UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

ON THE COMPOSITION OF NEURAL AND KERNEL LAYERS FOR MACHINE LEARNING

ALEX MARTORELL LOCASCIO

Thesis supervisor: LUIS ANTONIO BELANCHE MUÑOZ (Department of Computer Science)

Degree: Master's degree in data science

Thesis report

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Abstract

Deep Learning architectures in which neural layers alternate with mappings to infinite-dimensional feature spaces have been proposed in recent years, showing improvements on the results obtained when using either technique separately. However, these new algorithms have been presented without delving into the rich mathematical structure that sustains kernel methods.

The main focus of this thesis is not only to review these advances in the field of Deep Learning, but to extend and generalize them by defining a broader family of models that operate under the mathematical framework defined by the composition of a neural layer with a kernel mapping, all of which operate in reproducing kernel Hilbert spaces that are then concatenated. Each of these spaces has a specific reproducing kernel that we can characterize. Together all of this defines a regularization-based learning optimization problem, for which we prove that minimizers exist. This strong mathematical background is complemented by the presentation of a new a model, the Kernel Network, which manages to produce successful results on many classification problems.

"S'ha d'escriure amb llibertat, amb gust, amb plaer,
però amb la màxima observació possible"
Josep Pla

"Hi un amor profund en la memòria del que havies volgut ser"
Jordi Graupera

Acknowledgements

I want to start by thanking my professor and advisor Dr Lluís Belanche, who through countless meetings and too many coffees has given me helpful advice and most importantly, good ideas. The conversations we had helped me carry on with this thesis. I am certain that his lectures and his knowledge are present in my writing.

My parents also deserve a big acknowledgement, for all their support during my student life and all the difficulties of it. This thesis is a big milestone and I am grateful to share it with them.

My classmates (and close friends) Pim, Enric and Louis are also part of this. It has been 2 tough years, but being able to work with them has made everything a little bit easier.

Contents

List of Figures

List of Tables

1 Introduction

Deep Neural Networks and Kernel methods are two pillars of Machine Learning algorithms, and more specifically, of regularization-based learning. They have been studied as independent methods and both of them have gone through an uncountable amount of extensions, ramifications and generalizations. In fact, if someone now mentions these model families without giving more details, the reader asks for more specifications: which type of neural network are they referring to, what are we trying to predict, which kernel are we using, how many hyperparameters does it have, whether it is a deep model or not, ... as it is expected, the sophistication has not ceased to increase.

However, let us try to set some boundaries to orientate the reader. In neural networks, born with the multilayer perceptron (MLP), have have gone through many advancements, to produce complex models as LSTMs, convolutional neural networks (CNNs) and autoencoders. All these newer architectures belong to the framework defined by the umbrella term deep neural network, or sometimes an even bigger one, deep learning. The word "deep", usually just means more than one hidden layer; as the original models had just one hidden layer between the input and output.

With regards to Kernel methods, the limits are easier to set. The main idea is to try to solve a problem but in a different space. If we are discussing a classification problem, this is easy to explain: it is more often than not impossible to find a separating hyperplane between the two classes. However, what if these data points are mapped to a higher dimension ? Then, it may be possible to find a function the correctly classifies the two classes. This concept of optimizing a function in a different space is the fundamental notion behind kernel methods. They rely on a strong mathematical theory which is branch from functional analysis. To summarize, the data points are mapped to a Hilbert space of functions $\mathcal{H}$. Then, a function $f \in \mathcal{H}$ can be found such that a prediction $f(x)$ is a linear combination of the data points mapped to this new space.

In recent years, research seems to be willing to see if these two groups of models - in theory far apart from each other - have anything in common besides that they take data points as inputs. The authors of [1] refer to this process as *hybridization.* The idea behind it is to apply from the deep learning field to kernel methods and vice-versa. A starting point were gaussian processes, as the authors of [2] show us. They present a model that lays the first stone in this hybridization technique between deep networks and kernel methods. In their paper, the authors go over the comparison kernel-neural net: It seems like that in the 1990s, neural networks were seen as a revolutionary model that would take away the podium from kernel methods. This article , which was published in 2015, is a clear sign that this never happened, and subsequent research has proven that hybridization produces better results in many performance metrics, and what is more interesting, allows to solve new problems in Machine Learning for which an answer had yet to be found.

This thesis is structured in three main parts: The first one, to which Chapter 2 is dedicated, is a **review of the literature** published in the recent years about the main topics the are crucial to this writing. These are grouped in 4 categories: Firstly, what do we understand for the term "deep kernel learning" as of today. Secondly, it covers one of the better known hybridization proposals, the deep hybrid model. This proposal is what this thesis aims to formalize and generalize. After this, the two aspects that are more related to kernel methods are reviewed: representer theorem and approximations to kernels. The first aspect deals with the different versions of this essential result in kernel methods. The second one discusses the advances in approximating one of the more important kernels used in kernel methods, the gaussian RBF.

The second part is the **theoretical backbone** of the thesis and spans over three chapters: 3, 4 and 5. The first one, whilst trying to avoid excessive meanders and unnecessary technical detail that exceed our scope, it presents the essential mathematical tools behind the goal to establish connections between Kernel methods and Deep Learning. Chapter 4 is dedicated to stating a Representer Theorem for a multi class Deep Kernel Learning problem, a fundamental result to be able to characterize the Deep Kernel Learning model presented in Chapter 5, the **Kernel Network**. Aside from the equations of this new model and its mathematical details, a thorough training algorithm for it is also given.

Finally, the last part of the thesis, reserved to Chapter 6 and 7, is dedicated to presenting the **experimental results** obtained, as well as comparing them with other existing machine learning models. A rigorous approach is used to assess the quality of the model. Last but not least, and before some concluding thoughts, a brief chapter is included which a dissertation about some of the **limitations** encountered, as well as possible extensions and a future outlook.

2 Literature Review

There are several building blocks that come into place when discussing Deep Kernel Learning (DKL). Firstly, the term itself. Nowadays, there are many machine learning algorithms that are said to be DKL, hence, it becomes necessary to list a set of characteristics that such algorithms fulfill. It is also important to consider the theoretical framework behind DKL. Functional analysis and the representer theorem provide it for Kernel methods, and it is crucial to this thesis to see if this framework can be extended to DKL. Finally, kernel approximation is an important topic, in order to reduce the computational cost of learning algorithms.

In the whole of this thesis, a Learning problem in the context of machine learning is assumed. This consists of a set $X \subset \mathbb{R}^d$, $Y = \mathbb{R}$, a kernel function $k : X \times X \to \mathbb{R}$ which induces a mapping ϕ into an RKHS $\mathcal{H}$, a Hilbert space of functions $f : X \to \mathbb{R}$. (See Chapter 3 for more details).

2.1 The term "Deep Kernel Learning"

It is clear that the term *deep kernel learning* is already mainstream in the machine learning community. To put it into the context of our thesis, it can be useful to trace it back to its origins. Survey paper [3] is a good reference for the array of terminologies that have appeared in recent years, as it also focuses on finding points in common between Deep Kernel Learning (DKL) and Multiple Kernel Learning. (MKL)

In MKL, the kernel k is defined as a combination of predefined kernels $k = \sum_{m=1}^{M} \mu_m k_m$. The feature mapping is to a space induced by several kernels, defined as

$$\phi(x) = \big(\phi_1(x), \dots \phi_M(x)\big)^T$$

The term μ_m indicates the weight of each kernel, a combination that is learnt during training. Additional constraints may be imposed, such that $\sum_m \mu_m = 1$. [3]. One of the main references for Multiple Kernel Learning is, which states that combining kernels instead of 1 yields to better results in existing algorithms [4]. For instance, one of the more simple kernel combinations which is the sum $k_1 + k_2$ or the multiplication $k_1 \cdot k_2$ in a SVM arw shown to improve the test accuracy significantly.

DKL, on the other hand, is not based upon on a convex combination of kernels but rather in their composition. In general, a DKL model consists of several mappings into different feature spaces, which can be expressed as:

$$k(x_i, x_j) = \big(\phi_1 \circ \phi_2 \circ \dots \circ \phi_L(x_i)\big)^T \big(\phi_1 \circ \phi_2 \circ \dots \circ \phi_L(x_j)\big)$$

Article [2] is the first one to coin in 2015 the term "Deep Kernel Learning", as its title suggests. Their proposal is a Neural Network with L layers followed by a Gaussian Process (which produces a probabilistic mapping). Recall that a Gaussian Process $f(X) \sim \mathcal{GP}(\mu, k_\gamma)$ produces an output:

$$f(X) \sim [f(x_1) \dots f(x_n)]^T \sim N(\mu, K_{X,X}) \tag{2.1}$$

where $\mu = \mu(x_i)$ is the mean function and $(K_{X,X})_{i,j=1\div n} = k_\gamma(x_i, x_j)$ is the covariance kernel. γ is a hyperparameter, meaning that the covariances are parametrized by γ. It is clear then that a gaussian process is a collection of functions. The architecture proposed is:

$$\text{Input} \rightarrow \text{Hidden Layer } 1 \rightarrow \cdots \rightarrow \text{Hidden Layer } L \rightarrow \infty\text{-layer}$$

The novelty of this model is the ∞-layer. This is the author's way of referring to a gaussian process with an RBF Kernel, since as it is explained in Chapter 3, the feature mapping of an RBF kernel can be seen as an infinite basis function representation.

The justification for this choice can be found in the introduction of [2]: In the late 90s, there was a debate about whether gaussian processes would end up replacing neural networks. This rivalry between models produced an important amount of literature comparing neural networks (adpative finite basis functions, architecture, regularization) and gaussian processes (infinite basis functions, kernels), which brings research to a new stage: Combining the structural properties of neural networks with the non-parametric flexibility of kernel methods. Kernel methods are non-parametric in the sense that they do not make assumptions about the data distribution, something that is not common to many other ML models. The structural properties of neural networks (neurons, activation functions, stacking layers) succeed in learning complex data patterns.

Many developments have made since 2015. As of 2021, according to survey [3] there are three major families of models in the DKL context:

- Combination of deep learning architectures and kernel machines, the first one being the front-end and the latter the back-end part of the model. An example is a combination of a CNN with an SVM. The features learnt by a CNN are then used to train an SVM with the Gaussian kernel.
- Incorporation of kernel methods into deep architectures via the stacking principle: A first example is an SVM trained in a standard way and then applying a kernel activation on the support vectors, which are used as inputs for a second SVM. Another example is training deep neural networks with kernel blocks in between neural layers, which provides new representations of the data to possibly improve learning. This architecture, called **Deep Hybrid Model** was ideated by Mehrkanoon et al. and explained in [5, 6]. Since this architecture is one of the main inspirations and is studied throughout this thesis find a thorough introduction to its structure in section 2.2. The DKL model presented in this thesis, called Kernel Network, would also fall into this category. (Chapter 5)
- Integrating Deep Learning ideas into Kernel Learning. One of the more well-known examples of this is the arc-cosine kernels which mimic deep neural networks.

An attempt to combine MKL and DKL (which is explained below) is found in [7]. The authors generalize the MKL optimization problem as:

$$\min_{k \in K} \min_{f \in \mathcal{H}_k} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}_k} \tag{2.2}$$

where $\mathcal{H}_k$ is the reproducing kernel Hilbert space associated to kernel k, and K is the optimization domain of the candidate kernels. Since this is an MKL framework, one should expect that K is the domain obtained by a convex combination of kernels (see above). The extension proposed by the authors is Multiple Layer Multiple Kernel Learning (MLMKL), which is to generalize equation 2.2 to a concatenation of layers, which results in a domain of l-level multi layers:

$$K^{(l)} = \{k^{(l)}(\cdot,\cdot) = g^{(l)}([k_1^{(l-1)}(\cdot,\cdot), \ldots k_m^{(l-1)}(\cdot,\cdot)])\} \tag{2.3}$$

where $g^{(l)}$ is a function that combins all the kernels from the previous layer. Then, the authors delve into a Two-layer MKL scenario where the outer function is a gaussian RBF and the inner is a combination of m kernels. The two layer minimization problem is formulated as

$$\min_{k \in K^{(2)}} \min_{f \in \mathcal{H}_k} \frac{1}{2}\|f\|^2_{\mathcal{H}_k} + \sum_{i=1}^{N} L(f(x_i), y_i) + \sum_{k=1}^{m} \mu_k \tag{2.4}$$

Clearly, a penalty is added for the sum of μ parameters. Observe that the regularization is over the final function f, which expresses the concatenation of the two kernel layers. This model (2LMKL) returns better results than the SVM on many datasets. Alternatives to this formulation are seen in Chapter 4.

However, it must be said that MKL is not the focus of this thesis, however, as it has been shown, the representer theorem and many DKL algorithms have a clear connection with MKL.

2.2 Deep Hybrid models

The adjective *deep*, which already appeared in the previous section, does not only apply to kernels, but rather to a much broader field. Deep learning is where models composed of multiple processing layers learn representations of data with multiple levels of abstraction. [8]. Stacking different levels of layers allows the data points to be mapped to another vector space, which may have a higher dimension than the previous one, hence offering different feature representation, different levels of abstraction.

It has been effort of many research papers to see if a bridge can be built that establishes a clear connection between kernel methods and deep neural networks. This process can be referred to as *hybridization.* The lesson from deep kernel learning is that several layers may be needed to learn complex representations. For this reason, Kernel methods, which generally consist in finding similarities between data points through the matrix $(K_{X,X})_{i,j=1\div n} = k(x_i, x_j)$ may be insufficient. To study the incorporation of deep networks and kernel methods into a single architecture if is has been so far a good starting point for more expressive models [1].

The Deep Hybrid model was introduced in [5] and extended in [6] The main idea is to insert a kernel layer between two neural layers. A kernel layer consists of a mapping from a space to another with a different dimension. In figure 2.2, a simple illustration of this deep hybrid model is depicted.

The equations of the baseline model depicted in Figure 2.2 are [1]:

$$\begin{aligned} h_1 &= W_1 x + b_1 \\ h_2 &= \hat{\varphi}(h_1) \\ s &= W_2 h_2 + b_2 \end{aligned} \tag{2.5}$$

where $W_1 \in \mathbb{R}^{d_1 \times d}$ and $W_2 \in \mathbb{R}^{Q \times d_2}$ are the weight matrices, and $b_1 \in \mathbb{R}^{d_1}$, $b_2 \in \mathbb{R}^{Q}$ are the bias vectors. Note that s is the output of the neural layer, values to which a loss function is applied.

The question here is what exactly is $\hat{\varphi}(\cdot)$. As explained before, the model is named deep hybrid, meaning that it combines neural and kernel layers. Hence, $\hat{\varphi} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$

[1]The figures in this section are a reproduction of the ones that can be found in [5, 6]

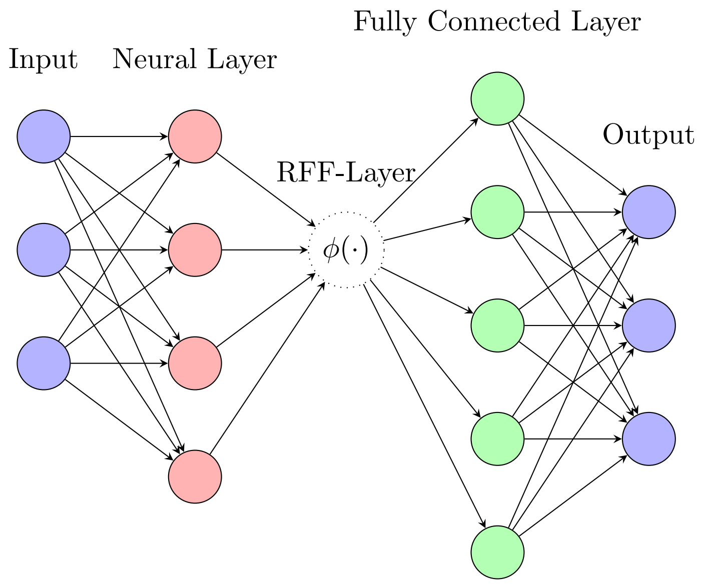

Figure 2.1: Graphical Representation of the Deep Hybrid Model

is an explicit mapping to the feature space (d_2 is the dimension of the feature space). More specifically, there exists a kernel $k : \mathbb{R}^{d_1} \times \mathbb{R}^{d_1} \to \mathbb{R}$ which induces a feature map $\phi : \mathbb{R}^{d_1} \to \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space. Note that two different spaces that seem to be similar objects have been mentioned here: A Hilbert space $\mathcal{H}$ and a vector space $\mathbb{R}^{d_2}$ (which is also in fact a Hilbert space). The differences between them and how they interact in the deep hybrid model are theoretical: They are explained in Chapter 3 and 5.

What is also explained in the next chapter this feature space need not be finite dimensional. It is known that some kernels induce a feature mapping to a Hilbert space that is infinite, such as the Gaussian RBF Kernel. The mathematical technicalities of this implication are outlined in Section 3.1. If the kernel trick ($k(x, y) = \varphi(x)^T \varphi(y)$) cannot be employed, the task becomes cumbersome. If $\varphi(x)$ is infinite, this becomes impossible. That is why the author uses an approximation to the explicit feature map, which is called Random Fourier Features. This approximation technique is central to this thesis, so find a detailed description of it in 3.2. In summary, what happens is that a map to a lower dimensional space $\varphi : \mathbb{R}^d \to \mathbb{R}^D$ is defined:

$$\hat{\varphi}(x) = \frac{1}{\sqrt{D}}\big(\cos(\omega_1^T x), \ldots, \cos(\omega_D^T x)\big) \quad \omega \sim N(0, \sigma^2 I_d) \tag{2.6}$$

Note that this definition is the one used by [5] and it is not the same as the one appearing in the original paper [9]. These distinctions are also discussed in section 3.2. At the end of the Fully Connected Layer, in the case of a multiclass the softmax $\sigma : \mathbb{R}^Q \to [0, 1]^Q$ is applied. To compute the difference between the prediction and the true value, the negative log-likelihood loss is used. The composition of the softmax and the negative log likelihood is called cross-entropy loss:

$$CE = -\sum_{i=1}^{d} t_i \log(p_i) \tag{2.7}$$

where t_i is the truth label for the i-th class and p_i the predicted probability for each class after the softmax has been applied. Clearly this is a standard function in neural network training. This is mentioned because the representer theorem will need to be adapted to this specific loss function. [10, 11]

A difference between the deep hybrid model and a neural network is the absence of an activation function between layers. However, the author expresses the similarities between an explicit feature mapping and activation function (e.g. the ReLU). Recall that the ReLU, which introduces non-linearity in the Neural Network is defined as $f(x) = \max\{0, x\}$. The explicit feature mapping, as defined in 2.6, receives input from all neurons. The product $\omega^T x$ involves the whole representation x for each coordinate $i = 1, \ldots, D$ of the new representation $\hat{\varphi}(x)$.

These differences can be visualized schematically in Figure 2.2.

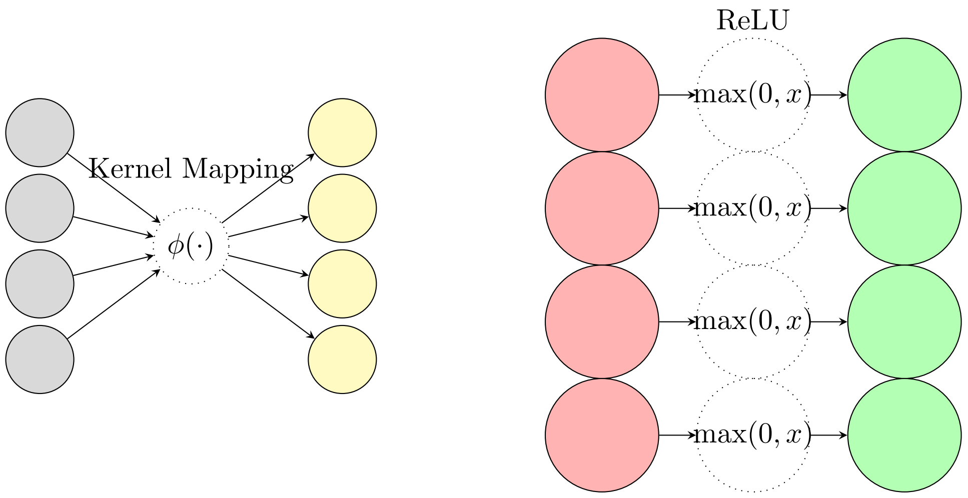

Figure 2.2: Comparison between the explicit feature mapping in a Deep Hybrid Model and the ReLU activation function in a Feed-Forward Neural Net

In [6], Mehrkanoon develops of an extension of the Deep Hybrid Model. This consists of a more complex architecture which uses the idea of a kernel mapping, but a more sophisticated version of it. The kernel mapping now receives the maximum of m neural passes, the average, or a convolutional opeartion. We focus briefly on the equations of the maxout kernel block, to see how it compares to the equations in 2.5.

The equations of a maxout kernel block are:

$$\begin{aligned} h^{(l)}_{\text{maxout}} &= \max_{k \in \{1, \ldots m\}} V^{(l)}_k h^{(l-1)} + b^{(l)}_k \\ h^{(l)} &= \hat{\varphi}_{(l)}\big(h^{(l)}_{\text{maxout}}\big) \end{aligned} \tag{2.8}$$

where (l) is the current layer, $V^{(l)}_k$ are the weight matrices and $b^{(l)}_k$ are the biases. Since a forward pass returns a vector in $\mathbb{R}^{d_l}$, the coordinate $i \in \{1, \ldots, d_l\}$ is defined as the maximum coordinate i in vectors $k = 1, \ldots, m$.

The average kernel block is analogous to the maxout. Finally, the convolutional block consists of a pointwise convolution. A pointwise convolution is a 1×1 convolution, meaning that each element gets affected by such operation.The intricacies present in defining convolutional filters is out of the scope of this thesis, but they are worth mentioning since CNNs appear again briefly in section 5.2.

Having said this, the equations of a deep maxout neural-kernel network are:

$$
\begin{aligned}
h^{(1)}_{\text{maxout}} &= \max_{k\in\{1,\ldots m\}} V^{(1)}_k x + b^{(1)}_k \\
h^{(1)} &= \hat{\varphi}_{(1)}\big(h^{(1)}_{\text{maxout}}\big) \\
h^{(2)}_{\text{maxout}} &= \max_{k\in\{1,\ldots m\}} V^{(2)}_k h^{(1)} + b^{(2)}_k \qquad (2.9)\\
h^{(2)} &= \hat{\varphi}_{(2)}\big(h^{(2)}_{\text{maxout}}\big) \\
s(x) &= Wh^{(2)} + b \qquad (2.10)
\end{aligned}
$$

where $h^{(1)}$ and $h^{(2)}$ each represent a maxout neural-kernel block and $W \in \mathbb{R}^{d_2\times Q}$ is the weight matrix of the fully connected layer.

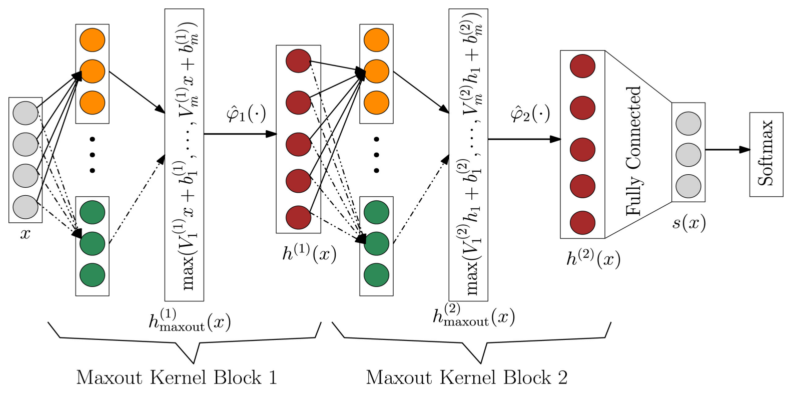
x
$\max(V_1^{(1)}x + b_1^{(1)}, \ldots, V_m^{(1)}x + b_m^{(1)})$
$\hat{\varphi}_1(\cdot)$
$h^{(1)}(x)$
$h^{(1)}_{\text{maxout}}(x)$
Maxout Kernel Block 1
$\max(V_1^{(2)}h_1 + b_1^{(2)}, \ldots, V_m^{(2)}h_1 + b_m^{(2)})$
$\hat{\varphi}_2(\cdot)$
$h^{(2)}(x)$
$h^{(2)}_{\text{maxout}}(x)$
Maxout Kernel Block 2
Fully Connected
$s(x)$
Softmax

Figure 2.3: Illustration of deep hybrid model with two maxout neural-kernel blocks. Figure extracted from article “Deep neural-kernel blocks”

In summary, what is seen is a more complex ”neural pass”, whereas the kernel activation remains the same, as an approximation of the gaussian kernel by RFFs.

Since the described model is present in several moments during the course of this thesis, some terminology around it must be set. The concatenation of h_1 and h_2 is referred to as a neural-kernel block. In [5], the author refers to the whole model as *deep hybrid model*. In the description of the algorithm, the name *deep hybrid neural-kernel network* is used. The term *block* only appears in the second paper ([6]), in a vague way. As stated previously, the extensions of the deep hybrid model are called *maxout kernel block*, *average kernel block* or *convolutional kernel block*. This second paper is in fact titled ”Deep neural-kernel blocks”, which is why, in this thesis, the structure neural layer + kernel mapping is referred to as *neural-kernel* block:

$$
\begin{aligned}
x' &= Wx + b \qquad (2.11)\\
x'' &= \hat{\varphi}(x)
\end{aligned}
$$

This is deemed a useful term since a concatenation of neural-kernel blocks can now be easily described. The concatenation of neural-kernel blocks with a fully connected layer at the end is called for the author Stacked layers model.

In the experimental section, the deep hybrid model is comapred with two other models beside One Layer or Two Layer Neural Networks: LS-SVM and TROP-ELM. The LS-SVM (least squares support vector machine) is a Kernel based machine learning algorithm which consists of optimizing the following functional:

$$J(w,e) = \frac{1}{2}w^T w + \gamma \frac{1}{2}\sum_{i=1}^{N} e_k^2$$
$$s.t \quad y_i = w^T \varphi(x_i) + b + e_i$$

The TROP-ELM is the acronym for Tikhonov-Regularized Optimally Pruned Extreme Learning Machine. This model, described in [12]. The authors present an extension of the OP-ELM. Extreme Learning Machine is a model where there is initialization of weights without backpropagation. This, of course is huge improvement in computational time, and produces surprising results. Optimally pruned means to solve the problem of irrelavant variables that can corrupt some of the neurons. Tikhonov-regularized is adding an L_2 penalty on regression weights. The authors claim that TROP-ELM gives better results, which is why it is used by authors of [5, 6] for comparison.

The deep hybrid model is tested using 15 different classifications data sets. The author claims that in average, the Deep Hybrid model with one neural-kernel block (i.e. Neural + Kernel + Fully Connected) performs equally or slightly better (On average an increment from 0 to 2%). The extension with different types of neural-kernel blocks produces better results on many of these datasets, with improvements in accuracy up to 6% with respect to the Deep Hybrid Model.

2.3 Representer Theorem and Deep Learning

The Representer Theorem is a fundamental result associated to Kernel methods [13, 14]. It defines a prediction function when mimizing the regularized risk as a linear combination of the Kernel matrix evaluated at the training data points x_i for $i = 1, \ldots, N$:

$$f(\cdot) = \sum_{i=1}^{N} \alpha_i k(\cdot, x_i) \tag{2.12}$$

In the survey [13] and in many subsequent papers, the problem to which it is applied is always a training set $X = \mathbb{R}^d$ considered is with training data set $X = \mathbb{R}^d$ but with output $Y = \mathbb{R}$. Also, the classical Representer Theorem (Wahba) is proved for one type of empirical risk (The mean square loss) and one quadratic regularizer. These two facts are the context of most statements of the Representer Theorem in Statistical Learning Literature. The output $Y = \mathbb{R}$ is applied to classification and regression problems. In [14] one can find a semi-parametric version of the Representer Theorem. The authors also generalize it to a larger group of regularizers.

The semi-parametric Representer Theorem, besides the hypothesis of the standard Representer Theorem (or non-parametric) considers a set of M real-valued functions $\{\psi_p\}_{p=1}^{M}$ over X. In addition, if we construct the matrix $(\psi_p(x_i))_{ip}$, where $i = 1, \ldots, n$, this matrix must have rank M. if that is the case, then when minimizing the regularized risk, admits a representation of the form $\tilde{f} := f + h$, with

$$\tilde{f}(\cdot) = \sum_{i=1}^{N} \alpha_i k(x_i, \cdot) + \sum_{p=1}^{M} \beta_p \psi_p(\cdot) \tag{2.13}$$

Observe that $f \in \operatorname{span}\{k(\cdot, x_i) \mid i = 1, \ldots, N\}$ and $h \in \operatorname{span}\{\psi_p\}$.

Why this Representer Theorem was named semi-parametric is now clear, as the prediction function $\tilde{f}$ has a set of real-valued functions that no depend on the number of data points. A simple example where this semiparmetric theorem can be applied is the SV classifier with an offset term b.

The results in [15] are born out of a limitation in interpolation and regression in RKHS. Sometimes, the Hilbert space H may not contain the functions that can approximate the solution of the problem. The authors define two similar functions $g_1 = (0.1 + |x|)^{-1}$ and $g_2 = (0.1 + |x - y|)^{-1}$, and produce 200 samples $(x_i, g_k(x_i))$ for each $k = 1, 2$. Then they try to approximate them with a function $f \in \mathcal{H}$ where $\mathcal{H}$ the Hilbert space defined by the tensor product of two Sobolev-Matern kernels (see page 17 of [15] for a definition). The problem arises when $g_1 \in H$ but $g_2 \notin H$. However, if the interpolant is searched in $\{f \circ R\} \mid f \in \mathcal{H}$ where R is a rotation, that solves the issue. The authors show this interpolant incurs in less error rather than using f as interpolator.

In summary, this example is very clear to set motivation for the authors, which also shared by this thesis: Two layers can be more expressive than a single one, which is restrictive for. That is a lesson that has already been extracted from Deep Neural Networks, which are able to extract more complex patterns.

The rest of the paper is dedicated to presenting a finite-sample (as well as an infinite-sample) representer theorem for the concatenation of L layers. A description of the Interpolation and Regression in the standard context (Wahba's Representer Theorem) is given, which is then extended to L layers. That implies describing two or more RKHS, one or more being vector-valued. A thorough description of the minimization problem and how the stated representer theorem applies to it is also provided.

The authors make a suggestion (Section 3.3.3 of [15]) to relate the finite sample representer theorem to Deep Kernel Learning. They assume that these models consists of an outer Kernel K applied to a concatenation of non linear functions $f_2, \ldots, f_L$

$$\tilde{K}(x, y) = K(f_2 \circ \ldots \circ f_L(x), f_2 \circ \ldots \circ f_L(y))$$

What if we assume that $f_l \in \mathcal{H}_{\updownarrow}$ for $l = 2, \ldots, L$ with $\mathcal{H}_l$ being an RKHS? This short paragraph, which can sound remarkable or prosaic to some readers, is one of the fundamental aspects of this thesis, which is to establish an array of connections between Kernel methods and Deep Networks.

In a different article [16], which then becomes part of PhD Thesis [17], Dinuzzo studies the case of a Kernel Machine with two layers. The justification is to consider an architecture that needs this composition of two functions to be described:

$$f = f_1 \circ f_2, \quad f_2 : X \to Y, \quad f_1 : Y \to Z \tag{2.14}$$

where X is a set and Y and Z are Hilbert spaces. No restrictions are made here over Y and Z, they are assumed to be vector valued. The author offers interpretations for this novel architecture: In many data modeling problems, one of them being that function g_1 can act as a preprocessing transformation on the data, by extracting features that will then be used in the predictor g_2.

A Representer Theorem for the problem 2.14 is stated (See Chapter 4). It is general result that allows for the kernel to be a combination of m kernels, which is a connection to MKL.The author moves to a problem where the kernel of the inner Hilbert space $\mathcal{H}_1$ has a matrix-valued kernel associated defined as $K^1 = \text{diag}(\tilde{K}_1, \ldots, \tilde{K}_m)$, where $\tilde{K}_i$ for $i = 1, \ldots, m$ are basis kernels to be defined.

Finally, a mention to developments from a mathematical point of view. Reserach has focused on generalizing and unifying the different statements that have appeared as Representer Theorems. [18] is a generalization by offering necessary and sufficient conditions for these theorems, a complicated issue as seen in section 3.3.2. Mathematical preliminaries are needed which clearly , such as linear operator theory and specifically subspace valued maps . On this same note, [19] states a representer theorem for Hilbert space valued functions (Vector Valued in our case).

2.4 Random Fourier Features

Random Fourier features are introduced by Rahimi in Recht in 2007 [9], as an approximation for the Gaussian RBF Kernel. (cf. Section 3.2)

$$k(x,y) = \langle \phi(x), \phi(y) \rangle \approx z(x)^T z(y) \tag{2.15}$$

with $z : X \to \mathbb{R}^D$. Instead of evaluating the kernel function, or relying on the kernel trick (first equality) the input is transformed via z. As it will be shown, there is not just one definition for z.

This is only possible for shift-invariant kernels, i.e. $k(x,y) = k(x-y)$. The authors in [9] show that this approximation $|k(x,y) - z(x,y)|$ can be bounded to error ε with only $D = O(D\varepsilon^{-2} \log \frac{1}{\varepsilon^2})$.

Another perspective which of special interest in this thesis is the error bounds for Random Fourier Features. In [9], a first error bound is given. The article entitled ”On the Error Bounds of Random Fourier Features”, [20] reviews the definition of Random Fourier Features in attempts to tighten the probability bound in [9]. However, the important addition to the study of this approximation is that the authors study the error in specific ML models, like Kernel Ridge Regression or Support Vector Machines. In other words, they study directly the difference in predictions due to using z instead of k. If $h(x) = \alpha^T k(x_i, x)$ and $\hat{h}(x) = \alpha^T \hat{k}_x$ is the function obtained through the Representer Theorem, it is possible to bound $|\hat{h}(x) - h(x)|$ by the differences $\|\hat{k}_x - k_x\|_2$ and $\|\hat{K} - K\|_2$.

What about the applications of Random Fourier Features in different settings? They have been used in Machine Learning algorithm since their appearance in 2007. Survey [21] lists many of them.

The simplicity and practical usage of this approximation sparked the question if other kernels, could have also a low-dimensional approximation. This applies to the inhomogeneous polynomial kernel $k(x,y) = (\langle x, x' \rangle + c)^p$ for which we know the feature mapping $\phi(x)$ has dimension $\binom{d+p}{p}$, a combinatorial number that explodes for large p. Spherical Random Features are presented in [22]. They provide an approximation for the polynomial kernel in the unit sphere. The impossibility of applying Bochner's Theorem to guarantee a non-negative Fourier Transform of the polynomial kernel is remedied by Spherical Random Features, which are similar to RFFs when it comes to defining the mapping ϕ, but different when it comes to defining $p(\omega)$ This finding is useful to extend Deep Hybrid models, in the sense that the kernel layer could be a mapping by a different kernel than the Gaussian RBF.

3 Important concepts in RKHS Theory

In this chapter, an overview is given of the necessary concepts that illustrate the results of Chapters 4 and 5. This consists of a brief summary of Kernel Methods and the Representer Theorem as well as detailed definition of two important kernels: The Gaussian RBF kernel and the Polynomial Kernel.

This allows to introduce further concepts in RKHS Theory, which are central to this thesis: Bochner's Theorem guarantees that a shift invariant kernel is positive definite if and only if it is the Fourier Transform of non-negative measure. This gives us a possibility to approximate the Gaussian RBF Kernel through computation of the fourier transform.

Since this is an approximation, an error with respect to the true value is incurred. In section 3.2 some of the error bounds are listed, which will be useful to further characterize the models in Chapter 5.

Section 3.3.2 is devoted to Vector Valued reproducing kernel Hilbert spaces, an extension to the definition of provided in Section 3.1. As shown, this consists of an extension to functions $f : X \to \mathbb{R}$, to the case $f : X \to \mathbb{R}^D$, for $D > 1$. representer theorems, which also are an extension from the real-valued function to the vector-valued function are presented. Vector valued RKHS are defined by matrix valued kernels, meaning that now there is a kernel matrix $k : X \times X \to \mathbb{R}^{D\times D}$. Studying matrix-valued kernels and its properties are not the objective of this thesis, but rather to study their relationship with a scalar-valued kernels, so that the models presented have a simpler expression.

3.1 Kernel methods, RKHS and The Representer Theorem

Let X be a non empty set. From now on, $X = \mathbb{R}^d$ is assumed. A symmetric function is called a positive semi definite kernel if $k : X \times X \to \mathbb{R}$ if for every $n \in \mathbb{N}$, for every $x_1, \dots, x_n \in X$ and $\forall a_i, a_j \sum_i \sum_j a_i a_j K(x_i, x_j) \geq 0$.

A simple example of a kernel is the linear kernel, $k(x, y) = \langle x, y\rangle_{\mathbb{R}^d}$ for $x, y \in \mathbb{R}^d$, which clearly is symmetric and positive semi definite.

The definition below links the concept of kernel and Hilbert space. Note that in some texts this appears as a Proposition.

Definition 3.1. Let X be a non-empty set. A function $k : X \times X \to \mathbb{R}$ is a kernel if and only if there exists a Hilbert space $\mathcal{H}$ and a map $\phi : X \to \mathcal{H}$ such that for all $x, x' \in \mathcal{H}$:

$$k(x, x') = \langle \phi(x), \phi(x')\rangle_{\mathcal{H}} \tag{3.1}$$

Equation is referred to as the *kernel trick.* This means that similarities can be computed in the Hilbert space directly by evaluating the kernel instead of computing the inner product in $\mathcal{H}$. This can be advantageous as we might not have an explicit expression ϕ, or such expression can be hard to obtain.

The following two definitions, which are intertwined, are the backbone of Kernel methods.

Definition 3.2. Let $X \subseteq \mathbb{R}^d$ and $\mathcal{H}$ a Hilbert space. A kernel function $k : X \times X \to \mathbb{R}$ is called a *reproducing kernel* of $\mathcal{H}$:

(1) $\forall x,\ k(x, \cdot) \in \mathcal{H}$

(2) $\forall f \in \mathcal{H}$ and $\forall x \in X$ $f(x) = \langle f(\cdot), k(x, \cdot)\rangle$ (reproducing property)

Then $\mathcal{H}$ is a *reproducing kernel Hilbert space.*

Example 3.3. Most Hilbert spaces are Reproducing Kernel Hilbert spaces. Spaces (with their respective inner product) such as $\mathbb{R}, \mathbb{R}^d, l^2, L^2$ are examples of RKHS.

A more formal definition for an RKHS follows.

Definition 3.4. An RKHS is a Hilbert space of functions where the linear evaluational functions are continuous

$$\begin{aligned}[x](\cdot) :& H \to \mathbb{R} \\ & f \to [x]f = f(x) \quad \forall f \in H \quad \forall x \in X\end{aligned}$$

Kernel methods and RKHS on their own cannot work without the representer theorem. This is a result that comes form Riesz's representation lemma.

In Section 2.3, a few mentions were made to the representer theorem and some extensions that have been tried. However, it is important to stress its importance in the context of machine learning. Observe that the optimization problem

$$\min_{f \in \mathcal{H}} L((x_1, f(x_1)), \ldots, (x_n, f(x_n))) + \lambda \|f\|^2 \tag{3.2}$$

is solved in a Hilbert space $\mathcal{H}$. As said before, the space $\mathcal{H}$ can be of high dimension or infinite. The representer theorem says that the solution to equation (3.2) is a linear combination

$$f(\cdot) = \sum_{i=1}^{N} K(\cdot, x^i)$$

Theorem 3.5. (Representer Theorem) $X = \mathbb{R}^d$, $Y = \mathbb{R}$ and $k : X \times X \to \mathbb{R}$ a kernel function. $\mathcal{H}$ is the RKHS induced by k, $\Theta : \mathcal{H} \to \mathbb{R}$ a non-decreasing function (regularizer) and the loss L. The optimization problem:

$$\min\nolimits_{f \in \mathcal{H}} \left\{ L\Big((f(x^1), y^1), \ldots (f(x^n), y^n) \Big) + \Theta(\|f\|^2) \right\}$$

has a solution of the form $f(\cdot) = \sum_{i=1}^{n} \alpha_i K(x^i, \cdot) \in \mathcal{H}$

Proof. Consider $Y = \text{span}\{K(x_i, \cdot) \quad i = 1, \ldots, N\} \subset H$, a closed set. By the Orthogonal decomposition Theorem $\mathcal{H} = Y \oplus Y^{\perp}$. Therefore, if $f \in \mathcal{H}$, $f = f_Y + f_Y^{\perp}$. By applying the reproducing property,

$$f(x) = \langle f(\cdot), K(x, \cdot)\rangle = \langle f_Y, K_x\rangle \quad \forall x \quad \forall f \in \mathcal{H}$$

Now, $f_Y = \sum_{i=1}^{N} \alpha_i K(x_i, \cdot)$. By definition $\|f\|^2 = \|f_Y + f_{Y^{\perp}}\|^2 \geq \|f_Y\|^2$. Since Θ is non decreasing, $\Theta(\|f\|^2) \geq \Theta(\|f_Y\|^2)$ for all $f \in \mathcal{H}$. □

The minimizer f need not be unique. For that, the convexity of the functional is necessary. (More on that in Section 3.3.2). It is important to point out that the monotonicity of Θ (i.e. it is a non decreasing function) is to guarantee that there always exists a minimizer of the form $f(\cdot) = \sum_{i=1}^{N} \alpha_i k(\cdot, x_i)$. This is used in the last line of the proof.

One of the references for Kernel methods and Support Vector Machines is [23], which sets it in the broader context of Statistical Learning. Chapter 4 is dedicated to Kernels and provides a formal theory about the gaussian rbf kernel and its RKHS, which this review on Kernels and RKHS Theory is partly based on.

In [23], a distinction is made between real valued kernels and complex valued kernels. That is not the case of [24], another important reference for Kernel methods.

Complex valued kernels are not considered in this thesis, as the possible solutions of the function to be approximated are in an $\mathbb{R}$-Hilbert space. For more information, the reader is referred to Chapter 4 in [23].

Let us move to discuss infinite feature mappings. The first concept needed is the one of sequence, but a particular one: a *square summable sequence* is $\{\alpha_k\}_{k\geq 1}$ for which $\|\alpha_k\|^2 = \sum_{k=1}^{\infty} \alpha_k^2 < \infty$.

Definition 3.6. The Hilbert space of square summable sequences l_2 is the complete vector space of sequences such that for $(\alpha_k)_k \in l_2$ then $\sum_{n=1}^{\infty} \alpha_k^2 < \infty$. l_2 is endowed with inner product $\langle \{\alpha_k\}_k, \{\beta_k\}_k \rangle = \sum_k \alpha_k \beta_k$.

Observe that feature map $\phi : X \to \mathcal{H}$ need not be unique. Consider the kernel matrix $K = (k(x_i, x_j))_{i,j=1,\dots,N}$. Since a kernel matrix K is positive semidefinite it can be decomposed as $K = VDV^T$. By definition of a kernel $K = \phi\phi^T$ where

$$\phi = \begin{bmatrix} \phi(x_1) \\ \dots \\ \phi(x_N) \end{bmatrix} \tag{3.3}$$

Which is why $\phi(x_i)^T = v_i^T D^{1/2}$ is a feature map. However, $\phi'(x) = Q\phi(x)$, when Q is an orthogonal matrix, is another valid feature map. This is clear since for all $i, j \in \{1, \dots, N\}$ we have $\phi'(x)\phi'(x') = \phi(x)^T Q^T Q \phi(x) = \phi(x)^T \phi(x)$.

The following lemma characterizes an important fact: Kernels can be described as the inner product between two sequences of l_2, meaning that it possible to have a kernel with a feature mapping to an infinite dimensional Hilbert space.

Lemma 3.7. Let X be a set and $(\phi_i(x))_{i\geq 1}$ a sequence of l_2, where $\phi_i(x)$ is the i-th coordinate of the feature map to the Hilbert space $\mathcal{H} = l_2$. The function defined as

$$k(x, x') = \sum_{i=1}^{\infty} \phi_i(x)\phi(x')$$

is a kernel.

This Lemma is a classical result in Kernel methods, and can be found in Section 4.1 of [23].

Proof. Hölder's inequality for l_2 spaces gives

$$\sum_{i=1}^{\infty} |\phi_i(x)\phi_i(x')| \leq \|\phi_i(x)\|_{l_2} \|\phi_i(x')\|_{l_2} < \infty$$

because $(\phi_i(x))$ belongs to l_2. Now, define $H := l_2$ and trivially $\phi(x) := (\phi_i(x))$ maps X to the Hilbert space. This proves that k is a kernel. □

It is clear that the Gaussian RBF Kernel and the Polynomial Kernel are widely used in machine learning [24, 13, 25]. Observe that they are both kernels with infinite or potentially infinite dimensional feature mappings, which In Chapter 5 these kernels will become necessary to describe the kernel network model, as well as a link between activation functions and RKHS. Find below detailed of these two kernels.

Notation. $[d]$ is used to indicate $\{1, \ldots, d\}$. Hence $[d]^p$ indicates the cartesian product $\{1, \ldots, d\} \times \overset{p}{\cdots} \times \{1, \ldots, d\}$

Example 3.8 (Homogeneous Polynomial Kernel)**.** The Polynomial kernel is defined as $k(x, x') = (\langle x, x\rangle + c)^d$, d is the dimension. Note that if $c = 0$ the polynomial kernel is referred to as homogeneous. For instance, given $p = 2$ and $d = 2$, the expansion of the product is

$$\Big(\langle (x_1, x_2), (x_1', x_2')\rangle\Big)^2 = (x_1x_1' + x_2x_2')^2 = x_1^2x_1^{2\prime} + 2x_1x_1'x_2x_2' + x_2^2x_2^{2\prime} =$$
$$= \sum_{j\in[2]^2} [x_1]_{j_1}[x_2]_{j_2} \cdot [x_1']_{j_1}[x_2']_{j_2}$$

More generally,

$$\langle x, x'\rangle^p = \sum_{j\in[d]^p} [x]_{j_1} \cdot \ldots \cdot [x]_{j_p}[x']_{j_1} \cdot \ldots \cdot [x']_{j_p} = \langle \phi(x), \phi(x')\rangle \tag{3.4}$$

where $\phi(x)$ has components being all possible p-th degree ordered products of the entries of x. In other words, for the case $d = 2, p = 2$, we have $\phi(x) = (x_2^2, x_2x_1, x_1^2)$

Example 3.9 (Gaussian RBF Kernel)**.** The Gaussian RBF kernel is defined as

$$k(x, x') = e^{-\frac{\|x-x'\|^2}{2\sigma^2}} = e^{-\gamma\|x-x'\|^2}$$

σ^2 or its equivalent γ is a hyperparameter. To find the feature mapping, observe

$$e^{-\gamma\|x-x'\|^2} = e^{-\gamma\|x\|^2}e^{2\gamma x^Tx'}e^{-\gamma\|x'\|^2}$$

Using the taylor expansion at $x = 0$

$$e^{2\gamma x^Tx'} = \sum_{k=0}^{\infty} \frac{1}{k!}(2\gamma\langle x, x'\rangle)^k$$

Using the results from the polynomial kernel, we obtain the feature map ϕ

$$e^{2\gamma x^Tx'} = \frac{2^k\gamma^k}{k!} \sum_{j\in[d]^k} [x]_{j_1} \cdot \cdots \cdot [x]_{j_k}[x']_{j_1} \cdot \cdots \cdot [x']_{j_k} \tag{3.5}$$

Clearly, the feature mapping for the Gaussian RBF kernel is:

$$\phi(x) = e^{-\gamma\|x\|^2}\left(\sqrt{\frac{2^k\gamma^k}{k!}}\prod_{i=0}^{k} x_{j_i}\right) \tag{3.6}$$

If the polynomial is of infinite degree, the feature mapping defined for both kernels is $\phi : \mathbb{R}^d \to l^2(\mathbb{N})$.

The next section is devoted to approximations to the Gaussian RBF Kernel via Random Fourier Features. An interesting discussion, which is found in [26] attemps to approximate the Gaussian RBF through the taylor expansion and comparing its error with the approximation through RFFs.

3.2 Approximations of the Gaussian RBF Kernel: Random Fourier Features

Let us begin with two important definitions.

Definition 3.10. The Fourier transform of a function on the eucliedan space $\mathbb{R}^n$ is defined as:

$$\hat{f}(\xi) = \int_{\mathbb{R}^n} f(x) e^{-i2\pi\langle \xi, x\rangle} dx \tag{3.7}$$

where ξ and x are vectors of $\mathbb{R}^n$.

Definition 3.11. A scale invariant kernel on $\mathbb{R}^d$ is a kernel function $k : X \times X \to \mathbb{R}$ such that $k(x, x') = k(x - x')$ for $x, x' \in \mathbb{R}^d$

The Gaussian, Laplacian and Cauchy Kernel are examples of shift-invariant kernels.

Notation. If $v \in \mathbb{C}^n$, v^* denotes the conjugate transpose, i.e. the operation of transposing v, and then applying the complex conjugate ($a + ib$ becomes $a - ib$)

Theorem 3.12 (Bochner)**.** A continuous shift-invariant kernel is positive definite if and only if it is the Fourier transform of a non-negative measure $p(\omega)$.

Observation 3.13. If $k(0) = 1$, $p(\omega)$ is a normalized probability density function.

Now, in application of Bochner's Theorem

$$k(x - y) = \int_{\mathbb{R}^d} p(\omega) e^{i\omega^T(x-y)} d\omega = E_\omega[\xi_\omega(x)\xi_\omega(y)^*] \tag{3.8}$$

where $\xi_\omega(x) = e^{i\omega^T x}$. The second equality is the definition of expected value. In particular, $\xi_\omega(x)\xi_\omega(y)^*$ is an unbiased estimate of $k(x, y)$ when ω is drawn form p (the probability distribution). [9]. Observe that the expected value can be approximated (Monte-Carlo):

$$E_\omega[\xi_\omega(x)\xi_\omega(y)^*] \approx \sum_{j=1}^{D} \xi_{\omega_j}(x)\xi_{\omega_j}(y) = \sum_{j=1}^{D} \exp(i\omega_j^T(x - y))$$

where ω_j are taken i.i.d from the probability distribution p. By Euler's Formula and $\forall j$:

$$\exp(i\omega_j^T(x - y)) = \cos(\omega_j^T(x - y)) - i\sin(\omega_j^T(x - y))$$

Since the kernel is assumed to be real-valued:

$$k(x - y) = \mathrm{Re}\,\big(E_\omega[\xi_\omega(x)\xi_\omega(y)^*]\big) = E_\omega\big[\mathrm{Re}\,(\xi_\omega(x)\xi_\omega(y)^*)\big] = E_\omega[\cos(\omega^T(x - y))]$$

Now, one can make the following observation (Law of total expectation)

$$E_\omega[\cos(\omega^T x + b)] = E_\omega[E_b[\cos(\omega^T x + b)] \mid \omega] = 0 \quad b \sim U[0, 2\pi]$$

which allows us to write

$$\begin{aligned} &E_\omega[\cos(\omega^T(x - y))] = E_\omega[\cos(\omega^T(x - y))] + E_\omega[\cos(\omega^T(x + y) + 2b)] \\ &= E_\omega[\cos(\omega^T(x - y))\cos(\omega^T(x + y) + 2b)] = E_\omega[\sqrt{2}\cos(\omega^T x + b)\sqrt{2}\cos(\omega^T y + b)] \end{aligned}$$

The last equality is trigonometry [2]. Observe that we now a mapping defined: $\xi_\omega(x) = z_\omega(x) = \sqrt{2}\cos(\omega^T x + b)$, with $\omega \sim p(\omega)$ and $b \sim U[0, 2\pi]$. In summary, we have proved the $\xi_\omega(x)\xi_{\omega(y)}$ has expected value $k(x, y)$. Now, the final step is to define the mapping $\phi(x)$, which has dimension D. As explained in the following algorithm, the mapping will be based on sampling D ω's and b's from their respective distributions.

The algorithm for calculating Random Fourier Features provided in [9] for the Gaussian RBF Kernel is summarized in the steps below:

(1) Compute the Fourier transform p of the shift-invariant kernel k

(2) Draw D iid samples $\omega_1, \ldots, \omega_D \in \mathbb{R}^d$ from $p(\omega)$. as well as D iid samples for b, $b_1, \ldots, b_D \in \mathbb{R}$ from the uniform(0, 2π) .

(3) Define $z(x) = \sqrt{\frac{2}{D}}\Bigg(\cos(\omega_1^T x + b_1) \ldots \omega_D^T x + b_D)\Bigg)^T$

First note that the fourier transform of the gaussian RBF kernel is also a gaussian, in particular $p(\omega) = (2\pi)^{-\frac{D}{2}} e^{-\frac{\|\omega\|_2^2}{2}}$. Also, note the differences between the feature mapping and the one the author of the Deep Hybrid model employs (see Section 2.2). These differences and their affect will be discussed in Chapter 5.

It is clear now that the definition $z(x)$ is not unique and that authors have used different versions. Research paper [20] attempts to list advantages and disadvantages between using the $z(x)$ defined previously or as:

$$\breve{z}(x) = \sqrt{\frac{2}{D}}\Bigg(\sin(w_1^T x)\cos(w_1^T x) \ldots \sin(w_{D/2}^T x)\cos(w_{D/2}^T x)\Bigg)^T$$

The reason why it is possible is direct consequence from trigonometry and noting that we have only sampled $D/2$. ω's from $p(\omega)$. The conclusion that $z(x)$ is superior than $\breve{z}(x)$ for the Gaussian Kernel.

Finally, to show the versatility of RFFs, recall the representer theorem from Section 3.1, and apply it to obtain a linear expansion of the RFFs:

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x) = \sum_{i=1}^{N} \alpha_i \langle \phi(x_i), \phi(x') \rangle \approx \sum_{i=1}^{N} \alpha_i z(x_i)^T z(x) = \hat{w}^T z(x) \tag{3.9}$$

with $w = \sum_{i=1}^{N} \alpha_i z(x_i)$.

3.3 Vector Valued Reproducing Kernel Hilbert Spaces

The representer theorem considers X to be set and $Y = \mathbb{R}$. [14], which is the most common application. As mentioned in section 2.3 there have been extensions of the representer theorem to the case $Y = \mathbb{R}^d$. More generally, the problem of learning real-valued function is extended to learning Hilbert-space valued functions.

This needs a more complex mathematical structure, which is an extension of the field of Learning in Reproducing Kernel Hilbert Spaces (RKHS). The case where the output is a vector, i.e. $Y = \mathbb{R}^d$ is referred vector-valued. Hence, a theory was developed for the

[2] $\cos(x + y) + \cos(x - y) = 2\cos(x)\cos(y)$

concept of Vector Valued Reproducing Kernel Hilbert Spaces (vector valued RKHS or VVRKHS from now on). It is not the goal of this thesis to provide a thorough presentation the theory surrounding vector valued RKHS, but rather to highlight the important definitions and results that allow to prove the statement for the two-layer representer theorem, one of the main results of the Thesis.

The following definitions and properties are extracted from the main reference in vector valued RKHS ([10]). A less formal introduction to vector valued RKHS, among other topics, can be found in [27].

Notation. In [10] two different notations are used for the inner product: $(\cdot,\cdot)$ and $\langle\cdot,\cdot\rangle$. The first one is used for the Hilbert space Y (i.e. the output), and the latter for the RKHS $\mathcal{H}$ Here we adopt $\langle\cdot,\cdot\rangle$ to denote the inner product of any Hilbert space.

3.3.1 Definitions and main properties

Let X be set, Y a Hilbert space, and H the Hilbert space of functions $f : X \to Y$. From now on, $Y = \mathbb{R}^D$

Definition 3.14. *H is a Vector Valued Reproducing Kernel Hilbert space if there exists a kernel function $\mathbf{K} : X \times X \to \mathbb{R}^{D\times D}$ such that*

(1) $\mathbf{K}(x,\cdot)z \in H(X,\mathbb{R}^D)\ \forall x \in X$ *and* $z \in \mathbb{R}^D$

(2) $z^T f(x) = \langle f, \mathbf{K}(\mathbf{x},\cdot)z\rangle_{H(X,\mathbb{R}^D)}\ \forall x \in X,\ \forall z \in \mathbb{R}^D$ *and* $\forall f \in H(X,\mathbb{R}^D)$

Observe the main difference with the RKHS theory described in [13], which is for kernels $K : X \times X \to \mathbb{R}$, which is called a **scalar-valued** kernel in comparison to the function defined above, which is a **matrix-valued** kernel. Also, the difference in the reproducing property with respect to a scalar-valued RKHS.

Example 3.15. *$K(x,y) = xy^T$ is a matrix-valued kernel. [28]*

When constructing matrix-valued kernels, a relationship with the scalar valued counterpart can be established.

Definition 3.16. Let $\mathbf{K} : X \times X \to \mathbb{R}^{D\times D}$, $k : X \times X \to \mathbb{R}$ and $A \in \mathbb{R}^{D\times D}$ positive semi-definite matrix. Define

$$\mathbf{K}(x,x') = k(x,x')A$$

$\mathbf{K}$ is called a *separable* kernel. There exists is an extension of this definition to a more general case,

$$\mathbf{K}(x,x') = \sum_{i=1}^{p} k_i(x,x')Q_i$$

where Q_i positive semi definite matrices.

Observe that the matrix A can be initialized in many different ways, and learnt. [27]. Definition 3.16 allows to work with scalar valued kernels in vector valued RKHS. For instance, for a matrix valued kernel $K : X \times X \to \mathbb{R}^{D\times D}$ the Gaussian RBF Kernel can be adapted to this new context by defining:

$$K(x, x') = e^{-\gamma\|x-y\|^2}\mathrm{diag}(A) \tag{3.10}$$

In other words, between two data points x and x', the matrix valued kernel K has non-zero values in the diagonal, and they are all the same. This means that the outputs are treated as unrelated, i.e. i-th component does not have any relationship with j-th component. In other words B encodes dependencies between the outputs.

The following section goes over the representer theorems in the context of vector-valued output. As one clearly notices, these are equivalent versions of the Representer Theorem for scalar output.

3.3.2 Representer Theorems

The following result is a representer theorem for the regularization problem in vector-valued spaces, with the squared error loss. [10]

Theorem 3.17. (Representer Theorem for vector valued functions) Consider $\{(x_i, y_i)\}_{i=1}^n \subseteq X \times Y$ with the following approximation scheme

$$J(f) = \sum_{j=1}^{n} \|y_i - f(x_i)\|^2 + \lambda\|f\|^2$$

If $\hat{f}$ minimizes J in $\mathcal{H}$, it is unique and has the form

$$\hat{f} = \sum_{i=1}^{n} K_{x_i} c_i$$

where the coefficients $c_j \in Y$ are the unique solution of the linear equations

$$\sum_{j=1}^{n} (K(x_i, x_l) + \mu\delta_{il})c_l = y_i \quad i = 1, \ldots, n$$

Proof. cf. p.7-8 [10]. □

The proof is similar to the scalar version of the representer theorem. Note that the statement of the theorem says the minimzer is *unique*. This is a rare case in representer theorems, as it is clear that for many functionals there will not be a unique minimizer. In this case, the uniqueness is thanks to the convexity of the loss function and the regularizer. Note the difference between the minimizer of the scalar case and the vector valued:

$$f = \sum_{i=1}^{n} \alpha_i k_{x_i} \qquad f = \sum_{i=1}^{n} c_i K_{x_i} \tag{3.11}$$

for $\alpha_i \in \mathbb{R}$ and $c_i \in \mathbb{R}^D$, $k : X \times X \to \mathbb{R}$ and $K : X \times X \to \mathbb{R}^{D\times D}$. In the previous section, the concept of separable kernel was introduced. For the vector valued case, we write the following:

$$f(\cdot) = \sum_{i=1}^{n} K(\cdot, x_i)c_i = \sum_{i=1}^{n} k(\cdot, x^i)Ac_i = \sum_{i=1}^{n} k(\cdot, x_i)\mathrm{diag}(a_j)c_i \tag{3.12}$$

where diag(a_j). for $a_j \in \mathbb{R}$ for $j = 1, \ldots, D$. Observe that the output vector $f(\cdot)$ has D components, with f_j beingthe sum across the training data set of the product $k(\cdot, x_i)a_j c_{j,i}$. This observation is the basis for the presentation of the kernel network in Chapter 5.

Notation. $Y^n = Y \times \overset{n}{\cdots} \times Y$ is used to express the space that contains all the data points. Note that it is a Hilbert space with inner product $\langle c, c' \rangle = \sum_{j=1}^{n} \langle c_j, c'_j \rangle$, for $c, c' \in Y^n$, i.e. $c = (c_j \in \mathbb{R}^D) \in Y^n$

This applies to the square error loss, but in general one can define the functional

$$E(f) = V(f(x_j), \|f\|^2) \tag{3.13}$$

where $V : Y^n \times \mathbb{R} \to \mathbb{R}$ as $V(f(x_j), \|f\|^2)$ for all $f \in \mathcal{H}$. This notation is a formal way of describing the minimization problem:

$$E(f) := \sum_{j=1}^{n} L(y_j, f(x_j)) + \Theta(\|f\|^2) \tag{3.14}$$

The authors in [10] prove a theorem for any function which minimizes the functional $E(f)$

Theorem 3.18. If for every $y \in Y^n$ the function $\Theta : \mathbb{R}_+ \to \mathbb{R}_+$ defined for $t \in \mathbb{R}_+$ by $h(t) := V(y, t)$ is strictly increasing and $f_0 \in \mathcal{H}$ minimizes the functional V, then $f_0 = \sum_{j \in [m]} K_{x_j} c_j$ for $c_j \in Y$. If V is strictly convex, the minimizer is unique.

Proof. cf. p. 9-10 [10]. □

The proof is very simple, and can be seen as a corollary of the previous theorem. Two other straightforward corollaries are stated below appear from this theorem. They appear under what is a common circumstance E might have one than more minimum, provided that it is a function of sveral variables. Is the form of the f_0 the same?

Corollary 3.19. If V satisfies the hypothesis of theorem 3.18 and $f_0 \in \mathcal{H}$ is a local minimum of E, then $f_0 = \sum_{j \in [m]} K_{x_j} c_j$ for $c_j \in Y$

Proof. cf. p. 10 [10]. □

This first theorem shows that there is a representation theorem for any global or local minimum of E, provided that V is strictly increasing. The corollary below is a result that can be applied when this strict convexity cannot be fulfilled.

Corollary 3.20. If V is differentiable at the local minimum $f_0 \in \mathcal{H}$ of E then there exists $c \in Y^n$ such that $f_0 = \sum_{j \in [m]} K_{x_j} c_j$

Proof. cf. p. 10 [10]. □

In summary, so far the following versions of the representer theorem have been covered:

(1) For the scalar-valued case, a representer theorem is presented with a general loss function, and non-decreasing regularizer Θ. This is called non-parametric representer theorem [14].

(2) Still in the scalar-valued case, a semi-parametric theorem was stated. This involves minimizing $\tilde{f}$ in the spaces H and span$\{\psi_p\}$ where ψ_p are real-valued functions, meaning we find $\tilde{f} = f + h$ [14].

(3) A vector-valued representer theorem for the standard approximation scheme with $L2$ regularization, which considers the squared loss and a constant times the squared norm of f as a regularizer. Also, the function f is found to be a unique minimizer, and in particular, the unique solution of a system of linear equations.

(4) For a general functional V, a vector-valued representer theorem is stated giving the expression form of f_0, which is $f_0 = \sum_{j\in[m]} K_{x_j} c_j$. Uniqueness is guaranteed provided the convexity of the minimizer.

(5) It is clear that in most practical cases is E will have more than one local minimum. Given the hypothesis of (4), which is Theorem , meaning that V is strictly increasing, a strong result states that in any local minimum of E also has the expression form $f_0 = \sum_{j\in[m]} K_{x_j} c_j$.

(6) Last but not least, an alternative to the strict convexity of V in the second argument is proposed, via the differentiability of V at f_0.

As mentioned previously, attempts to generalize and unify representer theorem for all functionals and regularizers have been made. This, however, requires moving a step up: operator theory and subspace valued maps are the framework for this approach. This exceeds the scope of this thesis, which does not need such a general for the pratical cases in which a result of the nature of the representer theorem is needed.

4 A Representer Theorem for the concatenation of L layers

The generalization of the Representer Theorem [13, 14] has been the objective of several research papers. In the previous chapter extensions to Vector Valued RKHS context have been studied. (see section 3.3.2 for more details)

Another extension to the Representer Theorem is to minimize a function f defined as a composition of two or more functions. Originally, this can be found in [16]. This article is part of a thesis by the same author on Kernel Methods [17].

In [16], the focus is the extension of kernel methods to a two layer problem. The setup is as follows: Learning a function $g : X \to Y$ (X a set and Y a Hilbert space) can be broken into two separate problems, by introducing $\mathcal{H}_1$, a Z-valued RKHS over X and $\mathcal{H}_2$ a Y-valued RKHS over Z . The spaces $\mathcal{H}_1$ and $\mathcal{H}_2$ are vector valued. The optimization problem consists of minimizing the functional

$$\min_{\substack{g_1 \in \mathcal{H}_1 \\ g_2 \in \mathcal{H}_2}} \sum_{i=1}^{N} L(y_i, g_2 \circ g_1(x_i)) + \Theta_1(\|g_1\|_{\mathcal{H}_1}) + \Theta_2(\|g_2\|_{\mathcal{H}_2}) \tag{4.1}$$

The first theorem for vector valued functions, proposes a sufficient condition.

Theorem 4.1 (Dinuzzo)**.** If the functional of problem 4.1 admits minimizers, then there exists minimizers of the form

$$g_1(x) = \sum_{i=1}^{N} K^1(x_i, x)c_i^1 \quad g_2(z) = \sum_{i=1}^{N} K^{(2)}(g_1(x_i), z)c_i^{(2)}$$

Letting $K(x_1, x_2) := K^2(g_1(x_1), g_1(x_2))$ denote the input-output kernel, there exists optimal learning architectures whose input-output map can be written as:

$$g(x) = (g_2 \circ g_1)(x) = \sum_{i=1}^{l} K(x_i, x)c_i^{(2)} \tag{4.2}$$

Proof. cf. p.112 - 113 [17]. □

The proof of this theorem uses a trick that and it is that points z_i can be seen as fixed points of the function $g_1(x_i)$ for $i = 1, \dots, N$ and the function minimized is

$$\min_{g_2 \in \mathcal{H}_2} L(g_2(z_1), \dots, g_2(z_N)) + \Theta_2(\|g_2\|_{\mathcal{H}_2})$$

In application of the Representer Theorem for vector valued RKHS (Theorem 3.18), there exists a minimizer of the form

$$g_2(z) = \sum_{i=1}^{l} K_{z_i}^2(z)c_i^2$$

Now that g_2 has been fixed, the problem to be solved is:

$$\min_{g_1 \in \mathcal{H}_1} \tilde{L}(g_1(x_1), \dots, g_1(x_N)) + \Theta_1(\|g_1\|_{\mathcal{H}_1})$$

Note $\tilde{L}(z) = L(g_2(z_1), \dots, g_2(z_N))$ which allows for the application of the Representer theorem again, obtaining the desired representation $g_2 \circ g_1$.

Section 3.1 covers the fact that Representer Theorems are not "Existence theorems", meaning that they do not guarantee an existence of minimizers, but rather they provide a representation for the minimizer in terms of the data points. Clearly, this also applies for the theorem above.

The existence of a minimizer can be ensured if there are conditions over g_1, g_2, Θ_1 and Θ_2. In the previous chapter, we showed that uniqueness of minimizers is not always guaranteed. Finally, as stated previously, if Θ_1 and Θ_2 are strictly increasing, all the minimizers have the form of the expansion presented in the theorem.

In the paper *A representer theorem for deep kernel learning*, authors B. Bohn, M.Griebel and C.Rieger state and proof a Representer Theorem for a concatenation of L layers. As explained in section 2.3, their motivation behind this new result comes from the shortfalls of the Representer Theorem in finding $f : X \to Y$ for some specific problems, that escape the mathematical structure of a single Hilbert space $\mathcal{H}$. For this reason, it is necessary to extend the framework to multiple layers, where each layer is a new RKHS, which contains arbitrary functions. The function $f : X \to Y$ is a now a concatenation of functions belonging to different RKHS. That allows for a better theoretical function approximation.

The main applications for this theorem are interpolation and regression, which means the learning problem is standard: $X = \mathbb{R}^d$ and $Y = \mathbb{R}$. L is the number of layers, i.e. the number of compositions of functions belonging to different Hilbert spaces.

Theorem 4.2. Let $\mathcal{H}_1, \dots, \mathcal{H}_l$ be reproducing kernel Hilbert spaces of functions with finite dimensional domains D_l and ranges $R_l \subseteq \mathbb{R}^{d_l}$ with $d_l \in \mathbb{N}$ for $l = 1, \dots, L$, such that $R_l \subseteq D_{l-1}$ for $l = 2, \dots, L$, $D_L = \Omega$ and $R_1 \subseteq \mathbb{R}$. Let furthermore $L : \mathbb{R}^2 \to [0, \infty]$ be an arbitrary loss function and $\Theta_1, \dots, \Theta_L : [0, \infty) \to [0, \infty)$ be strictly monotonically increasing functions. Then, a set of minimizers $\big(f_l\big)_{i=1}^{L}$ with $f_l \in H_l$ of

$$J(f_1, \dots, f_l) = \sum_{i=1}^{N} L(y_i, f_1 \circ \dots \circ f_L(x_i)) + \sum_{l=1}^{L} \Theta_l\big(\|f_l\|^2_{\mathcal{H}_l}\big)$$

fulfills $f_l \in V_l \subset \mathcal{H}_l$ for all $l = 1, \dots L$ with

$$V_l = \operatorname{span}\{K_l(f_{l+1} \circ \dots \circ f_L(x_i), \cdot)e_{k_l} \mid i = 1, \dots, N \text{ and } k_l = 1, \dots d_l\}$$

where K_l denotes the reproducing kernel of $\mathcal{H}_l$ and $e_{k_l} \in \mathbb{R}^{d_l}$ is the k_l-th unit vector.

Proof. cf. Theorem 1 p. 5-6 [15]. □

In this setting, a function $f_l \in \mathcal{H}_l$ is describe $f_l : D_l \to R_l \subseteq D_{l-1}$, where D_l, R_l and D_{l-1} are finite-dimensional. This a condition for this family of Representer Theorems, since the proof uses the orthogonal decomposition of a Hilbert space, which requires a finite dimensional Hilbert space. Generalizations of the representer theorem to infinite dimensional spaces have to undergo different techniques in operator theory and functional analysis.

The main lesson from this theorem is that an infinite-dimensional optimization problem is actually solved in a finite-dimensional space. This is the case for the representer theorems that have been presented in this thesis. The original problem

$$\arg \min_{\substack{f_l \in \mathcal{H}_l \\ l=1,\dots L}} J(f_1, f_2, \dots, f_L)$$

is now an optimization problem in a finite-dimension problem:

$$\arg \min_{\substack{f_l \in V_l \\ l=1,\dots L}} J(f_1, f_2, \dots, f_L)$$

The following Theorem is an equivalent of Theorem 4.2 with the difference that Y is a Vector Valued RKHS. It is the fundamental result of this thesis, as it supports the Kernel Network model developed in Chapter 5. The proof of it is based on the one of Theorem 4.2.

Theorem 4.3. Let $\mathcal{H}_1, \dots, \mathcal{H}_l$ be reproducing kernel Hilbert spaces of functions with finite dimensional domains D_l and ranges $R_l \subseteq \mathbb{R}^{d_l}$ with $d_l \in \mathbb{N}$ for $l = 1, \dots, L$, such that $R_l \subseteq D_{l-1}$ for $l = 2, \dots, L$, $D_L = \Omega$ and $R_1 \subseteq \mathbb{R}^m$. Let furthermore $L : Y \times \mathbb{R}^m \to [0, \infty]$ be an arbitrary loss function and $\Theta_1, \dots, \Theta_L : [0, \infty) \to [0, \infty)$ be strictly monotonically increasing functions. Then, a set of minimizers $\big(f_l\big)_{i=1}^{L}$ with $f_l \in H_l$ of

$$J(f_1, \dots, f_l) = \sum_{i=1}^{N} L(y_i, f_1 \circ \dots \circ f_L(x_i)) + \sum_{l=1}^{L} \Theta_l\big(\|f_l\|^2_{\mathcal{H}_l}\big)$$

fulfills $f_l \in V_l \subset \mathcal{H}_l$ for all $l = 1, \dots L$ with

$$V_l = \text{span}\{K_l(f_{l+1} \circ \dots \circ f_L(x_i), \cdot)e_{k_l} \mid i = 1, \dots, N \text{ and } k_l = 1, \dots d_l\}$$

where K_l denotes the reproducing kernel of $\mathcal{H}_l$ and $e_{k_l} \in \mathbb{R}^{d_l}$ is the k_l-th unit vector.

Proof. Since $\mathcal{H}_l$ are Hilbert spaces with a finite-dimensional domain, the orthogonal decomposition Theorem says that if $V_l \subset \mathcal{H}_l$ is a closed subset, it has an orthogonal complement $V_l^{\perp}$ such that $\mathcal{H}_l = V_l \oplus V_l^{\perp}$. Then f_l projects in two these two subspaces: $f_l = \Pi_{V_l}(f_l) + \Pi_{V_l^{\perp}}(f_l)$. This is for all $l = 1, \dots, L$ In virtue of the reproducing property for vector valued spaces:

$$f_l \circ f_{l+1} \circ \dots \circ f_L(x_i) = \sum_{k=1}^{d_1} \left\langle \Pi_{V_l}(f_l) + \Pi_{V_l^{\perp}}(f_l), K_l(f_{l+1} \circ \dots \circ f_L(x_i), \cdot)e_k \right\rangle e_k$$

Observe that $\langle \Pi_{V_l^{\perp}}(f_l), K_l(f_{l+1} \circ \dots \circ f_L(x_i), \cdot)e_k \rangle = 0$:

$$f_l \circ f_{l+1} \circ \dots \circ f_L(x_i) = \sum_{k=1}^{d_1} \left\langle \Pi_{V_l}(f_l), K_l(f_{l+1} \circ \dots \circ f_L(x_i), \cdot)e_k \right\rangle e_k$$

In application of the reproducing property once again $\langle f, K(x, \cdot)\rangle z = z^T f(x)$:

$$\sum_{k=1}^{d_1} \left(e_k^T \Pi_{V_l^{\perp}}(f_l)(f_{l+1} \circ \dots \circ f_L(x_i)) \right) e_k = \Pi_{V_l}(f_l)(f_{l+1} \circ f_L(x_i))$$

which is the evaluation of $\Pi_{V_l^{\perp}}$ at $f_{l+1} \circ \dots \circ f_L$. The last equality is clear. The process explained can be iterated to obtain:

$$f_1 \circ f_2 \circ \dots f_L(x_i) = \Pi_{V_1}(f_1) \circ \Pi_{V_2}(f_2) \circ \dots \circ \Pi_{V_L}(f_L)(x_i)$$

The functional $J(f_1, \dots f_L)$ is now written in function of the projections of $f_1, \dots, f_L$

$$J(f_1, \dots, f_L) = \sum_{j=1}^{N} L(y_i, \Pi_{V_l}(f_1) \circ \Pi_{V_l}(f_2) \circ \dots \circ \Pi_{V_L}(f_L)(x_i)) + \sum_{l=1}^{L} \Theta_l \left(\|\Pi_{V_l}(f_l)\|^2_{\mathcal{H}_l} + \|\Pi^{\perp}_{V_l}(f_l)\|^2_{\mathcal{H}_l} \right)$$

Where we have observed that $\|f_l\|^2_{\mathcal{H}_l} = \|\Pi_{V_l}(f_l)\|^2_{\mathcal{H}_l} + \|\Pi^{\perp}_{V_l}(f_l)\|^2_{\mathcal{H}_l}$ as in the proof for Theorem 3.5. Since Θ_l are strictly monotonically increasing functions:

$$J(f_1, \dots, f_L) \geq J(\Pi_{V_1}(f_1), \dots, \Pi_{V_L}(f_L))$$

which ensures that the minimizer f_l belongs to V_l for all $l = 1, \dots, L$ $\square$

5 A Deep Kernel Learning model: The Kernel Network

Notation. In the subsequent sections, a specific notation is used to differentiate the functions that define the Deep Hybrid Model from the ones obtained by the Representer theorem. For a single iteration of a Deep Hybrid Model (without stacked Neural-Kernel Blocks), the outputs of the Neural, Kernel, and FC Layer are denoted as h_1, h_2, and s respectively, following the notation in [5] in [6]. Whereas the functions representing the corresponding operations in their respective Reproducing Kernel Hilbert Spaces (RKHS) are denoted as f_3, f_2, and f_1. Observe that the numbers are reversed to allow for a more straightforward application of the Representer Theorem.

5.1 The Kernel Network

In the previous section, the Neural-Kernel structure was introduced. Recall that after a pass through a Neural-Kernel Block, the vector obtained is a representation of $x \in \mathbb{R}^d$ in a feature space. A Fully Connected Layer is added at the end of the block to solve the classification problem.

The Representer Theorem for L layers $L = 3$ to obtain an explicit expression for the *Kernel network*. Below we describe a forward pass through this structure:

$$\begin{tikzcd}
X \arrow[r, "f_3"] & \Omega \arrow[r, "f_2"] & \Phi \arrow[r, "f_1"] & Y \\
x_i \arrow[r, mapsto] & z_i = f_3(x_i) \arrow[r, mapsto] & t_i = f_2(z_i) \arrow[r, mapsto] & y_i = f_1(t_i)
\end{tikzcd}$$

Hence, the building block of a Kernel Network consists of the input and the three Hilbert spaces.

The Representer Theorem from Chapter 4. We define $h = f_1 \circ f_2 \circ f_3$. If this theorem is applied, the following expressions for f_3, f_2, f_1 are obtained:

$$f_3(\cdot) = \sum_{k=1}^{N}\sum_{o=1}^{d_3} c_{k,o}^{(3)} K^{(3)}(\cdot, x_k) e_o$$

$$g(\cdot) = f_2 \circ f_3(\cdot) = \sum_{j=1}^{N}\sum_{l=1}^{d_2} c_{j,l}^{(2)} K^{(2)} \left(\sum_{k=1}^{N}\sum_{o=1}^{d_3} c_{k,o}^{(3)} K^{(3)}(x_j, x_k) e_o, \cdot \right) e_l$$

$$h(\cdot) = f_1(\cdot) = \sum_{i=1}^{N}\sum_{p=1}^{d_1} c_{i,p}^{(1)} K^{(1)}(g(x_i), \cdot) e_p$$

where e_o, e_l, e_p are the respective unit vectors i.e. $e_o = (0, \ldots, 1, \ldots, 0)$ in the o-th coordinate.

Table 5.1 summarizes the dimensions as well as the number of parameters in the Kernel Network

Trainable parameters	Neural Layer	Kernel Layer	Fully Connected Layer
Weights	$d_3 \times d$	0	$d_2 \times d_1$
Coefficients	$N \times d_3$	$N \times d_2$	$N \times d_1$

Table 5.1: Number of parameters in a Neural-Kernel Block + Fully Connected Layer

A more simplified equivalent expression of the kernel network is:

$$f_1(\cdot) = \sum_{i=1}^{N} \sum_{p=1}^{d_1} c_{i,p}^{(1)} \mathcal{K}(x_i, \cdot) e_p \tag{5.1}$$

where the outer kernel $\mathcal{K}$ is defined as

$$\mathcal{K}(x, x') = \mathcal{K}(f_2 \circ f_3(x), f_2 \circ f_3(x'))$$

5.2 The RKHS in a Kernel Network

In the kernel network ,we start by making the following observation:

The first forward pass is defined as, $f(x) = Wx + b$, with $x \in \mathbb{R}^d$, $b \in \mathbb{R}^{d_3}$. In order to include the bias in the computation of the Kernel, we define $\tilde{W} = (b \quad W)^T$, meaning that $W \in \mathbb{R}^{(d_3+1)\times d}$ Observe that now $x = (1 \quad x_1 \quad \dots \quad x_d)^T$.

The linear kernel $k(x, y) = x^T y$ for $x, y \in \mathbb{R}^d$ can be extended to a more general kernel $k'(x, y) = x^T Ay$ where A is a psd matrix. (If $A = I_d$, $k'(x, y)$ is the linear kernel)

However, there is an important difference between the linear kernel and the general linear kernel. The linear kernel is just the dot product between two vectors, and the feature mapping, which is the identity, does not change the dimension of the space. This is why it is referred sometimes as a non-kernel. The general linear kernel does map the data points into another space, as its feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ is defined as $\phi(x) = W^T x$ (see propositions below)

Proposition 5.1. Given $x, y \in \mathbb{R}^d$, then $k(x, y) = x^T Ay$ where $A \in \mathbb{R}^{d\times d}$ is psd, is a kernel, and is called the general linear kernel.

Proof. k is a kernel if it is symmetric and positive semi definite. If A is psd , we can write $A = WW^T$ where W does not to be a square matrix. Now, for $x_1, \dots, x_N \in \mathbb{R}^d$, construct the matrix $K = (k_{ij})_{i,j=1,\dots,N}$ defined as $k(x_i, x_j) = x_i^T Ax_j$. Then, for every $c \in \mathbb{R}^N$:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k_{ij} = \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j x_i^T A x_j = \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j (x_i^T W)(W^T x_j) =$$
$$= \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j (W^T x_i)^T (W^T x_j) = \left\| \sum_{i=1}^{N} c_i (W^T x_i) \right\|^2 \geq 0$$

□

What is next is to describe the RKHS of the general linear kernel. In the proof above, we have seen that an inner product can be derived from the kernel $k(x, y) = x^T WW^T y$, defined as $\langle \phi(x), \phi(y) \rangle = \langle Wx, Wy \rangle_{\mathbb{R}_p}$. A candidate RKHS is needed. Given $f \in \mathcal{H}$,

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x) = \sum_{i=1}^{N} \alpha_i \langle W^T x_i, W^T x \rangle_{\mathbb{R}^p} = \left\langle \sum_{i=1}^{N} \alpha_i W^T x_i, W^T x \right\rangle_{\mathbb{R}^p} =$$
$$= \langle \alpha_1 W^T x_1 + \dots + \alpha_n W^T x_n, W^T x \rangle_{\mathbb{R}^p}$$

Clearly $\alpha_1 W^T x_1 + \dots + \alpha_n W^T x_n = s \in \mathbb{R}^p$ So, the functions are of the form $f(x) = \langle s, W^T x \rangle_{\mathbb{R}^p}$, with $s \in \mathbb{R}^p$. The candidate $\mathcal{H}$ is defined to have inner product $\langle f_1, f_2 \rangle_{\mathcal{H}} =$

$s_1^T s_2$, denoting the standard inner product in $\mathbb{R}^p$. That $\mathcal{H}$ is a Hilbert space is clear because $\mathcal{H}$ is isomorphic to $\mathbb{R}^p$, which is a Hilbert space. The remaining step is to check that it is indeed an RKHS:

- $\mathcal{H}$ contains $K_x = K(x, \cdot) : \mathbf{y} \mapsto \langle Wx, W\mathbf{y}\rangle$
- $f_s \in \mathcal{H}$, $x \in X$:

$$f(x) = \langle s, W^T x\rangle_{\mathbb{R}^p} = \langle f_s, f_{W^T x}\rangle_{\mathcal{H}} = \langle f_s, k_x\rangle$$

The following proposition summarizes this reasoning.

Proposition 5.2. The RKHS of the general linear kernel is the space of functions $f(x) = \langle s, W^T x\rangle_{\mathbb{R}^p}$, for $s \in \mathbb{R}^p$. The RKHS is endowed with the inner product $\langle f_{W,x}, f_{W,y}\rangle = \langle W^T x, W^T y\rangle$. The norm induced by the inner product is $\|f\|^2 = x^T A x$, where $A = WW^T$.

The RKHS of the Gaussian RBF Kernel can be quite complex if a rigorous approach is taken. Find details in Chapter 4 of [23]. Considering that the RKHS is already known, few things can be said. However, recall that in ther kernel network, the RBF Kernel is approximated through random Fourier features. The feature map is $z : X \to \mathcal{H}'$ defined as $z(x) = \cos(\omega^T x)$ where $\omega \sim N(0, \sigma^2 I_d)$ The approximate RBF kernel $k(x, y) \approx z(x)^T z(y)$ defines an RKHS H' that may not be contained in the RKHS $\mathcal{H}$ defined by the Gaussian RBF Kernel [21].

For the last layer, which is fully connected, observe that its equations are defined by $y = \sigma(Wh_2 + b)$, with $W \in \mathbb{R}^{d_2 \times d_1}$, and where $\sigma : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1}$ is an activation function. The question is if it possible to describe a pass through a fully connected with a function from an RKHS. In other words, an RKHS that contains the class of functions $\sigma(\langle w, x\rangle)$ for $w, x \in \mathbb{R}^{d_2}$. We have already characterized a neural pass has activation function.

The article titled *Convexified Convolutional Neural Networks* [11] by Zhang et al. aims to desribe a class of convolutional neural networks that are a convex optimization problem. This is done by developing the innovative idea that non linear convolutional filters ($h(z) = \sigma(\langle \omega, z\rangle)$) are seen as vector of a reproducing kernel hilbert space. An interesting result linked to statistical analysis is proven: For two-layer convolutional neural networks, the generalization error obtained by a convexified CNN converges to the generalization error of the best possible CNN. Establishing this strong bridge between deep neural networks and kernel methods comes with finding reproducing kernel hilbert spaces that can contain functions that represent the convolutional opreation. The authors introduce two important results, which we can study and apply to our problem. In the context of CNNs, the filters $h : z \mapsto \sigma\langle w, z\rangle$, for w, z in $\mathbb{R}^{d_2}$ are shown to be elements of the RKHS induced by the gaussian RBF kernel and the *inverse polynomial kernel*. Let us review the architecture they use to see if it can be compared to the kernel network.

In [11] the different building blocks of a CNN are described in the following way: Given an input $x \in \mathbb{R}^{d_0}$ a collection of patches $\{z_p(x)\}_{j=1}^P$ of the full input vector x. For each vector, $z_p(x) \in \mathbb{R}^{d_1}$. The convolutional operation is defined as:

$$h_j(z) := \sigma(\omega_j^T z) \quad z \in \mathbb{R}^{d_1} \tag{5.2}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$. Observe that $\omega_j \in \{\omega_j\}_{j=1}^r$. In the CNN context, each function h_j is called a filter. Observe that that $h(z) \in \mathbb{R}^r$. The final expression for this CNN, a vector

$f(x) = (f_1(x), \ldots, f_{d_2}(x)) \in \mathbb{R}^{d_2}$:

$$f_k(x) = \sum_{j=1}^{r} \sum_{p=1}^{P} \alpha_{k,j,p} h_j(z_p(x))$$

Observe the similarities between this framework and the last layer of a kernel network. In a kernel network, the fully connected layer can be written as:

$$f_j(x) = \sigma(\omega_j^T x) \quad \text{for } j = 1, \ldots, d_1 \tag{5.3}$$

i.e. a vector $f(x) = (f_1(x), \ldots, f_{d_1}(x))$

The equations for a Fully Connected Layer in the Deep hybrid model [5] are $s = \sigma(Wh_2 + b)$. Observe that the problem can be reducec to $s_i = \sigma_i(wx + b)$ for $w \in \mathbb{R}^{d_2}$. Applying the represnter theorem, we can obtain:

$$\sigma(\langle \omega, \cdot \rangle)_j = \sum_{i=1}^{N} c_i K(f_2(x^i), f_2(\cdot))_j \quad j = 1, \ldots, N$$

In this case, σ belongs to the RKHS induced by kernel K.

However, this is not a trivial matter. In [11] the explicit feature map for both kernels is needed to prove Lemma 5.4 and 5.5. (In [11], the RBF Kernel is defined as $\phi : \mathbb{R}^{d_1} \to l_2(\mathbb{N})$, which is the definition given in this thesis)

There exists a restriction over the activation function σ. A polynomial expansion is assumed: $\sigma(x) = \sum_{j=0}^{\infty} a_j t^j$. The RKHS of the gaussian RBF kernel captures activation functions that are either a polynomial or a sinuosidal function (which can be approximated by the polynomial). The polynomial expansions of the sigmoid function and the ReLU are not contained in the RKHS of the gaussian RBF kernel [11]. This is because the coefficients of the polynomial expansion do not converge quickly enough to zero.

These lemmas used in a CNN context study if the RKHS of these two respective kernels contain the function $h : z \mapsto \sigma(\langle \omega, z \rangle)$. As explained before, we consider an equivalent in the fully connected layer of the kernel network, where the last part of it is defined as a vector $x \in \mathbb{R}^{d_1}$ of components $x = (\sigma(\langle w_{1,\cdot}, x \rangle)), \ldots, \sigma(\langle w_{d_1,\cdot}, x \rangle))$, where $w_{1,\cdot}$ are the rows of the weight matrix.

Definition 5.3. The inverse polynomial kernel, for $x, x' \in \mathbb{R}^d$ is defined as

$$k(x, x') = \frac{1}{2 - \langle x, x' \rangle} \tag{5.4}$$

The following two lemmas have very similar proofs, so only the proof for the RBF kernel is reproduced.

Lemma 5.4. Assume that the function $\sigma(x)$ has a polynomial expansion $\sigma(t) = \sum_{j=0}^{\infty} a_j t^j$. Let $C_\sigma(\lambda) := \sqrt{\sum_{j=0}^{\infty} 2^{j+1} a_j^2 \lambda^{2j}}$. If $C_\sigma(\|\omega\|_2) < \infty$, then the RKHS induced by the inverse polynomial kernel contains the function $h : z \mapsto \sigma(\langle \omega, z \rangle)$ with Hilbert norm $\|h\|_{\mathcal{H}} = C_\sigma(\|\omega\|_2)$

Proof. Found in [11]. □

We assume $\|x\|^2 = \|x'\|^2 = 1$ for this lemma on the gaussian RBF kernel.

Lemma 5.5. Assume that the function $\sigma(x)$ has a polynomial expansion $\sigma(t) = \sum_{j=0}^{\infty} a_j t^j$. Let $C_\sigma(\lambda) := \sqrt{\sum_{j=0}^{\infty} \frac{j! e^{2\gamma}}{(2\gamma)^j} a_j^2 \lambda^{2j}}$. If $C_\sigma(\|\omega\|_2) < \infty$, then the RKHS induced by the Gaussian kernel contains the function $h : z \mapsto \sigma(\langle \omega, z \rangle)$ with Hilbert norm $\|h\|_{\mathcal{H}} = C_\sigma(\|\omega\|_2)$

Proof. Recall the feature mapping of the gaussian RBF kernel $\phi : \mathbb{R}^d \to l^2(\mathbb{N})$:

$$\phi(x) = e^{-\gamma\|x\|^2} \left(\sqrt{\frac{2^k \gamma^k}{k!}} \prod_{i=0}^{k} x_{j_i} \right) \tag{5.5}$$

(to the k-th coordinate) By assumption, $\|x\|^2 = \|x'\|^2 = 1$. Observe that the feature map can be written as follows. Similarly, a vector $\overline{\omega} \in l^2(\mathbb{N})$ is defined as

$$\overline{\omega} = e^{\gamma} \left(\sqrt{\frac{2^k \gamma^k}{k!}} \right)^{-\frac{1}{2}} a_j \prod_{i=0}^{k} \omega_{j_i}$$

Now, the image of $\langle w, x \rangle$ through the activation function is computed

$$\sigma(\langle \omega, x \rangle) = \sum_{j=0}^{\infty} a_j (\langle \omega, z \rangle)^j = \sum_{j=0}^{\infty} a_j \sum_{(j_1,\dots,j_k)\in[d_1]^k} \omega_{j_1} \cdot \dots \cdot \omega_{j_k} x_{j_1} \cdot \dots \cdot x_{j_k} = \langle \overline{\omega}, x \rangle \tag{5.6}$$

where the definition for the polynomial kernel is applied. Now, observe

$$e^{\gamma} \left(\sqrt{\frac{2^k \gamma^k}{k!}} \right)^{-\frac{1}{2}} e^{-\gamma} \left(\sqrt{\frac{2^k \gamma^k}{k!}} \right) = 1 \tag{5.7}$$

which is why $\overline{\omega}$ can be inferred from the product after the second equal sign of 5.6. This shows $h \in \mathcal{H}$ The second part of the proof has to do with verifying hilbert norm of $\mathcal{H}$:

$$\|\overline{\omega}\|_2^2 = \sum_{k=0}^{\infty} \frac{k! e^{2\gamma}}{(2\gamma)^k} a_k^2 \sum_{(j_1,\dots,j_k)\in[d_1]^k} \overline{\omega}_{j_1}^2 \overline{\omega}_{j_2}^2 \cdot \overline{\omega}_{j_k}^2 = \sum_{k=0}^{\infty} \frac{k! e^{2\gamma}}{(2\gamma)^k} a_j^2 \|\overline{\omega}\|_2^{2j} = C_\sigma^2(\|\overline{\omega}\|_2) < \infty$$

This concludes the proof, $\|h\|_{\mathcal{H}} = \|\overline{\omega}\|_2 = C_\sigma(\|\overline{\omega}\|_2)$ $\square$

Now, in the CNN framework described in [11], as well as ours, the representer theorem can be applied,

$$f_3(t)_j = \sigma(\langle \omega, t \rangle)_j = \sum_{i=1}^{N} c_i K(f_3(t), f_3(t_i))_j \quad j = 1, \dots, d_1$$

i.e. there exists a kernel function $K : \mathbb{R}^{d_1} \times \mathbb{R}^{d_1} \to \mathbb{R}$ such that $K(z, z') = \langle \phi(z), \phi(z') \rangle$, such that

$$\sigma(\langle \omega_j, z \rangle) = \langle \overline{\omega}_j, \phi(z) \rangle \tag{5.8}$$

However, both $\phi(z)$ and $\overline{\omega}_j$ belong to $l^2(\mathbb{N})$. The problem needs to be reduced to a finite-dimensional one, which the author tackles by projecting onto a linear space, the one spanned by $\phi(f_2(x_i))$.

Note that this is not done for the kernel network, as the idea is to replicate the deep hybrid model, where the last layer is a fully connected layer. This is considered as possible and important extension of the model.

5.3 Optimization of the Kernel Network

Notation. Sometimes, in attempt to simplify notation, it can be useful to refer to the row of the coefficient matrix $\mathbf{c}^{(i)}$ for $i = 1, 2, 3$. We write either $c_k = c_{k,\cdot}$ where k is the row.

The optimization problem described in Section 5.1 and extended in **??** is solved by minimizing the following functional given by the Representer Theorem:

$$J_{\lambda,\mu,\nu}(f_1, f_2, f_3) = \sum_{j=1}^{N} L(f_1 \circ f_2 \circ f_3(x_j), y_j) + \lambda \|f_1\|^2_{\mathcal{H}(\Phi,\mathbb{R}^{d_1})} + \mu \|f_2\|^2_{\mathcal{H}(\Gamma,\Phi)} + \nu \|f_3\|^2_{\mathcal{H}(\Omega,\Gamma)} \tag{5.9}$$

As explained in previous section, recall that the definition of matrix-valued kernels through its scalar-valued used in this thesis is:

$$\mathbf{K}(x_i, x_j) = K(x_i, x_j)A$$

where A is a diagonal matrix. $A = \text{diag}(a_1, \ldots, a_n)$

Let us compute the squared norm of f_3 (the rest are analogous):

$$\begin{aligned} \|f_3\|^2_\Gamma &= \left\langle \sum_{k=1}^{N} \sum_{o=1}^{d_3} c^{(3)}_{k,o} K(\cdot, x_k) e_o, \sum_{k'=1}^{N} \sum_{o=1}^{d_3} c^{(3)}_{k',o} K(\cdot, x'_k) e_o \right\rangle = \\ &= \sum_{k=1}^{N} \sum_{k'=1}^{N} \left\langle \sum_{o=1}^{d_3} c^{(3)}_{k,o} K(\cdot, x_k) e_o, \sum_{o=1}^{d_3} c^{(3)}_{k',o} K(\cdot, x'_k) e_o \right\rangle = \\ &= \sum_{k=1}^{N} \sum_{k'=1}^{N} \sum_{o=1}^{d_3} c^{(3)}_{k,o} c^{(3)}_{k',o} k(x_k, x'_k) \end{aligned}$$

where we have observed that $e_i^T e_j = \delta_{ij}$. To ease the notation, we can write:

$$\|f_3\|^2_\Gamma = c^T K c = \sum_{k,k'=1}^{N} c^{(3)}_{k,\cdot} k(x_k, x'_k) c^{(3)}_{k',\cdot} \tag{5.10}$$

As explained previously the Kernel Network has 5 sets of parameters:

$$\begin{aligned} \mathbf{c^{(1)}} &= \{c^{(1)}_{i,p} \mid i = 1, \ldots, N, p = 1, \ldots d_1\} \\ \mathbf{W^{(1)}} &= W \in \mathbb{R}^{d_1 \times d_2} \\ \mathbf{c^{(2)}} &= \{c^{(1)}_{j,l} \mid j = 1, \ldots, N, l = 1, \ldots d_1\} \\ \mathbf{c^{(3)}} &= \{c^{(3)}_{k,o} \mid k = 1, \ldots, N, o = 1, \ldots d_3\} \\ \mathbf{W^{(3)}} &= W \in \mathbb{R}^{d_3 \times d} \end{aligned}$$

Let us discuss the derivatives of the functional with respect to the parameters of the model. As it is clear, layers $L = 1, 2$ have parameters that depend from $L = 3$. Therefore, full expressions for the derivatives $\frac{\partial J}{\partial c^{(3)}_{k,o}}$ and $\frac{\partial J}{\partial w^{(3)}_{k,o}}$ contain the derivatives with respect to the other parameters of the kernel network and can be adapted accordingly. Ignoring the derivative of the cross-entropy (which is already known), we apply the chain rule.

$$\begin{aligned}\frac{\partial J}{\partial f_2 f_3(x_i)} &= \frac{\partial J}{\partial L}\frac{\partial L}{\partial f_1 f_2 f_3(x_i)}\frac{\partial f_1 f_2 f_3(x_i)}{\partial f_2(f_3(x_i))} = \\ &= \frac{\partial J}{\partial L}\frac{\partial L}{\partial f_1 f_2 f_3(x_i)}\sum_{i'=1}^{N}\sum_{p=1}^{d_1} c_{i,p}^{(1)}\frac{\partial}{\partial f_2 f_3(x_i)}\left[f_2(f_3(x_i)\right]^T W^T W f_2(f_3(x_i')) \end{aligned}$$

The derivative process can stop here (if f_3 is considered a constant) or continue

$$\frac{\partial f_2 f_3(x_i)}{\partial f_3(x_i)} = \sum_{j=1}^{N}\sum_{l=1}^{d_2} c_{j,l}^{(2)}\frac{\partial}{\partial f_3(x_i)}\varphi(f_3(x_i))^T\varphi(f_3(x_j)) \tag{5.11}$$

And now the expression of $f_3(x_i)$ given by the representer theorem would be used in case the goal is to obtain $\frac{\partial J}{\partial c_{k,o}^{(3)}}$, for instance.

As it is standard in neural network training $L(\theta) = \sum_{i=1}^{N} L_i(\theta)$, meaning that the derivatives are computed with respect to the functions evalueated at each data point x_i. Note that the partial derivative with respect to a coefficient is computed for each data point $i = 1, \dots, N$. This explains the i subscript.

Algorithm 1 below states the training process of the Kernel Network.

Algorithm 1 Learning the Kernel Network (with one Neural-Kernel Block)

Input. Data $\{(x_i, y_i)\}_{i=1}^{n}$, dimensions d_1, d_2, variance σ.

1. Initialize weight matrices $W \in \mathbb{R}^{d_3 \times d}, V \in \mathbb{R}^{d_2 \times d_1}$, as well as coefficient matrices $\mathbf{c^{(1)}} \in \mathbb{R}^{N \times d_1}, \mathbf{c^{(2)}} \in \mathbb{R}^{N \times d_2}, \mathbf{c^{(3)}} \in \mathbb{R}^{N \times d_3}$

2. Construct the kernel matrices $K_3 \in \mathbb{R}^{d \times d_3}$, $K_2 \in \mathbb{R}^{d_3 \times d_2}$, $K_1 \in \mathbb{R}^{d_2 \times d_1}$ while computing f_3, f_2, f_1 in this order for each $x \in X$:

$$\begin{aligned} K_3(x, x') &= x^T W^T W x' & f_3 x &= \sum_{k=1}^{N} c_k^{(3)} k(x_k, x) \\ K_2(f_3 x, f_3 x') &= \phi(f_3 x)^T \phi(f_3 x') & f_2 f_3 x &= \sum_{j=1}^{N} c_j^{(2)} k(f_3 x_j, f_3 x) \\ K_1(f_2 f_3 x, f_2 f_3 x') &= (f_2 x)^T V^T V f_2 x' & f_3 f_2 f_1 x &= \sum_{i=1}^{N} c_i^{(1)} k(f_3 f_2 x_i, f_3 f_2 x) \end{aligned}$$

3. Solve the following optimization problem:

$$\hat{h} \in \arg\min J(h)_{\lambda,\mu,\nu} \quad \text{where } J(h) = \sum_{i=1}^{N} L(h(x_i), y_i) + \lambda \|f_1\|_{\mathcal{H}_1}^2 + \mu \|f_2\|_{\mathcal{H}_2}^2 + \nu \|f_3\|_{\mathcal{H}_3}^2 \tag{5.12}$$

Output. A prediction $h(x) = \sum_{i=1}^{N} c_i^{(1)} \mathcal{K}(x_i, x)$

The optimization problem 5.12 involves performing backpropagation over the different sets of parameters described. Backpropagation is done using a gradient descent algorithm. In the next chapter, performance is evaluated using the following two algorithms: stochastic gradient descent and Adam. In summary, if θ denotes the set of all parameters in the kernel network:

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L$$

where η is the learning rate.

On a final note, recall that [5] and [6], the functional minimized is

$$J(W_1, W_2, b_1, b_2) = \frac{1}{n} \sum_{i=1}^{n} L(x^i, y^i) + \gamma [Tr(W_1 W_1^T) + Tr(W_2 W_2^T)] \tag{5.13}$$

Observe that the regularizer only affects the matrices of the neural layers.

6 Experimental Results

In this section, the accuracy and the generalization capability of the kernel network model, as well as other metrics are put to the test. This is performed in comparison with two other Machine Learning model: The deep hybrid model (The core Neural-Kernel model) presented in [5, 6] and the multi-layer perceptron (MLP)

In the Experimental Results section of [5] the deep hybrid model is put to test in 15 different data sets, varying in number of features and in size. The comparison of the deep hybrid model (One neural-kernel block or two) is made with two different models: neural networks (One hidden layer or two). and Shallow LS-SVM (primal and dual). In [6], the experiments section is dedicated to exploring any positive differences between the extended deep hybrid model (deep Neural-Kernel architectures: average, maxout and CNN) and the original deep hybrid model. Also, accuracies on the test set are stated for LS-SVM and TROP-ELM. In this second reference, the number of data sets put to the test is reduced to 12, a subset of the ones tested in the first reference, together with the Motor data set.

Table 6.1 lists the different data sets (retrieved from the UCI ML repository) tested in this report, with their dimension and number of attributes.

Data set	N	Partition train-test	Features	Classes
Australian	690	552/138	14	2
Titanic	2201	1760/441	3	2
Sonar	208	166/42	60	2
CNAE-9	1080	864/216	856	2
Adults [3]	1100	880/220	93	2

Table 6.1: Details of data sets used during experiments

The author of the deep hybrid model only provides the specific parameters for CNAE-9 dataset, which are listed in [5]. These are $h_1 = 300$, $h_2 = 400$, $\sigma = 0.7$ and an unspecified learning rate. In page 50-51 of [5], it is stated that the parameters for h_1 and h_2 for all the data sets are found using random search strategy. The author seems to indicate that he uses stochastic gradient descent (SGD) as the algorithm that optimizes the parameters of the deep hybrid model. The author claims accuracy results that are summarized in table 6.2

Deep Hybrid Model		Neural Networks	
One Neural-Kernel Block	Two Neural-Kernel Blocks	One layer	Two layers
0.93 ± 0.01	0.94 ± 0.02	0.91 ± 0.02	0.92 ± 0.01

Table 6.2: Results for CNAE-9 data set listed in deep hybrid model references

The deep hybrid model was also reproduced in PyTorch (cf. source code), which gave different results than the ones claimed in table above. This is important to note for the comparison, as the results stated for neural networks in [5, 6] were lower than the ones obtained.

When training the kernel network with an RFF layer, using SGD and the parameters that work for the deep hybrid model are not valid now, as shown in figures 6.1 and 6.2.

[3]Extracted random subset

In other words, the conclusion here is that the kernel network learns differently than the deep hybrid model, despite their similarities. A separate search must be conducted in order to maximize the performance of the model.

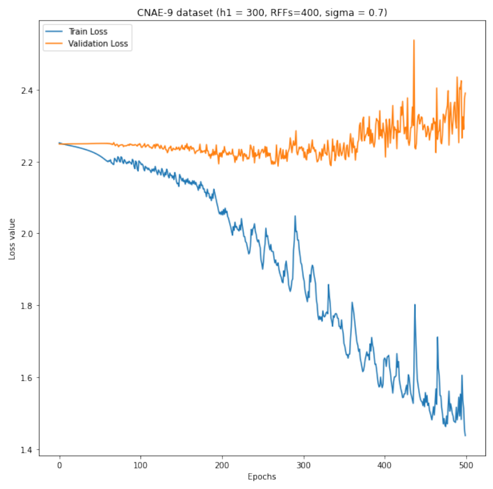

Figure 6.1: Loss for kernel network with deep hybrid model parameters

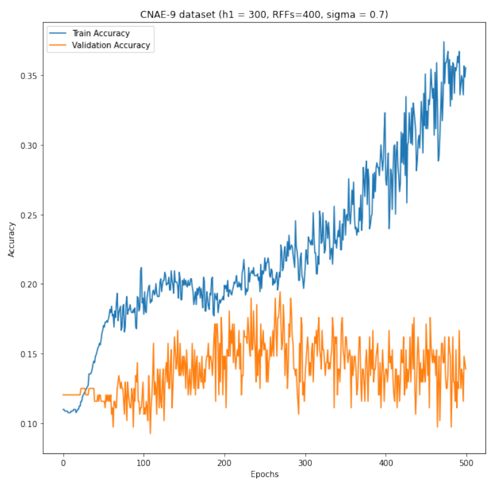

Figure 6.2: Accuracy for kernel network with deep hybrid model parameters

Reducing the number of neurons in the hidden layer to $h_1 = 50$ seems adequate to make a comparison between the different algorithms, as well as optimization strategies. As we will see, the kernel network seems to learn well with few neurons in the first layer, which is generally not the case for the deep hybrid model. This is because as shown later on, the learning happens especially through the RFF and the fully connected layer.

One of the strategies proposed is alternate optimization. Given a function $f : \mathbb{R}^s \to \mathbb{R}$ with $x = (x_1, \ldots, x_s)$, consider a partition of the variables and differentiate only with respect to those, leaving the others constant [29]. The parameters to optimize here are the weights for the neural layer and fully connected layer, and the coefficients obtained by the three layers in the kernel network. A conjecture is made that it can be beneficial to train the weights and the coefficients separately. In other words, during the even epochs, backpropagation is performed but the coefficients are considered to be constant, and hence only the weights are updated. The opposite happens for the odd epochs.

Also, two different optimizers are considered: Stochastic gradient descent and Adam. The optimizer in Pytorch refers to the algorithm used to update the parameters of the model. Some tests are performed using the CNAE-9 data set in order to compare the optimizers and the usage of alternate optimization. The results are summarized in Table 6.3.

	Algorithm	Alternate	Layer dim	Param.	Epochs	Time	Learns data	Acc.
1	SGD	Yes	$h_1 = 300, h_2 = 400$	612576	250×2	124 sec.	No	-
2	Adam	Yes	$h_1 = 300, h_2 = 400$	612576	250×2	146 sec.	No	-
3	SGD	Yes	$h_1 = 50, h_2 = 400$	396576	250×2	132 sec.	Yes	0.81
4	Adam	Yes	$h_1 = 50, h_2 = 400$	396576	250×2	127 sec.	Yes	0.68
5	SGD	No	$h_1 = 50, h_2 = 400$	443035	500	152 sec.	Yes	0.81
6	Adam	No	$h_1 = 50, h_2 = 400$	443035	500	168 sec.	Memorizes training	0.78
7	Adam	No	$h_1 = 50, h_2 = 200$	268435	500	89 sec.	Memorizes training	0.92

Table 6.3: Comparison between SGD and Adam via alternate optimization or not, with associated metrics

Figures 6.3, 6.4, 6.5 and 6.6 below show the loss function for different cases.

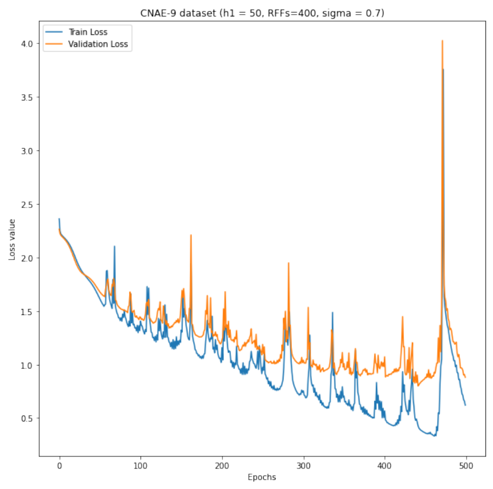

Figure 6.3: Loss for SGD with alternate optimization (3)

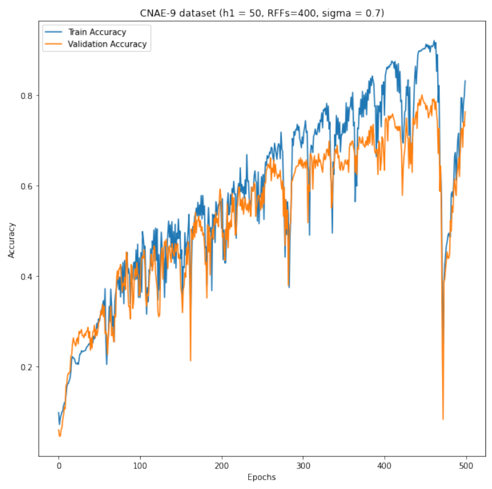

Figure 6.4: Accuracy for SGD with alternate optimization (3)

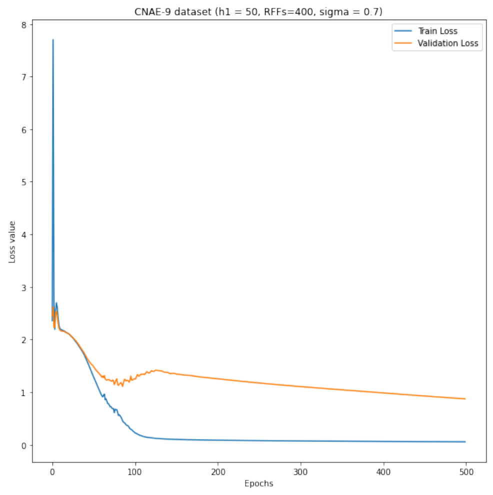

Figure 6.5: Loss for Adam with alternate optimization (6)

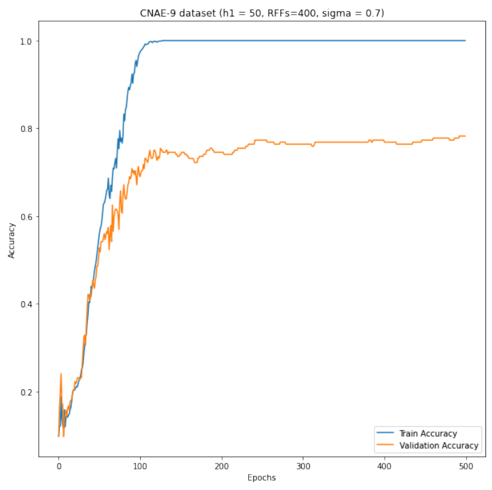

Figure 6.6: Accuracy for Adam with alternate optimization (6)

Clearly, to stabilize the training process of the kernel network is to use Adam with no alternate optimization.

6.1 Evaluating the kernel network and other ML models over different data sets

Four different models are evaluated in this section: firstly, a neural network with one layer, which is trained in two different ways: by passing the entire data set or batches of a specific dimension. The two studied hybrid models are Mehrkanoon's deep hybrid model and the kernel network.

To find the different hyperparameters, such as regularizers and layer dimensions, Random Search is used (as in the case of citing the article). For the regularizers, a uniform distribution from 1e-1 to 1e-5 is considered. As for the layer sizes, it depends specifically on each dataset, but the dimension of the first layer ranges from 5 to 300 neurons, and the number of random Fourier features layer ranges from 50 to 1000. Finally, the learning rate is also randomly searched, taking a value between 10^{-1} and 10^{-5}.

Regarding the presentation of results, once the hyperparameter values have been determined, the model is trained. To avoid bias in the results, the same train-test data partition is used for all four models. Each model is trained 50 times, with different random initializations of all parameters and coefficients each time to ensure that the presented results have minimal bias possible.

Figures 6.7 6.8 show the training of the kernel network for the australian data set. The parameters used are $\lambda = \mu = \nu = 0.01$, $h_1 = 10$, $h_2 = 100$, $\sigma = 0.6$. The learning rate is equal to 10^{-4}

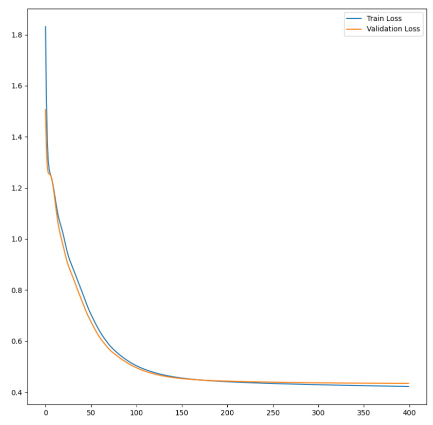

Figure 6.7: Loss for kernel network in the australian data set across epochs

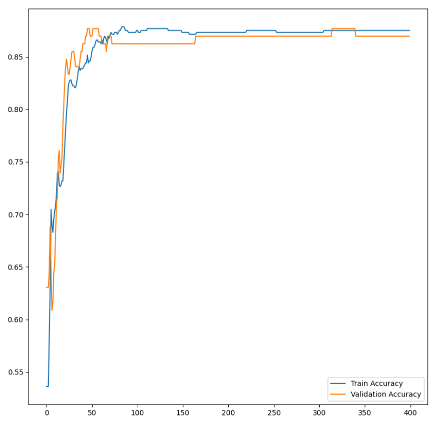

Figure 6.8: Accuracy for kernel network in the australian data set across epochs

Table 6.4 shows the comparison between 4 models: A neural network with one single layer, the deep hybrid model from Mehrkanoon and the kernel network.

Data set	NN 1 Layer (batches)	NN 1 Layer	Kernel network	Deep hyb. model
Australian	0.889 ± 0.025	0.882 ± 0.019	0.878 ± 0.013	0.873 ± 0.007
Titanic	0.780 ± 0.017	0.790 ± 0.014	$\mathbf{0.808 \pm 0.00}$	0.804 ± 0.00
Sonar	0.857 ± 0.05	0.874 ± 0.047	0.852 ± 0.05	0.848 ± 0.05
CNAE-9	0.96 ± 0.013	0.972 ± 0.003	0.888 ± 0.037	0.96 ± 0.03
Adults	0.839 ± 0.05	0.842 ± 0.051	0.805 ± 0.056	0.812 ± 0.055

Table 6.4: Results on different models

It is clear the the kernel network manages to learn at the same rate as the deep hybrid model, improving the accuracy slightly in some cases. In this case, the results are promising, since this means we have been able to replicate the deep hybrid model from a theoretical point of view. Nonetheless, so far, in most cases, the kernel network does not manage to surpass the neural network except in the Titanic data set. However, there are still many data sets to be tested, and that this model could show better performances. Also, the training process can still be refined, which has not been able to be done due to time constraints.

With regards to computational time, results for the australian data set are compiled in Table 6.5:

Data set	Training dim	NN batches	NN all	Kernel net.	Deep hyb.
Australian	690×14	$1.66s \pm 0.66s$	$0.18s \pm 0.04s$	$11.18s \pm 0.67s$	$1.15s \pm 0.28s$

Table 6.5: Running time for different models

It is clear that the kernel network is the most costly in terms of computational time. In the next chapter we consider some possible alternatives in order to reduce this cost.

Figures 6.9, 6.10, 6.11, 6.12 show the learning process for the RFF Layer and the FC Layer for the Australian data set. (parameters $\lambda = \mu = \nu = 0.01$, $h_1 = 10$, $h_2 = 100$, $\sigma = 0.7$) A fraction of 10 data points is taken in order to obtain a better visualization of the problem. Note that the class vector (true labels) are $y = [1, 0, 0, 0, 0, 0, 1, 1, 1, 0]$.

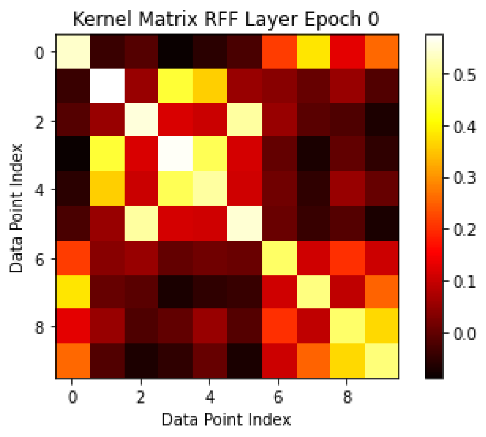

Figure 6.9: Kernel matrix for RFF Layer at epoch 0

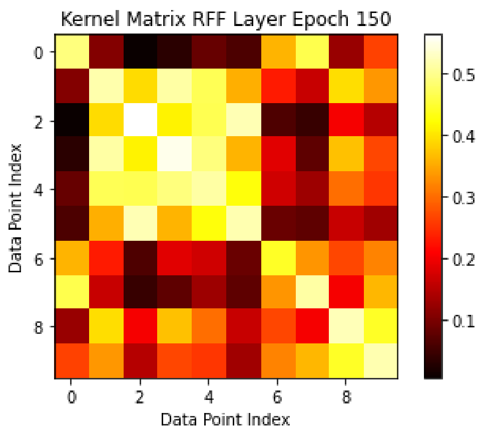

Figure 6.10: Kernel matrix for RFF Layer at epoch 150

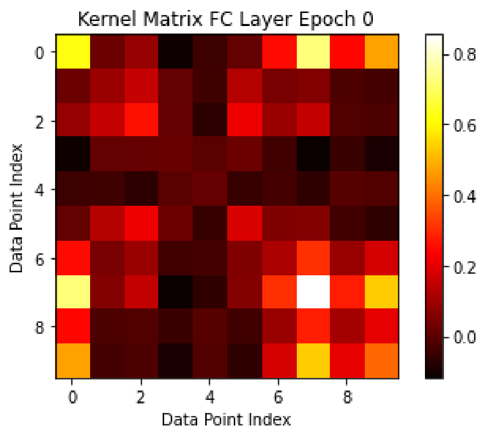

Figure 6.11: Kernel matrix for FC Layer at epoch 0

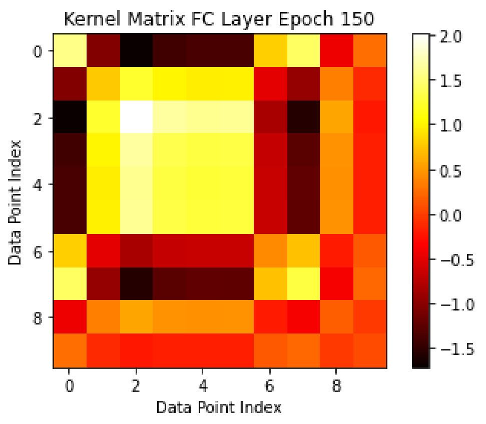

Figure 6.12: Kernel matrix for FC Layer at epoch 150

However, observe little change in the kernel matrices for the neural layer, as one can see in Figures 6.13 and 6.14.

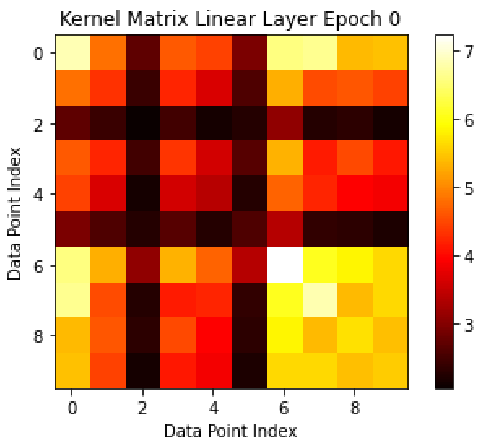

Figure 6.13: Kernel matrix for Neural Layer at epoch 0

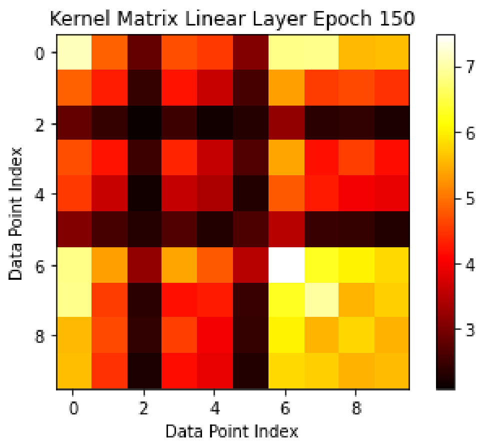

Figure 6.14: Kernel matrix for Neural Layer at epoch 150

This means that the weights of the NN layer after initizialization change very slightly. This is reinforced if we plot the norm of the weights, displayed in Figure 6.15. However,

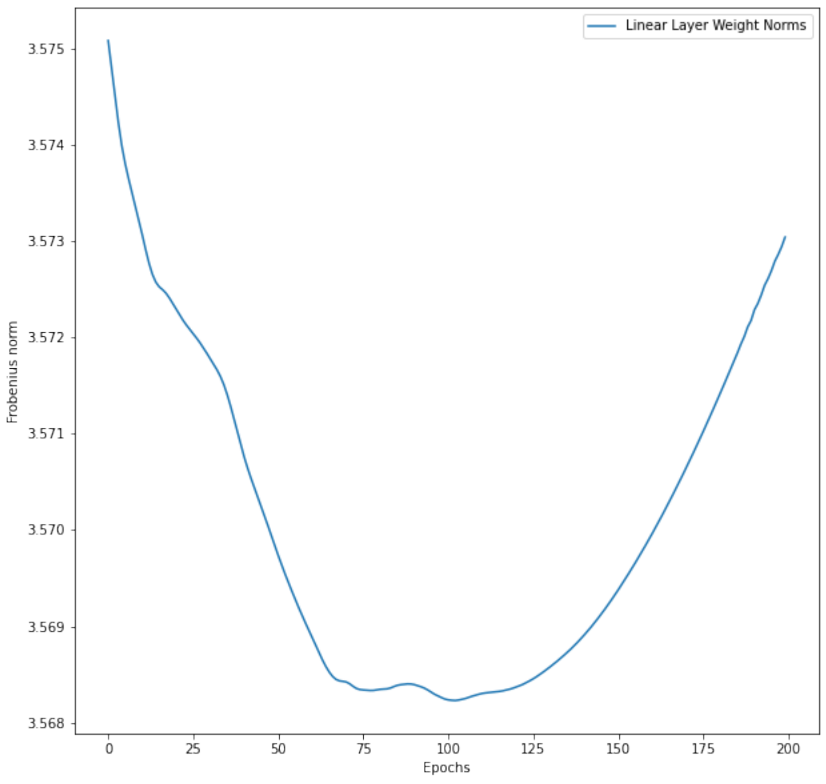

Figure 6.15: Norm of the weight matrix for the neural layer

it must be said this observation is only valid and the importance of the first layer in the whole of the model needs to be treated on a case by case basis.

7 Limitations and Future work

It is not surprising to say that the greatest limitation in this final master's thesis has been time. Once you start acquiring the necessary mathematical foundation to understand the latest models and have been able to recreate them, little time is left to present original ideas that combine the research conducted so far. And most importantly, ideas that work.

The theoretical path is long and even tortuous, but it allows us to establish foundations to subsequently build a coherent model based on this theory. Therefore, the author has deemed it necessary to develop a theory of kernel methods faithful to the underlying mathematics, and although the practical part may have been somewhat limited, the possibilities for expansion are remarkable. This chapter aims to list some of the proposals that have been left open-ended and can guide us towards a more definitive "Kernel Network" model. To do so, we rely on two articles that have proposed some ideas similar to those presented in this thesis.

As it is clear, one of the main problems as of writing is the scalability of the kernel network. The representer theorem for deep kernel learning guarantees that the minimizer has a linear expression that is the composition of several functions belonging to different RKHS. The model presented is able to learn well for small and some middle size datasets, but the amount of parameters it has severly impacts the computational time when the model. However, it must be said that no efforts have been made to optimize the algorithm, a complicated task that could be a starting point in order to reduce training time. Pytorch (the python framework used to develop all the different models), besides providing has an optimized backpropagation algorithm, which explains the low training time for NN-type models.

Another suggestion that could try and improve the computational time is to reconsider the linear layers. Chapter 5 has shown that there is an RKHS describing a neural layer, however, the functions belonging to the RKHS could be an unnecessary complication to the model, as they add parameters and do not provide a better representation of the data.

The article titled "Stacked Kernel Networks" (2017) [30] is one of the other papers that has had a significant influence on this master's thesis. It presents the "Stacked Kernel Network" model, a model that clearly falls into the category of hybrid models. Using the theory of RKHS (Reproducing Kernel Hilbert Spaces), it starts with a kernel layer after the input that is specified. This is followed by a neural layer, which defines the projection into a linear space. This is a main difference from the study conducted in the previous pages, differences that are clear in the following formula:

$$\begin{tikzcd} X \arrow[r, "f_3"] & \Omega \arrow[r, "f_2"] & \Phi \\ x_i \arrow[r, mapsto] & z_i = f_3(x_i) \arrow[r, mapsto] & t_i = f_2(z_i) \end{tikzcd}$$

$$\begin{tikzcd} X \arrow[r, "\phi"] & \mathcal{H}^{(1)} \arrow[r, "W"] & W^{(1)} \\ x_i \arrow[r, mapsto] & z_i = \phi(x_i) \arrow[r, mapsto] & t_i = W^T\phi(z_i) \end{tikzcd}$$

As we have demonstrated throughout this thesis, using the representation theorem involves a very large load of parameters that you carry along during training. For this reason, the authors of the article propose three methods to represent the functions of the RKHS. First, the same one that we said, that is, each output of a kernel layer is defined from a linear combination of the data points from the dataset. They call it non-parametric representation, as it depends on the data. Note that the authors use the representer theorem for scalar-valued functions, meaning that every component of the RFF layer has a function of the RKHS associated (hidden unit j and the layer l):

$$f_j^{(l)}(x) = \sum_{i=1}^{N} \alpha_i^{(l,j)} k^{(l)}(h_i^{(l-1)}, x)$$

where $k^{(l)}(\cdot,\cdot)$ is the kernel function associated with the RKHS $\mathcal{H}^{(l)}$.

As mentioned and proven in Section 5.2, the fully connected layer can be described as function from the RKHS. However, they admit that the number of parameters grows linearly with N, which is not scalable for large data sets. This is one of the main problems seen in the kernel network.

The second solution they propose is not to search in the entire RKHS, but rather in a subset of the RKHS, that being:

$$f = \{f_a(x) = k(a, x) \mid x \in \mathbb{R}^d\}$$

a is the learnable vector. This is clearly a subset of the RKHS because the kernel does not change and the function is no longer a linear combination from the span generated by the kernel evaluated at the training data points. The vector a is initialized using k-means clustering from the input samples. Observe that for a layer l, the function is

$$f_j^{(l)}(h_i^{(l-1)}) = k^{(l)}(a, h_i^{(l-1)}) \tag{7.1}$$

This could be considered in our case. In other words, discard the the parameters $c_{k,o}$ for the first layer, and learn the function $f_3(x)_j = k(a,x)_j$. Finally, the third option is to approximate the RBF kernel using RFFs, which has already been employed in this thesis giving good results.

The interpretability factor is one that is important nowadays, and is a pending discussion for this thesis. No one questions the fact that kernel methods are more interpretable than neural networks, and we have been able to visualize the learning process through the plots of the kernel matrices in the previous chapter. However, one could tackle more ambitious which is to track the values of the coefficients and see their intervention in predicting one class or another.

Another aspect could be to use the kernel function for the FC layer defined in section 5.2. As it shows succesful results in convexifying a CNN for the authors of [11]. Also, it is a better approximation to the gaussian RBF kernel than the one obtained via RFFs, as the RKHS does not change.

In summary, the kernel network has shown promising results, but can be still be tuned in many ways to improve its performance.

8 Conclusions

In this final master thesis, many different ideas have converged into an almost 40-page work: some from renowned authors in the field of machine learning, others incorporated with my own original contributions. Let us review this journey and offer some concluding thoughts.

Through the different chapters, we have studied the hybridization between deep learning and kernel methods. As we have seen from the beginning, this is an important research topic in the field of machine learning in the last decade. Having two a priori models, which have proven their success in their respective fields, the goal was to combine them from a formal perspective.

As we have shown by conducting an extensive review of the available literature - where the author admits that some papers might have not been included, considering that the literature is sometimes dense and sometimes unmanageable - this hybridization has yielded good results. Specifically, in a model that has been one of the main inspirations for this master's thesis: it has been proven that inserting a kernel function between two neural layers can give better results in classification and regression problems, since we remember that the objective of kernel methods is to send data points to a higher-dimensional space - often infinite, as we have shown throughout the chapters with mathematical rigor.

The existence of a well-known mathematical theory that underlies kernel methods - which we have reviewed, studied, and adapted to our problem over these forty pages - has allowed us to demonstrate a representation theorem for several layers, which is what defines a deep model: the concatenation of one layer behind the other. We have been able to construct a model that is defined as the composition of functions, each of which is determined by a linear combination of the data points.

As it has become clear, once the theoretically projected model has been put into practice, we have been able to verify that in most data sets, it works, producing competitive results, sometimes similar (or slightly better), than the reference model: the neural network with one layer. We have identified its main problem that should be addressed from now on: the number of parameters that need to be trained. We have proposed two ways to improve it: firstly, from the point of view of algorithm optimization. Then, from the theoretical point of view: The spaces where the theoretical minimum is sought can be reduced without having to give up the mathematical structure.

We have also seen how implementing this model leads us to analyze other hot branches of machine learning research, one of them being model interpretability. It is known that kernel methods are more interpretable than neural networks, and we have been able to visualize the learning process. The fact that we have an explicit expression of the functions we are minimizing, and conducting a study - which we have outlined in the practical part of this document - of the coefficients obtained by the representation theorem, could be a step to delve deeper into the interpretability of the predictions. This is the advantage of kernel methods, such as in support vector machines: we are able to know which vectors contribute to the prediction based on the values of the coefficients.

In summary, the presentation of this model opens several interesting paths from the research point of view. The author acknowledges some of the problems to be solved as well as the immediate needs to continue the study of the hybridization between kernel methods and neural networks, particularly, with the presented model.

References

[1] L. Belanche and M. Ruiz, "Bridging deep and kernel methods. a: European symposium on artificial neural networks. "esann2017: 25th european symposium on artificial neural networks: Bruges, belgium, april 26-27-28".," pp. 1–10, 2017.

[2] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing, "Deep kernel learning," 2015.

[3] T. Wang, L. Zhang, and W. Hu, "Bridging deep and multiple kernel learning: A review," *Information Fusion*, vol. 67, pp. 3–13, 2021.

[4] M. Gönen and E. Alpaydin, "Multiple kernel learning algorithms," *Journal of Machine Learning Research*, vol. 12, no. 64, pp. 2211–2268, 2011.

[5] S. Mehrkanoon and J. A. Suykens, "Deep hybrid neural-kernel networks using random fourier features," *Neurocomputing*, vol. 298, pp. 46–54, 2018.

[6] S. Mehrkanoon, "Deep neural-kernel blocks," *Neural Networks*, vol. 116, pp. 46–55, 2019.

[7] J. Zhuang, I. W. Tsang, and S. C. Hoi, "Two-layer multiple kernel learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 909–917, PMLR, 11–13 Apr 2011.

[8] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature Cell Biology*, vol. 521, pp. 436–444, May 2015.

[9] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in Neural Information Processing Systems* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), vol. 20, Curran Associates, Inc., 2007.

[10] C. A. Micchelli and M. A. Pontil, "On learning vector-valued functions," *Neural Comput.*, vol. 17, p. 177–204, jan 2005.

[11] Y. Zhang, P. Liang, and M. J. Wainwright, "Convexified convolutional neural networks," 2016.

[12] Y. Miche, M. van Heeswijk, P. Bas, O. Simula, and A. Lendasse, "Trop-elm: A double-regularized elm using lars and tikhonov regularization," *Neurocomputing*, vol. 74, no. 16, pp. 2413–2421, 2011. Advances in Extreme Learning Machine: Theory and Applications Biological Inspired Systems. Computational and Ambient Intelligence.

[13] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, jun 2008.

[14] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *Computational Learning Theory* (D. Helmbold and B. Williamson, eds.), (Berlin, Heidelberg), pp. 416–426, Springer Berlin Heidelberg, 2001.

[15] B. Bohn, M. Griebel, and C. Rieger, "A representer theorem for deep kernel learning," 2018.

[16] F. Dinuzzo, "Kernel machines with two layers and multiple kernel learning," *CoRR*, vol. abs/1001.2709, 2010.

[17] F. Dinuzzo, "Learning functions with kernel methods," 2011.

[18] A. Argyriou and F. Dinuzzo, "A unifying view of representer theorems," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 748–756, PMLR, 22–24 Jun 2014.

[19] S. Diwale and C. Jones, "A generalized representer theorem for hilbert space - valued functions," 2018.

[20] D. J. Sutherland and J. Schneider, "On the error of random fourier features," 2015.

[21] F. Liu, X. Huang, Y. Chen, and J. A. K. Suykens, "Random features for kernel approximation: A survey on algorithms, theory, and beyond," 2021.

[22] J. Pennington, F. X. X. Yu, and S. Kumar, "Spherical random features for polynomial kernels," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[23] I. Steinwart and A. Christmann, *Support Vector Machines.* Springer Publishing Company, Incorporated, 1st ed., 2008.

[24] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[25] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics).* Berlin, Heidelberg: Springer-Verlag, 2006.

[26] A. Cotter, J. Keshet, and N. Srebro, "Explicit approximations of the gaussian kernel," 2011.

[27] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, "Kernels for vector-valued functions: a review," 2012.

[28] "Approximation with matrix-valued kernels and highly effective error estimators for reduced basis approximations."

[29] J. C. Bezdek and R. J. Hathaway, "Some notes on alternating optimization," in *Advances in Soft Computing — AFSS 2002* (N. R. Pal and M. Sugeno, eds.), (Berlin, Heidelberg), pp. 288–300, Springer Berlin Heidelberg, 2002.

[30] S. Zhang, J. Li, P. Xie, Y. Zhang, M. Shao, H. Zhou, and M. Yan, "Stacked kernel network," 2017.