UCLouvain

École polytechnique de Louvain

Learning integral operators from diagonal-circulant neural networks

Author: **Julien Calbert**
Supervisors: **Pierre-Antoine Absil, Laurent Jacques**
Readers: **John Lee, Benoît Pairet**
Academic year 2019–2020
Master [120] in Mathematical Engineering

Abstract

There are imaging applications (e.g. optical calibration, exoplanet detection) where we observe an image "convolved" with a kernel whose shape depends on its localization. These operators are non shift-invariant and therefore cannot be modeled by a simple convolution. Moreover, these operators can appear in the resolution of an inverse problem. To solve such an optimization problem, it is necessary to evaluate at each iteration the operator and its derivatives with respect to its inputs. If the operator or its derivatives are too costly to evaluate, this may make the optimization problem intractable. Therefore, if we have an approximation model for which we can efficiently compute the derivatives and the result for a given input, we can replace the real operator by the approximation in the solving of the inverse problem.
The problem addressed in this paper concerns the approximation of these general integral operators based on a dataset containing the values of the operator for several inputs. The mathematical model used in this work to learn the integral operator is based on a feedforward neural network composed of an alternating product of diagonal and circulant matrices for one-dimensional applications, whereas it will be an alternating product of diagonal and doubly-block-circulant matrices for two-dimensional applications. In addition to the computational interest of these networks, this work has shown that they are useful for particular problems for which the physical system presents such factorization, for example in the context of exoplanet detection by direct imaging. In these problems, the amount of training data can be significantly reduced.

Contents

Notations

The next list describes several symbols and notations that will be later used within the body of the document.

$A_{i,j}/[\boldsymbol{A}]_{i,j}$ Two notations to designate the element in row i and column j of the matrix $\boldsymbol{A}$.

$\boldsymbol{A}_{i,:}$ Row i of matrix $\boldsymbol{A}$.

$\boldsymbol{A}_{:,i}$ Column i of matrix $\boldsymbol{A}$.

$\boldsymbol{A}^*$ The superscript $*$ represents the adjoint of a complex matrix (conjugate transposition).

$\boldsymbol{I_N}$ $\boldsymbol{I_N} \in \mathbb{R}^{N\times N}$ is the identity matrix.

$\boldsymbol{D}(\boldsymbol{w})$ Let $\boldsymbol{w} \in \mathbb{R}^N$, then $\boldsymbol{D}(\boldsymbol{w}) \in \mathbb{R}^{N\times N}$ is the diagonal matrix generated by the vector $\boldsymbol{w}$:

$$[\boldsymbol{D}(\boldsymbol{w})]_{i,j} = \begin{cases} w_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}, \quad 1 \leq i, j \leq N.$$

$\boldsymbol{C}(\boldsymbol{w})$ Let $\boldsymbol{w} \in \mathbb{R}^N$, then $\boldsymbol{C}(\boldsymbol{w}) \in \mathbb{R}^{N\times N}$ is the circulant matrix generated by the vector $\boldsymbol{w}$. A precise definition is given in (1.1).

$\boldsymbol{B}(\boldsymbol{w})$ Let $\boldsymbol{w} \in \mathbb{R}^{M\times N}$, then $\boldsymbol{B}(\boldsymbol{w}) \in \mathbb{R}^{MN\times MN}$ is the doubly-block-circulant matrix generated by the matrix $\boldsymbol{w}$. A precise definition is given in (1.2).

$\|\boldsymbol{w}\|_2$ Let $\|\boldsymbol{w}\|_2$ denotes the euclidean norm of the vector $\boldsymbol{w} \in \mathbb{R}^N$:

$$\|\boldsymbol{w}\|_2 = \sqrt{\sum_{i=1}^{N} w_i^2}.$$

$\|\boldsymbol{w}\|_1$ Let $\|\boldsymbol{w}\|_1$ denotes the following norm for a vector $\boldsymbol{w} \in \mathbb{R}^N$:

$$\|\boldsymbol{w}\|_1 = \sum_{i=1}^{N} |w_i|.$$

$\text{diag}(\boldsymbol{w})$ Let $\text{diag}(\boldsymbol{w}) \in \mathbb{R}^{N\times N}$ be the diagonal matrix with diagonal entries given by $\boldsymbol{w} \in \mathbb{R}^N$.

$\text{vec}(\boldsymbol{A})$ Let $\text{vec}(\boldsymbol{A})$ denotes the operator that creates a column vector from a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ by stacking the columns of $\boldsymbol{A} = [\boldsymbol{A}_{:,1}, ..., \boldsymbol{A}_{:,N}]$:

$$\text{vec}(\boldsymbol{A}) = \begin{pmatrix} \boldsymbol{A}_{:,1} \\ \boldsymbol{A}_{:,2} \\ \vdots \\ \boldsymbol{A}_{:,N} \end{pmatrix}.$$

$\odot$ Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{M \times N}$, then $\boldsymbol{A} \odot \boldsymbol{B}$ denotes the element-wise multiplication (Hadamard product):

$$[\boldsymbol{A} \odot \boldsymbol{B}]_{i,j} = A_{i,j} B_{i,j}, \quad 1 \leq i \leq M,\ 1 \leq j \leq N.$$

$\otimes$ Let $\boldsymbol{A} \in \mathbb{R}^{M_1 \times M_2}$ and $\boldsymbol{B} \in \mathbb{R}^{N_1 \times N_2}$, the Kronecker product between $\boldsymbol{A}$ and $\boldsymbol{B}$ is the matrix $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{M_1 N_1 \times M_2 N_2}$ defined as:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} A_{1,1}\boldsymbol{B} & A_{1,2}\boldsymbol{B} & \cdots & A_{1,M_2}\boldsymbol{B} \\ A_{2,1}\boldsymbol{B} & A_{2,2}\boldsymbol{B} & \cdots & A_{2,M_2}\boldsymbol{B} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M_1,1}\boldsymbol{B} & A_{M_1,2}\boldsymbol{B} & \cdots & A_{M_1,M_2}\boldsymbol{B} \end{pmatrix}.$$

$n!!$ Let $n!!$ denotes the double factorial of a number $n \in \mathbb{N}$:

$$n!! = \begin{cases} \prod_{k=0}^{\lceil \frac{n}{2} \rceil - 1} (n - 2k) & \text{if } n > 0, \\ 1 & \text{if } n = 0. \end{cases}$$

$\nabla_{\boldsymbol{x}} y$ Gradient of $\boldsymbol{y}$ with respect to $\boldsymbol{x}$.

$\operatorname{sign}(x)$ It returns the sign of the real x:

$$\operatorname{sign}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

$k_\sigma(u)$ This is the Gaussian function with a mean of zero and a standard deviation of σ:

$$k_\sigma(u) = \kappa_\sigma e^{\frac{-u^2}{2\sigma^2}} \text{ with } \kappa_\sigma = \frac{1}{\sqrt{2\pi}\sigma}.$$

$t_\gamma(u)$ This is the function that always returns γ: $t_\gamma(u) = \gamma$.

$\operatorname{fftshift}(\boldsymbol{A})$ Let $\operatorname{fftshift}(\boldsymbol{A})$ be the operator that exchanges the first quadrant of the matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ with the third, and the second quadrant with the fourth. The figures 1 and 2 illustrate this transformation.

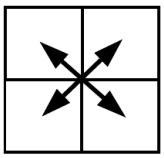


Figure 1: fftshift operator.

$$\left[\begin{array}{cc|ccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \xrightarrow{\text{fftshift}} \left[\begin{array}{ccc|cc} -4 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{array}\right]$$

Figure 2: Example of fftshift operator.

Introduction

Context

Since the first exoplanet (planet out of our solar system) detected in 1992 by the astronomer Wolszczan, more than 4000 have been detected [41]. The majority of the detections have been made indirectly using the transit method [17]. This method consists in detecting the decrease in brightness resulting from the passage of the planet in front of its star. Consequently, this technique only detects exoplanets that have an orbit in a plane formed by the observer and the star.

Direct imaging, another detection technique, can detect exoplanets orbiting in a plane perpendicular to the observer-star plane. As the name implies, the method seeks to directly observe the light reflected or emitted by the planet itself. This task is much more difficult than indirect detection for several reasons. In particular, it requires the use of a ground-based telescope equipped with a coronograph in order to reduce the light contrast between the exoplanet and its star. Consequently, due to atmospheric and instrumental disturbances, the observed image is noisy.

Therefore, in order to separate the potential planetary signals from the star light and the speckle field, it is necessary to use post-processing techniques based on the resolution of an inverse problem to correct the effects of the telescope on the observed image. In practice because of the coronograph, the optical system is non shift-invariant for small angular separation and therefore it cannot be modeled by a simple convolution. To solve the inverse problem, regardless of the optimization method used, we will often have to evaluate the model describing the telescope and its derivatives. So the goal is to design a model that approximates the telescope, for which we can quickly calculate the derivative relative to the inputs, and integrate it into the solving of the inverse problem. Additionally, we can not have access to the result of the telescope for each possible input. That is, we have to reduce the number of parameters of the model by exploiting the physics of the problem.

Objectives

There are imaging applications (e.g., in astronomy as previously presented) where we observe an image "convolved" with a kernel whose shape depends on its localization. These operators are non shift-invariant and therefore cannot be modeled by a simple convolution.
We consider the general form of the linear integral operator H:

$$H[f](u) = \int_{\Omega} K(u, x) f(x) \, \mathrm{d}x,$$

where K is called the kernel of the operator.
The problem addressed in this paper concerns the approximation of these general integral operators based on a dataset containing the values of the integral operator for several inputs.

Obviously, we wish to obtain a model able to generalize the result of the integral operator for inputs that are not part of the dataset. In addition, we wish to obtain an approximation model that verifies the following constraints:

- It should have a small generalization error for a dataset containing few data. Indeed, for some applications, it is only possible to probe the integral operator for a limited number of entries.
- The cost to evaluate the model for a given input and the cost to compute its derivatives with respect to its inputs should be low.

Indeed, in some applications (such as the problem of exoplanets detection), a partially known integral operator may appear in the solving of an inverse problem.
To solve such an optimization problem, it is necessary to evaluate at each iteration the operator and its derivatives with respect to its inputs. If the derivatives of the full operator are not available, or if they are too expensive to evaluate, this can make the optimization problem intractable. Therefore, if we have an approximation model for which we can efficiently compute the derivatives and the result for a given input, we can replace the real operator by the approximation in solving the inverse problem.

Main contributions

The mathematical model used in this work to learn the discrete matrix associated to the target integral operator is based on a feedforward neural network introduced in section 1.2.3. The network consists of an alternating product of diagonal and circulant matrices for a one-dimensional application, while it will be an alternating product of diagonal and doubly-block-circulant matrices for a two-dimensional application. This particular structure is motivated by several reasons:

- The computational interest both in inference and derivatives calculation of the diagonal/circulant neural network.
- The ability of circulant matrices, when combined with diagonal matrices, to represent any linear transformation provided the number of factors is sufficient.
- Depending of the application, the structure of the target operator can be close to this particular alternating product (typically in optics).

Through this work, we have highlighted the ability of these networks to learn non shift-invariant operators from a very small number of pairs (input, output) of the target operator. In particular, within the framework of the detection of exoplanets by direct imagery, it has been shown that the use of such a network increases the approximation of the behaviour of the telescope for a low angular separation compared to a model approaching the telescope by a simple convolution.

Structure of the document

The next chapter contains some preliminary concepts that will be useful for the rest of the document, in particular it deals with the properties of convolution and the machine learning. The chapter 2 presents a general overview of the diagonal/circulant neural network. In the chapter 3, the network and its properties are precisely defined. The chapter 4 gives some examples of areas where the particular diagonal/circulant structure appears. Chapter 5 contains the numerical results for the one-dimensional case. The chapter 6 concerns numerical experiments in the context of astronomy introduced previously for the detection of exoplanets. Finally, the last chapter concludes and lists the possible orientations for further research.

Chapter 1

Preliminaries

This chapter contains some preliminary concepts that will be useful for the rest of the document. The first part concerns the one-dimensional and two-dimensional convolution operation. The second part concerns the machine learning and in particular the neural network model.

1.1 Convolution

The diagonal/circulant neural network and the diagonal/doubly-block-circulant neural network investigated in this paper to approximate integral operators are based on the 1D and the 2D convolution operation, respectively. Therefore, the following sections contain a brief review of the convolution operation in 1D and in 2D. Inspired from [14, 31, 27] and [2, Appendix 1-2].

1.1.1 1D case

This section contains a brief review of the discrete Fourier transform, the circulant matrices and the convolution operation.

Discrete Fourier transform (DFT)

Let $f[n]$ be a signal of finite support: $0 \leq n \leq N-1$. The DFT and its inverse are defined as [1]:

$$\begin{cases} \hat{f}[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f[n]\omega_N^{nk} \quad , \quad k = 0, ..., N-1, \\ f[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \hat{f}[k]\omega_N^{-nk}, \quad n = 0, ..., N-1, \end{cases} \tag{1.1}$$

with $\omega_N = e^{-\mathrm{i}\frac{2\pi}{N}}$.
We can also rewrite the DFT as a matrix-vector product:

$$\hat{\boldsymbol{f}} = \boldsymbol{F_N} \boldsymbol{f}, \tag{1.2}$$

with $\boldsymbol{F_N} \in \mathbb{R}^{N \times N}$, the DFT matrix defined as: $(\boldsymbol{F_N})_{k,l} = \frac{1}{\sqrt{N}}\omega_N^{kl}$.
And where $\hat{\boldsymbol{f}}$ and $\boldsymbol{f}$ are vectors whose the elements are values of the discrete signals $\hat{f}[k]$ and $f[n]$, respectively. The DFT matrix is symmetric ${\boldsymbol{F_N}}^T = \boldsymbol{F_N}$ and its columns are orthonormal ${\boldsymbol{F_N}}^* \boldsymbol{F_N} = \boldsymbol{I_N}$. Therefore, ${\boldsymbol{F_N}}^{-1} = {\boldsymbol{F_N}}^*$.

[1] It is the symmetric normalization definition of the DFT. But beware that the NUMPY PYTHON package [32] uses another definition. For this reason, some of the theoretical results presented in the article differ from the practical implementation, for more details see section A.3.

So, $\boldsymbol{f} = \boldsymbol{F_N}^{-1}\hat{\boldsymbol{f}}$ is obviously equivalent to the inverse DFT rewritten in a matrix-vector product.

Circulant matrix

What follows is the precise definition of a circulant matrix, as well as some of its properties that will be useful for the rest of the paper.

Definition 1.1. *Let $\boldsymbol{C}(\boldsymbol{w}) \in \mathbb{R}^{N\times N}$ denotes the circulant matrix generated by the vector $\boldsymbol{w} = (w_0, w_1, ..., w_{N-1})^T$:*

$$\boldsymbol{C}(\boldsymbol{w}) = \begin{pmatrix} w_0 & w_{N-1} & \cdots & w_2 & w_1 \\ w_1 & w_0 & w_{N-1} & & w_2 \\ \vdots & w_1 & w_0 & \ddots & \vdots \\ w_{N-2} & \vdots & \ddots & \ddots & w_{N-1} \\ w_{N-1} & w_{N-2} & \cdots & w_1 & w_0 \end{pmatrix}.$$

Circulant matrix is a special kind of Toeplitz matrix where each row is a cyclic right shift of the previous one. We have:

$$[\boldsymbol{C}(\boldsymbol{w})]_{i,j} = w_{(i-j)mod\ N}, \quad 0 \leq i, j \leq N-1. \tag{1.3}$$

Theorem 1.1 (Diagonalization of circulant matrix)**.** *Let $\boldsymbol{C}(\boldsymbol{w}) \in \mathbb{R}^{N\times N}$ be the circulant matrix generated by $\boldsymbol{w}$. The columns of the inverse DFT matrix $\boldsymbol{F_N}^{-1}$ are the eigenvectors of any circulant matrix. The associated eigenvalues are the DFT values of the vector generating the matrix. Therefore, any circulant matrix can be diagonalized as:*

$$\begin{aligned} &\boldsymbol{C}(\boldsymbol{w}) = \boldsymbol{F_N}^{-1}\boldsymbol{D}\boldsymbol{F_N} = \boldsymbol{F_N}^{*}\boldsymbol{D}\boldsymbol{F_N}, \\ &\textit{with } \boldsymbol{D} = \mathrm{diag}(\sqrt{N}\boldsymbol{F_N}\boldsymbol{w}). \end{aligned}$$

Proof. see (A.1) □

Considering the diagonalization $\boldsymbol{C}(\boldsymbol{w}) = \boldsymbol{F_N}^{*}\boldsymbol{D}\boldsymbol{F_N}$, the transpose matrix can be expressed as:

$$\boldsymbol{C}(\boldsymbol{w})^T = \boldsymbol{C}(\boldsymbol{w})^* = (\boldsymbol{F_N}^{*}\boldsymbol{D}\boldsymbol{F_N})^* = \boldsymbol{F_N}^{*}\boldsymbol{D}^*\boldsymbol{F_N}. \tag{1.4}$$

Hence the following matrix-vector products with the vector $\boldsymbol{x} \in \mathbb{R}^N$ can be computed based on Theorem 1.1:

$$\begin{cases} \boldsymbol{C}(\boldsymbol{w})\boldsymbol{x} = \ \boldsymbol{F_N}^{-1}\mathrm{diag}(\sqrt{N}\boldsymbol{F_N}\boldsymbol{w})\boldsymbol{F_N}\boldsymbol{x} = \mathcal{F}^{-1}(\sqrt{N}\ \mathcal{F}(\boldsymbol{w}) \odot \mathcal{F}(\boldsymbol{x})), \\ \boldsymbol{C}(\boldsymbol{w})^T\boldsymbol{x} = \ \boldsymbol{F_N}^{-1}\mathrm{diag}(\sqrt{N}\boldsymbol{F_N}\boldsymbol{w})^*\boldsymbol{F_N}\boldsymbol{x} = \mathcal{F}^{-1}(\sqrt{N}\ \mathcal{F}(\boldsymbol{w})^* \odot \mathcal{F}(\boldsymbol{x})), \end{cases} \tag{1.5}$$

where $\mathcal{F}$ denotes the DFT defined according to (1.1). As we will see in section (3.1.3), $\mathcal{F}$ (and $\mathcal{F}^{-1}$) can be computed in $\mathcal{O}(N\log(N))$ using the FFT algorithm. Hence, based on (1.5) and the FFT algorithm, the complexity of the matrix-vector product is in $\mathcal{O}(N\log(N))$.
From the symmetry between $\boldsymbol{w}$ and $\boldsymbol{x}$ in the first equation of (1.5), we can deduce the following property of the circulant matrix:

$$\boldsymbol{C}(\boldsymbol{w})\boldsymbol{x} = \boldsymbol{C}(\boldsymbol{x})\boldsymbol{w}. \tag{1.6}$$

Circular convolution

Let $f[n]$ and $h[n]$ be two finite support signals: $0 \leq n \leq N-1$. The circular convolution of these two functions consists in computing the linear convolution between one function

and the periodic summation of the other, and in keeping only the values on the finite support of size N.
Let $g[n]$ be the circular convolution between $f[n]$ and $h[n]$:

$$\begin{aligned} g[n] &= h[n] \circledast f[n] = h_N[n] * f[n] \\ &= \sum_{k=0}^{N-1} h[(n-k) \mathrm{mod} N] f[k], \end{aligned} \tag{1.7}$$

with $h_N[n] = \sum_{k=-\infty}^{\infty} h[n - kN]$, the periodic summation of $h[n]$.
Let $\boldsymbol{g}$, $\boldsymbol{f}$ and $\boldsymbol{h}$ be the vectors whose the elements are the values of the discrete signals $g[n]$, $f[n]$ and $h[n]$, respectively. The circular convolution can also be expressed as a matrix-vector product:

$$\boldsymbol{g} = \boldsymbol{C}(\boldsymbol{h})\boldsymbol{f}, \tag{1.8}$$

where $\boldsymbol{C}(\boldsymbol{h}) \in \mathbb{R}^{N \times N}$ is the circulant matrix generated by $\boldsymbol{h}$.

1.1.2 2D case

This section contains a brief review of the 2D discrete Fourier transform, the doubly-block-circulant matrices and the convolution operation.

2D Discrete Fourier transform (2D-DFT)

Let $f[m, n]$ be a two-dimensional signal of finite support: $0 \leq m \leq M-1$, $0 \leq n \leq N-1$. The 2D-DFT and its inverse are defined as [2]:

$$\begin{cases} \hat{f}[k,l] = \dfrac{1}{\sqrt{MN}} \displaystyle\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f[m,n] \omega_M^{mk} \omega_N^{nl} \ , \ 0 \leq k \leq M-1, 0 \leq l \leq N-1, \\ f[m,n] = \dfrac{1}{\sqrt{MN}} \displaystyle\sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \hat{f}[k,l] \omega_M^{-mk} \omega_N^{-nl}, \ 0 \leq m \leq M-1, 0 \leq n \leq N-1. \end{cases} \tag{1.9}$$

Let $\hat{\boldsymbol{f}} \in \mathbb{R}^{M \times N}$ and $\boldsymbol{f} \in \mathbb{R}^{M \times N}$ be the matrices whose the elements are values of the discrete signals $\hat{f}[k, l]$ and $f[m, n]$, respectively. Considering the column stack version, we can rewrite the 2D-DFT as a matrix-vector product:

$$\mathrm{vec}(\hat{\boldsymbol{f}}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M}) \mathrm{vec}(\boldsymbol{f}). \tag{1.10}$$

Indeed, let $i = k + lM$ with $0 \leq k \leq M-1$ and $0 \leq l \leq N-1$:

$$\begin{aligned} [(\boldsymbol{F_N} \otimes \boldsymbol{F_M}) \mathrm{vec}(\boldsymbol{f})]_i &= \sum_{n=0}^{N-1} [\boldsymbol{F_N}]_{l,n} \sum_{m=0}^{M-1} [\boldsymbol{F_M}]_{k,m} [\mathrm{vec}(\boldsymbol{f})]_{m+nM} \\ &= \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f[m,n] \omega_M^{mk} \omega_N^{nl} = \hat{f}[k,l] = [\mathrm{vec}(\hat{\boldsymbol{f}})]_i. \end{aligned} \tag{1.11}$$

The matrix $(\boldsymbol{F_N} \otimes \boldsymbol{F_M}) \in \mathbb{R}^{MN \times MN}$ will be called the 2D-DFT matrix. It has the following properties:

[2] It is the symmetric normalization definition of the 2D-DFT. But beware that the NUMPY PYTHON package [32] uses another definition. For this reason, some of the theoretical results presented in the article differ from the practical implementation, for more details see section A.4.

- 2D-DFT is symmetric: using the transpose property of the Kronecker product and then the symmetry of the DFT matrix:

$$(\boldsymbol{F_N} \otimes \boldsymbol{F_M})^T = \boldsymbol{F_N}^T \otimes \boldsymbol{F_M}^T = \boldsymbol{F_N} \otimes \boldsymbol{F_M}. \tag{1.12}$$

- Its columns are orthonormal:

$$\begin{aligned}(\boldsymbol{F_N} \otimes \boldsymbol{F_M})^*(\boldsymbol{F_N} \otimes \boldsymbol{F_M}) &= (\boldsymbol{F_N}^* \otimes \boldsymbol{F_M}^*)(\boldsymbol{F_N} \otimes \boldsymbol{F_M}) = (\boldsymbol{F_N}^*\boldsymbol{F_N} \otimes \boldsymbol{F_M}^*\boldsymbol{F_M}) \\ &= (\boldsymbol{I_N}) \otimes (\boldsymbol{I_M}) = \boldsymbol{I_{MN}}.\end{aligned} \tag{1.13}$$

  Therefore:

$$(\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1} = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^*. \tag{1.14}$$

  So, $\text{vec}(\boldsymbol{f}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1}\text{vec}(\hat{\boldsymbol{f}})$ is obviously equivalent to the inverse 2D-DFT rewritten as a matrix-vector product.

Doubly-block-circulant matrix

What follows is the precise definition of a doubly-block-circulant matrix, as well as some of its properties that will be useful for the rest of the paper.

Definition 1.2. *A block matrix $\boldsymbol{B} \in \mathbb{R}^{MN\times MN}$ whose blocks $\boldsymbol{B}_i \in \mathbb{R}^{M\times M}$, $i = 0, ..., N-1$ is said block-circulant if the structure of $\boldsymbol{B}$ with respect to its blocks is circulant:*

$$\boldsymbol{B} = \begin{pmatrix} \boldsymbol{B}_0 & \boldsymbol{B}_{N-1} & \boldsymbol{B}_{N-2} & \cdots & \boldsymbol{B}_1 \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & \boldsymbol{B}_{N-1} & \cdots & \boldsymbol{B}_2 \\ \boldsymbol{B}_2 & \boldsymbol{B}_1 & \boldsymbol{B}_0 & \cdots & \boldsymbol{B}_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{B}_{N-1} & \boldsymbol{B}_{N-2} & \boldsymbol{B}_{N-3} & \cdots & \boldsymbol{B}_0 \end{pmatrix}.$$

If each individual block $\boldsymbol{B}_i$ is also circulant, then $\boldsymbol{B}$ is called doubly-block-circulant.
Considering a 2D signal of finite support: $w[m, n]$ with $0 \le m \le M-1$ and $0 \le n \le N-1$, we define $\boldsymbol{B}(\boldsymbol{w})$ as the doubly-block-circulant matrix generated by the matrix $\boldsymbol{w} \in \mathbb{R}^{M\times N}$. Where $\boldsymbol{w}$ is the matrix whose the entries are the value of signal $w[m, n]$. The block j is the circulant matrix generated by the j^{th} column of $\boldsymbol{w}$:

$$\boldsymbol{B}_j = \boldsymbol{C}(\boldsymbol{w}_{:,j}) = \begin{pmatrix} w[0,j] & w[M-1,j] & w[M-2,j] & \cdots & w[1,j] \\ w[1,j] & w[0,j] & w[M-1,j] & \cdots & w[2,j] \\ w[2,j] & w[1,j] & w[0,j] & \cdots & w[3,j] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w[M-1,j] & w[M-2,j] & w[M-3,j] & \cdots & w[0,j] \end{pmatrix}, \ j = 0, ..., N-1.$$

Theorem 1.2 (Diagonalization of doubly-block-circulant matrix)**.** *Let $\boldsymbol{B}(\boldsymbol{w}) \in \mathbb{R}^{MN\times MN}$ be the doubly-block-circulant matrix (DBC) generated by $w[m, n]$, a 2D signal of finite support: $0 \le m \le M-1$, $0 \le n \le N-1$. The columns of the inverse 2D-DFT matrix: $(\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1}$ are the eigenvectors of any doubly-block-circulant matrix. The associated eigenvalues are the 2D-DFT values of signal generating the matrix. Therefore, any DBC matrix can be diagonalized as:*

$$\begin{aligned}&\boldsymbol{B}(\boldsymbol{w}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1}\boldsymbol{D}(\boldsymbol{F_N} \otimes \boldsymbol{F_M}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^*\boldsymbol{D}(\boldsymbol{F_N} \otimes \boldsymbol{F_M}), \\ &\textit{with } \boldsymbol{D} = \text{diag}(\sqrt{MN}(\boldsymbol{F_N} \otimes \boldsymbol{F_M})\text{vec}(\boldsymbol{w})).\end{aligned}$$

Proof. see (A.2) □

Considering the diagonalization $\boldsymbol{B}(\boldsymbol{w}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^* \boldsymbol{D} (\boldsymbol{F_N} \otimes \boldsymbol{F_M})$, the transpose matrix can be expressed as:

$$\boldsymbol{B}(\boldsymbol{w})^T = \boldsymbol{B}(\boldsymbol{w})^* = ((\boldsymbol{F_N} \otimes \boldsymbol{F_M})^* \boldsymbol{D} (\boldsymbol{F_N} \otimes \boldsymbol{F_M}))^* = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^* \boldsymbol{D}^* (\boldsymbol{F_N} \otimes \boldsymbol{F_M}). \quad (1.15)$$

Hence the following matrix-vector products with a vector $\text{vec}(\boldsymbol{x}) \in \mathbb{R}^{MN}$ such that $\boldsymbol{x} \in \mathbb{R}^{M \times N}$, can be computed based on Theorem 1.2:

$$\begin{cases} \boldsymbol{B}(\boldsymbol{w})\text{vec}(\boldsymbol{x}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1} \boldsymbol{D} (\boldsymbol{F_N} \otimes \boldsymbol{F_M})\text{vec}(\boldsymbol{x}) = \mathcal{F}_{2D}^{-1}(\sqrt{MN}\mathcal{F}_{2D}(\boldsymbol{w}) \odot \mathcal{F}_{2D}(\boldsymbol{x})), \\ \boldsymbol{B}(\boldsymbol{w})^T\text{vec}(\boldsymbol{x}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1} \boldsymbol{D}^* (\boldsymbol{F_N} \otimes \boldsymbol{F_M})\text{vec}(\boldsymbol{x}) = \mathcal{F}_{2D}^{-1}(\sqrt{MN}\mathcal{F}_{2D}(\boldsymbol{w})^* \odot \mathcal{F}_{2D}(\boldsymbol{x})), \end{cases} \quad (1.16)$$

where $\mathcal{F}_{2D}$ denotes the 2D-DFT defined according to (1.9). As we will see in section (3.2.2), $\mathcal{F}_{2D}$ (and $\mathcal{F}_{2D}^{-1}$) can be computed in $\mathcal{O}(MN(\log(M) + \log(N)))$ using the FFT algorithm. Hence, based on (1.16) and the FFT algorithm, the complexity of the matrix-vector product is in $\mathcal{O}(MN(\log(M) + \log(N)))$.
From the symmetry between $\boldsymbol{w}$ and $\boldsymbol{x}$ in the first equation of (1.16), we can deduce the following property:

$$\boldsymbol{B}(\boldsymbol{w})\text{vec}(\boldsymbol{x}) = \boldsymbol{B}(\boldsymbol{x})\text{vec}(\boldsymbol{w}). \quad (1.17)$$

2D Circular convolution

Let $f[m,n]$ and $h[m,n]$ be two finite support signals: $0 \leq m \leq M-1$, $0 \leq n \leq N-1$. The circular convolution of these two functions consists in computing the linear convolution between one function and the periodic summation of the other, and in keeping only the values on the finite support.
Let $g[m,n]$ be the circular convolution between $f[m,n]$ and $h[m,n]$:

$$\begin{aligned} g[m,n] &= h[m,n] \circledast f[m,n] = h_{M,N}[m,n] * f[m,n] \\ &= \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} h[(m-k)\text{mod}M, (n-l)\text{mod}N] f[k,l], \end{aligned} \quad (1.18)$$

with $h_{M,N}[m,n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} h[m-kM, n-lN]$, the periodic summation of $h[m,n]$.
Let $\boldsymbol{g}$, $\boldsymbol{f}$ and $\boldsymbol{h}$ be the matrices whose the elements are the values of the discrete signals $g[m,n]$, $f[m,n]$ and $h[m,n]$, respectively.
The two-dimensional circular convolution can also be expressed as a matrix-vector product:

$$\text{vec}(\boldsymbol{g}) = \boldsymbol{B}(\boldsymbol{h})\text{vec}(\boldsymbol{f}), \quad (1.19)$$

where $\boldsymbol{B}(\boldsymbol{h}) \in \mathbb{R}^{MN \times MN}$ is the doubly-block-circulant matrix generated by $\boldsymbol{h}$.

1.2 Machine learning

This section contains the basic concepts concerning the machine learning which will be useful to present the particular structure of the investigated neural networks.

1.2.1 Overview

Machine learning algorithms learn mathematical model based on experiences (data) to perform some tasks. Most machine learning algorithms belongs to one of the two following categories [21, Chapter 5]:

- *Unsupervised learning:* the algorithm works on a dataset containing data with features and try to learn useful properties from the data. An example of such task is the clustering problem where the goal is to divide the dataset into subset containing similar data.
- *Supervised learning:* each element belonging to the dataset is associated with a label in case of classification or a target in case of regression. The name result from the fact that we feed the algorithm with training data (i.e. with the correct answer). The supervised algorithm try to model the relation between the input data and the target, such that it can predict the output value for new input data. The generalization is important because the training data can be only a sampling of the real target function.

For the rest of the article, we will focus on supervised algorithms for regression problems. Obviously, at the end of the training phase, we expect our model to work well on the training dataset. But, as mentioned, we also want our algorithm to work well on new data that are not part of our training dataset. The training phase of the model can be formalized as an optimization problem where we minimise the error on the training set, but machine learning differs from optimization precisely by the fact that we want this generalization ability. To measure the generalization error of the model, we can measure the performance of the model on a test set containing data that were not used during the learning process. The model's ability to generalize to unseen data is one of the main challenges of machine learning. Indeed, one can be faced with two difficulties [21, Chapter 5]:

- *Underfitting:* it describes the fact that the model is not able to obtain a low error on the training set and hence also on the test set. It can be easily detected by using a good metric to measure the error on the training set.
- *Overfitting:* it occurs when the gap between the training error and the test error is high. It is due to the fact that the model learn too well the training set with its irrelevant details and noise. It is the detection of this phenomenon that motivates the use of a test set.

One can control these phenomena by adjusting the complexity of the model which is often called: its capacity. The capacity of a model depends of many things: its structure, the number of parameters, the training algorithm used,...
Low-complexity models can not solve complex tasks (underfitting), while high-complexity models can solve complex tasks with the risk of overfitting. This behaviour is illustrated on Figure 1.1.

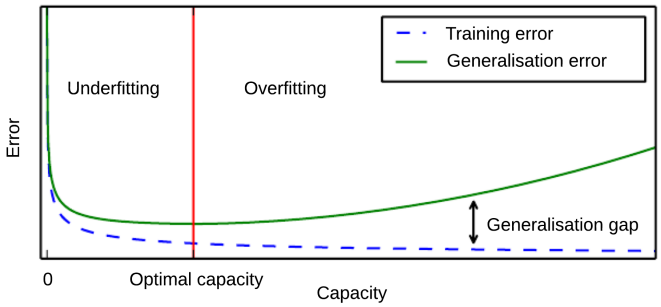


Figure 1.1: Relation between the error and the capacity of the model (Credit [42]).

The techniques used to reduce the generalization error, sometimes at the expense of training error are called regularization strategies [21, Chapter 7]. These techniques are designed to reduce the error on the test set.

1.2.2 Stochastic gradient descent

We have a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i),\ i = 1, ..., T\}$ where $\boldsymbol{x}_i$ are the inputs and $\boldsymbol{y}_i$ are the corresponding outputs.
We consider the problem of training a model, designated by $f(\boldsymbol{x}; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$ and taking as input the vector $\boldsymbol{x}$, to fit the dataset $\mathcal{D}$. We assume that the loss function can be decompose as a sum of per-example loss function:

$$E_T(\boldsymbol{\theta}) = \frac{1}{T}\sum_{i=1}^{T} \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}), \qquad \min_{\boldsymbol{\theta}} \quad E_T(\boldsymbol{\theta}), \tag{1.20}$$

where $\mathcal{L}$ is a per-example loss function that is supposed to be differentiable and remains general for now.
To solve this optimization problem, a commonly used method is the gradient descent where the parameters are updated as follows: $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta \nabla_{\boldsymbol{\theta}} E_T(\boldsymbol{\theta}^t)$ with $\eta > 0$ the learning rate. To be executed, it requires to compute the gradient of the loss function with respect to the parameters of the model:

$$\nabla_{\boldsymbol{\theta}} E_T(\boldsymbol{\theta}) = \frac{1}{T}\sum_{i=1}^{T} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}). \tag{1.21}$$

Therefore the computation of a gradient step is in $\mathcal{O}(T)$. And since to obtain a model with good generalization, it is necessary to have a large training set, the cost of an exact gradient step can become prohibitive.
The idea of stochastic gradient descent (SGD) is to use an update direction which is not exactly the gradient but an estimate computed from a randomly selected subset of the data $\mathcal{B}$ called a minibatch. The estimate of the gradient is given by:

$$\boldsymbol{g} = \frac{1}{|\mathcal{B}|}\sum_{i \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}). \tag{1.22}$$

Therefore, the complexity of one SGD step is in $\mathcal{O}(|\mathcal{B}|)$. Let $\boldsymbol{\theta}(t)$ denotes the parameters at step t, then the algorithm can be written as:

Algorithm 1: SGD algorithm

```
initialise the parameters θ(0);
t := 0;
while not converged do
    randomly shuffle the elements in the training set;
    divide the training set in minibatch of same size;
    for all minibatch do
        compute g(t) according (1.22);
        θ(t+1) = θ(t) − ηg(t);
        t := t + 1;
    end
end
```

One complete pass on the complete training set (i.e. on all the minibatches) is called an epoch. It has been shown that the SGD algorithm converges for convex optimization problem ([42, Theorem 14.8]). The SGD method as also the advantage to be able to escape from local minima in the case of non-convex optimization problem. Specifically, the SGD name corresponds to a minibatch containing only one element: $|\mathcal{B}| = 1$.

As explained, the SGD method does not use the exact gradient of our loss function but an estimation based on batches, which implies that at each iteration we have no guarantee that we are going in the optimal direction. Therefore, using an exponentially weighted moving average of the update direction, we can try to stabilise and keep a global direction which can be progressively improved over time as illustrated on Figure 1.2. From this, we deduce the following optimization scheme:

$$\begin{cases} \boldsymbol{z}(t+1) = \beta \boldsymbol{z}(t) + (1-\beta)\boldsymbol{g}(t), \\ \boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \eta \boldsymbol{z}(t+1), \end{cases} \tag{1.23}$$

where $\beta \approx 0.9$ is called the momentum parameter. The update direction $z(t)$ is computed with an exponentially weighted moving average, and as $\beta < 1$, the importance of previous gradient estimate will decrease over time. Scheme (1.23) is often written in the following way:

$$\begin{aligned} \boldsymbol{\theta}(t+1) &= \boldsymbol{\theta}(t) - \eta(1-\beta)\boldsymbol{g}(t) + \beta(\boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-1)) \\ &= \boldsymbol{\theta}(t) - \eta' \boldsymbol{g}(t) + \beta(\boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-1)), \end{aligned} \tag{1.24}$$

where $\eta' = \eta(1-\beta)$ is referred as the new learning rate. This scheme is called the SGD algorithm with momentum [39].

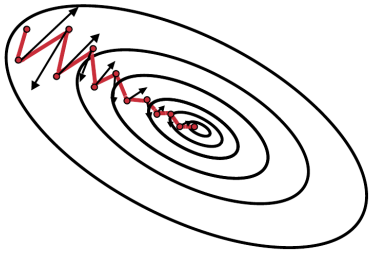


Figure 1.2: The path of the momentum algorithm is illustrated in red. The black arrows indicate the steps that the classical SGD will take.

1.2.3 Feedforward neural network

Neural networks [24, Chapter 11] are a particular model of computation used in machine learning. The model $f(.;\boldsymbol{\theta})$ parametrized by $\boldsymbol{\theta}$ consists in the composition of several functions $f_l(.;\boldsymbol{\theta}) : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$ called layers:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x};\boldsymbol{\theta}) = (f_L \circ ... \circ f_2 \circ f_1)(\boldsymbol{x}), \tag{1.25}$$

where $\boldsymbol{x}$ and $\hat{\boldsymbol{y}}$ are respectively the input and output of the model. L is called the depth of the network. Each component of the output of a layer function f_l can be seen as a unit working in parallel and corresponding to a vector to scalar mapping. These units are called neurons. Therefore, we usually represent the neural network with a graph where the nodes are the neurons and the edges connect the output of a neuron to the input of an other neuron of the next layer. Here we consider only feedforward model, i.e. model where the underlying graph is acyclic.
For what follows, we will consider a particular case of feedforward neural network where the function f_l can be written as: $f_l(\boldsymbol{x}) = \phi_l(\boldsymbol{w}^{(l)}\boldsymbol{x} + \boldsymbol{b}^{(l)})$, where $\boldsymbol{w}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ and $\phi_l(.)$ is a non linear function applied independently on each component. Such neural networks are called Multi-Layer Perceptron (MLP). The architecture of these neural networks is illustrated on Figure 1.3. The parameter vector $\boldsymbol{\theta}$ contains the weight matrices $\boldsymbol{w}^{(l)}$ and the bias vector $\boldsymbol{b}^{(l)}$ for $l = 1, ..., L$.

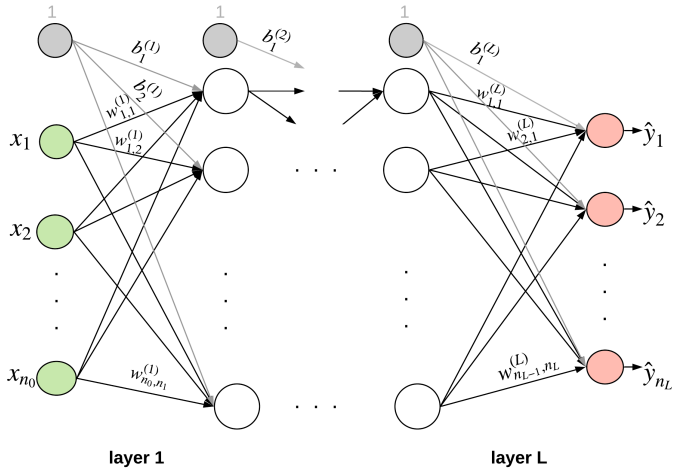


Figure 1.3: Classical Multi-Layer Perceptron architecture with input $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and output $\hat{\boldsymbol{y}} \in \mathbb{R}^{n_L}$.

The number of possible distinct configurations of a set of variables increase exponentially with the number of variables. Therefore for high-dimensional data, the number of possible configurations can become much larger than the size of the training set. By the lack of examples for a huge number of configuration of the input, the generalization task becomes more difficult. This phenomenon is known as the curse of dimensionality ([21, p.151]).

1.2.4 Train a neural network

Disposing of a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i),\ i = 1, ..., T\}$ and considering the Mean-Square-Error (MSE), the optimization problem for a feedforward neural network becomes:

$$\min_{\boldsymbol{\theta}} \quad E_T = \frac{1}{T}\sum_{i=1}^{T} \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) \ , \quad \text{with } \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \|\boldsymbol{y}_i - f(\boldsymbol{x}_i;\boldsymbol{\theta})\|_2^2. \tag{1.26}$$

To train a feedforward neural network model using the SGD (algorithm 1), we need to compute the estimation gradient given by expression (1.22). To compute the gradient of one element of the minibatch: $\nabla_{\boldsymbol{\theta}}\boldsymbol{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$, we will use the back-propagation algorithm [26][21, p.205]. Instead of calculating an analytical expression of the gradient which can be costly to evaluate, we will calculate the derivative with respect to the weights of each l layer by propagating the information throughout the network by recursively applying the chain rule. This computation will depend of the structure of the network. Let $E = \boldsymbol{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$ for some training data $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$. Let $\boldsymbol{a}^{(l)}$ and $\boldsymbol{z}^{(l)}$ denotes respectively the output vectors of the layer l before and after applying the non linear function ϕ_l: $a_i^{(l)} = \sum_{j=1}^{n_{l-1}} w_{i,j}^{(l)} z_j^{(l-1)} + b_i^{(l)}$, $z_i^{(l)} = \phi_l(a_i^{(l)})$ as illustrated on Figure 1.4.

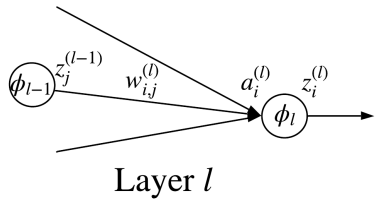


Figure 1.4

We can calculate the derivative with respect to the weight matrices:

$$\frac{\partial E}{\partial w_{i,j}^{(l)}} = \frac{\partial E}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial w_{i,j}^{(l)}} = \delta_i^{(l)} z_j^{(l-1)}, \quad \text{with } \delta_i^{(l)} = \frac{\partial E}{\partial a_i^{(l)}}. \tag{1.27}$$

Using the chain rule:

$$\delta_i^{(l)} = \sum_{k=1}^{n_l} \frac{\partial E}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_i^{(l)}} = \sum_{k=1}^{n_l} \delta_k^{(l+1)} \left(w_{k,i}^{(l+1)} \phi_l'(a_i^{(l)}) \right). \tag{1.28}$$

The values $a_i^{(l)}$ and $z_i^{(l)}$ can be obtained $\forall i \in \{1, ..., n_l\}$ $\forall l \in \{1, ..., L\}$ by evaluating the model at the given input and by memorizing during this forward-propagation the output values of each layer before and after applying the non linear function. Therefore, backward propagation is preceded by a forward-pass through the network.
Since the value $\delta_i^{(l)}$ depends only on those of the next layer and and since it can be calculated directly for the last layer: $\delta_i^{(L)} = -2(y_i - z_i^{(L)})\phi_L'(a_i^{(L)})$, we can calculate the derivative layer-by-layer starting from the last one. A similar result can be obtained for derivatives with respect to bias vectors.

1.2.5 Early stopping

When we train a model, it is important to determine when to stop. Often, the same behaviour as shown in Figure 1.1 appears when considering the number of epochs of the algorithm on the x-axis. The training error decreases steadily with the number of iterations, but the test error starts to increase after a while. Therefore, unlike an optimization problem, it is not enough to stop when the error on the training reaches a threshold. In order to detect when the overall trend of the validation error starts to increase (overfitting), we can use the so-called *early stopping algorithm* [21, p.239], which is an example of regularization strategy. There are several ways to implement early stopping. In this document, we consider the following implementation.
During the training, we evaluate the error on the validation set at each epoch. We record the parameters for which the validation set error is the lowest. If the validation error of the last epoch is greater than the average of the validation errors of the k precedent epochs, the

algorithm stops and we keep the model with the best saved parameters instead of the last ones. The parameters k giving the size of the averaging window is a hyperparameter of the model. Early stopping has the additional costs of evaluating the error of the validation set at each epoch and storing the best parameters seen so far. But these costs are negligible compared to the gain resulting from the reduction in the number of epochs and the gain in generalization.

Chapter 2

Integral operator

There are imaging applications (e.g., in astronomy as we will see in Chapter 6) where we observe an image "convolved" with a kernel whose shape depends on its localization. These operators are non shift-invariant and therefore cannot be modeled by a simple convolution.
We consider the general form of the linear integral operator H:

$$H[f](u) = \int_{\Omega} K(u,x)f(x)\ dx, \tag{2.1}$$

where K is called the kernel of the operator. If the kernel can be expressed as: $K(u,x) = g(u-x)$ for some function g, the operator is equivalent to a convolution operator.

The problem addressed in this paper concerns the approximation of these general integral operators based on a dataset containing the value of the integral operator for several inputs. Obviously, we wish to obtain a model able to generalize the result of the integral operator for inputs that are not part of the dataset. In addition, we wish to obtain an approximation model verifying the following constraints:

- It must have a low generalization error for a dataset containing few data. Indeed, for some applications, it is only possible to probe the integral operator for a restricted number of inputs.
- The cost to evaluate the model for a given input and the cost to compute its derivatives with respect to its inputs must be low.

Indeed, in some applications (as we will see in the Chapter 6 with the problem of exoplanet detection), a partially known integral operator may appear in the solving of an inverse problem. When solving such an optimization problem, it is necessary to evaluate at each iteration the operator and its derivatives with respect to its inputs. If the derivatives of the integral operator are not available, or if they are too costly to evaluate, this can make the optimization problem intractable. Therefore, if we have an approximation model for which we can efficiently compute the derivative and the result for a given input, we can replace the real operator by the approximation in the solving of the inverse problem.

This chapter first contains a general description of the model used to approximate an integral operator. Then, we will highlight the interest of having a model for which one can efficiently evaluate its value for a given input as well as its derivatives by means of the famous Lasso problem.

2.1 Overview of Diagonal/Circulant neural networks

Let $\boldsymbol{M} \in \mathbb{R}^{N\times N}$ be the discrete matrix associated with the integral operator H.
The mathematical model we are going to use to learn the matrix operator $\boldsymbol{M}$ is based on a feedforward neural network introduced in section 1.2.3 made up of an alternating product of diagonal and circulant matrices.
The model $f(.;\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$ consists of the composition of several functions $f_l(.;\boldsymbol{w}_l) : \mathbb{R}^N \to \mathbb{R}^N$ which are parameterized with $\boldsymbol{w} \in \mathbb{R}^N$. The vector $\boldsymbol{\theta}$ brings together the parameters from all the layers. Therefore, the model can be expressed as:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x};\boldsymbol{\theta}) = (f_L \circ ... \circ f_2 \circ f_1)(\boldsymbol{x}), \tag{2.2}$$

where $\boldsymbol{x}$ and $\hat{\boldsymbol{y}}$ are respectively the input and output of the model.
The function f_l is a linear mapping that can be defined by one of the following functions:

$$f_l(\boldsymbol{x};\boldsymbol{w}_l) = \boldsymbol{D}(\boldsymbol{w}_l)\boldsymbol{x} \quad \text{or} \quad f_l(\boldsymbol{x};\boldsymbol{w}_l) = \boldsymbol{C}(\boldsymbol{w}_l)\boldsymbol{x}. \tag{2.3}$$

Obviously, we have to alternate between diagonal and circulant matrices in the definition of the composition. Indeed two successive diagonal layers can be reduced into one diagonal layer as the product of two diagonal matrices is also a diagonal matrix. Similarly, the product of two circulant matrices is a circulant matrix. The idea is to combine the property of circulant matrices to model a shift-invariant operator for one-dimensional signals, with a diagonal operator in order to be able to reproduce a non shift-invariant behaviour. As we will see in details in Chapter 3.1.4, this network can model exactly any $\boldsymbol{M}$ operator with a sufficient number of factors.

Note, when working with two-dimensional signals, it has been shown in section 1.1.2 that the circular convolution which models a shift-invariant operator can be expressed in matrix form by means of a doubly-block-circulant matrix acting on the vectorized version of the 2D signal. Therefore, for two-dimensional signals, we will consider a network composed of an alternating product of diagonal and doubly-block-circulant matrices.

The precise description of the network layers is given in Chapter 3.

2.2 Computational motivation

As previously mentioned, in some applications, a partially known integral operator may appear in the solving of an inverse problem. We will now illustrate the interest of having an approximate model for which one can efficiently evaluate its value for a given input as well as its derivatives by means of the famous lasso problem.
The lasso problem (Least Absolute Shrinkage and Selection Operator) [12, 35] is an example of non-differentiable convex optimization problem. The problem consists in finding a sparse solution to a least square problem. It corresponds to the following optimization problem:

$$\min_{\boldsymbol{x}} \quad L(\boldsymbol{x}) = \frac{1}{2}\|\phi(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 + \lambda_\tau \|\boldsymbol{x}\|_1, \tag{2.4}$$

with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$ and $\lambda_\tau > 0$ which is a parameter that sets the trade-off between sparsity and the data fidelity of the solution.
Let $\phi(.) : \mathbb{R}^N \to \mathbb{R}^N$ be an operator (supposed to be differentiable) which can be partially known or for which the derivative are costly to evaluate.
We consider the following splitting $L(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$ with:

$$g(\boldsymbol{x}) = \frac{1}{2}\|\phi(\boldsymbol{x}) - \boldsymbol{y}\|_2^2, \qquad h(\boldsymbol{x}) = \lambda_\tau \|\boldsymbol{x}\|_1. \tag{2.5}$$

Both functions g and h are convex, but only the function g is differentiable.
Such particular convex optimisation problem where the objective function can be split into one differentiable and one non differentiable term can be solved by a proximal algorithm [35, 16].
In order to apply the proximal gradient method [35, p.148], we need to compute the following expressions:

- The gradient of g:
$$\nabla g(\boldsymbol{x}) = \boldsymbol{J}_\phi(\boldsymbol{x})^T(\phi(\boldsymbol{x}) - \boldsymbol{y}), \tag{2.6}$$
with $\boldsymbol{J}_\phi(\boldsymbol{x}) \in \mathbb{R}^{N\times N}$ which denotes the Jacobian matrix defined as $[\boldsymbol{J}_\phi(\boldsymbol{x})]_{i,j} = \frac{\partial \phi_i}{\partial x_j}(\boldsymbol{x})$.

- The proximal operator of the function h:
$$\begin{aligned}\mathbf{prox}_{\lambda h}(\boldsymbol{x}) &= \underset{\boldsymbol{u}\in\mathbb{R}^N}{\arg\min}\ \lambda_\tau\|\boldsymbol{u}\|_1 + \frac{1}{2\lambda}\|\boldsymbol{u}-\boldsymbol{x}\|_2^2 \\ &= \underset{\boldsymbol{u}\in\mathbb{R}^N}{\arg\min}\ \sum_{i=1}^{N}\left(|u_i| + \frac{1}{2\lambda\lambda_\tau}(u_i - x_i)^2\right).\end{aligned} \tag{2.7}$$
We define $l_i(u_i) = |u_i| + \frac{1}{2\lambda\lambda_\tau}(u_i - x_i)^2$. As the absolute value is non differentiable, we need to compute its subdifferential.

Definition 2.1. *The subdifferential of a continuous function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $\boldsymbol{x}_0 \in \mathbb{R}^n$ is defined as:*
$$\partial f(\boldsymbol{x}_0) = \{\boldsymbol{g} \in \mathbb{R}^n \mid \boldsymbol{g} \text{ is a subgradient of } f \text{ at } \boldsymbol{x}_0\}.$$
A vector $\boldsymbol{g} \in \mathbb{R}^n$ is a subgradient of f at $\boldsymbol{x}_0 \in \mathbb{R}^n$ iff:
$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}_0) + \boldsymbol{g}^T(\boldsymbol{y} - \boldsymbol{x}_0),\ \forall \boldsymbol{y} \in \mathbb{R}^n.$$
According to Definition 2.1, the subdifferential of the function $l_i(u_i)$ is given by:
$$\partial l_i(u_i) = \begin{cases}\operatorname{sign}(u_i) + \frac{1}{\lambda\lambda_\tau}(u_i - x_i) & \text{if } u_i \neq 0, \\ g + \frac{1}{\lambda\lambda_\tau}(u_i - x_i),\ \forall g \in [-1,1] & \text{if } u_i = 0.\end{cases} \tag{2.8}$$
As the function l_i is convex, the optimum is reached when:
$$\begin{aligned}0 \in \partial l_i(u_i) \iff u_i &= \begin{cases}\operatorname{sign}(x_i)(|x_i| - \lambda\lambda_\tau) & \text{if } |x_i| > \lambda\lambda_\tau, \\ 0 & \text{if } |x_i| \leq \lambda\lambda_\tau,\end{cases} \\ &= \mathcal{S}_{\lambda\lambda_\tau}(x_i).\end{aligned} \tag{2.9}$$
$\mathcal{S}_\tau$ denotes the soft-thresholding operator defined as $\mathcal{S}_\tau(x) = \operatorname{sign}(x)\max(|x| - \tau, 0)$.

We can now solve the optimization problem (2.4) using the proximal gradient method:
$$\begin{aligned}\boldsymbol{x}_{k+1} &= \mathbf{prox}_{\lambda_k h}(\boldsymbol{x}_k - \lambda_k \nabla g(\boldsymbol{x}_k)) \\ &= \mathcal{S}_{\lambda_k\lambda_\tau}\left(\boldsymbol{x}_k - \lambda_k \boldsymbol{J}_\phi(\boldsymbol{x}_k)^T(\phi(\boldsymbol{x}_k) - \boldsymbol{y})\right).\end{aligned} \tag{2.10}$$

The major computational cost of an iteration of scheme (2.10) can come from:

- The evaluation of the operator ϕ at given input: $\phi(\boldsymbol{x}_k)$.
- The computation of the derivatives of the operator at a given point: $\boldsymbol{J}_\phi(\boldsymbol{x}_k)$.
- The matrix-vector product $\boldsymbol{J}_\phi(\boldsymbol{x}_k)^T(\phi(\boldsymbol{x}_k) - \boldsymbol{y}))$.

This simple problem illustrates the advantage of having an approximation model $f(.;\boldsymbol{\theta})$ of $\phi(.)$ for which the evaluation and the derivatives can be calculated efficiently, because we could replace ϕ by f in (2.10).

If the operator $\phi(.)$ is such that: $\phi(\boldsymbol{x}) = \boldsymbol{M}\boldsymbol{x}$ with $\boldsymbol{M} \in \mathbb{R}^{N\times N}$ a dense and unstructured matrix, then the proximal gradient step can be rewritten as:

$$\boldsymbol{x}_{k+1} = \mathcal{S}_{\lambda_k\lambda_\tau}\left(\boldsymbol{x}_k - \lambda_k \boldsymbol{M}^T(\boldsymbol{M}\boldsymbol{x}_k - \boldsymbol{y})\right). \tag{2.11}$$

In that case, the computational bottleneck comes from the two matrix-vector products which are in $\mathcal{O}(N^2)$. Let $f(.,\boldsymbol{\theta})$ be the diagonal/circulant approximation model of $\boldsymbol{M}$, composed of k circulant layers and k diagonal layers. As the inference cost of a diagonal and a circulant layer are respectively in $\mathcal{O}(N)$ and $\mathcal{O}(N\log(N))$, the inference cost of the approximation model f is $\mathcal{O}(k(N\log(N)+N))$.
So, if we can approximate the matrix $\boldsymbol{M}$ with a product of few diagonal and circulant matrices (small k), the computational cost of an iteration of the proximal gradient method can be reduced.

Chapter 3

Diagonal/(doubly-block-)circulant neural network

The major computational bottleneck of the general neural network introduced in section (1.2.3) comes from the fully-connected layer which corresponds to a matrix-vector product followed by an element-wise activation function. Hence the cost of a forward in such a layer l is in $\mathcal{O}(n_{l-1}n_l)$ operations, where n_l indicates the dimensions of the output of the layer l. And as we have shown in section (1.2.4), the general backward propagation algorithm to compute the gradient requires also $\mathcal{O}(n_{l-1}n_l)$ for each layer l.
We can consider the two following approaches in order to reduce the computational cost:

- Classical neural network as the MLP fix at the beginning the architecture of the network, and the training phase can not improve it. To counteract this limitation, it exists method to learn during the training the important connections in the network and delete without loss of precision those that are not useful in order to reduce the storage and the computational complexity. Indeed, if we can get sparse layers, it will reduce the inference cost. It is called the weights pruning. An example of such a method given in [23]. It first trains the network in order to deduce the important connections and delete the unimportant connections, then it re-trains the network in order to improve the important weights in this reduced architecture. This process can be repeated several times. But it has two major drawbacks, the resulting layers will be unstructured after the pruning (so the inference cost can still be high) and the pruning has an additive cost during the training phase as it requires to train the network several times in order to learn the architecture.

- Sometimes, we can use prior information (as architecture or properties) on the real model in order to increase the generalization ability of a model. There are applications, as we will see later in Chapter 4, for which we know in advance that some of the parameters of the model should have the same value. Therefore, we can impose some of the parameters of the network to be equal. This procedure is called *parameters sharing* [21, p.246]. It is a regularization technique because it reduces the functional set of functions that can be learned by our model by constraining certain parameters to be equal. Moreover, we can also impose some parameters to a certain value. In that case, we talk of *parameters tying* [21, p.246].

For these reasons, we will focus on the second approach to reduce the bottleneck problem, which consists in directly replacing dense unstructured matrices with structured matrices whose matrix-vector product can be computed efficiently. In particular, we will analyze networks composed of the following types of structured matrices: diagonal and circulant/doubly-block-circulant. We will see that these layers reduce complexity for both inference and learning, while also reducing the size of parameter storage.

3.1 Diagonal/Circulant neural network

In this section, we will show that the cost of inference and the cost of training of a network where the layers are alternating diagonal and circulant matrices can be considerably reduced compared to a network made of dense and unstructured layers. We will also see the expressive power of such a network, i.e. when the circulant matrices are combined with the diagonal matrices.

3.1.1 Layers analysis

We will analyze the two kinds of layers constituting the network and see how to optimize their weights. Let $\boldsymbol{x} \in \mathbb{R}^N$ and $\boldsymbol{y} \in \mathbb{R}^N$ be respectively the input and the output of the layer. Let $\boldsymbol{w}$ be the parameters of the layer.
As we have shown in (1.2.4), we need to derive two expressions in order to execute the backward propagation algorithm and train the network: the derivative of the loss (E) defined in (1.26) with respect to the parameters and the inputs of the layer ($\frac{\partial E}{\partial \boldsymbol{w}}$ and $\frac{\partial E}{\partial \boldsymbol{x}}$).

Particular layer

We consider a linear layer of matrix $\boldsymbol{A}(\boldsymbol{w}) \in \mathbb{R}^{N\times N}$ generated by the vector of weights $\boldsymbol{w} \in \mathbb{R}^N$, then the relation input/output is given by:

$$\boldsymbol{y} = \boldsymbol{A}(\boldsymbol{w})\boldsymbol{x}. \tag{3.1}$$

We assume that the matrix $\boldsymbol{A}(\boldsymbol{w})$ has the following properties:

$$\boldsymbol{A}(\boldsymbol{w})\boldsymbol{x} = \boldsymbol{A}(\boldsymbol{x})\boldsymbol{w},\ \boldsymbol{A}(\boldsymbol{w})^T\boldsymbol{x} = \boldsymbol{A}(\boldsymbol{x})^T\boldsymbol{w}. \tag{3.2}$$

We can derive the expression of the derivative of the loss with respect to the inputs:

$$\frac{\partial E}{\partial x_i} = \sum_{j=1}^{N} \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_i} = \sum_{j=1}^{N} \frac{\partial E}{\partial y_j}\ [\boldsymbol{A}(\boldsymbol{w})]_{j,i}\ ,\ i = 1, ..., N. \tag{3.3}$$

Therefore, combining (3.3) with the properties (3.2), we have:

$$\begin{cases} \nabla_{\boldsymbol{x}} E = \boldsymbol{A}(\boldsymbol{w})^T \nabla_{\boldsymbol{y}} E = \boldsymbol{A}(\nabla_{\boldsymbol{y}} E)^T \boldsymbol{w}, \\ \nabla_{\boldsymbol{w}} E = \boldsymbol{A}(\boldsymbol{x})^T \nabla_{\boldsymbol{y}} E = \boldsymbol{A}(\nabla_{\boldsymbol{y}} E)^T \boldsymbol{x}. \end{cases} \tag{3.4}$$

Circulant layer

The idea is to learn how to perform invariant spatial transformations on the input signal in order to reduce the number of parameters. The circulant layer is based on the idea of parameter sharing. Indeed, by the Definition 1.1, several elements in the matrix are the same. Let a circulant layer generated by the weights $\boldsymbol{w} \in \mathbb{R}^N$, then the relation input/output is:

$$\boldsymbol{y} = \boldsymbol{C}(\boldsymbol{w})\boldsymbol{x}. \tag{3.5}$$

Using the properties of the circulant matrices shown in (1.5), the inference for a given input $\boldsymbol{x}$ can be computed in $\mathcal{O}(N\log(N))$ using FFT algorithm. But what about its gradients.
As we have shown in (1.6), the circulant matrix verify the property (3.2), therefore we get:

$$\begin{cases} \nabla_{\boldsymbol{x}} E = \boldsymbol{C}\left(\nabla_{\boldsymbol{y}} E\right)^T \boldsymbol{w} \Rightarrow \nabla_{\boldsymbol{x}} E = \mathcal{F}^{-1}\left(\sqrt{N}\ \mathcal{F}\left(\nabla_{\boldsymbol{y}} E\right)^* \odot \mathcal{F}(\boldsymbol{w})\right), \\ \nabla_{\boldsymbol{w}} E = \boldsymbol{C}\left(\nabla_{\boldsymbol{y}} E\right)^T \boldsymbol{x} \Rightarrow \nabla_{\boldsymbol{w}} E = \mathcal{F}^{-1}\left(\sqrt{N}\ \mathcal{F}\left(\nabla_{\boldsymbol{y}} E\right)^* \odot \mathcal{F}(\boldsymbol{x})\right), \end{cases} \tag{3.6}$$

where we used the matrix-vector property (1.5).
We find an interesting property of the circulant matrix: the computation of their gradient with respect to their weights/inputs can also be computed as a circulant matrix-vector product. Using again the convolution theorem, the gradients can be efficiently computed with FFT. Hence both, the forward of data $\boldsymbol{x}$ (the inference) and the computation of the gradients (the training) can be computed efficiently with FFT in $\mathcal{O}(N\log(N))$ instead of $\mathcal{O}(N^2)$ for a fully connected layer. The storage complexity also decrease from $\mathcal{O}(N^2)$ for a fully-connected layer to $\mathcal{O}(N)$.

Diagonal layer

The diagonal layer is based on the idea of parameter tying to reduce the number of parameters. Indeed, all the extra-diagonals elements are fixed to 0.
Let a diagonal layer generated by the weights $\boldsymbol{w} \in \mathbb{R}^N$, then the relation input/output is:

$$\boldsymbol{y} = \boldsymbol{D}(\boldsymbol{w})\boldsymbol{x}. \tag{3.7}$$

The diagonal matrix verify the property (3.2): $\boldsymbol{D}(\boldsymbol{w})\boldsymbol{x} = \boldsymbol{D}(\boldsymbol{x})\boldsymbol{w}$. Therefore, we get:

$$\begin{cases} \nabla_{\boldsymbol{x}} E = \boldsymbol{D}\left(\nabla_{\boldsymbol{y}} E\right)\boldsymbol{w}, \\ \nabla_{\boldsymbol{w}} E = \boldsymbol{D}\left(\nabla_{\boldsymbol{y}} E\right)\boldsymbol{x}. \end{cases} \tag{3.8}$$

The computation of the gradients of a diagonal layer with respect to their weights/inputs can be computed as a diagonal matrix-vector product. Hence both, the forward of data $\boldsymbol{x}$ and the computation of the gradients can be computed efficiently in $\mathcal{O}(N)$. The storage complexity is in $\mathcal{O}(N)$.

3.1.2 Evaluation of the derivative of the loss

We will apply the back-propagation algorithm presented for general fully connected layers in section (1.2.4) and show that it can be computed efficiently for these structured neural network. Let $\boldsymbol{a}^{(l-1)}$ and $\boldsymbol{a}^{(l)}$ be respectively the input and output of the layer l and $\boldsymbol{w}^{(l)}$ denotes the learnable parameters of layer l.

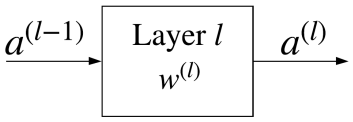


Figure 3.1: Input/output of layer l.

Considering layer l, we have shown with the second equation of (3.6) and (3.8) that:

$$\nabla_{\boldsymbol{w}^{(l)}} E = \begin{cases} \boldsymbol{C}\left(\nabla_{\boldsymbol{a}^{(l)}} E\right)^T \boldsymbol{a}^{(l-1)} & \text{if layer } l \text{ is a circulant layer,} \\ \boldsymbol{D}\left(\nabla_{\boldsymbol{a}^{(l)}} E\right)\boldsymbol{a}^{(l-1)} & \text{if layer } l \text{ is a diagonal layer.} \end{cases} \tag{3.9}$$

All the $\boldsymbol{a}^{(l)}$ values are known as they are evaluated during the forward of the input $\boldsymbol{x}$. But as $\boldsymbol{a}^{(l)}$ is the input of the next layer $l+1$, we can use the first equation of (3.6) and (3.8) to determine:

$$\nabla_{\boldsymbol{a}^{(l)}} E = \begin{cases} \boldsymbol{C}\left(\nabla_{\boldsymbol{a}^{(l+1)}} E\right)^T \boldsymbol{w}^{(l+1)} & \text{if layer } (l+1) \text{ is a circulant layer,} \\ \boldsymbol{D}\left(\nabla_{\boldsymbol{a}^{(l+1)}} E\right)\boldsymbol{w}^{(l+1)} & \text{if layer } (l+1) \text{ is a diagonal layer.} \end{cases} \tag{3.10}$$

Hence in order to compute the gradient of layer l, we need those of layer $(l+1)$. That is why, we need to compute the gradient for the last layer and then back-propagate their values to the previous layers. The gradients of the last layer L can be easily computed as: $\boldsymbol{a}^{(L)} = f(\boldsymbol{x};\boldsymbol{\theta})$. So:

$$\nabla_{\boldsymbol{a}^{(L)}} E = -2(\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})), \tag{3.11}$$

where $f(.;\boldsymbol{\theta})$ gives the output of the network.

3.1.3 Fast Fourier Transform

The FFT algorithm [19, 7] which efficiently computes the discrete Fourier transform, is the basic computational block of the diagonal-circulant neural network. Indeed, as we have shown, both inference and training requires the calculation of some DFT.
Starting from the definition of the DFT given in (A.15) and considering that N is a power of 2, we can rewrite the DFT: $\hat{f}[k]$ for $k = 0, ..., N-1$ as:

$$\begin{aligned}\hat{f}[k] &= \sum_{r=0}^{\frac{N}{2}-1} f[2r]\omega_N^{k2r} + \sum_{r=0}^{\frac{N}{2}-1} f[2r+1]\omega_N^{k(2r+1)} \\ &= \left(\sum_{r=0}^{\frac{N}{2}-1} f[2r]\omega_{\frac{N}{2}}^{kr}\right) + \omega_N^k \left(\sum_{r=0}^{\frac{N}{2}-1} f[2r+1]\omega_{\frac{N}{2}}^{kr}\right).\end{aligned} \tag{3.12}$$

We define $\hat{f}_e[k]$ and $\hat{f}_o[k]$ (for $k = 0, ..., \frac{N}{2}$), the DFT of the signals of size $\frac{N}{2}$ made up respectively of the even and odd index elements of $f[n]$. Using the fact that $\omega_{\frac{N}{2}}^{kr}$ is periodic of period $\frac{N}{2}$ in k and that: $\omega_N^k = -\omega_N^{k-\frac{N}{2}}$, we can rewrite (3.12) as:

$$\hat{f}[k] = \begin{cases} \hat{f}_e[k] + \omega_N^k \hat{f}_o[k] & \text{if } k \in \left\{0, ..., \dfrac{N}{2} - 1\right\}. \\ \hat{f}_e\left[k - \dfrac{N}{2}\right] - \omega_N^{k-\frac{N}{2}} \hat{f}_o\left[k - \dfrac{N}{2}\right] & \text{if } k \in \left\{\dfrac{N}{2}, ..., N-1\right\}. \end{cases} \tag{3.13}$$

Therefore we can compute the DFT of size N by computing two DFT of size $\frac{N}{2}$ plus an additional recombination step called: butterfly algorithm given in (3.13). It is also illustrated on Figure 3.2. We can apply recursively (3.13) on the even and odd signal of size $\frac{N}{2}$, leading to a divide-and-conquer implementation of the DFT computation: the FFT algorithm. Let $C(N)$ denotes the complexity for the execution of the FFT algorithm on an input of size N, we have the following relation:

$$\begin{aligned} C(N) &= 2C\left(\frac{N}{2}\right) + \mathcal{O}(N) \\ &\Rightarrow C(N) \in \mathcal{O}(N\log(N)). \end{aligned} \tag{3.14}$$

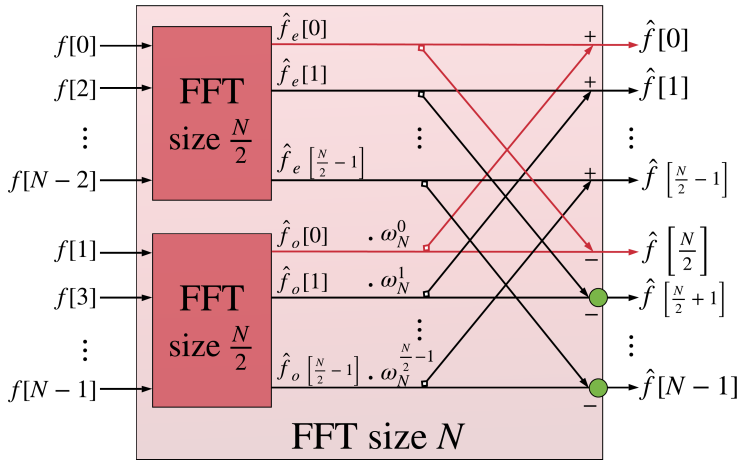


Figure 3.2: Recursive property of the FFT algorihtm (N is a power of 2).

As we only consider real inputs (without imaginary part) for the neural network, we can save computations based on the hermitian-symmetry property of the DFT for purely real signal. Indeed the value at frequency k is the complex conjugate of the value at frequency $N-k$ for $k = 1, ..., N-1$:

$$\hat{f}[N-k] = (\hat{f}[k])^* \quad , \quad k = 1, ..., N-1. \tag{3.15}$$

Therefore, we can avoid the redundant computation done by FFT by computing only the $\left\lfloor \frac{N}{2} \right\rfloor + 1$ first elements of the DFT of the inputs. It is illustrated on Figure 3.2 where we can avoid the computation of the green points. The complexity does not change but the computation is faster. This improved procedure is called the real-fast-Fourier transform (RFFT) [43]. It is this algorithm for the DFT computation that is used in the neural network implementation.

3.1.4 Motivations

Because of the computational (inference and training) and storage complexity reduction, the diagonal/circulant neural networks seems tempting. But it stays to analyze the ability of those networks to approximate functions. It is an appropriate question as the number of parameters is much lower than the unstructured dense matrices based networks. Moreover, the network is only linear without activation functions.
But the Theorem 3.1 states that circulant matrices when combined with diagonal matrices can be used to represent any linear transformation.

Theorem 3.1 (Huhtanen & Perämäki, 2015)**.** *[25, Expression 1.2],[4, Theorem 1] Any matrix $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ can be factorized as the product of circulant and diagonal matrices with the number of factors being at most $2n-1$.*
Precisely: $\exists\ m \leq n$ such that:

$$\boldsymbol{A} = \left(\prod_{i=1}^{m-1} \boldsymbol{D}_i \boldsymbol{C}_i \right) \boldsymbol{D}_m,$$

with $\boldsymbol{D}_i$ $(i = 1, ..., m)$ a diagonal matrix and $\boldsymbol{C}_i$ $(i = 1, ..., m-1)$ a circulant matrix.

But, the Theorem 3.1 has two major drawbacks: first the number of matrices in the product to approximate the matrix $\boldsymbol{A}$ depends only of the size of the matrix: n and not of the matrix itself. Secondly, the theorem provides nothing about the expressive power of a diagonal-circulant product when the number of factors is lower than the require number by the theorem: at most $2n-1$. Moreover, some real matrices have a decomposition involving a smaller number of factors.

3.2 Diagonal/Doubly-block-circulant neural network

In this section, we will show that the cost of inference and the cost of training such a network can be significantly reduced compared to networks based on fully connected layers. We will also see the expressive power of such a network, i.e. when the doubly-block-circulant (DBC) matrices are combined with the diagonal matrices.

3.2.1 Layers analysis

We will analyze the two kinds of layers constituting the network and see how to optimize their weights. Let $\boldsymbol{w} \in \mathbb{R}^{M \times N}$ be the parameters of the layer. Let $\boldsymbol{x} \in \mathbb{R}^{M \times N}$ and $\boldsymbol{y} \in \mathbb{R}^{M \times N}$ be respectively the input and the output of the layer.

Doubly-block-circulant layer

As for circulant layer, the DBC layer is based on the idea of parameter sharing. Indeed, by the Definition 1.2, several elements in the matrix are the same. Let a DBC layer generated by the weights $\boldsymbol{w}$, then the relation input/output is:

$$\text{vec}(\boldsymbol{y}) = \boldsymbol{B}(\boldsymbol{w})\text{vec}(\boldsymbol{x}), \tag{3.16}$$

which is, as we have shown in (1.19), equivalent to a two-dimensional circular convolution between the two-dimensional signals defined by the matrices $\boldsymbol{w}$ and $\boldsymbol{x}$.
As we have shown in (1.17), the DBC matrix verify the property (3.2), therefore we get:

$$\begin{cases} \nabla_{\boldsymbol{x}} E = \boldsymbol{B}\left(\nabla_{\boldsymbol{y}} E\right)^T \text{vec}(\boldsymbol{w}) \Rightarrow \nabla_{\boldsymbol{x}} E = \mathcal{F}_{2D}^{-1}\left(\sqrt{MN}\ \mathcal{F}_{2D}\left(\nabla_{\boldsymbol{y}} E\right)^* \odot \mathcal{F}_{2D}(\boldsymbol{w})\right), \\ \nabla_{\boldsymbol{w}} E = \boldsymbol{B}\left(\nabla_{\boldsymbol{y}} E\right)^T \text{vec}(\boldsymbol{x}) \Rightarrow \nabla_{\boldsymbol{w}} E = \mathcal{F}_{2D}^{-1}\left(\sqrt{MN}\ \mathcal{F}_{2D}\left(\nabla_{\boldsymbol{y}} E\right)^* \odot \mathcal{F}_{2D}(\boldsymbol{x})\right), \end{cases} \tag{3.17}$$

where we used the matrix-vector property (1.16).
We get an interesting property of the DBC layers: the computation of their gradient with respect to their weights/inputs can also be computed as a DBC matrix-vector product.
Using again the convolution theorem, the gradients can be efficiently computed with 2D-FFT. Hence both, the forward of data $\boldsymbol{x}$ (the inference) and the computation of the gradients (the training) can be computed efficiently with FFT in $\mathcal{O}(MN(\log(M)+\log(N))$ instead of $\mathcal{O}(M^2N^2)$ for a fully connected layer. The storage complexity also decrease from $\mathcal{O}(M^2N^2)$ for a fully-connected layer to $\mathcal{O}(MN)$.

Diagonal layer

The diagonal layer is similar to the one analyzed in (3.1.1) excepted that we define the relation input/output as:

$$\text{vec}(\boldsymbol{y}) = \boldsymbol{D}(\text{vec}(\boldsymbol{w}))\text{vec}(\boldsymbol{x}), \tag{3.18}$$

which is equivalent to a component-wise product in matrix form:

$$\boldsymbol{y} = \boldsymbol{w} \odot \boldsymbol{x}. \tag{3.19}$$

Hence, the costs of inference and training are in $\mathcal{O}(MN)$.

3.2.2 2D-Fast Fourier Transform

As we have shown, the computation of the 2D-DFT is the bottleneck of the diagonal/doubly-block-circulant neural network. We consider a signal $\boldsymbol{f} \in \mathbb{R}^{M \times N}$. Based on the matrix-vector form of the 2D-DFT (1.10) and using Theorem 3.3, the 2D-DFT can be rewritten as:

$$\begin{aligned} \text{vec}(\hat{\boldsymbol{f}}) &= (\boldsymbol{F_N} \otimes \boldsymbol{F_M})\text{vec}(\boldsymbol{f}) \\ &= \text{vec}(\boldsymbol{F_M} \boldsymbol{f} {\boldsymbol{F_N}}^T). \end{aligned} \tag{3.20}$$

Therefore using the symmetric property of the Fourier matrix, we have:

$$\hat{\boldsymbol{f}} = \boldsymbol{F_M} \boldsymbol{f} \boldsymbol{F_N}. \tag{3.21}$$

Using the following decomposition:

$$\begin{cases} \hat{\boldsymbol{p}} = \boldsymbol{F_M} \boldsymbol{f}, \\ \hat{\boldsymbol{f}} = (\boldsymbol{F_N} \hat{\boldsymbol{p}}^T)^T, \end{cases} \tag{3.22}$$

we can compute the 2D-DFT by computing N 1-DFT of size M of the columns of $\boldsymbol{f}$ and then M 1-DFT of size N of the lines of $\hat{\boldsymbol{p}}$. Therefore, using the FFT algorithm, the complexity is in $\mathcal{O}(MN(\log(M) + \log(N)))$.

3.2.3 Motivations

Additionally to the computational interest (both on inference and training) of using the diagonal/doubly-block-circulant neural network introduced before, we will see that these networks can theoretically approximate any linear transformation applied on a matrix.
We want to obtain for the 2D case a result similar to the Theorem 3.1 for the 1D case, in order to justify the use of diagonal/doubly-block-diagonal neural network when we work with two-dimensional problem as it is the case in image processing.
To obtain such a result, we will need several preliminary theorems/properties involving the Kronecker product.

Theorem 3.1. *Let $\boldsymbol{A}_i \in \mathbb{R}^{M \times M}$ and $\boldsymbol{B}_i \in \mathbb{R}^{N \times N}$ with $i = 1, ..., n$. Then the following relation holds:*

$$\left(\prod_{i=1}^{n} \boldsymbol{A}_i \right) \otimes \left(\prod_{i=1}^{n} \boldsymbol{B}_i \right) = \prod_{i=1}^{n} (\boldsymbol{A}_i \otimes \boldsymbol{B}_i).$$

Theorem 3.2. *If matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible, then $(\boldsymbol{A} \otimes \boldsymbol{B})$ is invertible and its inverse is given by:*

$$(\boldsymbol{A} \otimes \boldsymbol{B})^{-1} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}.$$

Theorem 3.3. *For any matrices $\boldsymbol{B} \in \mathbb{R}^{m \times m}$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{C} \in \mathbb{R}^{n \times n}$, the following relation holds:*

$$\text{vec}(\boldsymbol{BAC}) = (\boldsymbol{C}^T \otimes \boldsymbol{B})\text{vec}(\boldsymbol{A}).$$

Theorem 3.4. *The matrix resulting from the Kronecker product of two circulant matrices is a doubly-block-circulant matrix.*

Proof. Let $\boldsymbol{C}_1 \in \mathbb{R}^{M \times M}, \boldsymbol{C}_2 \in \mathbb{R}^{N \times N}$, two circulant matrices and $\boldsymbol{B} \in \mathbb{R}^{MN \times MN}$, the matrix resulting from the Kronecker product of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$:

$$\begin{aligned} \boldsymbol{B} &= \boldsymbol{C}_1 \otimes \boldsymbol{C}_2 \\ &= \left({\boldsymbol{F_M}}^{-1} \boldsymbol{D}_1 \boldsymbol{F_M} \right) \otimes \left({\boldsymbol{F_N}}^{-1} \boldsymbol{D}_2 \boldsymbol{F_N} \right) && \text{(Theorem 1.1)} \\ &= ({\boldsymbol{F_M}}^{-1} \otimes {\boldsymbol{F_N}}^{-1})(\boldsymbol{D}_1 \otimes \boldsymbol{D}_2)(\boldsymbol{F_M} \otimes \boldsymbol{F_N}) && \text{(Theorem 3.1)} \\ &= (\boldsymbol{F_M} \otimes \boldsymbol{F_N})^{-1}(\boldsymbol{D}_1 \otimes \boldsymbol{D}_2)(\boldsymbol{F_M} \otimes \boldsymbol{F_N}). && \text{(Theorem 3.2)} \end{aligned} \tag{3.23}$$

$\boldsymbol{D}_1 \otimes \boldsymbol{D}_2$ is a diagonal matrix because the Kronecker product of two diagonal matrices is a diagonal matrix. Therefore, the matrix $\boldsymbol{B}$ verify the Theorem 1.2 and so, $\boldsymbol{B}$ is a doubly-block-circulant matrix. □

Theorem 3.5. *Let $\boldsymbol{f} \in \mathbb{R}^{M\times N}$, $\boldsymbol{A}^{(1)} \in \mathbb{R}^{M\times M}$ and $\boldsymbol{A}^{(2)} \in \mathbb{R}^{N\times N}$, then $\exists\ m \leq \max(M,N)$ such that:*

$$\operatorname{vec}\left(\boldsymbol{A}^{(1)}\boldsymbol{f}\boldsymbol{A}^{(2)}\right) = \left[\left(\prod_{i=1}^{m-1} \boldsymbol{D}_i\boldsymbol{B}_i\right)\boldsymbol{D}_m\right]\operatorname{vec}(\boldsymbol{f}),$$

with $\boldsymbol{D}_i$ $(i = 1,...,m)$ a diagonal matrix and $\boldsymbol{B}_i$ $(i = 1,...,m-1)$ a doubly-block-circulant matrix.

Proof. Let $\boldsymbol{A}^{(1)} \in \mathbb{R}^{M\times M}$ and $\boldsymbol{A}^{(2)} \in \mathbb{R}^{N\times N}$. Theorem 3.1 states that $\exists\ m \leq M$ and $\exists\ n \leq N$ such that:

$$\begin{cases} \boldsymbol{A}^{(1)} = \left(\prod_{i=1}^{m-1} \boldsymbol{D}_i^{(1)}\boldsymbol{C}_i^{(1)}\right)\boldsymbol{D}_m^{(1)}, \\ \boldsymbol{A}^{(2)} = \left(\prod_{i=1}^{n-1} \boldsymbol{D}_i^{(2)}\boldsymbol{C}_i^{(2)}\right)\boldsymbol{D}_n^{(2)}. \end{cases} \tag{3.24}$$

We will do the proof for $m \geq n$, but a similar proof leading to a similar result can be done for $m < n$.

$$\begin{aligned}
\operatorname{vec}(\boldsymbol{A}^{(1)}\boldsymbol{f}\boldsymbol{A}^{(2)}) &= \left[\left(\boldsymbol{A}^{(2)}\right)^T \otimes \boldsymbol{A}^{(1)}\right]\operatorname{vec}(\boldsymbol{f}) && \text{(Theorem 3.3)} \\
&= \left[\left(\boldsymbol{D}_n^{(2)}\prod_{i=1}^{n-1}\left(\boldsymbol{C}_{n-i}^{(2)}\right)^T\boldsymbol{D}_{n-i}^{(2)}\right)\otimes\left(\boldsymbol{D}_1^{(1)}\prod_{i=1}^{m-1}\boldsymbol{C}_i^{(1)}\boldsymbol{D}_{i+1}^{(1)}\right)\right]\operatorname{vec}(\boldsymbol{f}) && (\ 3.24) \\
&= \left[\boldsymbol{D}_n^{(2)}\otimes\boldsymbol{D}_1^{(1)}\right]\left[\prod_{i=1}^{n-1}\left(\left(\boldsymbol{C}_{n-i}^{(2)}\right)^T\otimes\boldsymbol{C}_i^{(1)}\right)\left(\boldsymbol{D}_{n-i}^{(2)}\otimes\boldsymbol{D}_{i+1}^{(1)}\right)\right] \\
&\quad \left[\prod_{i=n}^{m-1}\left(\boldsymbol{I}_{\boldsymbol{N}}\otimes\boldsymbol{C}_i^{(1)}\right)\left(\boldsymbol{I}_{\boldsymbol{N}}\otimes\boldsymbol{D}_{i+1}^{(1)}\right)\right]\operatorname{vec}(\boldsymbol{f}) && \text{(Theorem 3.1)} \\
&= \left[\left(\prod_{i=1}^{m-1}\boldsymbol{D}_i\boldsymbol{B}_i\right)\boldsymbol{D}_m\right]\operatorname{vec}(\boldsymbol{f}),
\end{aligned}$$

with

$$\boldsymbol{D}_i = \begin{cases} \boldsymbol{D}_{n+1-i}^{(2)}\otimes\boldsymbol{D}_i^{(1)} & i = 1,...,n, \\ \boldsymbol{I}_{\boldsymbol{N}}\otimes\boldsymbol{D}_i^{(1)} & i = n+1,...,m, \end{cases} \tag{3.25}$$

$$\boldsymbol{B}_i = \begin{cases} \left(\boldsymbol{C}_{n-i}^{(2)}\right)^T\otimes\boldsymbol{C}_i^{(1)} & i = 1,...,n-1, \\ \boldsymbol{I}_{\boldsymbol{N}}\otimes\boldsymbol{C}_i^{(1)} & i = n,...,m-1. \end{cases} \tag{3.26}$$

The matrices $\boldsymbol{D}_i \in \mathbb{R}^{MN\times MN}$ with $i = 1,...,m$ are diagonal as the Kronecker product of two diagonal matrices is diagonal.
The matrices $\boldsymbol{B}_i \in \mathbb{R}^{MN\times MN}$ with $i = 1,...,m-1$ are doubly-block-circulant as the Kronecker product of two circulant matrices is doubly-block-circulant (Theorem 3.4). □

Theorem 3.6. *Let $\boldsymbol{f} \in \mathbb{R}^{M\times N}$ and $\boldsymbol{A}^{(1)} \in \mathbb{R}^{M\times M}$, then $\exists\ m \leq M$ such that:*

$$\operatorname{vec}\left(\boldsymbol{A}^{(1)}\boldsymbol{f}\right) = \left[\left(\prod_{i=1}^{m-1}\boldsymbol{D}_i\boldsymbol{B}_i\right)\boldsymbol{D}_m\right]\operatorname{vec}(\boldsymbol{f}),$$

with $\boldsymbol{D}_i$ $(i = 1,...,m)$ a diagonal matrix and $\boldsymbol{B}_i$ $(i = 1,...,m-1)$ a doubly-block-circulant matrix.

Proof. Based on the proof of Theorem 3.5 with $\boldsymbol{A}^{(2)} = \boldsymbol{I_N}$ (and so $n = 1$), we get:

$$\begin{aligned} \operatorname{vec}(\boldsymbol{A}^{(1)} \boldsymbol{f} \boldsymbol{A}^{(2)}) &= \operatorname{vec}(\boldsymbol{A}^{(1)} \boldsymbol{f}) \\ &= \left[\left(\prod_{i=1}^{m-1} \boldsymbol{D}_i \boldsymbol{B_i} \right) \boldsymbol{D}_m \right] \operatorname{vec}(\boldsymbol{f}), \end{aligned} \tag{3.27}$$

with

$$\begin{cases} \boldsymbol{D}_i = \boldsymbol{I_N} \otimes \boldsymbol{D}_i^{(1)} & i = 1, ..., m, \\ \boldsymbol{B}_i = \boldsymbol{I_N} \otimes \boldsymbol{C}_i^{(1)} & i = 1, ..., m-1. \end{cases} \tag{3.28}$$

□

Theorems 3.5 and 3.6 states that any linear transformation applied (both at left and at right or only at left) to a matrix can be rewritten as a product composed of alternating product of diagonal and doubly-block-circulant matrices. Therefore it seems pertinent to use neural network having such a structure to learn linear operators applied on two-dimensional signal (typically images). But as for the 1D case (Theorem 3.1), Theorems 3.5 and 3.6 state nothing about the expressive power of diagonal/doubly-block-circulant network when the number of factors used is lower than the required number.

Chapter 4

Theoretical justification of the decomposition

In Chapter 3, we saw that any linear operator applying to vectors and any linear operator applying to matrices can be decomposed as a product of diagonal-circulant matrices and diagonal/doubly-block-circulant matrices, respectively. But the number of factors required may be large, of the order of the dimension of the associated matrix of the operator. In this section, we will see that linear operators requiring few factors appear in several domains. Typically, we will see such a decomposition in the optical field. Then, we will see that a variant of this decomposition appears in the context of solving differential equations with finite difference schemes.

4.1 Diffractive optical systems

As we will see, the diagonal/doubly-block-circulant factorization of matrices appears in the problem of designing a particular optical system. First, we will state the inherent properties of thin lens [22, Chapter 5] in the context of coherent optical system, and then motivate our particular factorization.

4.1.1 Thin lens

The most important component of optical systems is the lens. By studying the lens in the wave-optics approach, that is by considering the diffraction effects, we can show some interesting Fourier properties of the lens. A lens is made of a material through which the light passes. Light moves there more slowly than in air, so it is characterized by a refraction index n (considered normalized with respect to that of air). We consider thin lens, that is to say, a ray entering in (x, y) in the lens comes out approximately at the same position on the opposite face. This assumption involves that the lens only delays the incident wave-front proportionally to the thickness of the lens at each point. Let $\Delta(x, y)$ denotes the thickness of the lens at position (x, y) and $\Delta_0 = \Delta(0, 0)$, the maximum thickness of the lens (see Figure 4.1). Let U_l, the incident field on the first plane l, then the field on the second plane is given by:

$$U_r(x, y) = h(x, y)U_l(x, y) = e^{-\mathrm{i}\phi(x,y)}U_l(x, y), \tag{4.1}$$

with $\phi(x, y)$ the total phase delay caused by the lens between the planes l and r at position (x, y). It is given by:

$$\phi(x, y) = \underbrace{kn\Delta(x, y)}_{\phi_1(x,y)} + \underbrace{k(\Delta_0 - \Delta(x, y))}_{\phi_2(x,y)} = k\Delta_0 + k(n - 1)\Delta(x, y), \tag{4.2}$$

with $\phi_1(x,y)$ which is the phase delay resulting of the lens and $\phi_2(x,y)$ which is the phase delay resulting from the remaining free space between the plane l and r. The parameter $k = \frac{2\pi}{\lambda}$ is the coefficient of proportionality and λ is the wavelength.

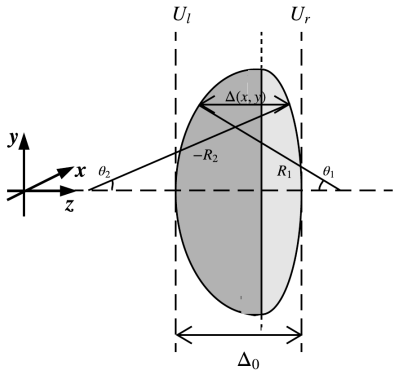


Figure 4.1: Thin lens with spherical boundary.

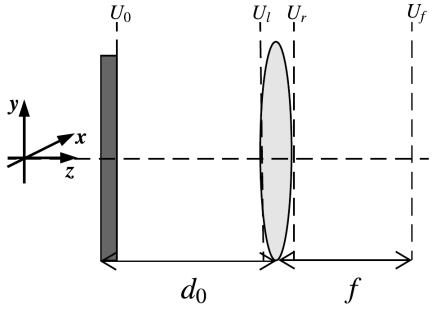


Figure 4.2: Object in front of the lens.

Considering a spherical lens (illustrated on Figure 4.1), that is both surface of the lens are given by a semi-sphere whose the radius of convergence are respectively R_1 and R_2 (by convention: $R_1{>}0$, $R_2 < 0$), we can get the following expression of $\Delta(x,y)$:

$$\Delta(x,y) = \Delta_0 - R_1\left(1 - \sqrt{1 - \frac{x^2+y^2}{R_1^2}}\right) + R_2\left(1 - \sqrt{1 - \frac{x^2+y^2}{R_2^2}}\right). \tag{4.3}$$

Let θ_i denotes the angle between the z-axis and the (x,y) point considered measured from the center of convergence of semi-sphere i.
Expression (4.3) can be simplified under the paraxial approximation: we only consider the part of the wave-front near the lens axis ($|\theta| \ll$). Therefore $\theta_i \approx \sin(\theta_i) = \frac{\sqrt{x^2+y^2}}{R_i}$. Then, using the second order approximation of $\cos(\theta) \approx 1 - \frac{\theta^2}{2}$, we can approximate:

$$\cos(\theta_i) = \sqrt{1 - \sin(\theta_i)^2} = \sqrt{1 - \frac{x^2+y^2}{R_i^2}} \approx 1 - \frac{x^2+y^2}{2R_i^2} \quad i = 1,2. \tag{4.4}$$

The thickness function becomes:

$$\Delta(x,y) = \Delta_0 - \frac{x^2+y^2}{2}\left(\frac{1}{R_1} - \frac{1}{R_2}\right). \tag{4.5}$$

Therefore, the transformation function of the lens becomes:

$$h(x,y) = e^{-\mathrm{i}kn\Delta_0} e^{\mathrm{i}k(n-1)\left(\frac{1}{R_1}-\frac{1}{R_2}\right)\left(\frac{x^2+y^2}{2}\right)} = e^{-\mathrm{i}kn\Delta_0} e^{\frac{\mathrm{i}k}{2f}(x^2+y^2)}, \quad \text{with } \frac{1}{f} = (n-1)\left(\frac{1}{R_1} - \frac{1}{R_2}\right), \tag{4.6}$$

where we recombine the physicals parameters of the lens into one, the focal length f.
We consider the field U_0 in front of the lens at a distance d_0 (Figure 4.2). Using the Fresnel approximation [22, Chapter 4] and the expression of the phase transformation applied by a lens (4.6), it can be shown that the field at the back focal plane is given by:

$$\begin{aligned} U_f(x,y) &= \frac{-1}{\mathrm{i}\lambda f} e^{-\mathrm{i}k(f+d_0+n\Delta_0)} e^{\frac{-\mathrm{i}k}{2f}\left(1-\frac{d_0}{f}\right)(x^2+y^2)} \iint_{-\infty}^{\infty} U_0(x',y') e^{\mathrm{i}\frac{2\pi}{\lambda f}(xx'+yy')}\,\mathrm{d}x'\mathrm{d}y' \\ &= -\frac{C}{\mathrm{i}\lambda f} e^{\frac{-\mathrm{i}k}{2f}\left(1-\frac{d_0}{f}\right)(x^2+y^2)} \mathcal{F}\{U_0\}\left(\frac{-x}{\lambda f}, \frac{-y}{\lambda f}\right), \end{aligned} \tag{4.7}$$

with C, a constant phase factor.
In the particular case where the input is placed in the front focal plane: $d_0 = f$, the quadratic phase factor disappears. Therefore, the wave-front located at the back focal plane is equal at a constant phase factor to the two-dimensional Fourier transform of the wave-front at the front focal plane. This setup is called the $2f-$system.

4.1.2 Design of an optical system

This ability of the converging lens ($f > 0$) to perform a two-dimensional Fourier transform can be combine with diffractive elements (mask) placed between the lens to filter frequency content [30, 40]. For example, if we consider two $2f-$systems without filter in between, we get on the output plane a reversed version of the input plane: $U_f(x, y) = \frac{C^2}{\lambda^2 f^2} U_0(-x, -y)$. By adding a filter in the Fourier plane, we can change the frequency content. A $4f-$system, illustrated on Figure 4.3, consists in a diffractive element placed in between two $2f$-systems containing the same lens.

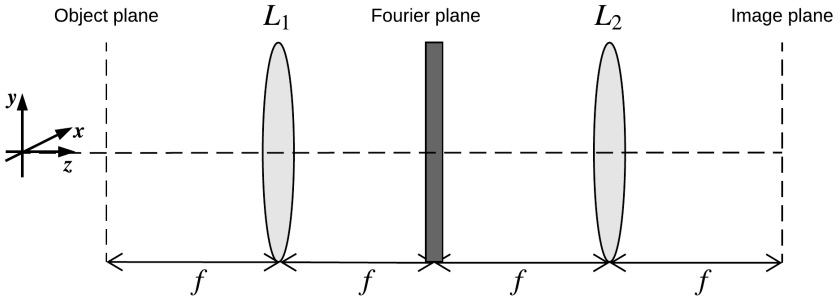


Figure 4.3: $4f$-system

The system performs first a 2D-Fourier transform, followed by a pointwise multiplication with the transmission function of the diffractive element and finally a 2D-Fourier transform. The mask can change the amplitude and/or the phase of the image.
Using the sampling theorem for band limited signal, we can work with discrete finite length version of the field and discrete optical systems of resolution $N \times N$. Therefore, considering the column stacked version of the input field $\boldsymbol{U}_0$, the $4f-$optical system can be represented as matrix $\boldsymbol{M} \in \mathbb{C}^{N^2 \times N^2}$. More precisely, the optical system can be modelled by a product of matrices which are the 2D-DFT matrix for the lens L_1 and L_2 and a diagonal matrix ($\boldsymbol{D} \in \mathbb{C}^{N^2 \times N^2}$) for the transmittance of the diffractive element:

$$\text{vec}(\boldsymbol{U}_f) = \boldsymbol{M}\text{vec}(\boldsymbol{U}_0) = (\boldsymbol{F_N} \otimes \boldsymbol{F_N})\boldsymbol{D}(\boldsymbol{F_N} \otimes \boldsymbol{F_N})\text{vec}(\boldsymbol{U}_0). \tag{4.8}$$

For what follows, we will denote $\boldsymbol{F} = (\boldsymbol{F_N} \otimes \boldsymbol{F_N})$.
Remarks, apply the discrete Fourier transform instead of the inverse discrete Fourier transform to a vector will produce a upside down result which in optics is usually ignored. That is why, for simplicity of notation we can avoid to always distinguish $\boldsymbol{F}$ and $\boldsymbol{F}^{-1}$. Therefore, neglecting the upside down effect of the lens, we can write: $\boldsymbol{M} = \boldsymbol{F}^{-1}\boldsymbol{D}\boldsymbol{F}$, which corresponds to a doubly-block-circulant matrix according to Theorem 1.2.

We are interested in the problem of designing a particular optical system $\boldsymbol{M}$ based on the concatenation of several $2f$-system. And typically, we are interested by the class of linear operator which requires few diffractive elements.
For example, we could want to design an optical system composed of only 3 diffractive elements. Such system is illustrated on Figure 4.4 and is called the $8f$-system.

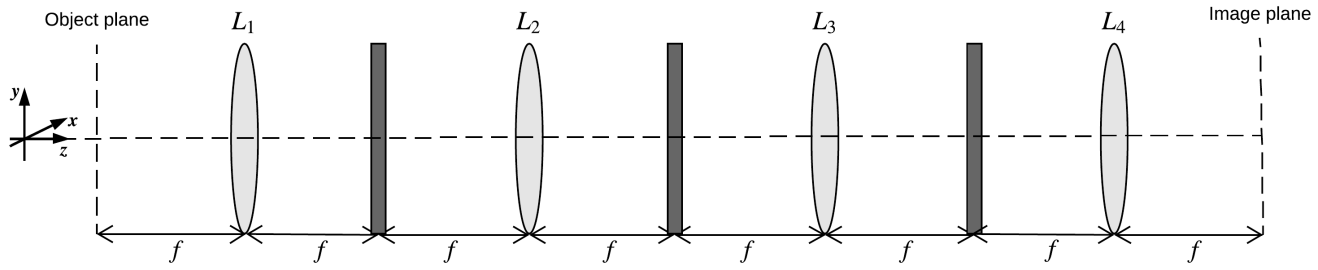


Figure 4.4: $8f$-system (composed of 3 filters and 4 lens).

The problem therefore consists in solving the following equation system:

$$\boldsymbol{F}\boldsymbol{D}_3\boldsymbol{F}\boldsymbol{D}_2\boldsymbol{F}\boldsymbol{D}_1\boldsymbol{F} - \boldsymbol{M} = \boldsymbol{0}, \tag{4.9}$$

where the variables are the elements of the diagonal matrices $\boldsymbol{D}_1, \boldsymbol{D}_2, \boldsymbol{D}_3$ which characterise the filters. We can show that this problem can be reformulate as computing the diagonal/doubly-block-circulant factorization of a matrix. Indeed, we have:

$$\begin{aligned} &\boldsymbol{F}\boldsymbol{D}_3\boldsymbol{F}\boldsymbol{D}_2\boldsymbol{F}\boldsymbol{D}_1\boldsymbol{F} = \boldsymbol{M}, \\ \Rightarrow &\boldsymbol{D}_3\boldsymbol{F}\boldsymbol{D}_2\boldsymbol{F}\boldsymbol{D}_1 = \boldsymbol{F}^{-1}\boldsymbol{M}\boldsymbol{F}^{-1} \triangleq \tilde{\boldsymbol{M}}, \\ \Rightarrow &\boldsymbol{D}_3\boldsymbol{B}\boldsymbol{D}_1 = \tilde{\boldsymbol{M}}, \end{aligned} \tag{4.10}$$

with $\boldsymbol{B} = \boldsymbol{F}\boldsymbol{D}_2\boldsymbol{F}$ a doubly-block-circulant matrix. Therefore, the problem to solve is equivalent to factorize the matrix $\tilde{\boldsymbol{M}}$ into a product of a diagonal matrix, a doubly-block-circulant matrix and a diagonal matrix.
This formulation of the optic design problem as a matrix factorization into diagonal/doubly-block-circulant matrices can be obtained for any optics system where the number of diffractive elements is a power of 2.

4.2 Finite difference schemes

The objective of this section is to show that factorization in diagonal/circulant matrices can appear in the process of solving differential equations when they are solved with finite difference schemes. In practice, the differential equations are solved numerically by discretizing the domain on a lattice and by approximating the differential equation by an iterative process. We can approximate the differential operators by finite difference schemes.
As we will see, the finite difference schemes applied on all the lattice at once can be reformulated as a matrix-vector product where the matrix is circulant.
We will highlight the appearance of the diagonal/circulant factorization for the particular case of the diffusion problem.

4.2.1 Isotropic diffusion

For example, we consider the two-dimensional diffusion equation:

$$\begin{cases} \dfrac{\partial I}{\partial t} = c\nabla^2 I, \\ I(x, y, 0) = I_0(x, y), \end{cases} \tag{4.11}$$

with $I_0(x, y)$ our initial condition and c the diffusion coefficient which is as a first step assumed constant. It can be shown [29] that the solution of (4.11) is given by:

$$\begin{cases} I(x,y,t) = \iint_{\mathbb{R}^2} H_t(x-x', y-y') I_0(x', y') \, \mathrm{d}x' \mathrm{d}y' = (H_t * I_0)(x, y), \\ H_t(x,y) = \dfrac{1}{2\pi\sigma_t^2} e^{\frac{-(x^2+y^2)}{2\sigma_t^2}}, \quad \text{with } \sigma_t = \sqrt{2c}\sqrt{t}. \end{cases} \tag{4.12}$$

So, for each time t, the solution results from a convolution between the initial condition function and an isotropic Gaussian filter whose width increases with t ($\sigma_t \sim \sqrt{t}$). Therefore, this diffusion process is a linear and space-invariant transformation of the initial function.

4.2.2 Anisotropic diffusion

The anisotropic diffusion [37] is a generalization of the isotropic diffusion, where the diffusion coefficient is now considered as a function of the position:

$$\begin{cases} I(x,y,0) = I_0(x,y), \\ \dfrac{\partial I}{\partial t} = \nabla \cdot (c(x,y,t) \nabla I) = c(x,y,t) \nabla^2 I + \nabla c \cdot \nabla I. \end{cases} \tag{4.13}$$

We assume here that we can neglect the second term, therefore, we have the following problem:

$$\begin{cases} I(x,y,0) = I_0(x,y), \\ \dfrac{\partial I}{\partial t} = c(x,y,t) \nabla^2 I. \end{cases} \tag{4.14}$$

Considering the discretized problem made of a lattice of $n \times n$ points equidistant, the Laplacian operator can be for example approximated by the following finite difference scheme:

$$(\overline{\nabla^2} I)^t_{i,j} = I^t_{i-1,j} + I^t_{i+1,j} + I^t_{i,j-1} + I^t_{i,j+1} - 4I^t_{i,j}, \tag{4.15}$$

where $\overline{\nabla^2} I$ denotes the approximation of the Laplacian.
Therefore, the approximated Laplacian of the entire function can be computed by the 2D circular convolution between the function and the following kernel centred in (0,0):

$$\boldsymbol{K} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

In order to write the circular convolution in the form of (1.18), we have to pad the kernel with 0 in order to reach the size of the lattice and then to apply a fftshift.

$$\begin{aligned} h &= \text{fftshift}(\text{pad}(\boldsymbol{K})), \\ \overline{\nabla^2} I &= h \circledast I. \end{aligned} \tag{4.16}$$

As we have shown in (1.19), the 2D circular convolution can be expressed as a matrix-vector product by considering the column stacked version of the signals. Therefore we obtain:

$$\text{vec}(\overline{\nabla^2} I) = \boldsymbol{B} \text{vec}(I), \tag{4.17}$$

where $\boldsymbol{B}$ is the doubly-block circulant matrix generated by h: $\boldsymbol{B}(h)$.
To solve (4.14), we will use the Euler integration scheme:

$$\begin{cases} I^{t+1}_{i,j} = I^t_{i,j} + \Delta t \left(\dfrac{\overline{\partial I}}{\partial t} \right)^t_{i,j} = I^t_{i,j} + \Delta t c^t_{i,j} (\overline{\nabla^2} I)^t_{i,j}, \\ \text{vec}(I^{t+1}) = (\boldsymbol{I} + \boldsymbol{D}^t \boldsymbol{B}) \text{vec}(I^t), \end{cases} \tag{4.18}$$

with $\boldsymbol{I}$ the identity matrix ($\in \mathbb{R}^{n^2 \times n^2}$) and $\boldsymbol{D}^t$ a diagonal matrix where $D_{k,k} = \Delta t c_{i,j}^t$ with $k = jn + i$. So, the solution at time t can be obtained as:

$$\text{vec}(I^t) = \left(\prod_{i=1}^{t} (\boldsymbol{I} + \boldsymbol{D}^i \boldsymbol{B}) \right) \text{vec}(I_0). \tag{4.19}$$

We got a variant of our diagonal/doubly-block-circulant matrices product.
But, if we assume that $D_{k,k} \gg 1$, the identity matrix becomes negligible and therefore we obtain our particular factorization:

$$\text{vec}(I^t) = \left(\prod_{i=1}^{t} (\boldsymbol{D}^i \boldsymbol{B}) \right) \text{vec}(I_0). \tag{4.20}$$

If we also assume that $c(x, y, t)$ is not function of I, we could try to learn with the diagonal/doubly-block-circulant neural network this particular anisotropic diffusion process based on the result at time t of a set of different initial condition functions.

Chapter 5

Numerical results

The purpose of this chapter is to show that some interesting non-shift-invariant linear operators can be approached by our factorization with only a few factors.
First, the section 5.1 contains descriptions of the datasets that will be used in the experiments. Next, in section 5.2, we will look at integral operators which have an exact factorization in diagonal and circulant matrices with few factors. In particular, we will focus on a non-shift-invariant operator behaving similarly to a convolution at the periphery of the domain but exhibiting a damping effect as we approach the center of the domain. As we will see in Chapter 6, non-shift-invariant operators with this behaviour are of interest because they appear in astronomy in the context of direct imaging of exoplanets.
Finally, in section 5.3, we are going to look at warped convolution, which is an additional example of non shift-invariant operator.

5.1 Datasets

We will use two different datasets: the first composed of perfectly localized inputs in space and the second composed of perfectly localized inputs in frequency.

Dataset $\mathcal{D}_1$

Let $\mathcal{D}_1 = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \ | \boldsymbol{x}_i \in \mathbb{R}^N, \boldsymbol{y}_i \in \mathbb{R}^N\}_{i=1}^N$ where the inputs are some Dirac deltas :

$$[\boldsymbol{x}_i]_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad , \quad j = 1, ..., N. \tag{5.1}$$

And where the targets $\boldsymbol{y}_i$ are the result of the discrete integral operator applied on $\boldsymbol{x}_i$. We have $\boldsymbol{y}_i = \boldsymbol{M}\boldsymbol{x}_i$, where $\boldsymbol{M} \in \mathbb{R}^{N \times N}$ is the matrix associated to the integral operator. Therefore, each target $\boldsymbol{y}_i$ is a column of the integral operator matrix $\boldsymbol{M}$: $\boldsymbol{y}_i = \boldsymbol{M}_{:,i}$ for $i = 1, ..., N$.
The problem of learning an integral operator where the dataset at our disposal is generated by inputs that are Dirac delta may seem completely useless since we already know the operator through the dataset.
But we will show that by using our well designed diagonal/circulant structure for the neural network, we are able to learn the whole operator (each columns of $\boldsymbol{M}$) with only few data (few columns of $\boldsymbol{M}$).
Moreover, the application that we will discuss more in detail in Chapter 6 (exoplanet detection) is a particular application where the inputs at our disposal for the dataset are effectively Dirac delta function (corresponding to a star or exoplanets).

Dataset $\mathcal{D}_2$

Let $\mathcal{D}_2 = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \ | \boldsymbol{x}_i \in \mathbb{R}^N, \boldsymbol{y}_i \in \mathbb{R}^N\}_{i=1}^N$ where the inputs are perfectly localized in frequency:

$$[\boldsymbol{x}_i]_j = \cos(2\pi(i\Delta f)X_j) \quad , \quad j = 1, ..., N, \tag{5.2}$$

with $\boldsymbol{X} \in \mathbb{R}^N$ the domain considered and Δf, the frequency discretization step.
The target values $\boldsymbol{y}_i = \boldsymbol{M}\boldsymbol{x}_i$ where $\boldsymbol{M} \in \mathbb{R}^{N \times N}$ is the matrix associated to the integral operator.

5.2 Factorizable integral operators with few factors

We are going to test the neural network on integral operators of the form:

$$H[f](u) = \int K(u, x)f(x) \ \mathrm{d}x = (D_{\rho_2} \circ C_\sigma \circ D_{\rho_1} \circ C_\sigma)[f](u), \tag{5.3}$$

where C_σ and D_ρ are functional operators defined as:

$$\begin{cases} C_\sigma[f](u) = \int k_\sigma(u - x)f(x) \ \mathrm{d}x, \quad \text{with } k_\sigma(u) = \kappa_\sigma e^{\frac{-u^2}{2\sigma^2}}, \ \kappa_\sigma = \dfrac{1}{\sqrt{2\pi}\sigma}, \\ D_\rho[f](u) = \rho(u)f(u). \end{cases} \tag{5.4}$$

Therefore, the matrix $\boldsymbol{M}$ corresponding to the discretized version of this integral operator can be expressed as: $\boldsymbol{M} = \boldsymbol{D}_2\boldsymbol{C}\boldsymbol{D}_1\boldsymbol{C}$, where $\boldsymbol{D}_1$, $\boldsymbol{D}_2$ are diagonal matrices and $\boldsymbol{C}$ is a circulant matrix. We are going to consider two integral operators having a factorization of 4 factors.

"Exponential" integral operator

It corresponds to the factorizable integral operator (5.3) with $\rho_1(v) = 1 - ce^{\frac{-av^2}{2\sigma^2}}$.
The closed-form expression of the associated kernel function is given by:

$$K(u, x) = \rho_2(u)k_{\sqrt{2}\sigma}(u - x)\left[1 - c\sqrt{\frac{2}{2+a}}e^{\frac{-a(u+x)^2}{2(2+a)\sigma^2}}\right]. \tag{5.5}$$

This integral operator is non shift-invariant. The kernel is composed of two terms, ones being a Gaussian function and the other being a function of $(u + x)$. It will be shift-invariant in the case where $\rho_2 \equiv 1$ and $\rho_1 \equiv 1$ $(c = 0)$. Indeed, in that case the kernel and the associated integral operator are reduced to:

$$\begin{cases} K(u, x) = k_{\sqrt{2}\sigma}(u - x) = \kappa_{\sqrt{2}\sigma}e^{\frac{-(u-x)^2}{2(\sqrt{2}\sigma)^2}}, \\ H[f](u) = (C_\sigma \circ C_\sigma)[f](u) = (C_{\sqrt{2}\sigma})[f](u), \end{cases} \tag{5.6}$$

which corresponds to a simple Gaussian convolution operator.
The interest of the operator (5.5) is to obtain an operator relatively shift-invariant everywhere except in the center.
For a large value of $|u|$, the exponential term decreases to 0, and the kernel becomes similar to a Gaussian kernel. Therefore, the integral operator behaves as a convolution for points at the domain boundary. But for a small value of $|u|$ (i.e. for points at the center of the domain), the exponential term will perturb and dampen the signal. The parameter a controls the diameter of the region in which the integral operator's behaviour is non shift-invariant.
We have therefore found a non shift-invariant operator with the desired properties that can be factorized with only 4 factors. The motivation of this operator will make sense when we will see

the applications in the field of direct imaging of exoplanets in Chapter 6.

"Polynomial" integral operator

It corresponds to the factorizable integral operator (5.3) with $\rho_1(v) = \sum_{i=0}^{n} a_i v^i$.
In the case where ρ_1 is a polynomial of degree n, the closed-form expression of the kernel function is given by:

$$K(u,x) = \rho_2(u) k_{\sqrt{2}\sigma}(u-x) \sum_{i=0}^{n} a_i \left[\sum_{k=0}^{i} \binom{i}{k} \left(\frac{\sigma}{\sqrt{2}}\right)^k l_k \left(\frac{u+x}{2}\right)^{i-k} \right], \tag{5.7}$$

$$\text{with } l_k = \begin{cases} 0 & \text{if } k \text{ is odd,} \\ (k-1)!! & \text{if } k \text{ is even.} \end{cases} \tag{5.8}$$

Again, the kernel is composed of two parts, one being a Gaussian shift-invariant kernel and the other being a polynomial in $(u+x)$.
Therefore, the integral operator is non shift-invariant except in the case where $\rho_2 \equiv 1$ and $n = 0$. This shows that non-trivial non shift-invariant integral operators such as kernel (5.7) can be factorized with very few factors.

The proofs of the factorization of the integral operators associated to the kernels (5.5) and (5.7) are in Appendix A.5.

5.2.1 Results

In this section, we will show by means of two numerical examples that the diagonal/circulant neural network is able to learn a factorizable integral operator with very little training data. In particular, we will evaluate the performance on non shift-invariant integral operators whose kernel is given by (5.5) and (5.7).

"Exponential" integral operator

The target operator is the one that has its kernel given by (5.5) with $\rho_1(v) = 1 - ce^{\frac{-av^2}{2\sigma^2}}$ and $\rho_2(v) = 1$ and with the parameters given in Table 5.1.

Parameters	Values
$\boldsymbol{X}$	$[-100, 100]$
N	500
a	0.05
c	1
σ	4

Table 5.1: Model parameters for the "exponential" operator.

Layers	Initialization of $\boldsymbol{w}$
$\boldsymbol{C}_1(\boldsymbol{w})$	$k_\sigma(u)$ with $\sigma = 1$
$\boldsymbol{D}(\boldsymbol{w})$	$t_\gamma(u)$ with $\gamma = 1$
$\boldsymbol{C}_2(\boldsymbol{w})$	$k_\sigma(u)$ with $\sigma = 1$

Table 5.2: Network layers initialization. Initialization is done by evaluating the functions on the domain $\boldsymbol{X}$.

The domain $\boldsymbol{X}$ is discretized in $N = 500$ equidistant points.
The neural network structure used is identical to the structure of the target operator: a factorization $\boldsymbol{CDC}$ where the layers are initialized according to Table 5.2. The training set is composed of 70% of the dataset. The hyperparameters of the optimization algorithm for the learning are given in section A.6. The stopping criterion used for the learning is early stopping presented in section 1.2.5 with a window size of 10. The datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ are defined respectively

according to (5.1) and (5.2) (with $\Delta f = 2 \cdot 10^{-4}$).

On Figure 5.1, we see that for the two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, the learned intermediate factors are not equivalent to the real factors of the target operator. Their shape is similar, but they differs by their scale. Indeed, there is an indeterminacy on the absolute intensities of each component of the factorization (more details are given in 5.2.2). But the combination of all components is as desired, equivalent to the complete target operator.

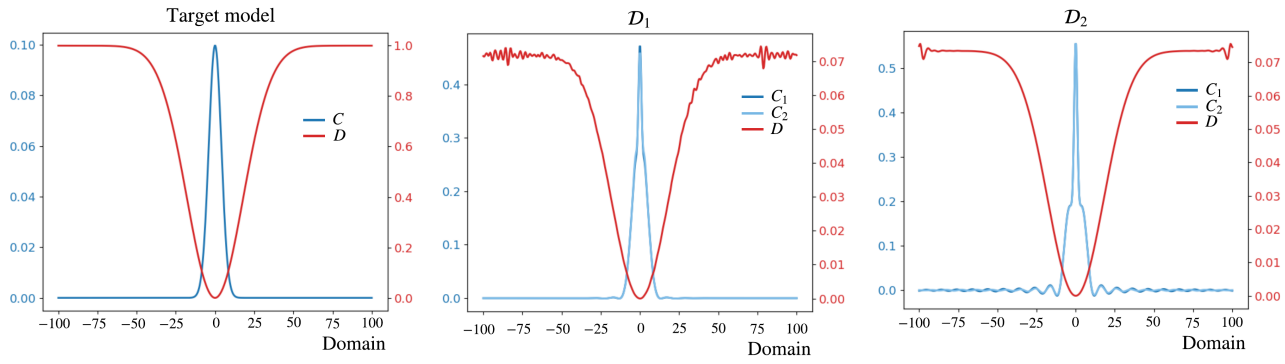


Figure 5.1: The figure on the left shows the weights $\boldsymbol{w} \in \mathbb{R}^N$ that generated the circulant matrix $\boldsymbol{C} = \boldsymbol{C}(\boldsymbol{w})$ and the diagonal matrix $\boldsymbol{D} = \boldsymbol{D}(\boldsymbol{w})$ of the target model. The middle figure shows the weights of the neural network resulting from the training phase on the dataset $\mathcal{D}_1$. The figure on the right gives the weights of the neural network resulting from the training phase on the dataset $\mathcal{D}_2$.

In order to analyze the ability of the diagonal/circulant neural network to generalize according to the size of the training set, we can draw the learning curves. Figure 5.3 gives the training and validation error at the end of the network training for a training set of increasing size (see Figure 5.2 for the principle). It can be seen that very little training data is required to learn the integral operator. The amount of training data needed is even lower for the dataset $\mathcal{D}_2$ where the inputs are frequency-localized. In addition, the gap between the training and validation error is smaller for the use of frequency-localized inputs for the training, indicating a better generalization ability in this case.

Figure 5.2: To build the learning curves: we build a validation set at the beginning which will remain the same, then, we gradually train the neural network on a training set whose size increases progressively.

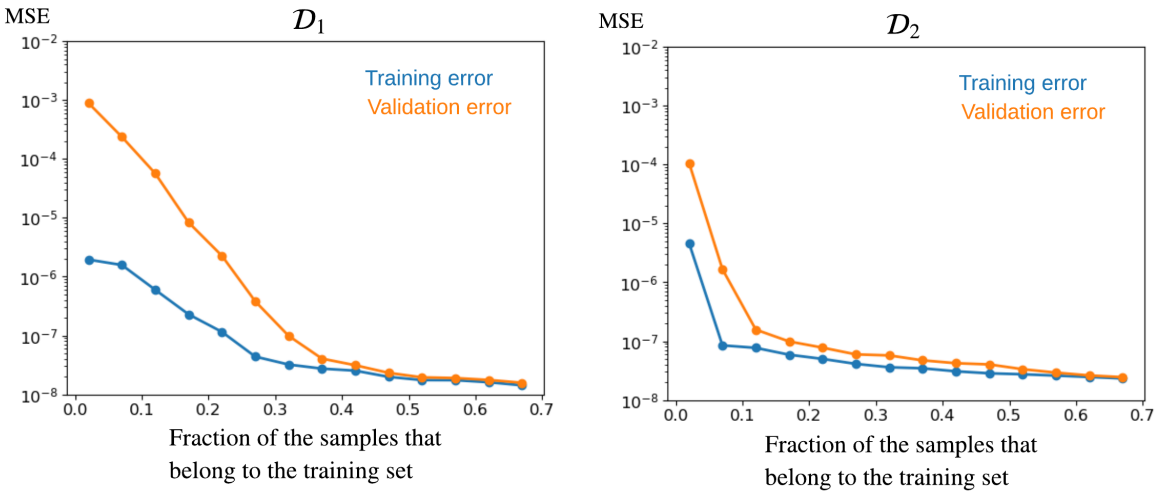


Figure 5.3: The y-axis is the mean-squared-error. On the left are the learning curves for the network trained on the $\mathcal{D}_1$ dataset. On the right, the learning curves for the network trained on the data set $\mathcal{D}_2$.

"Polynomial" integral operator

The target operator is the one that has its kernel given by (5.7) with $\rho_1(v) = \sum_{i=0}^{n} a_i v^i$ and $\rho_2(v) = 1$ and with the parameters given in Table 5.3.

Parameters	Values
$\boldsymbol{X}$	$[-50, 50]$
N	500
n	3
$[a_0, a_1, a_2, a_3]$	$10^{-3} \cdot [-300, -4, 1, 0.015]$
σ	4

Table 5.3: Model parameters for the "polynomial" operator.

Layers	Initialization of $\boldsymbol{w}$
$\boldsymbol{C}_1(\boldsymbol{w})$	$k_\sigma(u)$ with $\sigma = 1$
$\boldsymbol{D}(\boldsymbol{w})$	$t_\gamma(u)$ with $\gamma = 1$
$\boldsymbol{C}_2(\boldsymbol{w})$	$k_\sigma(u)$ with $\sigma = 1$

Table 5.4: Network layers initialization. Initialization is done by evaluating the functions on the domain $\boldsymbol{X}$.

The interval of the domain $\boldsymbol{X}$ is discretized in points $N = 500$ equidistant points.
The neural network structure used is identical to the structure of the target operator: a factorization $\boldsymbol{CDC}$ where the layers are initialized according to Table 5.4. The training set is composed of 70% of the dataset. The hyperparameters of the optimization algorithm for the learning are given in section A.6. The stopping criterion used for the learning is early stopping presented in section 1.2.5 with a window size of 10. The datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ are defined respectively according to (5.1) and (5.2) (with $\Delta f = 2 \cdot 10^{-4}$).

On the Figure 5.4, we can do the same remarks than for the "exponential" integral operator. We see that for the two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, the learned intermediate factors are not equivalent to the real factors of the target operator. But the combination of all components is as desired, equivalent to the complete target operator.

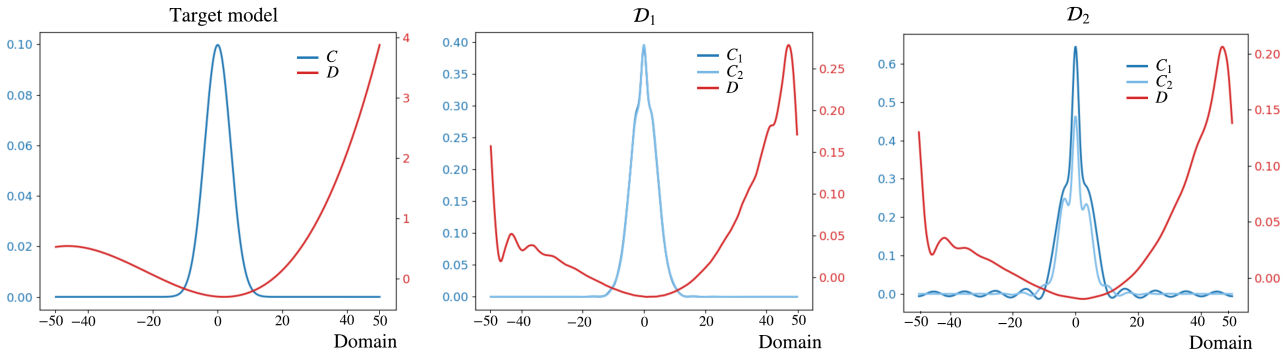


Figure 5.4: The figure on the left shows the weights $\boldsymbol{w} \in \mathbb{R}^N$ that generated the circulant matrix $\boldsymbol{C} = \boldsymbol{C}(\boldsymbol{w})$ and the diagonal matrix $\boldsymbol{D} = \boldsymbol{D}(\boldsymbol{w})$ of the target model. The middle figure shows the weights of the neural network resulting from the training phase on the dataset $\mathcal{D}_1$. The figure on the right gives the weights of the neural network resulting from the training phase on the dataset $\mathcal{D}_2$.

Using the Figure 5.5, it can be seen that very little training data is required to learn the operator. The amount of training data needed is even lower for the dataset $\mathcal{D}_2$ where the inputs are frequency-localized. In addition, the gap between the training and validation error is smaller for the use of frequency-localized inputs for the training, indicating a better generalization ability in this case.

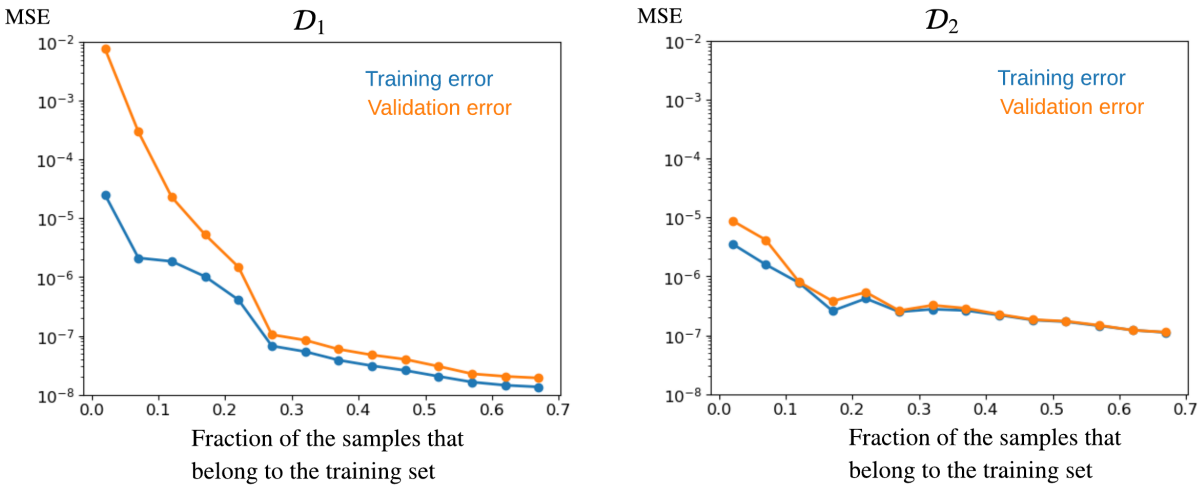


Figure 5.5: On the left are the learning curves for the network trained on the $\mathcal{D}_1$ dataset. On the right, the learning curves for the network trained on the dataset $\mathcal{D}_2$.

5.2.2 Conclusion

Let $\boldsymbol{M}$ be the discrete matrix associated with the integral operator which is factorizable with few factors $\boldsymbol{M} = \boldsymbol{C}_n\boldsymbol{D}_n...\boldsymbol{C}_1\boldsymbol{D}_1$. Through the previous observations, we highlight two facts:

- The diagonal/circulant neural network having the same structure as the target operator is not able to learn the exact factorization but it is able to learn the complete integral operator (the matrix $\boldsymbol{M}$). Indeed, several identical factorizations can lead to the same integral operator.
  For example, if we consider the following operator: $\boldsymbol{M} = \boldsymbol{C}\boldsymbol{D}$. Then, we can obtain the same operator with a different factorization: $\boldsymbol{M} = \boldsymbol{C}'\boldsymbol{D}'$ with $\boldsymbol{C}' = k\boldsymbol{C}$ and $\boldsymbol{D}' = \frac{1}{k}\boldsymbol{D} \quad \forall k \in \mathbb{R}_0$.

  Therefore, without additional regularization, the network is not guaranteed to learn the original factorization. But this is not a problem because our objective is to approximate the integral operator and not its factorization.

- The network diagonal/circulant is able to learn the integral operator with few data. Through the two numerical examples studied, the learning of the integral operator by means of frequently localized inputs seems to require less training data than the use of perfectly spatially localized data.

5.3 Warped convolution

In this section, we will test the diagonal/circulant neural network on an operator that we will name warped convolution.

5.3.1 Operator description

There are imaging applications (e.g., in astronomy or lensless endoscopy) where we observe an image "convolved" with a kernel whose shape depends on its localization. In the continuous setting and in 1-D, the action of this spatially-variant blurring can be modeled as follows.
Let a function $w : \mathbb{R}^+ \to \mathbb{R}^+$ be such that $w(0) = 0$, $w(x)$ is increasing ($w'(x) \geq 0$), and $\lim_{x\to+\infty} w(x) = +\infty$. Moreover, we assume that w is differentiable and invertible over $\mathbb{R}^+$, that is, there exists a w^{i} such that $[w^{\mathrm{i}} \circ w](x) = x$ for $x \geq 0$.

We define the following functions:

$$\begin{cases} w_{\mathrm{s}} : x \in \mathbb{R} \mapsto w_{\mathrm{s}}(x) \ = \mathrm{sign}(x)w(|x|), \\ w_{\mathrm{s}}^{\mathrm{i}} : x \in \mathbb{R} \mapsto w_{\mathrm{s}}^{\mathrm{i}}(x) = \mathrm{sign}(x)w^{\mathrm{i}}(|x|), \end{cases} \tag{5.9}$$

with w_s^{i} the inverse function of w_s: $[w_s^{\mathrm{i}} \circ w_s](x) = x \ \forall x$.
From this function we define a warping operator:

$$W : f(x) \mapsto w'(|x|)f\Big(w_s(x)\Big). \tag{5.10}$$

This operator "warp" a function f. The simplest case amounts to taking $w(x) = ax$ (for some $a > 0$) for which $Wf(x) = af(ax)$, which is a contraction with factor a. Moreover, we can define the inverse warping:

$$W^{\mathrm{i}} : f(x) \mapsto \tfrac{1}{w'(w^{\mathrm{i}}(|x|))} f\Big(w_s^{\mathrm{i}}(x)\Big). \tag{5.11}$$

Therefore, we have the following relation:

$$(W^{\mathrm{i}} \circ W)[f](x) = f(x). \tag{5.12}$$

We can now define the convolution warped by w as:

$$G_w[f](x) = (W^{\mathrm{i}} \circ G \circ W)[f](x), \tag{5.13}$$

with $G[f](x) = \int g(y-x)f(y)\,\mathrm{d}y = g * f(x)$ the convolution of f by g (e.g., a Gaussian kernel). The so-called warped convolution thus consists in first warping the space (and thus the function f) with W, then in convolving with a kernel g, and then in unwarping with W^{i}. We can show that this warped convolution can be rewritten as an integral operator where the shape of the kernel depends on its localization. Indeed, by writing explicitly G_w, we have:

$$G_w[f](x) = \frac{1}{w'(w^{\mathrm{i}}(|x|))} \int g(y - w_s^{\mathrm{i}}(x)))f(w_s(y))w'(|y|)\ \mathrm{d}y, \tag{5.14}$$

and then by applying the following change of variable $\tilde{y} = w_s(y)$, we obtain:

$$G_w[f](x) = \frac{1}{w'(w^{\mathrm{i}}(|x|))} \int g(w_s^{\mathrm{i}}(\tilde{y}) - w_s^{\mathrm{i}}(x))f(\tilde{y})\ \mathrm{d}\tilde{y}, \tag{5.15}$$

which effectively corresponds to a non shift-invariant integral operator.
For instance, taking $f(x) = \delta(x - x_0)$ for some x_0 (for δ a Dirac) gives:

$$G_w[f](x) = \tfrac{1}{w'(w^{\mathrm{i}}(|x|))} g(w_s^{\mathrm{i}}(x_0) - w_s^{\mathrm{i}}(x)). \tag{5.16}$$

In other words, within a normalization factor, $G_w[f](x)$ is equivalent to a convolution with the Gaussian kernel $g(x_0' - x')$ with $u' := w_s^{\mathrm{i}}(u)\ \forall u$, after having warped the space with w_s^{i}. The impulse response is simply a matter of having "warped" the space (with w_s^i) before defining the filter in it.

5.3.2 Results

The target operator is the one given by (5.13) or equivalently by (5.15) with the particular choice:

$$\begin{cases} w(x) = |x|^{\alpha}, \\ w^{\mathrm{i}}(x) = |x|^{1/\alpha}, \end{cases} \tag{5.17}$$

for some $\alpha > 0$. In this case, for $\alpha > 1$, W corresponds to dilate the argument of the function on which it applies, which corresponds to contract this function.
The parameters are given in Table 5.5.

Parameters	Values
$\boldsymbol{X}$	$[-10, 10]$
N	300
α	1.5
σ	0.5

Table 5.5: Model parameters for the warped operator.

Layers	Initialization of $\boldsymbol{w}$
$\boldsymbol{C}_1(\boldsymbol{w})$	$k_\sigma(u)$ with $\sigma = 2$
$\boldsymbol{D}_1(\boldsymbol{w})$	$t_\gamma(u)$ with $\gamma = 0.1$
$\boldsymbol{C}_2(\boldsymbol{w})$	$k_\sigma(u)$ with $\sigma = 0.5$
$\boldsymbol{D}_2(\boldsymbol{w})$	$t_\gamma(u)$ with $\gamma = 0.1$
$\boldsymbol{C}_3(\boldsymbol{w})$	$0.5k_\sigma(u)$ with $\sigma = 0.5$
$\boldsymbol{D}_3(\boldsymbol{w})$	$t_\gamma(u)$ with $\gamma = 0.5$

Table 5.6: Network layers initialization. Initialization is done by evaluating the functions on the domain $\boldsymbol{X}$.

The interval of the domain $\boldsymbol{X}$ is discretized $N = 300$ equidistant points.

Figure 5.6 shows the result of the warped operator for several spatial Dirac delta given in input. Its behaviour is very different from that of a Gaussian convolution. We observe that near the center, the Gaussian shape ceases to be symmetrical because the influence of the deformation depends on the localization. As we move away from the center, we get a Gaussian that spreads.

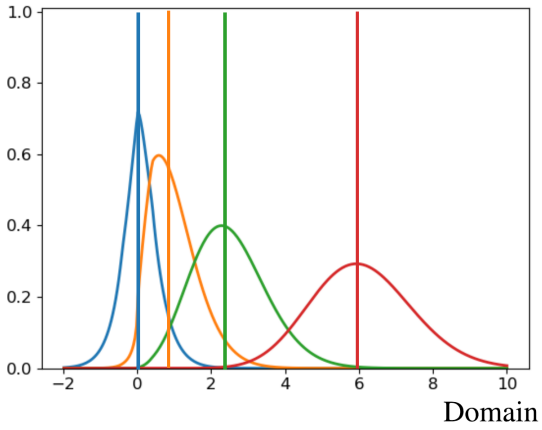


Figure 5.6: Result of the warped operator (5.13) with parameters given in Table 5.5 for several spatial Dirac delta given in input. The straight line corresponds to the input, and the curve of the same color to the associated result.

The training set is composed of 80% of the dataset. The hyperparameters of the optimization algorithm for the learning are given in section A.6. The stopping criterion used for the learning is the early stopping presented in section 1.2.5 with a window size of 50. The datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ are defined respectively according to (5.1) and (5.2) (with $\Delta f = 8 \cdot 10^{-4}$).

On Figure 5.7, we note that the increase in the number of layers in the network makes it possible to model with an higher precision the non shift-invariant behaviour of this warped-convolution operator. As expected, the worst result is obtained for a network consisting of a single circulant layer because, as has been shown, the warped-convolution operator is different from a convolution.

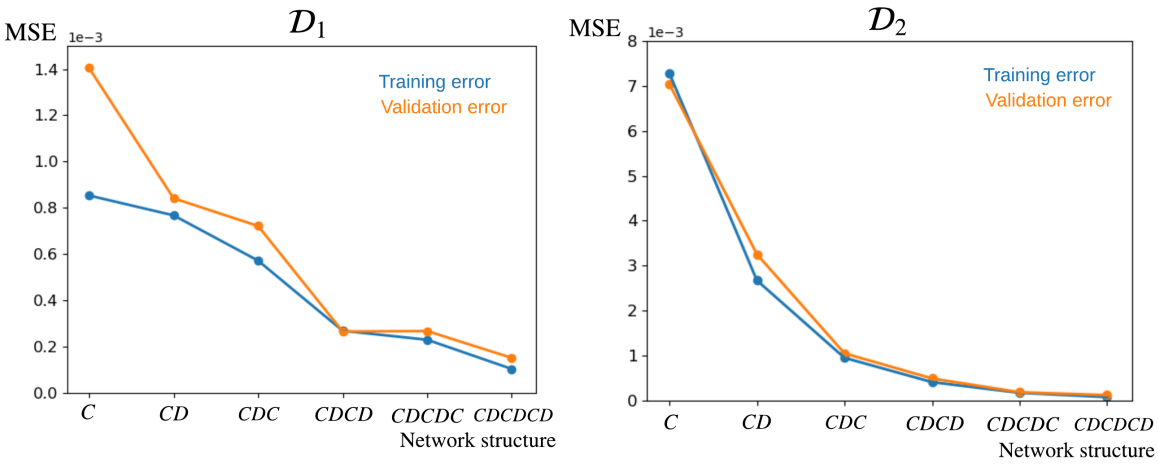


Figure 5.7: Mean squared error on the training and validation set at the end of the training. Left figure with training set $\mathcal{D}_1$ and right figure with training set $\mathcal{D}_2$. The structure of the network is given on the x-axis. The layers are applied on the input from left to right.

The learning curves for the network having a structure of 4 factors $\boldsymbol{CDCD}$ are given on Figure 5.8. It can be seen that: adding additional data to the training set quickly becomes useless. And this is even more marked for the dataset $\mathcal{D}_2$ containing perfectly localized entries.

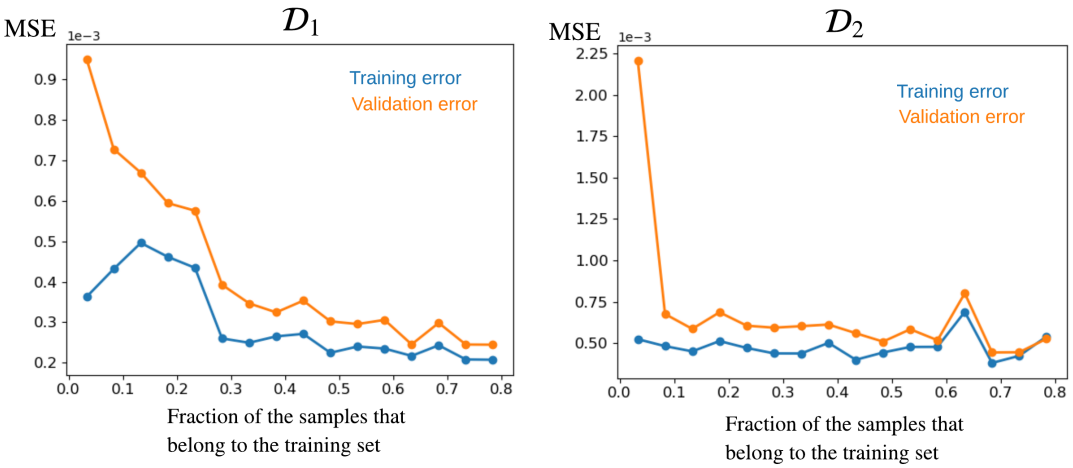


Figure 5.8: On the left are the learning curves for the network trained on the $\mathcal{D}_1$ dataset. On the right, the learning curves for the network trained on the dataset $\mathcal{D}_2$. The network has the following structure: $\boldsymbol{CDCD}$.

Chapter 6

Application in astronomy

As we will see in this chapter, the detection of exoplanets by direct imaging presents several difficulties that require the resolution of an inverse problem in post-processing. Several models have been proposed to describe the link between the true space signal and the image actually observed on the screen of the ground-based telescope (PCA [3], LLSG [20], ANDROMEDA [11]). But none of these techniques took into account the effects of the telescope on the observation. Recently, [6] proposed to integrate the effect of the telescope by modeling the telescope with a shift-invariant operator. However, we know that for small angular separations, the telescope is non shift-invariant. We will see that the diagonal/doubly-block-circulant neural network can be used in a first step to learn the non shift-invariant behaviour of the telescope and in a second step, it can be integrated in the inverse resolution process to increase the detectability of exoplanets having a small angular separation with their host star.

6.1 Context

Most exoplanets are detected by the radial velocity method and the transit method [41]. These two methods infer the presence of an exoplanet indirectly by observing their effect on the star light. Another technique: direct imaging attempts to directly observe the light reflected by the planet. The two main challenges of this technique are:

- The contrast difference between the light of the host star and the planet is very high (in the order of 10^3 to 10^{10}).
  In order to reduce the contrast between the star and the planet, the telescope are equipped with a coronagraph [10, Chapter 1] that block the light on the axis (of the star) while allowing off-axis light (of potential compagnions) to pass through. The behaviour of a Lyot-stop coronograph is illustrated on Figure 6.1. An opaque mask placed in the focal plane of lens 1 stops most of the starlight. The mask is designed to redirect starlight that is not blocked to the edges of the beam. It then suppresses the effect of diffraction caused by the edges of the mask by blocking the light at the edge of the beam with the Lyot stop. When the light is finally focus by the lens 3 on the screen, most of the starlight has been suppressed. We are now able to see objects that are more than a million times fainter than the star. As the telescope points directly at the star, the light from the planet forms an angle and misses the mask.

- The observed system is so far away that the angular distance between the star and its expolanets is very small.
  The resolution power of a telescope is limited by the diffraction of the light which introduced a blurring. The smallest angular difference which can be resolve by a system is characterized by the following ratio: $\frac{\lambda}{D}$, where λ denote the wavelength and D the diameter

of the lens. Therefore, for a given wavelength, the resolution power of a telescope can be improved by increasing the diameter of the lens. Space telescope limited by their size can not reach a sufficient resolution. To overcome these problems, we need large ground-based telescope. But such telescopes suffer of the atmospheric perturbations. The light coming from a distant star can be considered as a plane wave. However, the wavefront is distorted because it will pass through regions of the Earth's atmosphere with different temperatures and pressures (and therefore a different refractive index). The final image will contains additionally to the star light some synthetic light patterns called the speckles field [18]. To address this problem, the telescope is equipped with an adaptive optics system [5] which consists in a deformable mirror that can correct in real time the wavefront.

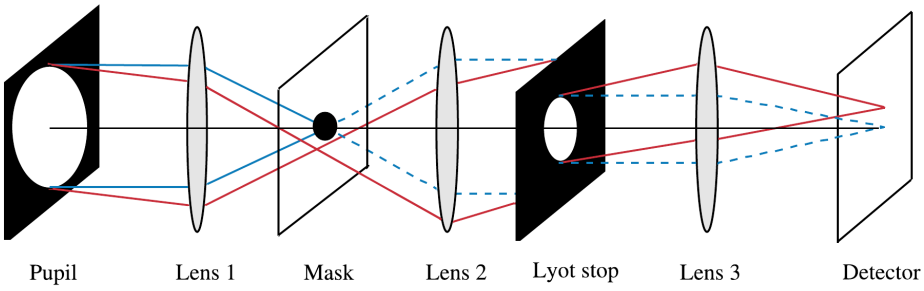


Figure 6.1: Lyot coronograph, with in blue the starlight (on-axis), in red the planet's light.

But even with the adaptive optic system, some quasi-static speckles will appear due to residual static aberrations.
To distinguish the signal of a potential exoplanet from the speckle field, additional information is needed. For example, temporal information can be used. The pupil of the telescope is stabilized during the night so that the aberrations (speckles) remain fixed. However, by the rotation of the Earth, the field of view rotates and all the planets with it. We can then exploit this difference in temporal behaviour between the planets and the speckles. The technique of observation to obtain this data is called: Angular Differential Imaging (ADI) [28].

Therefore, in order to separate the planetary signals from the star light and the speckle field, it is necessary to use post-processing techniques based on the resolution of an inverse problem to correct the effects of the telescope on the observed image. By opposition to the forward model which consists in determining the effects from knowing the causes by modelling the link between the observation and the real images, the inverse problem [15] consists in determining the causes from knowing the effects.
We can therefore see that the inverse problems are likely to pose difficulties. Indeed, it is easy to imagine that the same effects could come from different causes. The presence of several solutions is one of the main difficulties for solving inverse problems. Therefore, it is necessary to have additional knowledge to discriminate among them.
Moreover, in physical processes, the measurement is often disturbed by measurement noise, it is therefore necessary to dispose of the statistical distribution of the noise in order to improve the quality of the solution.

6.2 Forward model

Let T be the number of frames of size $N \times N$ resulting from an ADI observation sequence and $\boldsymbol{M} \in \mathbb{R}^{N^2 \times T}$ be a matrix whose each column corresponds to a vectorized frame. And each row corresponds to the evolution of a specific pixel over time. Gonzalez et al [20] proposed a local three terms decomposition (known as the LLSG algorithm): $\boldsymbol{M} = \boldsymbol{L} + \boldsymbol{S} + \boldsymbol{G}$ with $\boldsymbol{L}$, a low rank matrix, $\boldsymbol{S}$ a sparse matrix and $\boldsymbol{G}$ a noise matrix. Since the starlight and the quasi-static speckle field do not change much over time, they are among the few main components of the $\boldsymbol{M}$ matrix. This is why the low-rank term includes most of the starlight and the speckle field. Since planets are small moving objects, they will not appear in the low rank term but in the sparse term. This decomposition is illustrated in Figure 6.2.

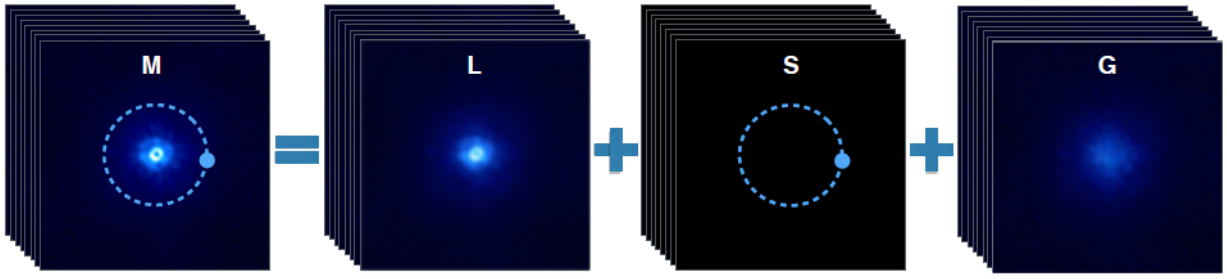


Figure 6.2: LLSG decomposition: $\boldsymbol{L}$ is a low rank term, $\boldsymbol{S}$ is a sparse term and $\boldsymbol{G}$ is a noise term (Credit [20]).

We will use the following forward model proposed in [6]:

$$\boldsymbol{M} = \boldsymbol{L} + \phi * \boldsymbol{S} + \boldsymbol{G}, \tag{6.1}$$

with ϕ, the PSF of the telescope and * denotes the two-dimensional convolution applied separately on each frame of $\boldsymbol{S}$.
As LLSG, it uses a 3 terms decomposition: the low rank matrix $\boldsymbol{L}$ to absorb most of the star and speckles signal, the sparse matrix $\boldsymbol{S}$ which will contain the planetary signal and an additive noise $\boldsymbol{G}$.
But there is two main differences with LLSG:

- It takes into account the convolution resulting of the instruments.
- It imposes a stronger constraint on the structure of the planetary sequence of frames to be recovered: in addition to the constraint of sparsity on $\boldsymbol{S}$, it also imposes a structure of rotation between the different images modelling the apparent rotation of the planets around the star:

  $$\boldsymbol{S} = \mathcal{R}(\boldsymbol{x}\mathbf{1}^T), \tag{6.2}$$

  where $\boldsymbol{x} \in \mathbb{R}^{N^2}$ corresponds to the first frame containing the planetary signal, and $\mathbf{1} \in \mathbb{R}^T$ is the vector with all components being 1. $\mathcal{R} : \mathbb{R}^{N^2 \times T} \to \mathbb{R}^{N^2 \times T}$ is a linear operator that rotates each frame of the volume, increasing the rotation on each planetary frames (which are the columns of $(\boldsymbol{x}\mathbf{1}^T) \in \mathbb{R}^{N^2 \times T}$).

6.3 Limitations

If the optical system is linear and shift-invariant, the observed image result in the 2D convolution of the true intensity distribution and the impulse response of the telescope called: Point Spread Function (PSF). In order to recover the original image without the instrumental disturbances it is necessary to deconvolve the image obtained with the telescope.

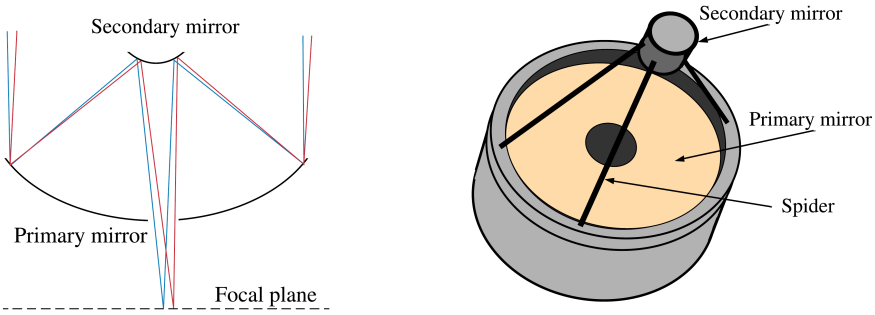


Figure 6.3: Reflecting telescope

But in practice, the telescope system is non shift-invariant.

- The pupil is not just a circular aperture. Indeed, to capture and focus light, most ground-based telescopes do not use a lens but a system of two mirrors (reflecting telescope). As illustrated on Figure 6.3, the system consists of a primary (parabolic) mirror that will focus the light in front of it, so a secondary (hyperbolic) mirror is placed near the focal length in front of the primary mirror to redirect the light back to the optical system. Therefore it partially prevents the light from reaching the primary mirror. In addition to the loss of light collected by the system, it causes diffraction effects due to the obstruction caused by the secondary mirror and the support beams of the secondary mirror which is called the spider structure. These diffraction effects are illustrated on Figure 6.4.

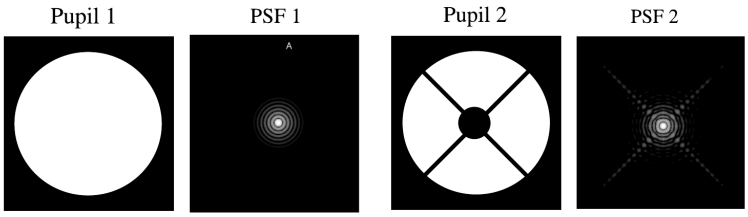


Figure 6.4: Pupil 1 is a perfect circular aperture, the corresponding PSF is the Airy pattern. Pupil 2 is composed of an aperture and the support structure. Its associated PSF contains some spikes caused by the diffraction with the spider structure.

  Therefore, since the diffraction effects will depends of the position of the observed point source, the forward system of the telescope can not be described by a shift-invariant operator as the convolution.

- Moreover, the effects caused by the coronograph are also non shift-invariant. Indeed, the coronograph illustrated on Figure 6.1 has a structure similar to an optical system build from the $2f-$system described in section 4.1. The lens will compute the Fourier transform of the field, and the opaque mask and the Lyot mask correspond to component-wise multiplication [10, p.31].

The deformation caused by the coronograph is only present for a small angular separation. Indeed, as illustrated on Figure 6.1, for a sufficient angular separation, the light (red beams) will avoid the deformation caused by the mask. Thus, at a sufficiently large distance from the center of the image, the telescope can be modelled by convolution.

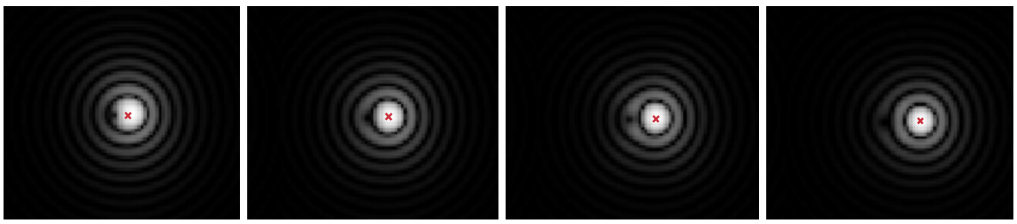


Figure 6.5: Coronograph-induced deformation for a point light source moving away from the telescope axis (data genererated with HEEPS [1]).

The Figure 6.5 illustrates the deformation caused by the coronograph for a point light source (a star, an exoplanet) moving away from the telescope axis. At a sufficient angular separation, we notice that the deformation disappears to give a result similar to an Airy pattern, resulting from a simple convolution with the pupil indicator function [44] as shown in Figure 6.4.

Therefore, the model (6.1) will give good results for the detection of exoplanets having a sufficiently large angular separation with their host star. But to detect exoplanets close to their star, we will have to take into account the real operator that characterises the telescope. This integral operator is a non shift-invariant operator that behaves like a convolution at the periphery of the domain but has a damping effect when approaching the center of the domain, exactly like the integral operators (5.5) studied in the Chapter 2.
As we have shown in Chapter 2, the diagonal/circulant neural network are efficient to learn such integral operator with only few factors.
The forward model becomes:

$$\boldsymbol{M} = \boldsymbol{L} + \phi(\boldsymbol{S}) + \boldsymbol{N}, \tag{6.3}$$

with $\phi(.)$ denote the non shift-invariant integral operator characterising the telescope applied here on each frame (column) of $\boldsymbol{S}$ independently.

6.4 Inverse problem

The inverse problem is solved with a negative log-likelihood Estimate approach for which the Hubert loss has been shown [33] to be well suited for the sub-exponential distribution of the speckle field. We add the priors of low rank, sparsity and also of positivity (since the variables corresponds to an intensity of images and therefore are non negative).

$$\begin{aligned} \boldsymbol{L}^*, \boldsymbol{x}^* = \arg\min_{\boldsymbol{L},\boldsymbol{x}} \quad & \|\boldsymbol{M} - \boldsymbol{L} - \phi(\mathcal{R}[\boldsymbol{x}\mathbf{1}^T]\|_\delta^H \\ s.t. \quad & \operatorname{rank}(\boldsymbol{L}) \leq r \qquad \text{low rank prior} \\ & \|\boldsymbol{x}\|_0 \leq s \qquad \text{sparse prior} \\ & \boldsymbol{L}, \boldsymbol{x} \geq 0 \qquad \text{positivity} \end{aligned} \tag{6.4}$$

The metric $\|\boldsymbol{a}\|_\delta^H$ for $\boldsymbol{a} \in \mathbb{R}^n$ is defined as $\sum_{i=1}^n |a_i|_\delta$, where $|a|_\delta$ denote the Hubert function:

$$|a|_\delta = \begin{cases} \frac{1}{2\delta}a^2 & \text{if } |a| \leq \delta, \\ |a| - \frac{\delta}{2} & \text{if } |a| > \delta. \end{cases} \tag{6.5}$$

The optimization problem (6.4) is NP-hard and not convex because of the low rank and l_0 constraints. Hence, we consider a tractable approach employing convex relaxations to obtain a convex optimization problem:

$$\begin{aligned} \boldsymbol{L}^*, \boldsymbol{x}^* = \arg\min_{\boldsymbol{L},\boldsymbol{x}} \quad & \|\boldsymbol{M} - \boldsymbol{L} - \phi(\mathcal{R}[\boldsymbol{x}\mathbf{1}^T])\|_\delta^H \\ s.t. \quad & \boldsymbol{L} \in \text{span}(U_r^*) && \text{low rank prior} \\ & \|\boldsymbol{x}\|_1 \leq \tau_s && \text{sparse prior} \\ & \boldsymbol{L}, \boldsymbol{x} \geq 0 && \text{positivity} \end{aligned} \tag{6.6}$$

For $\delta > 0$, the Huber loss is convex and differentiable. The l_1-norm has been shown to be an effective surrogate for the number of non-zero entries of a vector [9]. To relax the rank constraint, [6] proposed to impose to $\boldsymbol{L}$ to lie in the span of the first r columns of $\boldsymbol{U}^*$, where $\boldsymbol{U}^*$ is given by the SVD decomposition of a good estimate of $\boldsymbol{L}$ obtained with a fixed-point algorithm presented in [34].

6.5 Proposed solution

The problem of detecting exoplanets by the inverse problem approach (6.6) can be divided into two sub-problems:

- Build a model that approximates ϕ and for which we can compute the derivative with respect to the inputs in order to integrate it into the resolution of the inverse problem.
- Solve the inverse problem for a particular data cube using the telescope model to detect exoplanets. The problem (6.6) can be solved for example using the generalized forward backward algorithm [38]. This method require the computation at each iteration of the value $\phi(\boldsymbol{x})$ and the computation of the gradient of ϕ with respect to its inputs: $\nabla_{\boldsymbol{x}}\phi(\boldsymbol{x})$.

To solve the inverse problem, regardless of the optimization method used, we will often have to evaluate the model describing the telescope and its derivatives.
So the goal is to design a model that approximates the telescope, for which we can quickly calculate the derivative relative to the inputs, and integrate it into the solving of the inverse problem.
To achieve this, we will use the diagonal/doubly-block-circulant network for the following reasons:

- We can not have access to the result of the telescope for each possible input. That is we reduce the number of parameters by exploiting the physics of the problem. Indeed, our observations of section (6.3) states that the telescope can be modelled as an alternance of $2D$ convolution and component-wise multiplications.
- The computational interest both in inference and derivative calculation of the diagonal/doubly-block-circulant neural network.

6.6 Numerical results

Let $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \text{ with } \boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^{N^2},\ i = 1, ..., T\}$ be the dataset (ADI datacube). The vectors $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ correspond to the column-stacked version of the pixels of the input and output image of the telescope, respectively. The ADI datacube which will be used for the numerical simulation is generated with HEEPS [1, 13] which is an High-contrast End-to-End Performance Simulator

mostly geared towards studying the performance of ELT instrument: "METIS".
This simulator is very complex and therefore, it is long to calculate and in particular difficult to integrate in an inverse problem for which one must be able to calculate the derivative with respect to the inputs efficiently. Which is why we are going to learn it with a neural network.

In order to get a simple case, the datacube is generated by:

Model 6.6.1. *a classical vortex coronagraph with a "perfect" circular entrance pupil (without central obstruction, spiders, or segments), and without wavefront aberrations.*

Precisely, the dataset is composed of a pair set (input, target), where the inputs are point light sources located on a grid in the upper left quarter. The Dirac delta inputs appear in a grid 13×13 in the upper left quarter as shown schematically in the Figure 6.6. Therefore, the dataset is composed of $T = 169$ elements of size 200×200 ($N = 200$).

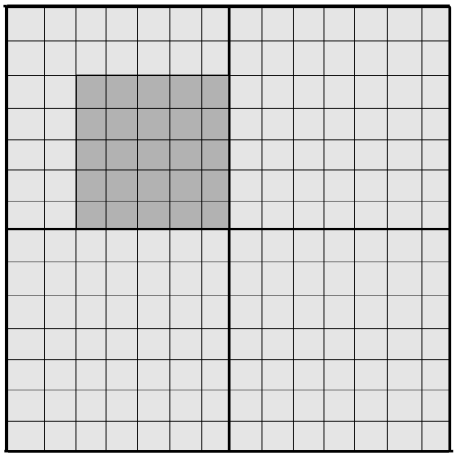


Figure 6.6: This is a frame with light gray pixels. Darker gray is the region where the light source points are placed to form the entries in the dataset.

We consider the following optimization problem:

$$\min_{\boldsymbol{\theta}} \quad E_T = \frac{1}{T} \sum_{i=1}^{T} \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}), \tag{6.7}$$

with

$$\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \frac{1}{N^2} \|\boldsymbol{y}_i - \phi(\boldsymbol{x_i}; \boldsymbol{\theta})\|_2^2. \tag{6.8}$$

$\boldsymbol{\theta}$ denotes the parameters of the network function ϕ. E_T is the error metric that will be used for the training, it is the mean-square-error (noted MSE) evaluated on the dataset. While $\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ is the mean-square-error for the particular input $\boldsymbol{x}_i$.
We define also the relative squared error between a given target and the corresponding prediction of the network :

$$\mathcal{L}_r(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \frac{\|\boldsymbol{y}_i - \phi(\boldsymbol{x}_i; \boldsymbol{\theta})\|_2}{\|\boldsymbol{y}_i\|_2}, \tag{6.9}$$

which will be used to analyse the results.
The hyperparameters of the training strategy are given in section A.6.

Through this section, we will answer to several questions about the network's ability to learn the dataset generated by the simulator 6.6.1.

6.6.1 Initialization strategy

In this section, we are interested in analyzing the initialization strategy of the network.
As we know from section 6.3, the telescope behaves approximately like a convolution away from the center of the image. Therefore, the doubly-block-diagonal matrices of the network are initialized with the image of the telescope for the farthest entry point from the center, refocused (see Figure 6.7).
In order to approximate the damping at center of the image observed on Figure 6.5, the diagonal matrices are initialized such that it does not change the value outside a radius R, and inside a radius R from the center, it decrease according to $\frac{r}{R}$, where r is the radial distance to the center (see Figure 6.8).

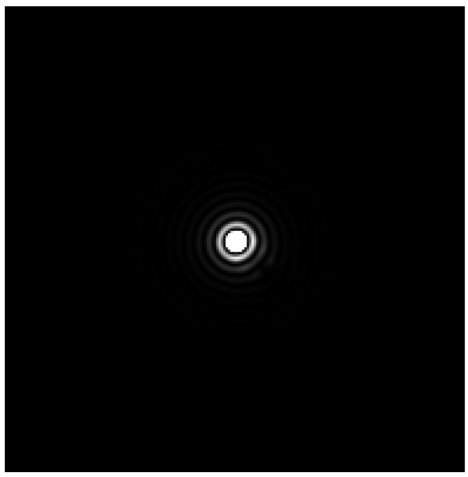


Figure 6.7: Initialization of the matrix $\boldsymbol{w}$ which generates the doubly-block-circulant matrix $\boldsymbol{B}(\boldsymbol{w})$.

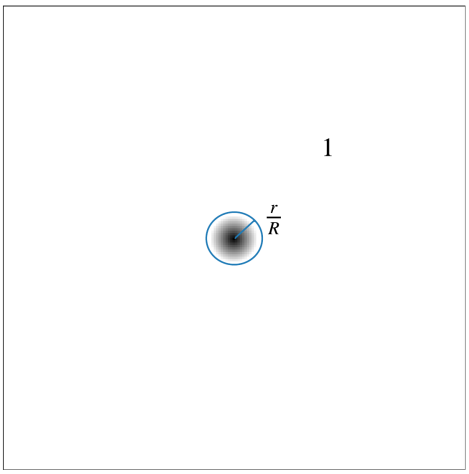


Figure 6.8: Initialization of the matrix $\boldsymbol{w}$ which generates the diagonal matrix $\boldsymbol{D}(\boldsymbol{w})$.

Figure 6.9 gives the result for a particular input for this initialization scheme.

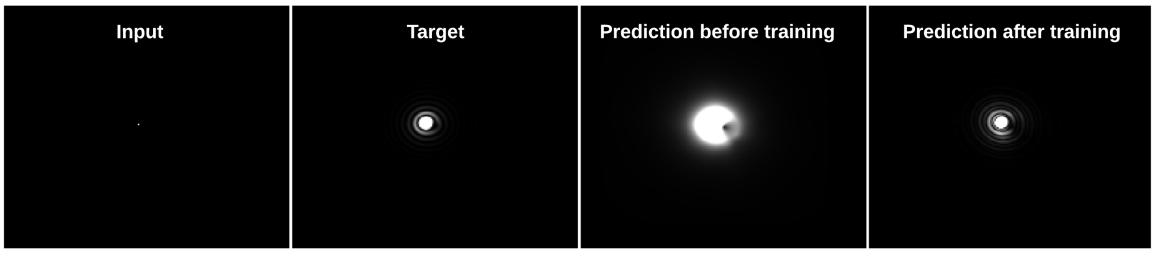
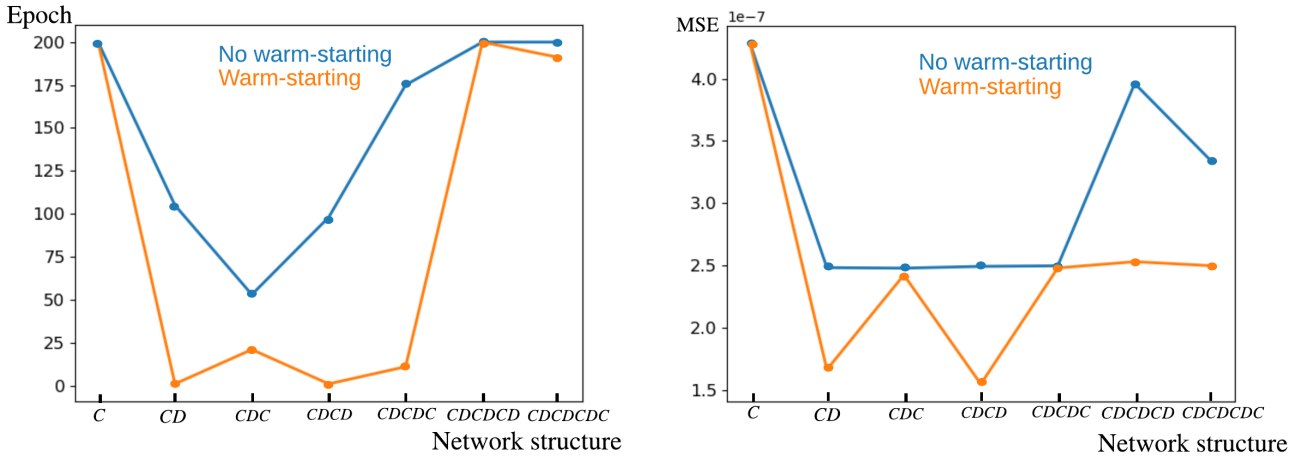


Figure 6.9: From left to right: input signal, target, prediction before the training of the network and the prediction after the training. Neural network with structure $\boldsymbol{CDCDCD}$ with $\boldsymbol{C}$ denoting a 2D convolution layer and $\boldsymbol{D}$ a component-wise multiplication layer. The layers are applied on the input from left to right.

In addition to the initialization mentioned above, we can also test a *warm-starting strategy* [8]. Normally, neural network training begins with the random initialization of weights. Warm-starting consists in initializing the network weights by copying them from a previously trained model. This allows to start training from a better starting point. The strategy here is to progressively increase the number of network layers. Once the network is trained with a certain

number of layers, we use the pre-trained weights from the previous layers and add a new layer. Here, only the weights of the new layer are trained, the pre-trained ones are frozen during the learning process. This has the advantage of reducing the computational cost.

We can see on Figure 6.10a that the warm-starting network actually starts with a better initial condition and therefore a lower number of epochs is required to achieve a given accuracy. With the figure 6.10b, we observe that for a fixed number of epochs, the neural network with hot start reaches a better accuracy.



(a) Number of epochs needed to achieve a given MSE precision ($2.5 \cdot 10^{-7}$) on the dataset.

(b) MSE of the dataset resulting of a given number of epochs (200).

Figure 6.10: In blue for a network initialized according to Figures 6.7 and 6.8 with a training restarting from scratch. In orange, for a network trained with the warm-starting strategy (include only the number of epoch for the last layer).

6.6.2 Leave-one-out

It has been shown in section 5.2 that for one-dimensional integral operators with diagonal/circulant factorization with few factors, good results can be obtained with a reduced number of data (input,target). Guided by this observation, we wish to know if it also applies to the telescope 6.6.1.

For that, we are gonna do a leave-one-out [24, Chapter 7]. The neural network is trained on all the data except for one point which belong to the validation set. Therefore the validation set contains only one element. Usually, the operation is repeated by changing the data belonging to the validation set as shown in Figure 6.11.

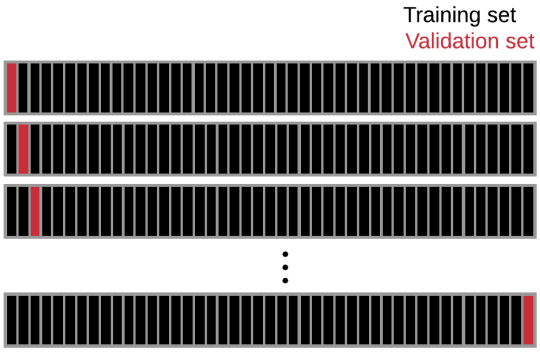


Figure 6.11: Illustration of the general leave-one-out process: the validation set contains successively all the data separately.

However, here we want to measure the importance of the data in the dataset as a function of their radial distance from the center of the image. Therefore, a leave-one-out is performed in such a way that the points which belong successively to the validation set are aligned along a line starting from the center. For a given validation set, the stopping criterion used for learning the network is early stopping presented in section 1.2.5 with a window size of 10.

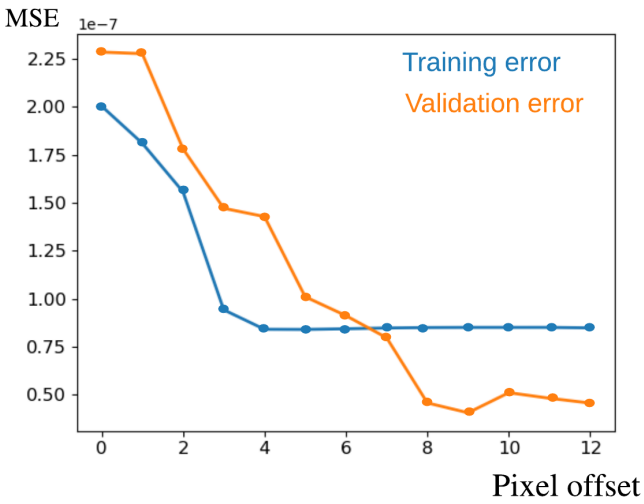


Figure 6.12: Evolution of the MSE of the training set and the validation set for a leave-one-out where the points belonging successively to the validation set are on a line passing through the center of the image. The abscissa is the radial distance (in pixels) of the data removed from the training set to be put into the validation set. The structure of the network is as follows: ***CDC***.

The Figure 6.12 gives the error evolution of the training set and validation set as a function of the radial distance (in pixels) of the data transferred from the training to the validation set. The following observations can be made:

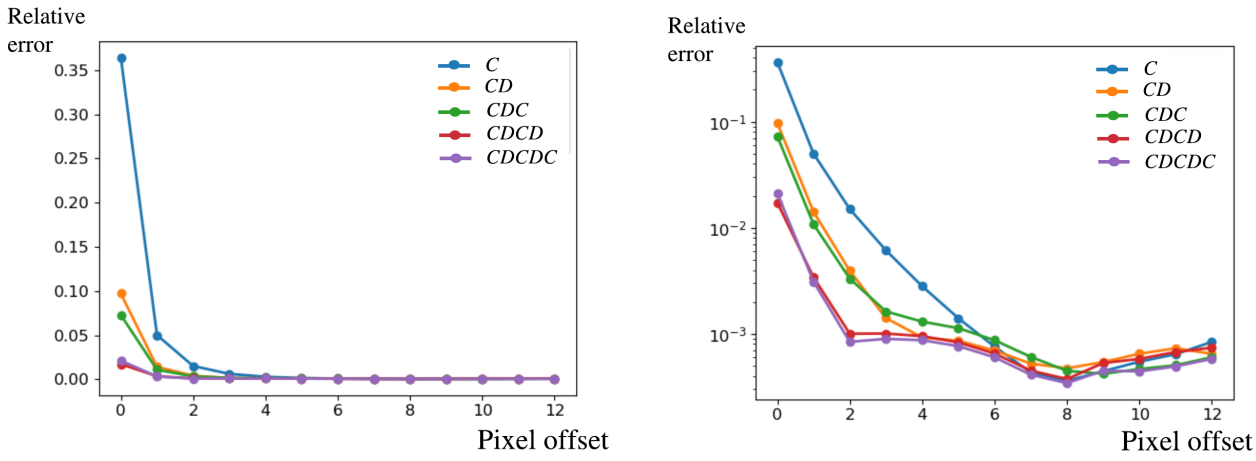
- We can see that the training error is greater for a data suppressed from the training set that is close to the center of the image. This is in line with what was expected. Indeed, the effects (deformations) of the coronograph occur close to the center as shown in Figure 6.5.
  Therefore, data close to the center of the image are important for learning, given the non-shift invariant behaviour near the center.

- The training error decreases with the distance to the center until it is constant from a radial distance of 5 pixels. This means that beyond this distance, deleting one of the data in the training set does not affect the learning and generalization performance of the network. This is due to the reduction of coronographic deformations.
- The validation error, i.e. the error on the data deleted from the training set decreases with the distance for the same reasons as the training error. On the other hand it continues to decrease with distance because the behaviour of the telescope looks more and more like a convolution as you move away from the center.

Therefore, the number of training data far from the center can be reduced for an equivalent performance as the telescope behaviour becomes convolutional (shift-invariant).

6.6.3 Error distribution

Here we will analyze the error distribution according to the position of the input Dirac delta. The Figure 6.13, gives the evolution of the error as a function of the distance to the center of the input delta for several neural network structures. We find in blue the decreasing behavior of the error with the distance to the center for a network having only a convolution structure. As expected, the error of the network decreases with the radial distance to the center.
We also notice that the addition of a diagonal layer immediately reduces the error at the center. Moreover, the addition of layers reduces the error at the center by modeling the non shift-invariant behaviour of the telescope at the center.



(a) Relative squared error (6.9).

(b) Relative squared error (6.9).

Figure 6.13: Evolution of the error for particular inputs $\boldsymbol{x}_i$ which are Dirac delta placed on a line passing through the center of the image. The abscissa is the radial distance (in pixels) of the input delta with the center.

6.6.4 Dataset-augmentation

In order to obtain a model which can better generalize on new input data, we can train it on more data. Having a limited dataset, we can build new dummy data based on assumptions about the real model and add it to the training set. This process is called *dataset augmentation* [21, p.233]. The simplified telescope 6.6.1 is rotation invariant. Therefore, we can apply a synthetic rotation on the data at our disposition in order to increase the size of the the training

set. Precisely, each pair (input,target) will be added to the dataset for the 90°, 180°, 270° rotations as shown in the Figure 6.14.

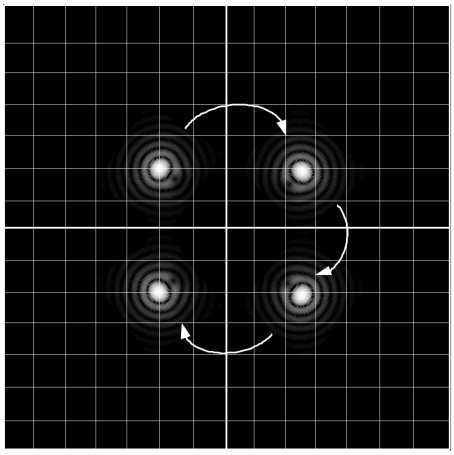


Figure 6.14: Dataset augmentation: each data of the original dataset rotated by 90°, 180° and 270° is added.

We can therefore compare two networks, the first trained on the initial dataset (illustrated on Figure 6.6) and the second formed on dataset resulting of the dataset augmentation (illustrated on Figure 6.14).
Figure 6.15 gives the error map with respect to the position of the input delta for both networks. We can see that for the network trained on the initial dataset, the error is higher in the quadrants for which it has no training data (top right, bottom left and right). But for the network trained on the augmented dataset, we obtain a result with identical behavior in all quadrants (rotation invariant). The MSE evaluated on the augmented dataset is of $3.45 \cdot 10^{-6}$ for the $\boldsymbol{CDC}$ network trained on the initial dataset while it is of $1.72 \cdot 10^{-6}$ for the $\boldsymbol{CDC}$ network trained on the augmented dataset.
Therefore, the MSE on the augmented dataset is smaller for the network trained on the augmented data set than the MSE of the first network trained on the initial dataset.

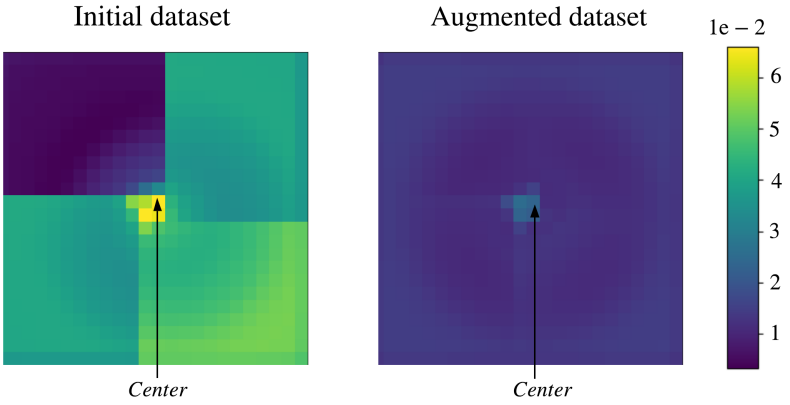


Figure 6.15: This is a map of the error. Each pixel i in these figures has the relative squared error value (6.9) between the prediction of the network and the target value for the particular data pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ where $\boldsymbol{x}_i$ is the Dirac delta function placed at the pixel i. The neural network has the structure: $\boldsymbol{CDC}$. The figures on the left and on the right correspond to a network trained respectively on the initial and augmented dataset.

Conclusion

Summary

We have shown that for a particular application where the structure of the target operator is or is close to the particular factorization in diagonal/circulant matrices, that the diagonal/circulant neural network can efficiently learn the operator with a reduced amount of training data.

Furthermore, we have shown that diagonal/doubly-block-circulant neural network reveal to be well adapted to model with few parameters and few data the non shift-invariant behaviour of the telescope which was shown in section 6.3. Moreover, the ability to efficiently evaluate the gradient of this neural network with respect to the input enables to performs fast implementation of the optimization method to solve the inverse problem (6.6) for the exoplanets detection.
Even if an improvement in the learning of the telescope operator has been demonstrated using a diagonal/circulant network compared to a simple convolution modeling of the telescope, it remains to effectively integrate the learned operator in solving the inverse problem and test its performance. In addition, this improvement is noticeable only for very small angular separations.

Possible improvements

First, the parameters for the learning algorithm given in section A.6 were hand-picked. Ideally, we would want some kind of cross-validation to select appropriate values. It might also be interesting to analyse the performance of the network for a telescope that does not include as many simplifications (6.6.1), for example a telescope having a pupil that contains the spiders.

In this paper, we have studied a purely linear model (2.2):

$$f(\boldsymbol{x};\boldsymbol{\theta}) = (f_L \circ ... \circ f_2 \circ f_1)(\boldsymbol{x}),$$

were function f_l is a linear mapping that can be defined by one of the following functions:

$$f_l(\boldsymbol{x};\boldsymbol{w}_l) = \boldsymbol{D}(\boldsymbol{w}_l)\boldsymbol{x} \quad \text{or} \quad f_l(\boldsymbol{x};\boldsymbol{w}_l) = \boldsymbol{C}(\boldsymbol{w}_l)\boldsymbol{x}.$$

Instead, and especially for integral operators for which we do not expect diagonal/circulant factorization with few factors, we could try the following structure that adds bias and non-linear functions to the layers:

$$f_l(\boldsymbol{x};\boldsymbol{w}_l) = \phi(\boldsymbol{D}(\boldsymbol{w}_l)\boldsymbol{x} + \boldsymbol{b}_l) \quad \text{or} \quad f_l(\boldsymbol{x};\boldsymbol{w}_l) = \phi(\boldsymbol{C}(\boldsymbol{w}_l)\boldsymbol{x} + \boldsymbol{b}_l),$$

with $\phi(.)$ a non linear function and $\boldsymbol{b}_l$ a vector of bias parameters. It could be interesting to measure the impact of the non-linearities in the network on its accuracy and its convergence rate.

Appendix A

Appendix

A.1 Proof of the Theorem (1.1)

Proof. Let $\boldsymbol{C}(\boldsymbol{w})$ be the circulant matrix generated by vector $\boldsymbol{w} \in \mathbb{R}^N$.
Let $\boldsymbol{v}_k$ be the k^{th} column of ${\boldsymbol{F_N}}^{-1} = {\boldsymbol{F_N}}^*$. We want to prove that $\boldsymbol{v}_k$ is an eigenvector of $\boldsymbol{C}(\boldsymbol{w})$.

$$[\boldsymbol{C}(\boldsymbol{w})\boldsymbol{v}_k]_m = \sum_{n=0}^{N-1} [\boldsymbol{C}(\boldsymbol{w})]_{m,n} [\boldsymbol{v}_k]_n = \sum_{n=0}^{N-1} w_{(m-n)\text{mod}N} \frac{1}{\sqrt{N}} \omega_N^{-kn}. \tag{A.1}$$

By a change in variable: $j = (m-n)\text{mod}N \Rightarrow \exists s \in \mathbb{Z} \mid j = m - n + sN$ and by using the periodicity of the complex exponential function, we obtain: $\omega_N^{-kn} = \omega_N^{-k(m-j+sN)} = \omega_N^{-k(m-j)}$. Then, we can rewrite:

$$[\boldsymbol{C}(\boldsymbol{w})\boldsymbol{v}_k]_m = \left(\sum_{j=0}^{N-1} w_j \omega_N^{kj} \right) \frac{1}{\sqrt{N}} \omega_N^{-km} = \sqrt{N} [\boldsymbol{F_N}\boldsymbol{w}]_k [\boldsymbol{v}_k]_m. \tag{A.2}$$

Therefore, we proved that $\boldsymbol{v}_k$ is an eigenvector of the matrix $\boldsymbol{C}(\boldsymbol{w})$ with an eigenvalue given by $\sqrt{N} [\boldsymbol{F_N}\boldsymbol{w}]_k$:

$$\boldsymbol{C}(\boldsymbol{w})\boldsymbol{v}_k = \sqrt{N} [\boldsymbol{F_N}\boldsymbol{w}]_k \boldsymbol{v}_k. \tag{A.3}$$

By concatenating the result for each column of ${\boldsymbol{F_N}}^{-1}$, we have:

$$\boldsymbol{C}(\boldsymbol{w}){\boldsymbol{F_N}}^{-1} = {\boldsymbol{F_N}}^{-1}\text{diag}(\sqrt{N}\boldsymbol{F_N}\boldsymbol{w}). \tag{A.4}$$

And finally:

$$\boldsymbol{C}(\boldsymbol{w}) = {\boldsymbol{F_N}}^{-1}\text{diag}(\sqrt{N}\boldsymbol{F_N}\boldsymbol{w})\boldsymbol{F_N}. \tag{A.5}$$

□

A.2 Proof of the Theorem (1.2)

Proof. Let $\boldsymbol{B}(\boldsymbol{w})$ be the doubly-block-circulant matrix generated by matrix $\boldsymbol{w} \in \mathbb{R}^{M \times N}$.
Let $\boldsymbol{v}_k$ be the k^{th} column of $(\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1} = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^*$ where $k = k_1 + k_2 M$ $(0 \leq k_1 < M$, $0 \leq k_2 < N)$. We want to prove that $\boldsymbol{v}_k$ is an eigenvector of $\boldsymbol{B}(\boldsymbol{w})$.
Let l be such that $l = l_1 + l_2 M$ $(0 \leq l_1 < M$, $0 \leq l_2 < N)$.

$$[\boldsymbol{B}(\boldsymbol{w})\boldsymbol{v}_k]_l = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} [\boldsymbol{B}_{(l_2-n)\text{mod}N}]_{l_1,m} [\boldsymbol{v}_k]_{m+nM}. \tag{A.6}$$

By definition of DBC matrix, the block $\boldsymbol{B}_i = \boldsymbol{C}(\boldsymbol{w}_{:,i})$, therefore, we can rewrite:

$$[\boldsymbol{B}(\boldsymbol{w})\boldsymbol{v}_k]_l = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} w_{(l_1-m)\text{mod}M,(l_2-n)\text{mod}N}[\boldsymbol{v}_k]_{m+nM}. \tag{A.7}$$

By two changes of variable:

$$\begin{aligned} j_1 = (l_1 - m)\text{mod}M &\Rightarrow \exists s_1 \in \mathbb{Z} \mid j_1 = l_1 - m + s_1 M, \\ j_2 = (l_2 - n)\text{mod}N &\Rightarrow \exists s_2 \in \mathbb{Z} \mid j_2 = l_2 - n + s_2 N, \end{aligned} \tag{A.8}$$

we obtain:

$$[\boldsymbol{B}(\boldsymbol{w})\boldsymbol{v}_k]_l = \sum_{j_1=0}^{M-1}\sum_{j_2=0}^{N-1} w_{j_1,j_2}[\boldsymbol{v}_k]_{m+nM}, \tag{A.9}$$

with $m = (l_1 - j_1 + s_1 M)$ and $n = (l_2 - j_2 + s_2 N)$. And we know that:

$$\begin{aligned} [\boldsymbol{v}_k]_{m+nM} &= [(\boldsymbol{F}_N \otimes \boldsymbol{F}_{\boldsymbol{M}})^*]_{m+nM,k} \\ &= [(\boldsymbol{F}_N^* \otimes \boldsymbol{F}_{\boldsymbol{M}}{}^*)]_{m+nM,k_1+k_2M} \\ &= [\boldsymbol{F}_N^*]_{n,k_2}[\boldsymbol{F}_{\boldsymbol{M}}{}^*]_{m,k_1} \\ &= \frac{1}{\sqrt{N}}\omega_N^{-k_2 n}\frac{1}{\sqrt{M}}\omega_M^{-k_1 m}. \end{aligned} \tag{A.10}$$

By using the periodicity of the complex exponential function, we obtain:
$\omega_M^{-k_1 m} = \omega_M^{-k_1(l_1-j_1+s_1M)} = \omega_M^{-k_1(l_1-j_1)}$. The same can be done for $\omega_N^{-k_2 n}$. Therefore, we obtain:

$$\begin{aligned} [\boldsymbol{B}(\boldsymbol{w})\boldsymbol{v}_k]_l &= \sum_{j_1=0}^{M-1}\sum_{j_2=0}^{N-1} w_{j_1,j_2}\frac{1}{\sqrt{MN}}\omega_M^{-k_1(l_1-j_1)}\omega_N^{-k_2(l_2-j_2)} \\ &= \left(\sum_{j_1=0}^{M-1}\sum_{j_2=0}^{N-1} w_{j_1,j_2}\omega_M^{k_1 j_1}\omega_N^{k_2 j_2}\right)\frac{1}{\sqrt{MN}}\omega_M^{-k_1 l_1}\omega_N^{-k_2 l_2} \\ &= \sqrt{MN}[(\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}})\text{vec}(\boldsymbol{w})]_{k_1+k_2M}\ [\boldsymbol{v}_k]_l. \end{aligned} \tag{A.11}$$

As it is valid for any component l, we obtain:

$$\boldsymbol{B}(\boldsymbol{w})\boldsymbol{v}_k = \sqrt{MN}[(\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}})\text{vec}(\boldsymbol{w})]_{k_1+k_2M}\ \boldsymbol{v}_k. \tag{A.12}$$

Therefore, we proved that $\boldsymbol{v}_k$ is an eigenvector of the matrix $\boldsymbol{B}(\boldsymbol{w})$ with an eigenvalue given by $\sqrt{MN}[(\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}})\text{vec}(\boldsymbol{w})]_{k_1+k_2M}$.
As it is valid for any column k, we obtain:

$$\boldsymbol{B}(\boldsymbol{w})(\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}})^{-1} = (\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}})^{-1}\text{diag}(\sqrt{MN}(\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}})\text{vec}(\boldsymbol{w})). \tag{A.13}$$

Finally:

$$\boldsymbol{B}(\boldsymbol{w}) = (\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}})^{-1}\text{diag}(\sqrt{MN}(\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}})\text{vec}(\boldsymbol{w}))(\boldsymbol{F}_{\boldsymbol{N}} \otimes \boldsymbol{F}_{\boldsymbol{M}}). \tag{A.14}$$

□

A.3 DFT in the numpy package of python

In the numpy package of python, the DFT and its inverse are defined as:

$$\begin{cases} \hat{f}[k] = \sum_{n=0}^{N-1} f[n]\omega_N^{nk} \quad , \quad k = 0, ..., N-1, \\ f[n] = \frac{1}{N}\sum_{k=0}^{N-1} \hat{f}[k]\omega_N^{-nk}, \quad n = 0, ..., N-1, \end{cases} \tag{A.15}$$

with $\omega_N = e^{-\mathrm{i}\frac{2\pi}{N}}$.
Then the DFT can be rewritten as matrix-vector product: $\hat{\boldsymbol{f}} = \boldsymbol{F_N f}$ with $\boldsymbol{F_N} \in \mathbb{R}^{N\times N}$, the DFT matrix defined as $(\boldsymbol{F_N})_{k,l} = \omega_N^{kl}$.
In that case, the DFT matrix is only symmetric and orthogonal: $\boldsymbol{F_N}^{-1} = \frac{1}{N}\boldsymbol{F_N}^*$.
Based on this definition of the DFT matrix, the product between a circulant matrix and a vector differs from (1.5):

$$\boldsymbol{C(w)x} = \boldsymbol{F_N}^{-1}\mathrm{diag}(\boldsymbol{F_N w})\boldsymbol{F_N x} = \mathcal{F}^{-1}(\mathcal{F}(\boldsymbol{w}) \odot \mathcal{F}(\boldsymbol{x})), \tag{A.16}$$

where $\mathcal{F}$ denotes the DFT defined according to (A.15).

A.4 2D-DFT in the numpy package of python

In the numpy package of python, the 2D-DFT and its inverse are defined as:

$$\begin{cases} \hat{f}[k,l] = \sum\limits_{m=0}^{M-1}\sum\limits_{n=0}^{N-1} f[m,n]\omega_M^{mk}\omega_N^{nl}, \\ f[m,n] = \dfrac{1}{MN}\sum\limits_{k=0}^{M-1}\sum\limits_{l=0}^{N-1} \hat{f}[k,l]\omega_M^{-mk}\omega_N^{-nl}. \end{cases} \tag{A.17}$$

with $\omega_N = e^{-\mathrm{i}\frac{2\pi}{N}}$.
Considering the column stack version, we can rewrite the 2D-DFT as a matrix-vector product:

$$\mathrm{vec}(\hat{\boldsymbol{f}}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})\mathrm{vec}(\boldsymbol{f}), \tag{A.18}$$

with $(\boldsymbol{F_N})_{k,l} = \omega_N^{kl}$.
In that case, the 2D-DFT matrix is only symmetric and orthogonal:
$(\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1} = \frac{1}{MN}(\boldsymbol{F_N} \otimes \boldsymbol{F_M})^*$.
Based on this definition of the DFT matrix, the product between a doubly-block-circulant matrix and a vector differs from (1.16):

$$\boldsymbol{B(w)}\mathrm{vec}(\boldsymbol{x}) = (\boldsymbol{F_N} \otimes \boldsymbol{F_M})^{-1}\boldsymbol{D}(\boldsymbol{F_N} \otimes \boldsymbol{F_M})\mathrm{vec}(\boldsymbol{x}) = \mathcal{F}_{2D}^{-1}(\mathcal{F}_{2D}(\boldsymbol{w}) \odot \mathcal{F}_{2D}(\boldsymbol{x})), \tag{A.19}$$

where $\mathcal{F}_{2D}$ denotes the 2D-DFT defined according to (A.17).

A.5 Derivation of the kernel of some factorizable integral operators

In this section, we prove the factorization in a product of diagonal and circulant matrices of some non-shift invariant integral operators.
Let the integral operator and the associated kernel be defined as:

$$\begin{aligned} H[f](u) &= [D_{\rho_2} \circ C_\sigma \circ D_{\rho_1} \circ C_\sigma][f](u) \\ &= \rho_2(u)\int k_\sigma(u-v)\left(\rho_1(v)\int k_\sigma(v-x)f(x)\,\mathrm{d}x\right)\,\mathrm{d}v \\ &= \int K(u,x)f(x)\,\mathrm{d}x, \\ K(u,x) &= \rho_2(u)\int k_\sigma(u-v)\rho_1(v)k_\sigma(v-x)\,\mathrm{d}v, \end{aligned} \tag{A.20}$$

with k_σ, C_σ, D_ρ defined in (5.4).

"Exponential" integral operator

First, we consider the integral operator (A.20) with $\rho_1(v) = e^{\frac{-av^2}{2\sigma^2}}$. The kernel is given by:

$$\begin{aligned}
K(u,x) &= \rho_2(u) \int \left(\kappa_\sigma e^{\frac{-(u-v)^2}{2\sigma^2}}\right)\left(e^{\frac{-av^2}{2\sigma^2}}\right)\left(\kappa_\sigma e^{\frac{-(v-x)^2}{2\sigma^2}}\right) \mathrm{d}v \\
&= \rho_2(u)\kappa_\sigma^2 e^{\frac{-(u^2+x^2)}{2\sigma^2}} \int e^{\frac{-((2+a)v^2-2(u+x)v)}{2\sigma^2}} \mathrm{d}v \\
&= \rho_2(u)\kappa_\sigma^2 e^{\frac{-(u^2+x^2)}{2\sigma^2}} e^{\frac{(u+x)^2}{2(2+a)\sigma^2}} \int e^{\frac{-\left(\sqrt{2+a}v - \frac{(u+x)}{\sqrt{2+a}}\right)^2}{2\sigma^2}} \mathrm{d}v \\
&= \rho_2(u)\kappa_\sigma e^{\frac{-((1+a)(u^2+x^2)-2ux)}{2(2+a)\sigma^2}} \frac{1}{\sqrt{2+a}} \int \frac{1}{\sqrt{2\pi}\left(\frac{\sigma}{\sqrt{2+a}}\right)} e^{\frac{-\left(v-\frac{(u+x)}{2+a}\right)^2}{2\left(\frac{\sigma}{\sqrt{2+a}}\right)^2}} \mathrm{d}v \\
&= \rho_2(u)\kappa_{\sqrt{2+a}\sigma} e^{\frac{-((1+a)(u^2+x^2)-2ux)}{(2+a)(\sqrt{2}\sigma)^2}} \\
&= \rho_2(u) k_{\sqrt{2}\sigma}(u-x) \left[\sqrt{\frac{2}{2+a}} e^{\frac{-a(u+x)^2}{2(2+a)\sigma^2}}\right].
\end{aligned} \tag{A.21}$$

From result (A.21) and by the linearity of the kernel with respect to ρ_1, we can deduce the following expression for the kernel when $\rho_1(v) = 1 - ce^{\frac{-av^2}{2\sigma^2}}$:

$$K(u,x) = \rho_2(u) k_{\sqrt{2}\sigma}(u-x) \left[1 - c\sqrt{\frac{2}{2+a}} e^{\frac{-a(u+x)^2}{2(2+a)\sigma^2}}\right]. \tag{A.22}$$

"Polynomial" integral operator

It corresponds to the integral operator (A.20) with $\rho_1(v) = \sum_{i=0}^n a_i v^i$. The kernel is given by:

$$\begin{aligned}
K(u,x) &= \rho_2(u) \int \left(\kappa_\sigma e^{\frac{-(u-v)^2}{2\sigma^2}}\right)\left(\sum_{i=0}^n a_i v^i\right)\left(\kappa_\sigma e^{\frac{-(v-x)^2}{2\sigma^2}}\right) \mathrm{d}v \\
&= \rho_2(u)\kappa_\sigma^2 e^{\frac{-(u^2+x^2)}{2\sigma^2}} \sum_{i=0}^n a_i \int v^i e^{\frac{-(2v^2-2(u+x)v)}{2\sigma^2}} \mathrm{d}v \\
&= \rho_2(u)\kappa_\sigma^2 e^{\frac{-(u^2+x^2)}{2\sigma^2}} e^{\frac{(u+x)^2}{4\sigma^2}} \sum_{i=0}^n a_i \int v^i e^{\frac{-\left(\sqrt{2}v - \frac{(u+x)}{\sqrt{2}}\right)^2}{2\sigma^2}} \mathrm{d}v \\
&= \rho_2(u)\kappa_\sigma e^{\frac{-(u-x)^2}{4\sigma^2}} \sum_{i=0}^n a_i \frac{1}{\sqrt{2}} \int v^i \left(\frac{1}{\sqrt{2\pi}\left(\frac{\sigma}{\sqrt{2}}\right)} e^{\frac{-\left(v-\frac{(u+x)}{2}\right)^2}{2\left(\frac{\sigma}{\sqrt{2}}\right)^2}}\right) \mathrm{d}v \\
&= \rho_2(u)\kappa_{\sqrt{2}\sigma} e^{\frac{-(u-x)^2}{2(\sqrt{2}\sigma)^2}} \sum_{i=0}^n a^i \mathbb{E}_X[X^i] = \rho_2(u) k_{\sqrt{2}\sigma}(u-x) \sum_{i=0}^n a^i \mathbb{E}_X[X^i],
\end{aligned} \tag{A.23}$$

where X is a random variable following a normal distribution: $X \sim \mathcal{N}\left(\frac{u+x}{2}, \left(\frac{\sigma}{\sqrt{2}}\right)^2\right)$.
Let $Z \sim \mathcal{N}(0,1)$ and $Y = \mu + \sigma Z \Leftrightarrow Y \sim \mathcal{N}(\mu, \sigma^2)$, we have:

$$\mathbb{E}_Y[Y^i] = \mathbb{E}_Z[(\mu + \sigma Z)^i] = \sum_{k=0}^i \binom{i}{k} \mu^{i-k} \sigma^k \mathbb{E}_Z[Z^k]. \tag{A.24}$$

$\mathbb{E}_Z[Z^k]$ is the k^{th} moment of a unit normal distribution and is given by:

$$l_k \triangleq \mathbb{E}_Z[Z^k] = \begin{cases} 0 & \text{if } k \text{ is odd.} \\ (k-1)!! & \text{if } k \text{ is even.} \end{cases} \tag{A.25}$$

Therefore, we deduce: $\mathbb{E}_X[X^i] = \sum_{k=0}^{i} \binom{i}{k} \left(\frac{u+x}{2}\right)^{i-k} \left(\frac{\sigma}{\sqrt{2}}\right)^k l_k$. Finally, we get the closed form of the kernel function:

$$K(u,x) = \rho_2(u) k_{\sqrt{2}\sigma}(u-x) \sum_{i=0}^{n} a_i \left[\sum_{k=0}^{i} \binom{i}{k} \left(\frac{\sigma}{\sqrt{2}}\right)^k l_k \left(\frac{u+x}{2}\right)^{i-k} \right]. \tag{A.26}$$

A.6 Hyperparameters setting

To solve the optimization problems, we used the stochastic gradient descent algorithm with minibatch and momentum introduced in section 1.2.2. The numerical simulations were executed using the PyTorch library [36] on Python which is a machine learning framework, with the optimizer torch::optim::SGD. Table A.1 contains the values of the hyperparameters of the optimization algorithm used.

Parameters	Values
η	2
β	0.9
$\lvert\mathcal{B}\rvert$	5

Table A.1: Hyperparameters setting: η denotes the learning rate, β the momentum and $|\mathcal{B}|$ the batch size.

Bibliography

[1] Heeps: High-contrast end-to-end performance simulator. `https://github.com/vortex-exoplanet/HEEPS`, 2020.

[2] M. A. Abidi, A. V. Gribok, and J. Paik. Optimization Techniques in Computer Vision: Ill-posed problems and regularization. Springer, 2016.

[3] A. Amara and S. P. Quanz. Pynpoint: an image processing package for finding exoplanets. Monthly Notices of the Royal Astronomical Society, 427(2):948–955, 2012.

[4] A. Araujo, B. Negrevergne, Y. Chevaleyre, and J. Atif. The expressive power of deep neural networks with circulant matrices. 2018.

[5] J. M. Beckers. Adaptive optics for astronomy: principles, performance, and applications. Annual review of astronomy and astrophysics, 31(1):13–62, 1993.

[6] L. J. Benoît Pairet, Faustine Cantalloube. Blind source separation for circumstellar disks imaging. Proceedings of the fifth "international Traveling Workshop on Interactions between low-complexity data models and Sensing Techniques"(iTWIST'20).

[7] C. S. Burrus. Fast fourier transforms. Lulu. com, 2008.

[8] Y. Z. By Jennifer Villa. Warm starting for efficient deep learning resource utilization. `https://determined.ai/blog/warm-stating/`, 2018.

[9] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? Journal of the ACM (JACM), 58(3):1–37, 2011.

[10] F. Cantalloube. Detection and characterization of exoplanets in high contrast images by the inverse problem approach. PhD thesis, 2016.

[11] F. Cantalloube, D. Mouillet, L. Mugnier, J. Milli, O. Absil, C. G. Gonzalez, G. Chauvin, J.-L. Beuzit, and A. Cornia. Direct exoplanet detection and characterization using the andromeda method: Performance on vlt/naco data. Astronomy & Astrophysics, 582:A89, 2015.

[12] S. Canu, R. Flamary, and D. Mary. Introduction to optimization with applications in astronomy and astrophysics. EAS Publications Series, 78:127–161, 2016.

[13] B. Carlomagno, O. Absil, M. Kenworthy, G. Ruane, C. U. Keller, G. Otten, M. Feldt, S. Hippler, E. Huby, D. Mawet, et al. End-to-end simulations of the e-elt/metis coronagraphs. In Adaptive Optics Systems V, volume 9909, page 990973. International Society for Optics and Photonics, 2016.

[14] C.-Y. Chao. Circulant matrices (philip j. davis). SIAM Review, 24(3):356, 1982.

[15] C. Charles. Introduction aux problèmes inverses. Notes de Statistique et d'Informatique, 2014.

[16] P. L. Combettes and J.-C. Pesquet. Proximal splitting methods in signal processing. In Fixed-point algorithms for inverse problems in science and engineering, pages 185–212. Springer, 2011.

[17] H. J. Deeg and R. Alonso. Transit photometry as an exoplanet discovery method. Handbook of Exoplanets, page 633–657, 2018.

[18] G. Duchêne. High-angular resolution imaging of disks and planets. New Astronomy Reviews, 52(2-5):117–144, 2008.

[19] P. J. Fessler. Eecs 451 (chapter 6), digital signal processing and analysis.

[20] C. G. Gonzalez, O. Absil, P.-A. Absil, M. Van Droogenbroeck, D. Mawet, and J. Surdej. Low-rank plus sparse decomposition for exoplanet detection in direct-imaging adi sequences-the llsg algorithm. Astronomy & Astrophysics, 589:A54, 2016.

[21] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. MIT press, 2016.

[22] J. W. Goodman. Introduction to Fourier optics. Roberts and Company Publishers, 2005.

[23] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In Advances in neural information processing systems, pages 1135–1143, 2015.

[24] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media, 2009.

[25] M. Huhtanen and A. Perämäki. Factoring matrices into the product of circulant and diagonal matrices. Journal of Fourier Analysis and Applications, 21(5):1018–1033, 2015.

[26] L. John and V. Michel. Machine learning : regression, dimensionality reduction and data visualization. 2018.

[27] H. Khalil. Matrices structurées et matrices de Toeplitz par blocs de Toeplitz en calcul numérique et formel. PhD thesis, 2008.

[28] C. Marois, D. Lafreniere, R. Doyon, B. Macintosh, and D. Nadeau. Angular differential imaging: a powerful high-contrast imaging technique. The Astrophysical Journal, 641(1):556, 2006.

[29] S. Meignen. Traitement d'image avancé.

[30] J. Müller-Quade, H. Aagedal, T. Beth, and M. Schmid. Algorithmic design of diffractive optical systems for information processing. Physica D: Nonlinear Phenomena, 120(1-2):196–205, 1998.

[31] C. Nikou. Filtering in the frequency domain (circulant matrices and convolution),university of ioannina - department of computer science and engineering. 2011.

[32] T. E. Oliphant. A guide to NumPy, volume 1. Trelgol Publishing USA, 2006.

[33] B. Pairet, F. Cantalloube, C. A. Gomez Gonzalez, O. Absil, and L. Jacques. Stim map: detection map for exoplanets imaging beyond asymptotic gaussian residual speckle noise. Monthly Notices of the Royal Astronomical Society, 487(2):2262–2277, 2019.

[34] B. Pairet, F. Cantalloube, and L. Jacques. Reference-less algorithm for circumstellar disks imaging. arXiv preprint arXiv:1812.01333, 2018.

[35] N. Parikh, S. Boyd, et al. Proximal algorithms. Foundations and Trends® in Optimization, 1(3):127–239, 2014.

[36] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

[37] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. IEEE Transactions on pattern analysis and machine intelligence, 12(7):629–639, 1990.

[38] H. Raguet, J. Fadili, and G. Peyré. A generalized forward-backward splitting. SIAM Journal on Imaging Sciences, 6(3):1199–1226, 2013.

[39] S. Ruder. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2016.

[40] M. Schmid, R. Steinwandt, J. Müller-Quade, M. Rötteler, and T. Beth. Decomposing a matrix into circulant and diagonal factors. Linear Algebra and its Applications, 306(1-3):131–143, 2000.

[41] J. Schneider. Interactive extra-solar planet catalog. the extrasolar planets encyclopedia. `http://exoplanet.eu/catalog/`, 2018. (visited on 12/06/2020).

[42] S. Shalev-Shwartz and S. Ben-David. Understanding machine learning: From theory to algorithms. Cambridge university press, 2014.

[43] H. V. Sorensen, D. Jones, M. Heideman, and C. Burrus. Real-valued fast fourier transform algorithms. IEEE Transactions on acoustics, speech, and signal processing, 35(6):849–863, 1987.

[44] R. van Boekel and C. P. Dullemond. Beobachtende astronomie (mkep5), chapter 3 (on diffraction). Max-Planck-Institut fuer Astronomie, 2010.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**