described in Section 5.4 and compared to several classical pooling baselines, including the original CKN's Gaussian pooling with pooling size equal to 6. The hyperparameters are entropic regularization ε and bandwidth for position encoding σ_{pos}. Their search grids are shown in Table 5.E.2. The results are shown in Table 5.E.3. We notice that the position encoding is crucial to this task, and substantially improves the performance of its counterpart without it.

5.E.3. Protein Fold Recognition

Models setting and hyperparameters. We consider here the one-layer models followed by a global mean pooling for the baseline methods CKN (Chen et al., 2019a) and RKN (Chen et al., 2019b). We build our OTK on top of the one-layer CKN model, where κ can be seen as a Gaussian kernel on the k-mers in sequences. The only difference between our model and CKN is thus the pooling operation, which is given by our embedding introduced in Section 5.4. The bandwidth parameter of the Gaussian kernel κ on k-mers is fixed to 0.6 for unsupervised models and 0.5 for supervised models, the same as used in CKN which were selected by the accuracy on the validation set. The filter size k is fixed to 10 and different numbers of anchor points in Nyström for κ are considered in the experiments. The other hyperparameters for OTKs are the entropic regularization parameter ε, the number of supports in a reference p, the number of references q, the number of iterations for Sinkhorn's algorithm and the regularization

Table 5.E.4.: Hyperparameter search grid for SCOP 1.75

Hyperparameter	Search range
ε for Sinkhorn	$[1.0; 0.5; 0.1; 0.05; 0.01]$
λ for classifier (unsupervised OTKs)	$1/2^{\text{range}(5,20)}$
λ for classifier (supervised OTKs)	[1e-6;1e-5;1e-4;1e-3]

Table 5.E.5.: Classification accuracy (top 1/5/10) results of unsupervised OTKs for SCOP 1.75. We show the results for different combinations of (number of references q × number of supports p). The reference measures $\mathbf{z}$ are learned with either K-means or Wasserstein barycenter for updating centroids.

Nb. filters	Method	q	Embedding size ($q \times p$)			
			10	50	100	200
128	K-means	1	76.5/91.5/94.4	77.5/91.7/94.5	79.4/92.4/94.9	78.7/92.1/95.1
		5	72.8/89.9/93.7	77.8/91.7/94.6	78.6/91.9/94.6	78.1/92.1/94.7
		10	62.7/85.8/91.1	76.5/91.0/94.2	78.1/92.2/94.9	78.6/92.2/94.7
	Wass. bary.	1	64.0/85.9/91.5	71.6/88.9/93.2	77.2/91.4/94.2	77.5/91.9/94.8
		5	70.5/89.1/93.0	76.6/91.3/94.4	78.4/91.7/94.3	77.1/91.9/94.7
		10	63.0/85.7/91.0	75.9/91.4/94.3	77.5/91.9/94.6	77.7/92.0/94.7
1024	K-means	1	84.4/95.0/96.6	84.6/95.0/97.0	85.7/95.3/96.7	85.4/95.2/96.7
		5	81.1/94.0/96.2	84.9/94.8/96.8	84.7/94.4/96.7	85.2/95.0/96.7
		10	79.8/93.5/95.9	83.1/94.6/96.6	84.4/94.7/96.7	84.8/94.9/96.7

parameter λ in the linear classifier. The search grid for ε and λ is shown in Table 5.E.4 and they are selected by the accuracy on validation set. ε plays an important role in the performance and is observed to be stable for the same dataset. For this dataset, it is selected to be 0.5 for all the unsupervised and supervised models. The effect of other hyperparameters will be discussed below.

Learning unsupervised OTKs. The kernel embedding φ, which is infinite dimensional for the Gaussian kernel, is approximated with the Nyström method using K-means on 300000 k-mers extracted from the same training set as in Chen et al. (2019b). The reference measures are learned by using either K-means or Wasserstein to update centroids in 2-Wasserstein K-means on 3000 subsampled sequences for RAM-saving reason. We evaluate OTKs on top of features extracted from CKNs of different dimensions, representing the number of anchor points used to approximate κ. The number of iterations for Sinkhorn is fixed to 100 to ensure the convergence. The results for different combinations of q and p are provided in Table 5.E.5. Increasing the number of supports p can improve the performance and then saturate it when p is too large. On the other hand, increasing the number of references while keeping the embedding dimension (*i.e.* $p \times q$) constant is not significantly helpful in this unsupervised setting. We also notice that Wasserstein Barycenter for learning the references does not outperform K-means, while the latter is faster in terms of computation.

Table 5.E.6.: Classification accuracy (top 1/5/10) of supervised models for SCOP 1.75. The accuracies obtained by averaging 10 different runs. We show the results of using either one reference with 50 supports or 5 references with 10 supports. Here DeepSF is a 10-layer CNN model.

Method	Top 1/5/10 accuracy on SCOP 2.06	
PSI-BLAST (Hou et al., 2018)	84.53/86.48/87.34	
DeepSF (Hou et al., 2018)	73.00/90.25/94.51	
Number of filters	128	512
CKN (Chen et al., 2019a)	76.30±0.70/92.17±0.16/95.27±0.17	84.11±0.11/94.29±0.20/96.36±0.13
RKN (Chen et al., 2019b)	77.82±0.35/92.89±0.19/95.51±0.20	85.29±0.27/94.95±0.15/96.54±0.12
Ours		
OTK ($\Phi_{\mathbf{z}}$ 1×50)	82.83±0.41/93.89±0.33/96.23±0.12	88.40±0.22/95.76±0.13/97.10±0.15
OTK ($\Phi_{\mathbf{z}}$ 5×10)	**84.68±0.50/94.68±0.29/96.49±0.18**	**88.66±0.25/95.90±0.15/97.33±0.14**

Learning supervised OTKs. The supervised OTKs are initialized with the unsupervised method and then trained in an alternating fashion which was also used for CKN: we use an Adam algorithm to update anchor points in Nyström and reference measures $\mathbf{z}$, and the L-BFGS algorithm to optimize the classifier. The learning rate for Adam is initialized with 0.01 and halved as long as there is no decrease of the validation loss for 5 successive epochs. In practice, we notice that using a small number of Sinkhorn iterations can achieve similar performance to a large number of iteration, while being much faster to compute. We thus fix it to 10 throughout the experiments. The accuracy results are obtained by averaging on 10 runs with different seeds following the setting in Chen et al. (2019b). The results are shown in Table 5.E.6 with error bars. The effect of the number of supports q is similar to the unsupervised case, while increasing the number of references can indeed improve performance.

5.E.4. Detection of Chromatin Profiles

Model architecture and hyperparameters. For the above reason and fair comparison, we use here the supervised OTK as a module in Deep NNs. The architecture of our model is shown in Table 5.E.7. We use an Adam optimizer with initial learning rate equal to 0.01 and halved at epoch 1, 4, 8 for 15 epochs in total. The number of iterations for Sinkhorn is fixed to 30. The whole training process takes about 30 hours on a single GTX2080TI GPU. The dropout rate is selected to be 0.4 from the grid $[0.1; 0.2; 0.3; 0.4; 0.5]$ and the weight decay is 1e-06, the same as Zhou and Troyanskaya (2015). The σ_{pos} for position encoding is selected to be 0.1, by the validation accuracy on the grid $[0.05; 0.1; 0.2; 0.3; 0.4; 0.5]$. The checkpoint with the best validation accuracy is used to evaluate on the test set. Area under ROC (auROC) and area under precision curve (auPRC), averaged over 919 chromatin profiles, are used to measure the performance. The hidden size d is chosen to be either 1024 or 1536.

Results and importance of position encoding. We compare our model to the state-of-the-art CNN model DeepSEA (Zhou and Troyanskaya, 2015) with 3 convolutional layers. Our model outperforms DeepSEA, while requiring fewer layers. The positional information is known to be important in this task. To show the efficacy of our position encod-

Table 5.E.7.: Model architecture for DeepSEA dataset.

Model architecture
Conv1d(in channels=4, out channels=d, kernel size=16) + ReLU
OTKLayer(in channels=d, supports=64, references=1, $\varepsilon = 1.0$, PE=True, $\sigma_{\text{pos}} = 0.1$)
Linear(in channels=d, out channels=d) + ReLU
Dropout(0.4)
Linear(in channels=$d \times 64$, out channels=919) + ReLU
Linear(in channels=919, out channels=919)

Table 5.E.8.: Results for prediction of chromatin profiles on the DeepSEA dataset. The metrics are area under ROC (auROC) and area under PR curve (auPRC), averaged over 919 chromatin profiles. The accuracies are averaged from 10 different runs. Armed with the positional encoding (PE) described in Section 5.4, the OTK outperforms the state-of-the-art model and an OTK with the PE proposed in Vaswani et al. (2017).

Method	DeepSEA Zhou and Troyanskaya (2015)	OTK	OTK ($d = 1024$)	OTK ($d = 1536$)
Position encoding	-	Sinusoidal 200	Ours	Ours
auROC	0.933	0.917	0.935	**0.936**
auPRC	0.342	0.311	0.354	**0.360**

ing, we compare it to the sinusoidal encoding used in the original transformer (Vaswani et al., 2017). We observe that our encoding with properly tuned σ_{pos} requires fewer layers, while being interpretable from a kernel point of view. We also find that larger hidden size d performs better, as shown in Table 5.E.8. ROC and PR curves for all the chromatin profiles and stratified by transcription factors, DNase I-hypersensitive sites and histone-marks can also be found in Figure 5.E.1.

5.E.5. SST-2

Dataset description. The data set contains 67349 training samples and 872 validation samples and can be found at `https://gluebenchmark.com/tasks`. The test set contains 1821 samples for which the predictions need to be submitted on the GLUE leaderboard, with limited number of submissions. As a consequence, our training and validation set are extracted from the original training set (80% of the original training set is used for our training set and the remaining 20% is used for our validation set), and we report accuracies on the standard validation set, used as a test set. The reviews are padded with zeros when their length is shorter than the chosen sequence length (we choose 30 and 66, the latter being the maximum review length in the data set) and the BERT implementation requires to add special tokens [CLS] and [SEP] at the beginning and the end of each review.

Model architecture and hyperparameters. In most transformers such as BERT, the embedding associated to the token [CLS] is used for classification and can be seen in some sense as an embedding of the review adapted to the task. The features we used are the word features provided by the BERT base-uncased version, available at `https:`

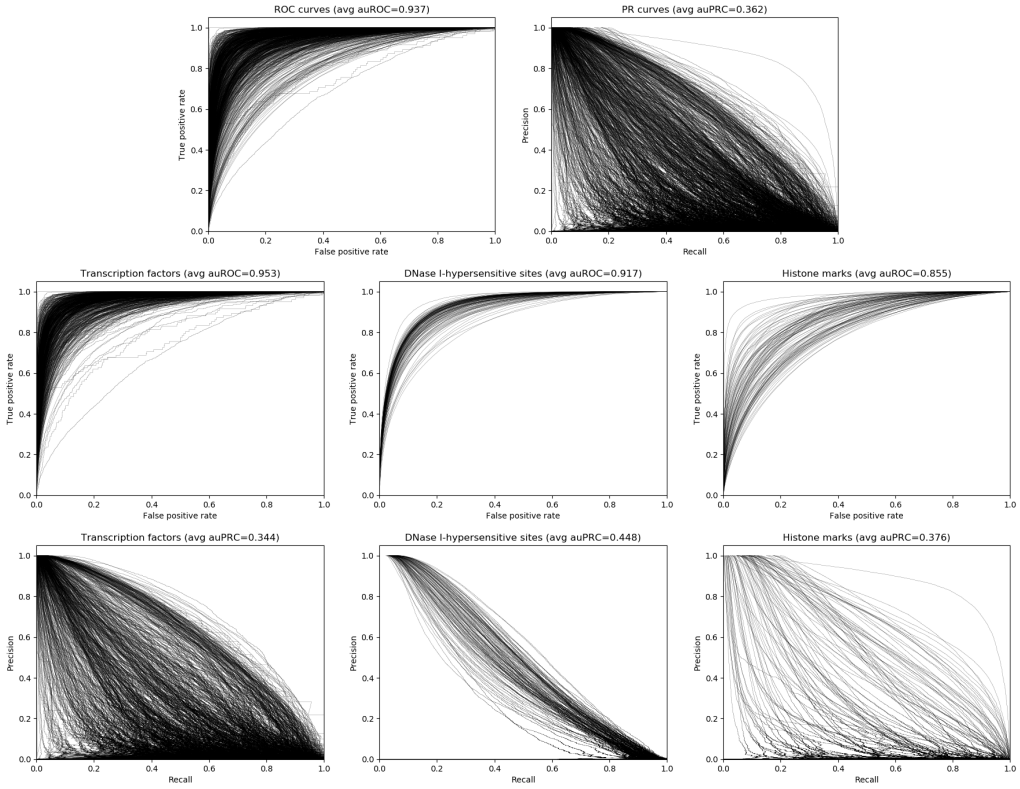


Figure 5.E.1.: ROC and PR curves for all the chromatin profiles (first row) and stratified by transcription factors (left column), DNase I-hypersensitive sites (middle column) and histone-marks (right column). The profiles with positive samples fewer than 50 on the test set are not taken into account.

Table 5.E.9.: Accuracies on standard validation set for SST-2 with unsupervised OTK features depending on the number of references and supports. The references were computed using K-means on samples for multiple references and K-means on patches for multiple supports. The size of the input BERT features is (length × dimension). The accuracies are averaged from 10 different runs.

BERT Input Feature Size	(30 × 768)		(66 × 768)	
Features	Pre-trained	Fine-tuned	Pre-trained	Fine-tuned
[CLS]	84.6±0.3	90.3±0.1	86.0±0.2	**92.8±0.1**
Flatten	84.9±0.4	**91.0±0.1**	85.2±0.3	92.5±0.1
Mean pooling	85.3±0.3	90.8±0.1	85.4±0.3	92.6±0.2
OTK (1 × 3)	85.5±0.1	90.9±0.1	86.5±0.1	92.6±0.1
OTK (1 × 10)	85.1±0.4	90.9±0.1	85.9±0.3	92.6±0.1
OTK (1 × 30)	86.3±0.3	90.8±0.1	86.6±0.5	92.6±0.1
OTK (1 × 100)	85.7±0.7	90.9±0.1	86.6±0.1	92.7±0.1
OTK (1 × 300)	**86.8±0.3**	90.9±0.1	**87.2±0.1**	92.7±0.1

//huggingface.co/transformers/pretrained_models.html. For this version, the dimension of the word features is 768. In the unsupervised case, the word embeddings of the reviews are kept as is, *i.e* we do not embed it using a Gaussian kernel. In this setting, the OTK linearly recombines the word features based on the transport plan. The resulting features are used to train a large-scale linear classifier using the Cyanure library (Mairal, 2019). In the supervised case, the OTK uses a Gaussian Nyström embedding with varying number of filters before the pooling layer, and the parameters of the two layers, $\mathbf{w}$ and $\mathbf{z}$, are optimized end-to-end. In this case, we have to tune the bandwidth of the Gaussian kernel as well as the learning rate. The classifier is here a fully-connected layer. In both case, we tune the entropic regularization parameter of optimal transport and the regularization parameter (or weight decay) of the classifier so as to get the best accuracy on the standard validation set, which is our test set. The parameters in the search grid are summed up in Table 5.E.10. The best entropic regularization and Gaussian kernel bandwidth are typically ans respectively 3.0 and 0.5 for this data set. In BERT models, the positional information is integrated in the initial word embeddings. As a consequence, we do not use our own positional encoding. The supervised training process takes between half an hour for smaller models (typically 128 filters in $\mathbf{w}$ and 3 supports in $\mathbf{z}$) and a few hours for larger models (256 filters and 100 supports) on a single GTX2080TI GPU.

Results and discussion. As explained in Section 5.5, our unsupervised OTK improves the BERT pre-trained features while still using a simple linear model as shown in Table 5.E.9, and its supervised counterpart enables to get even closer to the state-of-the art (for the BERT base-uncased model) accuracy, which is usually obtained after fine-tuning of the BERT model on the whole data set. This can be seen in Table 5.E.11.

Table 5.E.10.: Hyperparameter search grid for SST-2.

Hyperparameter	Search range
Entropic regularization ε	$[3.0; 1.0; 0.5]$
λ for classifier (unsupervised OTKs)	$10^{\text{range}(-10,1)}$
λ for classifier (supervised OTKs)	[1e-4;1e-3;1e-2]
Gaussian kernel bandwidth (supervised OTKs)	$[0.5; 1.0; 1.5]$
Learning rate (supervised OTKs)	$[0.1; 0.01; 0.001]$

Table 5.E.11.: Accuracies on standard validation set for SST-2 with supervised OTK features from pre-trained BERT (30×768) depending on the number of supports in the reference. The accuracies are averaged from 10 different runs, and 30 Sinkhorn iterations were used.

Number of supports p		3	10	30
Nyström filters	128	87.59	87.59	87.53
	256	87.45	87.44	87.50
	512	87.27	87.16	87.29

6

Conclusion

In this thesis, we developed a general kernel framework for handling structured data, notably sequence- and graph-structured data, with several applications in bioinformatics. On the one hand, our framework provides tools for efficiently learning representations without supervision as traditional kernels. On the other hand, it also enables supervised representation learning like deep neural networks, which significantly bridges the gap between kernel methods and deep learning.

6.1. Summary of Contributions

Here, we summarize the contributions presented in the thesis.

Modeling biological sequences with convolutional and recurrent kernel networks. Our first and second contributions investigate data-efficient models for biological sequences, by leveraging various prior knowledge used in classical string kernels. In Chapter 2, we introduce a new convolutional kernel named CKN-seq to tackle the motif discovery problem, by assuming genetic determinants (sequence motif) to be contiguous. Our method allows efficiently predicting from relatively short sequences while providing simple interpretation, through the lens of kernel approximation methods. It is shown to be effective on transcription factor binding prediction and protein homology recognition tasks. In the large-data setting, it substantially reduces the gap between classical string kernels and deep convolutional networks. On the other hand, it outperforms convolutional networks in the small-data regime.

In Chapter 3, we present a natural extension of the above work to modeling sequence gaps, based on substring kernels, to deal with protein homology recognition. In this work, the above contiguous assumption on motifs is relaxed to allow gaps, which is shown to be a helpful prior for this task especially for detecting remote homologies. Furthermore, our model can be viewed as a new type of recurrent neural networks (RNNs), which uncovers links between many existing deep models and kernel methods. Consequently, it opens the door to better regularization and architecture design of RNNs.

Modeling graph-structured data with convolutional kernel networks. The contribution of Chapter 4 consists in providing a general view of many existing graph kernels based on substructure counting and introducing a new multilayer kernel relying on fixed-length paths in the graphs. Rather than the neighbor features aggregation used in graph neural networks, our approach relies on aggregating path features, which makes it more expressive. By leveraging kernel approximation techniques, the resulting graph representations of our approach can be learned without supervision, or in a task-driven way as graph neural networks. Moreover, controlling the length of paths in our model allows

compromising computational complexity and expressiveness. Our work gives a novel view to designing more expressive kernels or deep networks for graph-structured data.

Feature aggregation in RKHSs. Our last contribution presented in Chapter 5 addresses the problem of feature aggregation for long sequences or more generally large sets of features with positional information that exhibit long-range dependencies. To this end, we introduce two valid kernel embeddings for feature aggregation in RKHSs. While the first one enables valid max pooling in RKHSs, the second one consists of a parametrized kernel embedding based on Wasserstein distance, where the parameters can be adaptively learned in both supervised and unsupervised fashions. Empirically, our second embedding combined with the above CKN-seq brings further improvement on protein homology recognition. We also provide an implementation of our embedding that can be used as a module in deep networks, which is shown to be very effective in the detection of chromatin profiles for genetic sequences.

6.2. Future Research and Perspectives

Based on the contributions of the thesis, several questions and research directions arise and would be interesting to investigate in the future. The goal of my research is to develop a unifying framework for analyzing and interpreting data with arbitrary structure. This would require several basic ingredients: computationally efficient models for representing structured data, subsequent models for performing analysis and prediction from the representations, and interpretation tools that give explanations to the prediction. While in deep learning the two first aspects are generally considered and trained jointly, I would like to separate them as they play different roles in a learning model from a kernel perspective. So far, the work presented in the thesis is mostly focused on the two first aspects, I think all of these aspects are equally important for real-world applications and advancing in one aspect would be complementary to the other aspects.

Efficient models for representing structured data. Regarding the first aspect on efficiently representing structured data, I think there is still room for improvement on representing huge-dimensional data like genome-wide sequences. Since the learning models considered in Chapter 2, 3 and 5 were designed to handle relatively short sequences, they may not be adequate to directly work with genome-scale data where a single phenotype (*e.g.*, antimicrobial resistance) can be associated to the entire genome of very large size. For instance, the genome of a bacteria ranges in size from 100 kbp to 10 Mbp. Within such huge-scale genome, yet only few proportions of the sequences are associated to the phenotype of interest. Predicting these phenotypes from sequences is thus achieved by introducing intermediate tasks which are tightly related to the phenotype while providing precise annotations at sub-region or even base-pair level. However, obtaining intermediate annotations at such high resolution could be very expensive and directly working with the original sparse annotations is also infeasible. In this respect, efficient statistical selection methods that can be used to highlight the most related genomic areas are promising approaches, such as *causal feature selection* (Guyon et al., 2007; Aliferis et al., 2010) or *k-mer based selection* (Saeys et al., 2007; Aghazadeh et al., 2018). Once the areas most relevant to the phenotype are marked, applying the models

studied in this thesis to these areas may result in more precise prediction and more reliable interpretation. More generally, these approaches can be considered as the possible preprocessing methods before applying deep neural networks to sequence prediction tasks.

Another direction to improve the current representations is to improve unsupervised models to leverage large amounts of unannotated data. While we used in the thesis a greedy approach for unsupervised feature learning which approximates a hierarchical kernel layer-by-layer using the Nyström approximation method, some recent work such as Shankar et al. (2020) has shown that computing the *exact* hierarchical kernel can pronouncedly improve performance despite the requirement of lots of computing resources. By consequence, more accurate methods are needed to better approximate such hierarchical kernels with more compact representations. A possible direction is to directly approximate the entire hierarchical kernel in the final feature space, rather than in a layer-by-layer manner, using clustering and back-propagation techniques (Caron et al., 2018, 2020). Once such unsupervised representations are pre-trained with large amounts of unannotated data, they can be easily transferred to some downstream tasks where only few annotations are provided for training.

In order to learn representations for data with complex structure, we presented in Chapter 5 an optimal transport based feature aggregation that is able to adaptively aggregate local features exhibiting long-range dependencies. To make full use of optimal transport to deal with large-scale datasets, one needs efficient algorithms to solve a huge number of optimal transport problems of potentially large-scale discrete distributions. To alleviate the computational burden, one direction is to perform low-rank approximation of Wasserstein distance matrices. In Chapter 5, we proposed a simple approach to tackle this problem, though the main focus was to develop models that efficiently capture long-range dependencies in biological sequences. A more thorough study on the approximation error of our approach is crucial to understand and improve the method. Besides, other aggregation approaches from kernel literature, such as Fisher kernels (Perronnin and Dance, 2007; Sánchez et al., 2013), are also of interest to study and build deep models.

Subsequent models for prediction and analysis. Regarding the second aspect on subsequent models for prediction or analysis, rather than the discriminative models considered in most of the current work for supervised learning tasks, *generative models* can also be of high interest for structured data modeling since they can perform several tasks that discriminative models cannot tackle. First, generative models can be learned with unannotated data, which is useful for the tasks where little labeled data is available. They provide several ways to incorporate information of the unlabeled data to boost the performance of prediction. Applications include protein function prediction where functional annotation of proteins is very hard to be experimentally verified while the number of protein sequences is enormous. Second, generative models may be useful for learning distributions of structured data and therefore for designing new sequences or graphs. Examples include *DNA or protein sequence design* (Gupta and Zou, 2019; Wang et al., 2020), *protein structural modeling* (Gao et al., 2020), which offer the possibility of solving many real problems in biomedicine and material science.

More interpretable learning models. Regarding the last aspect on interpretability, we proposed a few interpretation techniques to visualize the sequence or graph motifs for trained models that could be relevant to the outcome in Chapter 2 and 4. However, the statistical significance of the association of these motifs with the outcome is harder to define. Designing proper statistical tests to measure the contributions of different motifs is crucial for understanding deep learning and its adoption in real-world applications. In addition to interpreting existing deep models, there still exists a performance gap between the latest deep residual networks (ResNets) and deep kernels. Some recent works have even shown that ResNet can perform a kind of feature selection at different scales while classical kernel methods cannot do (Allen-Zhu and Li, 2019). Fortunately, more advanced learning techniques involving multiple kernels namely multiple kernel learning allow to perform selection of kernels. Therefore, incorporating such approaches into hierarchical kernels may be useful to reduce the gap to the deep ResNets and eventually explain these models.

Finally, I would like to conclude this thesis by drawing attention to the importance of exploring both fields of kernel methods and deep learning. Borrowing ideas from one field to the other would be crucial to understand the inductive bias of existing models and construct appropriate models for various structured data. Classical tools and concepts from kernel methods could be used to understand and design more recent deep networks, while techniques from deep learning would also be useful to enable large scale learning for classical kernels. Bridging performance gap between state-of-the-art deep models (*e.g.* residual networks) and kernel methods would also pave the way for explainable and interpretable deep learning.

A

Software

Along with the contributions described above, several software packages developed during the thesis are available online:

- CKN-seq (https://gitlab.inria.fr/dchen/CKN-seq)
  CKN-seq is a Pytorch implementation to model biological sequences (DNA, protein etc.) with convolutional kernel networks developed in Chapter 2. It provides prediction tool as well as interpretation tools to visualize sequence motifs. A more user-friendly interface was also implemented by François Gindraud under "fgindraud" branch.

- CKN-Pytorch-image (https://github.com/claying/CKN-Pytorch-image)
  CKN-Pytorch-image is a Pytorch implementation to perform image classification with convolutional kernel networks. It provides both supervised and unsupervised representations for images. This implementation is based on the work of Mairal (2016).

- RKN (https://github.com/claying/RKN)
  RKN is a Pytorch implementation to model biological sequences (DNA, protein etc.) with recurrent kernel networks presented in Chapter 3. It is useful for prediction tasks where there exist gapped motifs. The code is implemented with a CUDA-Pytorch interface to have comparable speed as Pytorch's recurrent neural networks implementation.

- GCKN (https://github.com/claying/GCKN)
  GCKN is a Pytorch implementation to model graphs with node attributes using graph convolutional kernel networks presented in Chapter 4. It provides both supervised and unsupervised graph representations. An interpretation tool is also provided to visualize the most important subgraph for a given graph and a trained model. Our implementation shows state-of-the-art performances in a couple of graph classification benchmarks.

- OTK (https://github.com/claying/OTK)
  Optimal transport kernel (OTK), presented in Chapter 5, is a Pytorch implementation for feature aggregation. It allows performing adaptive pooling (attention + pooling) for arbitrary structured objects. Principally, it can be useful to model any data represented as sets of features (sequences, images, graphs etc.). In this implementation, it can be used as a module in neural networks, or alone as a kernel method. It outperforms many existing pooling operations such as average and max pooling. And it is shown to be effective for modeling long biological sequences with potentially long-range dependencies.

- Scikit-Learn interface of Cyanure (http://thoth.inrialpes.fr/people/mairal/cyanure/welcome.html)
  Cyanure is an open-source C++ software package with a Python 3 interface. The goal of Cyanure is to provide state-of-the-art solvers for learning linear models, based on stochastic variance-reduced stochastic optimization with acceleration mechanisms and Quasi-Newton principles. The core code was implemented by Julien Mairal and its Scikit-Learn interface was implemented by me.

Bibliography

A. Aghazadeh, R. Spring, D. Lejeune, G. Dasarathy, A. Shrivastava, et al. Mission: Ultra large-scale feature selection using count-sketches. In *International Conference on Machine Learning (ICML)*, 2018.

B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. Molecular biology of the cell, sixth edition. *Garland Science*, 2014.

C. F. Aliferis, A. Statnikov, I. Tsamardinos, S. Mani, and X. D. Koutsoukos. Local causal and markov blanket induction for causal discovery and feature selection for classification part i: algorithms and empirical evaluation. *Journal of Machine Learning Research (JMLR)*, 11(1), 2010.

B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature Biotechnology*, 33(8):831–838, 2015.

Z. Allen-Zhu and Y. Li. What can resnet learn efficiently, going beyond kernels? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning (ICML)*, 2019.

A. Andreeva, D. Howorth, C. Chothia, E. Kulesha, and A. G. Murzin. Scop2 prototype: a new approach to protein structure mining. *Nucleic Acids Research*, 42(D1):D310–D314, 2014.

C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle. Deep learning for computational biology. *Molecular Systems Biology*, 12(7):878, 2016.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

S. Arora, S. S. Du, W. Hu, Z. Li, R. R. Salakhutdinov, and R. Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

E. Asgari and M. R. Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS One*, 10(11):e0141287, 2015.

A. Auton, G. R. Abecasis, D. M. Altshuler, R. M. Durbin, G. R. Abecasis, D. R. Bentley, A. Chakravarti, A. G. Clark, P. Donnelly, E. E. Eichler, P. Flicek, S. B. Gabriel, R. A. Gibbs, E. D. Green, M. E. Hurles, B. M. Knoppers, J. O. Korbel, E. S. Lander, C. Lee, H. Lehrach, E. R. Mardis, G. T. Marth, G. A. McVean, D. A. Nickerson, J. P. Schmidt, S. T. Sherry, J. Wang, R. K. Wilson, R. A. Gibbs, E. Boerwinkle, H. Doddapaneni, Y. Han, V. Korchina, C. Kovar, S. Lee, D. Muzny, J. G. Reid, Y. Zhu, J. Wang, Y. Chang, Q. Feng, X. Fang, X. Guo, M. Jian, H. Jiang, X. Jin, T. Lan, G. Li, J. Li, Y. Li, S. Liu, X. Liu, Y. Lu, X. Ma, M. Tang, B. Wang, G. Wang, H. Wu, R. Wu, X. Xu, Y. Yin, D. Zhang, W. Zhang, J. Zhao, M. Zhao, X. Zheng, E. S. Lander,

D. M. Altshuler, S. B. Gabriel, N. Gupta, N. Gharani, L. H. Toji, N. P. Gerry, A. M. Resch, P. Flicek, J. Barker, L. Clarke, L. Gil, S. E. Hunt, G. Kelman, E. Kulesha, R. Leinonen, W. M. McLaren, R. Radhakrishnan, A. Roa, D. Smirnov, R. E. Smith, I. Streeter, A. Thormann, I. Toneva, B. Vaughan, X. Zheng-Bradley, D. R. Bentley, R. Grocock, S. Humphray, T. James, Z. Kingsbury, H. Lehrach, R. Sudbrak, M. W. Albrecht, V. S. Amstislavskiy, T. A. Borodina, M. Lienhard, F. Mertes, M. Sultan, B. Timmermann, M.-L. Yaspo, E. R. Mardis, R. K. Wilson, L. Fulton, R. Fulton, S. T. Sherry, V. Ananiev, Z. Belaia, D. Beloslyudtsev, N. Bouk, C. Chen, D. Church, R. Cohen, C. Cook, J. Garner, T. Hefferon, M. Kimelman, C. Liu, J. Lopez, P. Meric, C. O'Sullivan, Y. Ostapchuk, L. Phan, S. Ponomarov, V. Schneider, E. Shekhtman, K. Sirotkin, D. Slotta, H. Zhang, G. A. McVean, R. M. Durbin, S. Balasubramaniam, J. Burton, P. Danecek, T. M. Keane, A. Kolb-Kokocinski, S. McCarthy, J. Stalker, M. Quail, J. P. Schmidt, C. J. Davies, J. Gollub, T. Webster, B. Wong, Y. Zhan, A. Auton, C. L. Campbell, Y. Kong, A. Marcketta, R. A. Gibbs, F. Yu, L. Antunes, M. Bainbridge, D. Muzny, A. Sabo, Z. Huang, J. Wang, L. J. M. Coin, L. Fang, X. Guo, X. Jin, G. Li, Q. Li, Y. Li, Z. Li, H. Lin, B. Liu, R. Luo, H. Shao, Y. Xie, C. Ye, C. Yu, F. Zhang, H. Zheng, H. Zhu, C. Alkan, E. Dal, F. Kahveci, G. T. Marth, E. P. Garrison, D. Kural, W.-P. Lee, W. Fung Leong, M. Stromberg, A. N. Ward, J. Wu, M. Zhang, M. J. Daly, M. A. DePristo, R. E. Handsaker, D. M. Altshuler, E. Banks, G. Bhatia, G. del Angel, S. B. Gabriel, G. Genovese, N. Gupta, H. Li, S. Kashin, E. S. Lander, S. A. McCarroll, J. C. Nemesh, R. E. Poplin, S. C. Yoon, J. Lihm, V. Makarov, A. G. Clark, S. Gottipati, A. Keinan, J. L. Rodriguez-Flores, J. O. Korbel, T. Rausch, M. H. Fritz, A. M. Stütz, P. Flicek, K. Beal, L. Clarke, A. Datta, J. Herrero, W. M. McLaren, G. R. S. Ritchie, R. E. Smith, D. Zerbino, X. Zheng-Bradley, P. C. Sabeti, I. Shlyakhter, S. F. Schaffner, J. Vitti, D. N. Cooper, E. V. Ball, P. D. Stenson, D. R. Bentley, B. Barnes, M. Bauer, R. Keira Cheetham, A. Cox, M. Eberle, S. Humphray, S. Kahn, and . Murray. A global reference for human genetic variation. *Nature*, 526 (7571):68–74, 2015.

F. Bach. Graph kernels between point clouds. In *International Conference on Machine Learning (ICML)*, 2008.

F. Bach. On the equivalence between kernel quadrature rules and random feature expansions. *Journal of Machine Learning Research (JMLR)*, 18(1):714–751, 2017.

F. R. Bach, G. R. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *International Conference on Machine Learning (ICML)*, 2004.

D. Bahdanau, K. Cho, and J. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

G. Baid. An attention-based model for transcription factor binding site prediction. 2018.

A. Barla, F. Odone, and A. Verri. Histogram intersection kernel for image classification. In *International Conference on Image Processing*, 2003.

A. Ben-Hur and D. Brutlag. Remote homology detection: a motif based approach. *Bioinformatics*, 19(suppl_1):i26–i33, 2003.

A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support vector machines and kernels for computational biology. *PLoS Computational Biology*, 4(10), 2008.

R. Bhatia, T. Jain, and Y. Lim. On the bures-wasserstein distance between positive definite matrices. *Expositiones Mathematicae*, 2018.

A. Bietti and J. Mairal. Invariance and stability of deep convolutional representations. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6210–6220, 2017.

A. Bietti and J. Mairal. Group invariance, stability to deformations, and complexity of deep convolutional representations. *Journal of Machine Learning Research (JMLR)*, 20(1):876–924, 2019.

A. Bietti, G. Mialon, D. Chen, and J. Mairal. A kernel perspective for regularizing deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

M. Bishop and E. A. Thompson. Maximum likelihood alignment of dna sequences. *Journal of Molecular Biology*, 190(2):159–165, 1986.

L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

C. Bock, M. Togninalli, E. Ghisu, T. Gumbsch, B. Rieck, and K. Borgwardt. A wasserstein subsequence kernel for time series. In *International Conference on Data Mining (ICDM)*, 2019.

K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *International Conference on Data Mining (ICDM)*, 2005.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings in Computational Statistics (COMPSTAT)*. Springer, 2010.

S. Boughorbel, J. P. Tarel, and N. Boujemaa. The intermediate matching kernel for image local features. In *International Joint Conference on Neural Networks*, 2005.

C. J. Burges et al. Simplified support vector decision rules. In *International Conference on Machine Learning (ICML)*, 1996.

C. Cai and Y. Wang. A simple yet effective baseline for non-attributed graph classification. *arXiv preprint arXiv:1811.03508*, 2018.

M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *European Conference on Computer Vision (ECCV)*, 2018.

M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research (JMLR)*, 11(4), 2010.

D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. *Bioinformatics*, 35(18):3294–3302, 2019a.

D. Chen, L. Jacob, and J. Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b.

D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. In *Research in Computational Molecular Biology (RECOMB)*, 2019c.

D. Chen, L. Jacob, and J. Mairal. Convolutional kernel networks for graph-structured data. In *International Conference on Machine Learning (ICML)*, 2020.

L. Chen, G. Wang, C. Tao, D. Shen, P. Cheng, X. Zhang, W. Wang, Y. Zhang, and L. Carin. Improving textual network embedding with global attention via optimal transport. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019d.

Y. Chen, Y. Li, R. Narayan, A. Subramanian, and X. Xie. Gene expression inference with deep learning. *Bioinformatics*, 32(12):1832–1839, 2016.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Y. Cho and L. K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, 2017.

M. Collins and N. Duffy. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

T. C. P.-G. Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 10 2016. ISSN 1477-4054.

F. Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970.

M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

M. Cuturi and A. Doucet. Fast computation of wasserstein barycenters. In *International Conference on Machine Learning (ICML)*, 2013.

M. Cuturi and J.-P. Vert. The context-tree kernel for strings. *Neural Networks*, 18(8): 1111–1123, 2005.

A. Daniely, R. Frostig, and Y. Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances In Neural Information Processing Systems (NIPS)*, 2016.

A. Drouin, S. Giguère, M. Déraspe, M. Marchand, M. Tyers, V. G. Loo, A.-M. Bourgault, F. Laviolette, and J. Corbeil. Predictive computational phenotyping and biomarker discovery using reference-free genome comparisons. *BMC Genomics*, 17(1):754, 2016.

S. S. Du, K. Hou, R. R. Salakhutdinov, B. Poczos, R. Wang, and K. Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

I. Dunham, A. Kundaje, S. F. Aldred, P. J. Collins, C. A. Davis, F. Doyle, C. B. Epstein, S. Frietze, J. Harrow, R. Kaul, J. Khatun, B. R. Lajoie, S. G. Landt, B.-K. Lee, F. Pauli, K. R. Rosenbloom, P. Sabo, A. Safi, A. Sanyal, N. Shoresh, J. M. Simon, L. Song, N. D. Trinklein, R. C. Altshuler, E. Birney, J. B. Brown, C. Cheng, S. Djebali, X. Dong, I. Dunham, J. Ernst, T. S. Furey, M. Gerstein, B. Giardine, M. Greven, R. C. Hardison, R. S. Harris, J. Herrero, M. M. Hoffman, S. Iyer, M. Kellis, J. Khatun, P. Kheradpour, A. Kundaje, T. Lassmann, Q. Li, X. Lin, G. K. Marinov, A. Merkel, A. Mortazavi, S. C. J. Parker, T. E. Reddy, J. Rozowsky, F. Schlesinger, R. E. Thurman, J. Wang, L. D. Ward, T. W. Whitfield, S. P. Wilder, W. Wu, H. S. Xi, K. Y. Yip, J. Zhuang, B. E. Bernstein, E. Birney, I. Dunham, E. D. Green, C. Gunter, M. Snyder, M. J. Pazin, R. F. Lowdon, L. A. L. Dillon, L. B. Adams, C. J. Kelly, J. Zhang, J. R. Wexler, E. D. Green, P. J. Good, E. A. Feingold, B. E. Bernstein, E. Birney, G. E. Crawford, J. Dekker, L. Elnitski, P. J. Farnham, M. Gerstein, M. C. Giddings, T. R. Gingeras, E. D. Green, R. Guigó, R. C. Hardison, T. J. Hubbard, M. Kellis, W. J. Kent, J. D. Lieb, E. H. Margulies, R. M. Myers, M. Snyder, J. A. Stamatoyannopoulos, S. A. Tenenbaum, Z. Weng, K. P. White, B. Wold, J. Khatun, Y. Yu, J. Wrobel, B. A. Risk, H. P. Gunawardena, H. C. Kuiper, C. W. Maier, L. Xie, X. Chen, M. C. Giddings, B. E. Bernstein, C. B. Epstein, N. Shoresh, J. Ernst, P. Kheradpour, T. S. Mikkelsen, S. Gillespie, A. Goren, O. Ram, X. Zhang, L. Wang, R. Issner, M. J. Coyne, T. Durham, M. Ku, T. Truong, L. D. Ward, R. C. Altshuler, M. L. Eaton, M. Kellis, S. Djebali, C. A. Davis, A. Merkel, A. Dobin, T. Lassmann, A. Mortazavi, A. Tanzer, J. Lagarde, W. Lin, F. Schlesinger, C. Xue, G. K. Marinov, J. Khatun, B. A. Williams, C. Zaleski, J. Rozowsky, M. Röder, F. Kokocinski, R. F. Abdelhamid, T. Alioto, I. Antoshechkin, M. T. Baer, P. Batut, I. Bell, K. Bell, S. Chakrabortty, X. Chen, J. Chrast, J. Curado, T. Derrien, J. Drenkow, E. Dumais, J. Dumais, R. Duttagupta, M. Fastuca, K. Fejes-Toth, P. Ferreira, S. Foissac, M. J. Fullwood, H. Gao, D. Gonzalez, A. Gordon, H. P. Gunawardena, C. Howald, S. Jha, R. Johnson, P. Kapranov, B. King, C. Kingswood, G. Li, O. J. Luo, E. Park, J. B. Preall, K. Presaud, P. Ribeca, B. A. Risk, D. Robyr, X. Ruan, M. Sammeth, K. S. Sandhu, L. Schaeffer, L.-H. See, A. Shahab, J. Skancke, A. M. Suzuki, H. Takahashi, H. Tilgner, D. Trout, N. Walters, H. Wang, J. Wrobel, Y. Yu, Y. Hayashizaki, J. Harrow, M. Gerstein, T. J. Hubbard, A. Reymond, S. E. Antonarakis, G. J. Hannon, M. C. Giddings, Y. Ruan, B. Wold, P. Carninci, R. Guigó, T. R. Gingeras, K. R. Rosenbloom, C. A. Sloan, K. Learned, V. S. Malladi, M. C. Wong, G. P. Barber, M. S. Cline, T. R. Dreszer, S. G. Heitner, D. Karolchik, W. J. Kent, V. M. Kirkup, L. R. Meyer, J. C. Long, M. Maddren, B. J. Raney, T. S. Furey, L. Song, L. L. Grasfeder, P. G. Giresi, and L... An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, 2012.

R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

G. Elgar and T. Vavouri. Tuning in to the signals: noncoding sequence conservation in vertebrate genomes. *Trends in Genetics*, 24(7):344–352, 2008.

A. Elmas, X. Wang, and J. M. Dresch. The folded k-spectrum kernel: A machine learning approach to detecting transcription factor binding sites with gapped nucleotide dependencies. *PloS One*, 12(10):e0185570, 2017.

A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research (JMLR)*, 2(Dec):243–264, 2001.

L. Flagel, Y. Brandvain, and D. R. Schrider. The unreasonable effectiveness of convolutional neural networks in population genetic inference. *Molecular Biology and Evolution*, 36(2):220–238, 2019.

H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *International Conference on Machine Learning (ICML)*, 2005.

W. Gao, S. P. Mahajan, J. Sulam, and J. J. Gray. Deep learning in protein structural modeling and design. *Patterns*, page 100142, 2020.

A. Gardner, C. A. Duncan, J. Kanno, and R. R. Selmic. On the definiteness of earth mover's distance and its relation to set intersection. *Transactions on Cybernetics*, 48 (11):3184–3196, 2017.

A. Garriga-Alonso, C. E. Rasmussen, and L. Aitchison. Deep convolutional networks as shallow gaussian processes. In *International Conference on Learning Representations (ICLR)*, 2019.

T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pages 129–143. Springer, 2003.

A. Genevay, M. Cuturi, G. Peyré, and F. Bach. Stochastic optimization for large-scale optimal transport. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

M. Ghandi, D. Lee, M. Mohammad-Noori, and M. A. Beer. Enhanced regulatory sequence prediction using gapped k-mer features. *PLoS Computational Biology*, 10(7): e1003711, 2014.

M. B. Giles. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In *Advances in Automatic Differentiation*, pages 35–44. Springer, 2008.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, 2017.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016.

K. Grauman and T. Darrell. The pyramid match kernel: Efficient learning with sets of features. *Journal of Machine Learning Research (JMLR)*, 8(Apr):725–760, 2007.

A. Gupta and J. Zou. Feedback gan for dna optimizes protein functions. *Nature Machine Intelligence*, 1(2):105–111, 2019.

S. Gupta, J. A. Stamatoyannopoulos, T. L. Bailey, and W. S. Noble. Quantifying similarity between motifs. *Genome Biology*, 8(2):R24, 2007.

I. Guyon, C. Aliferis, and A. Elisseeff. Causal feature selection. *Computational Methods of Feature Selection*, pages 63–82, 2007.

W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

T. Håndstad, A. J. Hestnes, and P. Sætrom. Motif kernel generated by genetic programming improves remote homology and fold detection. *BMC bioinformatics*, 8(1): 23, 2007.

S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems (NIPS)*, 1989.

Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007.

D. Haussler. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

C. Helma, T. Cramer, S. Kramer, and L. De Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of chemical information and computer sciences*, 44(4):1402–1411, 2004.

S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

S. Hochreiter, M. Heusel, and K. Obermayer. Fast model-based protein homology detection without alignment. *Bioinformatics*, 23(14):1728–1736, 2007.

T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

J. Hou, B. Adhikari, and J. Cheng. Deepsf: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics*, 34(8):1295–1303, 2018.

S. Ivanov and E. Burnaev. Anonymous walk embeddings. In *International Conference on Machine Learning (ICML)*, pages 2186–2195, 2018.

E. J. Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*, 25, 01 2019.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2):95–114, 2000.

T. S. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies. In *International Conference on Intelligent Systems for Molecular Biology (ISMB)*, 1999.

A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

T. Jebara, R. Kondor, and A. Howard. Probability product kernels. *Journal of Machine Learning Research (JMLR)*, 5(Jul):819–844, 2004.

H. Jégou, M. Douze, and C. Schmid. On the burstiness of visual elements. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 34(9):1704–1716, 2011.

A. Jha, M. R. Gazzara, and Y. Barash. Integrative deep models for alternative splicing. *Bioinformatics*, 33(14):i274–i282, 2017.

F. D. Johansson and D. Dubhashi. Learning with similarity functions on graphs using matchings of geometric embeddings. In *International Conference on Knowledge Discovery and Data Mining*, 2015.

N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2014.

M. Karimzadeh and M. M. Hoffman. Virtual chip-seq: Predicting transcription factor binding by learning from the transcriptome. *bioRxiv*, 2018.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning (ICML)*, 2003.

S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30 (8):595–608, 2016.

D. R. Kelley, J. Snoek, and J. L. Rinn. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome research*, 26(7): 990–999, 2016.

D. R. Kelley, Y. A. Reshef, M. Bileschi, D. Belanger, C. Y. McLean, and J. Snoek. Sequential regulatory activity prediction across chromosomes with convolutional neural networks. *Genome research*, 28(5):739–750, 2018.

K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. Benchmark data sets for graph kernels, 2016. http://graphkernels.cs.tu-dortmund.de.

P. Kheradpour and M. Kellis. Systematic discovery and characterization of regulatory motifs in encode tf binding experiments. *Nucleic Acids Research*, 42(5):2976–2987, 2013.

G. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *Journal of mathematical analysis and applications*, 33(1):82–95, 1971.

T. Kin, K. Tsuda, and K. Asai. Marginalized kernels for rna sequence data analysis. *Genome Informatics*, 13:112–122, 2002.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. 2015.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

P. W. Koh, E. Pierson, and A. Kundaje. Denoising genome-wide histone chip-seq with convolutional neural networks. *Bioinformatics*, 33(14):i225–i233, 2017.

N. M. Kriege, P.-L. Giscard, and R. Wilson. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

N. M. Kriege, C. Morris, A. Rey, and C. Sohler. A property testing framework for the theoretical expressivity of graph kernels. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018.

N. M. Kriege, F. D. Johansson, and C. Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. *Journal of Bioinformatics and Computational Biology*, 3(03):527–550, 2005.

P. P. Kuksa, P.-H. Huang, and V. Pavlovic. Scalable algorithms for string kernels with inexact matching. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning (ICML)*, 2015.

J. Lanchantin, R. Singh, B. Wang, and Y. Qi. Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks. In *Pacific Symposium on Biocomputing*, 2017.

G. R. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research (JMLR)*, 5(Jan):27–72, 2004a.

G. R. Lanckriet, T. De Bie, N. Cristianini, M. I. Jordan, and W. S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20(16):2626–2635, 2004b.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations (ICLR)*, 2018.

T. Lei, W. Jin, R. Barzilay, and T. Jaakkola. Deriving neural architectures from sequence and graph kernels. In *International Conference on Machine Learning (ICML)*, 2017.

C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research (JMLR)*, 5(Nov), 2004.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Biocomputing*. World Scientific, 2001.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: a string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, 2002a.

C. Leslie, E. Eskin, J. Weston, and W. Noble. Mismatch String Kernels for SVM Protein Classification. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

C. S. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, 2002b.

C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.

L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships. *Journal of Computational Biology*, 10(6):857–868, 2003.

D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

F. Liu, X. Huang, Y. Chen, and J. A. Suykens. Random features for kernel approximation: A survey in algorithms, theory, and beyond. *arXiv preprint arXiv:2004.11154*, 2020.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research (JMLR)*, 2(Feb), 2002.

S. Lyu. Mercer kernels for object recognition with local features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

S. Lyu. Mercer kernels for object recognition with local features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Graph kernels for molecular structure- activity relationship analysis with support vector machines. *Journal of chemical information and modeling*, 45(4):939–951, 2005.

J. Mairal. End-to-end kernel learning with supervised convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

J. Mairal. Cyanure: An open-source toolbox for empirical risk minimization for python, C++, and soon more. *arXiv preprint arXiv:1912.08165*, 2019.

J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations (ICLR)*, 2018.

A. G. d. G. Matthews, J. Hron, M. Rowland, R. E. Turner, and Z. Ghahramani. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Q. Mérigot, A. Delalande, and F. Chazal. Quantitative stability of optimal transport maps and linearization of the 2-wasserstein space. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

S. Merity, N. S. Keskar, and R. Socher. Regularizing and optimizing lstm language models. In *International Conference on Learning Representations (ICLR)*, 2018.

G. Mialon*, D. Chen*, A. d'Aspremont, and J. Mairal. A trainable optimal transport embedding for feature aggregation. In *International Conference on Learning Representations (ICLR)*.

C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *International Conference on Data Mining*. IEEE, 2016.

C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, 2019.

A. Morrow, V. Shankar, D. Petersohn, A. Joseph, B. Recht, and N. Yosef. Convolutional kitchen sinks for transcription factor binding site prediction. *arXiv preprint arXiv:1706.00125*, 2017.

N. Murray and F. Perronnin. Generalized max pooling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–540, 1995.

N. Navarin, D. V. Tran, and A. Sperduti. Pre-training graph neural networks with kernels. *arXiv preprint arXiv:1811.06930*, 2018.

R. M. Neal. *Bayesian learning for neural networks*. Springer, 1996.

M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning (ICML)*, 2016.

G. Nikolentzos, P. Meladianos, A. J.-P. Tixier, K. Skianis, and M. Vazirgiannis. Kernel graph convolutional neural networks. In *International Conference on Artificial Neural Networks (ICANN)*, 2018.

R. Novak, L. Xiao, Y. Bahri, J. Lee, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, and J. Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations (ICLR)*, 2019.

F. Orsini, P. Frasconi, and L. De Raedt. Graph invariant kernels. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

P. Pavlidis, J. Weston, J. Cai, and W. S. Noble. Learning gene functional classifications from multiple data types. *Journal of Computational Biology*, 9(2):401–411, 2002.

F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Conference on computer vision and pattern recognition (CVPR)*, 2007.

G. Peyré, M. Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

D. Quang and X. Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *Nucleic Acids Research*, 44(11):e107–e107, 2016.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. In *arXiv preprint arXiv 1910.10683*, 2019.

A. Raganato, Y. Scherrer, and J. Tiedemann. Fixed encoder self-attention patterns in transformer-based machine translation. *arXiv preprint arXiv:2002.10260*, 2020.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

A. Rakotomamonjy, F. R. Bach, S. Canu, and Y. Grandvalet. Simplemkl. *Journal of Machine Learning Research (JMLR)*, 9(Nov), 2008.

P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens. Stand-alone self-attention in vision models. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *International Workshop on Mining Graphs, Trees and Sequences*, 2003.

H. Rangwala and G. Karypis. Profile-based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, 21(23):4239–4247, 2005.

A. Rives, S. Goyal, J. Meier, D. Guo, M. Ott, C. L. Zitnick, J. Ma, and R. Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. In *bioRxiv 622803*, 2019.

Y. Rubner, C. Tomasi, and L. J. Guibad. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

Y. Saeys, I. Inza, and P. Larranaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.

S. Saitoh. *Integral transforms, reproducing kernels and their applications*, volume 369. CRC Press, 1997.

J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision (IJCV)*, 105 (3):222–245, 2013.

I. J. Schoenberg. Positive definite functions on spheres. *Duke Mathematical Journal*, 9 (1):96–108, 03 1942.

B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT Press, 2001.

B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel methods in computational biology*. MIT Press, 2004.

J. Schur. Bemerkungen zur theorie der beschränkten bilinearformen mit unendlich vielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 1911(140):1–28, 1911.

M. Seeger. Covariance kernels from bayesian generative models. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

V. Shankar, A. Fang, W. Guo, S. Fridovich-Keil, L. Schmidt, J. Ragan-Kelley, and B. Recht. Neural kernels without tangents. In *International Conference on Machine Learning (ICML)*, 2020.

J. Shawe-Taylor, N. Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, 2009.

N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12 (9), 2011.

A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning (ICML)*, 2017a.

A. Shrikumar, P. Greenside, and A. Kundaje. Reverse-complement parameter sharing improves deep learning models for genomics. *bioRxiv*, 2017b.

M. Slattery, T. Zhou, L. Yang, A. C. D. Machado, R. Gordân, and R. Rohs. Absence of a simple code: how transcription factors read the genome. *Trends in Biochemical Sciences*, 39(9):381–399, 2014.

T. F. Smith, M. S. Waterman, et al. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

A. J. Smola and P. L. Bartlett. Sparse greedy gaussian process regression. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1), 2014.

A. J. Stewart, S. Hannenhalli, and J. B. Plotkin. Why transcription factor binding sites are ten nucleotides long. *Genetics*, 192(3):973–985, 2012.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–133, 1974.

J. Suzuki, H. Isozaki, and E. Maeda. Convolution kernels with feature selection for natural language processing tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

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

A. N. Tikhonov and V. Y. Arsenin. Solutions of ill-posed problems. *New York*, pages 1–30, 1977.

M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt. Wasserstein weisfeiler-lehman graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.-R. Müller. A new discriminative kernel from probabilistic models. In *Advances in Neural Information Processing Systems (NIPS)*, 2002a.

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18(suppl_1):S268–S275, 2002b.

A. B. Tsybakov. *Introduction to nonparametric estimation*. Springer Science & Business Media, 2008.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

N. Verma, E. Boyer, and J. Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. 2018.

J.-P. Vert. The optimal assignment kernel is not positive definite. *arXiv preprint arXiv:0801.4061*, 2008.

J.-P. Vert, H. Saigo, and T. Akutsu. Convolution and local alignment kernels. *Kernel methods in Computational Biology*, pages 131–154, 2004.

J.-P. Vert, R. Thurman, and W. S. Noble. Kernels for gene regulatory regions. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research (JMLR)*, 11, 2010.

E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

M. J. Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge University Press, 2019.

C. Wallraven, B. Caputo, and A. Graf. Recognition with local features: the kernel recipe. In *International Conference on Computer Vision (ICCV)*, 2003.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: a multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*, 2019.

M. Wang, J. Yang, G.-P. Liu, Z.-J. Xu, and K.-C. Chou. Weighted-support vector machines for predicting membrane protein types based on pseudo-amino acid composition. *Protein Engineering Design and Selection*, 17(6):509–516, 2004.

S. Wang, J. Peng, J. Ma, and J. Xu. Protein secondary structure prediction using deep convolutional neural fields. *Scientific reports*, 6(1):1–11, 2016.

W. Wang, D. Slepcev, S. Basu, J. A. Ozolek, and G. K. Rohde. A linear optimal transportation framework for quantifying and visualizing variations in sets of images. *International Journal of Computer Vision*, 101(2):254–269, 2013.

Y. Wang, H. Wang, L. Wei, S. Li, L. Liu, and X. Wang. Synthetic promoter design in escherichia coli based on a deep generative network. *Nucleic Acids Research*, 48(12): 6403–6412, 2020.

C. Watkins. Dynamic alignment kernels. In *Advances in Neural Information Processing Systems (NIPS)*, 1999.

Y. Weiqiu, S. Sun, and M. Iyyer. Hard-coded gaussian attention for neural machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

C. K. Williams. Computation with infinite neural networks. *Neural Computation*, 10 (5):1203–1216, 1998.

C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.

M. Wu, B. Schölkopf, and G. Bakir. Building sparse large margin classifiers. In *International Conference on Machine Learning (ICML)*, 2005.

Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *Transactions on Neural Networks and Learning Systems*, 2020.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Y. Yamanishi, J.-P. Vert, and M. Kanehisa. Protein network inference from multiple genomic data: a supervised approach. *Bioinformatics*, 20(suppl_1):i363–i370, 2004.

T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances Neural Information Processing Systems (NeurIPS)*, 2018.

Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In *Advances Neural Information Processing Systems (NeurIPS)*, 2019.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision (ECCV)*, 2014.

H. Zeng, M. D. Edwards, G. Liu, and D. K. Gifford. Convolutional neural network architectures for predicting dna–protein binding. *Bioinformatics*, 32(12):i121–i127, 2016.

K. Zhang, I. W. Tsang, and J. T. Kwok. Improved nyström low-rank approximation and error analysis. In *International Conference on Machine Learning (ICML)*, 2008.

M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, 2018a.

S.-W. Zhang, Q. Pan, H.-C. Zhang, Y.-L. Zhang, and H.-Y. Wang. Classification of protein quaternary structure with support vector machine. *Bioinformatics*, 19(18): 2390–2396, 2003.

Y. Zhang, P. Liang, and M. J. Wainwright. Convexified convolutional neural networks. In *International Conference on Machine Learning (ICML)*, 2017.

Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *Advances Neural Information Processing Systems (NeurIPS)*, 2018b.

J. Zhou and O. G. Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature methods*, 12(10):931–934, 2015.

C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.