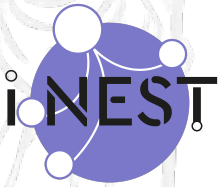
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

PhD Program in Mathematical Sciences
Computational Mathematics Curriculum
XXXVIII cycle

Deep Unfolding: bridging the gap between model-based and data-driven approaches

Thesis written with the financial contribution of the European Union - Next Generation EU - National Recovery and Resilience Plan (NRRP), M4C2



Investment 1.5 : Creazione e Rafforzamento di "Ecosistemi dell'Innovazione per la Sostenibilità", Costruzione di "Leader Territoriali di R&S"
Program : "Interconnected Nord-Est Innovation Ecosystem - iNEST"
MUR ID : ECS00000043
CUP : C43C22000340006

Coordinator
Prof. Giovanni Colombo

Candidate
Erik Chinellato
2097747

Supervisor
Prof. Fabio Marcuzzi

Co-supervisor
Prof. Mario Putti

February 2, 2026

Academic Year 2025-2026

Abstract

This dissertation investigates recent machine learning architectures that aim to bridge traditional model-based methods with data-driven approaches, with a particular focus on interpretability, structure, and consistency. Central to this work is the paradigm of deep unfolding, which provides a unifying framework to reinterpret iterative algorithms as trainable neural networks. We show that deep unfolding naturally manifests through two complementary mechanisms: algorithm unrolling, which defines the network architecture by unfolding iterative procedures into layered structures, and backpropagation through time, which enables the optimization of temporally structured models. Within this framework, bilevel optimization emerges as a principled mathematical foundation that connects classical optimization algorithms with modern learning-based formulations, yielding neural architectures that remain grounded in well-understood mathematical principles.

The first part of the dissertation focuses on audio processing applications, specifically source separation and detection. We revisit nonnegative matrix factorization (NMF) and its variants as a powerful model-based framework for time-frequency analysis. Algorithm unrolling is then employed to enhance NMF-based methods, leading to competitive deep architectures such as PAD-NMF and Deep-NMFD. These approaches overcome limitations of traditional NMF, including slow convergence and limited adaptability, while preserving interpretability. We further analyze how temporal dependencies are incorporated through different mechanisms, highlighting complementary strengths across practical scenarios.

The second part of the dissertation addresses state estimation in dynamical systems, a domain traditionally dominated by Kalman filtering. After reviewing the Kalman filter, ensemble Kalman methods, and particle filters, we frame state estimation as a general data assimilation problem and explore learning-enhanced extensions of classical filtering techniques. Conditional diffusion models are presented as a data-driven alternative to particle filters, capable of sampling from complex conditional posteriors without requiring explicit likelihood models. In parallel, KalmanNet and the proposed Deep Kalman Filter are introduced as learned extensions of the Kalman filter that preserve its predictor-corrector structure while overcoming classical assumptions such as linearity and known noise statistics. Both methods exemplify deep unfolding, combining algorithmic structure with learning-based optimization.

Contents

Notation

In the upcoming notation, we consider:

$$f : \underset{x}{\mathbb{R}^n} \longrightarrow \underset{f(x)}{\mathbb{R}^k} \qquad g : \underset{(x,y)}{\mathbb{R}^n \times \mathbb{R}^m} \longrightarrow \underset{g(x,y)}{\mathbb{R}} \qquad h : \underset{M}{\mathbb{R}^{n\times m}} \longrightarrow \underset{h(M)}{\mathbb{R}}$$

$\mathcal{M}_{m\times n}(X)$	Set of $m \times n$ matrices with elements in X
$\mathbb{1}_n$	$n \times n$ identity matrix
$\mathbb{1}_{m\times n}$	$m \times n$ ones matrix
$\mathbb{0}_{m\times n}$	$m \times n$ zeroes matrix
δ_{ij} or $\delta_{i,j}$	Kronecker delta
M_{ij} or $M_{i,j}$	Element (i, j) of M
$M_{:,j}$	j-th column of M
$M_{i,:}$	i-th row of M
$\mathcal{C}(X)$	Set of continuous functions with domain X
$\mathcal{C}^k(X)$	Set of k-times differentiable and continuous functions with domain X
$\nabla_x f(x)$	For $k = 1$: $n \times 1$ gradient vector of f For $k > 1$: $k \times n$ Jacobian matrix of f
$\nabla^2_x f(x)$	For $k = 1$: $n \times n$ Hessian of f
$\nabla^2_{xy} g(x, y)$	$m \times n$ Hessian block of g
$\nabla^2_{yx} g(x, y)$	$n \times m$ Hessian block of g
$\nabla_M h(M)$	$n \times m$ reshaped gradient of h
$\mathring{X}$	Interior of X
$B_\delta(x)$	Ball of radius δ centered at x
$Y \sim p_Y$	Random variable Y with distribution p_Y

Introduction

The development of methods for inference, optimization, and dynamical system analysis has historically been shaped by two methodological traditions that were long regarded as conceptually distinct. Model-based approaches rely on explicit mathematical formulations derived from physical principles, statistical assumptions, or expert knowledge, and emphasize interpretability, stability, and theoretical guarantees [15, 85]. In contrast, data-driven approaches, and in particular modern deep learning, prioritize expressive parametric models trained end-to-end from data, often achieving remarkable empirical performance but at the expense of transparency and structural insight [78, 53]. As real-world systems grow in complexity and data availability continues to increase, the limitations of treating these paradigms as mutually exclusive have become increasingly apparent. This has motivated a growing body of research aimed at bridging the gap between model-based and learning-based methods [110, 23, 26, 51, 27], giving rise to hybrid approaches that seek to combine the structure and reliability of classical algorithms with the flexibility and adaptivity of data-driven learning.

Within this evolving landscape, *Deep Unfolding* [57, 96] has emerged as a powerful and unifying conceptual framework. The key observation underlying deep unfolding is that many algorithms used across signal processing, optimization and control are inherently iterative, defined by the repeated application of an update rule that progressively refines an estimate or state. Classical examples include gradient-based optimization methods [98], proximal algorithms [104], matrix factorization schemes [80], and recursive estimators for dynamical systems [72]. Deep unfolding reinterprets such iterative procedures as layered computational graphs, where each iteration corresponds to a layer of a deep network. Parameters that were traditionally fixed or manually tuned, such as step sizes, thresholds, gains, or regularization weights, are instead treated as learnable quantities and optimized from data using gradient-based techniques [57, 64]. This reinterpretation preserves the inductive biases, constraints, and interpretability of the original algorithm while enabling data-driven adaptation and performance improvements. A central theme of this work is that deep unfolding is not a single technique but rather a general paradigm that manifests in different forms depending on the role played by the underlying iterative process. On the one hand, algorithm unrolling [57, 96] provides an architectural design principle, in which the structure and depth of a neural network are explicitly derived from a known algorithm. The learning process is then focused on optimizing the algorithmic parameters so that a truncated number of iterations yields high-quality solutions, often with emphasis on convergence behavior or task-specific objectives. On the other hand, backpropagation through time [132, 112] offers an optimization mechanism for models with temporal or recursive structure, such as recurrent or auto-regressive neural networks. In this case, the unfolding occurs implicitly during training, as the recurrent computation is expanded over

time to enable gradient propagation. Although these two perspectives are often presented separately in the literature, this work adopts the viewpoint that they represent complementary expressions of the same underlying idea: iterative evolution, whether algorithmic or dynamical, can be represented as a deep computation and optimized end-to-end. From this standpoint, the distinction between model-based and learning-based approaches becomes a matter of how much structure is imposed a priori, rather than a fundamental conceptual divide.

Framing deep unfolding in this way has important implications for optimization and learning. In particular, it naturally leads to bilevel optimization formulations [55, 99, 14, 86, 123], where the objective of interest depends on the solution of an inner problem defined implicitly through an iterative procedure. Such structures arise when learning algorithmic parameters, hyperparameters, or surrogate models whose performance is evaluated through the outcome of an optimization or inference process. Bilevel optimization provides a rigorous mathematical framework for these problems, but also introduces significant computational and analytical challenges, especially when the inner solution cannot be computed exactly or in closed form [122]. Iterative differentiation techniques [115] address this difficulty by replacing the inner problem with a truncated iterative scheme and differentiating through it explicitly, a strategy that closely mirrors algorithm unrolling. This connection highlights how deep unfolding can serve not only as a modeling paradigm but also as a practical tool for scalable and interpretable learning in complex optimization settings.

The relevance of deep unfolding is further illustrated through its application to nonnegative matrix factorization (NMF), a classical unsupervised learning technique grounded in constrained optimization [79]. NMF decomposes nonnegative data into additive components that are often highly interpretable, making it particularly attractive in domains such as audio processing [118, 134], document analysis [136], and bioinformatics [16]. However, classical NMF formulations are limited by their unsupervised nature and by the rigidity of their cost functions and update rules. Deep unfolded variants address these limitations by mapping iterative NMF updates onto multilayer architectures and training them discriminatively [64, 22], thereby guiding the factorization toward task-specific solutions while retaining nonnegativity constraints and interpretability. These models exemplify how deep unfolding enables the integration of domain knowledge and learning, producing architectures that are both structured and adaptable.

A complementary and equally important application domain considered in this work is data assimilation [40, 109], which concerns the estimation of latent states of dynamical systems from noisy and incomplete observations. Data assimilation lies at the intersection of modeling and learning, as it explicitly combines predictions derived from a dynamical model with observational data in a probabilistic framework. The Kalman filter [72] and its many extensions [39, 70] constitute a cornerstone of this field, offering recursive estimation schemes with well-understood optimality properties under specific assumptions. From the perspective adopted in this work, Kalman filtering can be viewed as an iterative inference algorithm whose prediction and update steps define a structured computational graph. This interpretation opens the door to learning-enhanced variants in which components of the filter, such as dynamics, observation models, or gain matrices, are adapted from data while preserving the overall recursive structure. Recent developments in learning-enhanced data assimilation, including neural and hybrid Kalman-based

architectures [110, 20, 23] as well as diffusion-based models [30, 121, 120], exemplify the broader trend toward blurring the line between model-based inference and data-driven learning. These approaches leverage neural networks to compensate for model mismatch, unknown noise statistics, or incomplete physical knowledge, while still relying on the algorithmic backbone of classical filters. Viewed through the lens of deep unfolding, they represent a natural continuation of the same paradigm: iterative inference procedures are unfolded into trainable architectures that combine structure, interpretability, and learning capacity.
Overall, this work advocates deep unfolding as a unifying framework for understanding and designing hybrid models that merge classical algorithms with modern learning techniques. By consistently interpreting optimization methods, matrix factorizations, and state estimation algorithms as unfolded computational graphs, it demonstrates how architectural design, learning, and inference can be integrated in principled manner across diverse application domains.

The structure of the thesis is as follows. Chapters 1 and 2 lay the mathematical and conceptual foundations of deep unfolding. In particular, Chapter 1 introduces algorithm unrolling and backpropagation through time as two complementary manifestations of the unfolding paradigm, while Chapter 2 elucidates the connection between unrolled architectures and iterative differentiation methods for bilevel optimization. Chapters 3 and 4 then focus on nonnegative matrix factorization, presenting the classical formulation and its variants before exploring deep unfolded architectures for audio analysis tasks such as speech enhancement, source separation, and event detection. Here, the original contributions of the thesis include the PAD-NMF [22] and Deep-NMFD [24] architectures, as well as the ARD-NMFD optimization algorithm. Finally, Chapters 5 and 6 review the Kalman filter and selected generalizations, and examine recent learning-based extensions within the broader context of learning-enhanced data assimilation. Within this application domain, the original contributions include an audio tracking algorithm based on particle filters and the Deep Kalman Filter [21, 23].

Deep Unfolding

This chapter introduces deep unfolding as a unifying framework bridging classical model-based algorithms and data-driven learning. We show how algorithm unrolling and back-propagation through time can be interpreted as complementary unfolding paradigms, corresponding respectively to architectural design and optimization mechanisms.

1.1 Introduction

In recent years, the boundaries between classical model-based algorithms and data-driven learning have become increasingly blurred. A prominent example of this convergence is the paradigm known as *Deep Unfolding*, which provides a principled framework for integrating domain knowledge, iterative algorithms, and modern deep learning techniques. Within this perspective, both *Algorithm Unrolling* and *Backpropagation Through Time* (BPPT) can be understood as complementary manifestations of the same underlying idea: representing iterative or dynamical processes as deep computational graphs, and optimizing them using gradient-based methods. In this view, algorithm unrolling primarily serves as an *architectural design paradigm*, while BPTT functions as an *optimization mechanism* for training models with temporal or iterative structure.

The concept of algorithm unrolling originates form the observation that many signal processing, control, and inference problems are traditionally solved using iterative algorithms with well-defined update rules. Examples include gradient descent, proximal methods, Kalman filtering, and message-passing algorithms. By mapping each iteration of such an algorithm to a layer in a neural network, one obtains a structured deep architecture whose forward pass of the original algorithm. Crucially, parameters that are fixed or heuristically chosen in classical formulations, such as step sizes, thresholds, or gains, can be made learnable and optimized form data. This idea has led to influential works such as LISTA, which unrolls the iterative shrinkage-thresholding algorithm into a trainable network for sparse coding, achieving significantly faster convergence with fewer iterations than its classical counterpart [57]. Section 1.2.1, in particular, provides a general introduction to algorithm unrolling and briefly presents LISTA as a paradigmatic example. Since then, algorithm unrolling has been successfully applied across a wide range of domains, including inverse problems, communications, control, and state estimation, offering improved performance, interpretability, and data efficiency compared to generic black-box models [96].

While algorithm unrolling focuses on the explicit design of network architectures inspired by known algorithms, recurrent neural networks (RNNs) [78, 65] and auto-regressive neu-

ral networks (ARNNs) [128, 126] address a related challenge from a different angle: modeling sequential and dynamical data. RNNs and ARNNs define a recursive relationship between (hidden) states across time, enabling the representation of temporal dependencies through shared parameters. Training such models requires accounting for the influence of parameters across multiple time steps, which is achieved through backpropagation through time. As we will see in Section 1.2.2, BPTT conceptually unfolds the recurrent computation over time, transforming it into a deep feedforward graph to which standard backpropagation can be applied [132, 112]. Although this unfolding is often viewed as a purely computational step for gradient calculations, it implicitly reveals that RNNs and ARNNs correspond to deep networks whose depths are determined by the sequence length. From the standpoint of Deep Unfolding, BPTT can therefore be interpreted as applying the same unrolling principle, not to a hand-crafted algorithm, but to a learned dynamical system. In contrast to algorithm unrolling, where the depth and structure are fixed by design, the unfolded depth in BPTT is data-dependent. Nonetheless, both approaches rely on the same fundamental mechanism: representing iterative evolution as a layered computation and optimizing it end-to-end via gradient descent. This unifying interpretation highlights that the distinction between "model-based" and "learning-based" approaches is often one of degree rather than kind.
Recognizing algorithm unrolling and BPTT as complementary aspects of deep unfolding has important practical and conceptual implications. It provides a common language for hybrid models that embed learned components within principled algorithmic structures such as learned Kalman filters, unrolled optimization networks, and physics-informed recurrent architectures. Moreover, it clarifies how prior knowledge and interpretability can be preserved without sacrificing the expressive power and adaptability of deep learning. As highlighted in the introduction to this work, this perspective serves as a guiding framework for developing models that combine the stability and structure of classical algorithms with the flexibility of neural networks, while relying on deep unfolding as the unifying design and training principle.

1.2 Unfolding paradigms

The purpose of this section is to briefly introduce the mathematical frameworks underlying algorithm unrolling and backpropagation through time. This discussion provides the foundation for Chapter 2, where algorithm unrolling is shown to yield a natural and efficient approach to bilevel optimization through iterative differentiation (ITD) methods. In a similar vein, backpropagation through time offers a valuable perspective for analyzing and comparing several of the algorithms developed in the subsequent chapters.

1.2.1 Algorithm unrolling

Algorithm unrolling can be viewed as an architectural design paradigm that enables the construction of deep networks from iterative algorithms. Consider a θ-parametric update map $f_\theta : \Omega_X \to \Omega_X$ acting on a state $x \in \Omega_X$. Many iterative algorithms explore the search space Ω_X through updates of the form:

$$x_k = f_\theta(x_{k-1}) \qquad \forall k \geq 1 \tag{1.2.1}$$

with the objective of converging to a desired optimal state. A broad and widely used instance of this paradigm is gradient descent, where the map f_θ encodes the direction of steepest descent of a given objective function. In this setting, θ typically corresponds to the step size, which is commonly treated as a hyperparameter governing the convergence speed. Through the recursive updates in (1.2.1), the map f_θ fully specifies the optimization procedure, while the sequence $\{x_k\}_{k\geq 0}$ represents the trajectory generated from a given initial condition x_0. It is important to emphasize that, within this classical optimization perspective, the role of f_θ is purely instrumental: it serves as the mechanism by which the optimization trajectory $\{x_k\}_{k\geq 0}$ is guided towards the desired optimal state, and is therefore intended to be applied repeatedly. Algorithm unrolling, as schematized in Figure 1.2.1, arises from a similar shift in viewpoint, one that places emphasis on the *algorithm* induced by f_θ, rather than on the update map in isolation.

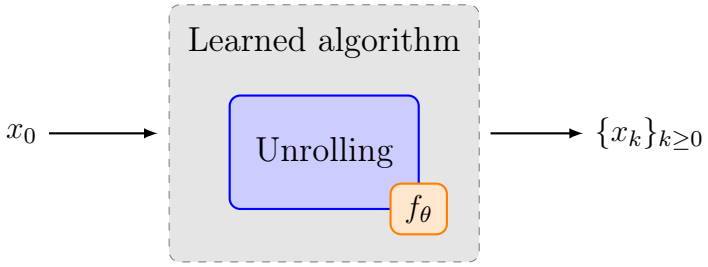


Figure 1.2.1: General algorithm unrolling paradigm. The algorithm defined by the update map f_θ is optimized to produce output sequences with desired properties.

Under this reinterpretation, the iterative scheme in (1.2.1) is viewed as a deep architecture in which each iteration corresponds to a layer, and the iterate x_k plays the role of the hidden state at the k-th layer. The map f_θ is then naturally interpreted as the (nonlinear) transformation that propagates the state from one layer to the next, akin to activations in deep neural networks. By truncating the number of iterations to a fixed integer $K \geq 1$, this construction yields a finite-depth network in which θ becomes a set of learnable parameters. A schematic illustration of such an unrolled architecture is provided in Figure 1.2.2.

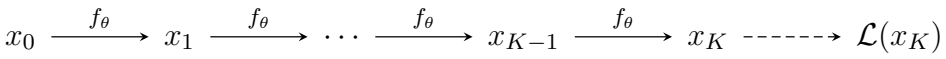


Figure 1.2.2: Unrolled iterations (1.2.1) into a K-layer network trainable end-to-end with the outer objective $\mathcal{L}$ as a loss function.

The resulting network can be trained end-to-end using an appropriate loss function $\mathcal{L}$.

Learned-ISTA

As a design paradigm, algorithm unrolling was first introduced in [57], a seminal work that reformulated the Iterative Shrinkage-Thresholding Algorithm (ISTA) within a machine learning framework. ISTA specifies an iterative update rule for computing a sparse representation $x \in \mathbb{R}^n$ of a given observation $y \in \mathbb{R}^m$, using a fixed dictionary $D \in \mathcal{M}_{m\times n}(\mathbb{R})$. Starting from the initialization $x_0 = 0$, the ISTA iterations take the form:

$$x_k = \mathcal{S}_{\frac{\alpha}{L}}\left(x_{k-1} - \frac{1}{L}D^\top\left(Dx_{k-1} - y\right)\right) \tag{1.2.2}$$

where $\alpha > 0$, $L > 0$ is an upper bound on the largest eigenvalue of $D^\top D$, and $\mathcal{S}_\beta$ is the soft-thresholding operator, defined as:

$$\mathcal{S}_\beta(x) = \begin{cases} x - \beta & \text{if} \quad x > \beta \\ 0 & \text{if} \quad x \in [-\beta, \beta] \\ x + \beta & \text{if} \quad x < -\beta \end{cases} \tag{1.2.3}$$

To address one of ISTA's most well-known limitations, namely its slow convergence, the algorithm unrolling paradigm was introduced. By unrolling the ISTA iterations into a fixed-depth network, this approach enabled the learning of an enhanced variant of the algorithm, known as Learned ISTA (LISTA). For a prescribed (small) depth K, LISTA is trained to approximate the optimal sparse representation within only K iterations, thereby significantly outperforming the classical ISTA algorithm in terms of convergence speed and practical performance.
The unrolled network was obtained by considering the following update map:

$$f_\theta(x) = \mathcal{S}_{\theta_1}\left(\theta_2 y + \theta_3 x\right) \tag{1.2.4}$$

where $\theta = \{\theta_i\}_{i=1}^3$, with θ_1, θ_2, and θ_3 serving as learnable surrogates for $\frac{\alpha}{L}$, $\frac{1}{L}D^\top$, and $\mathbb{1}_n - \frac{1}{L}D^\top D$, respectively[1]. Moreover, for a training pair (y, x_y), where x_y is the optimal sparse representation of y provided by ISTA, the network is trained using the loss:

$$\mathcal{L}(x_K) = \frac{1}{2}\|x_K - x_y\|_2^2 \tag{1.2.5}$$

1.2.2 Backpropagation through time

Backpropagation through time was originally introduced in [132] as an optimization mechanism for training models endowed with temporal or iterative structure. Consider again a θ-parametric map $f_\theta : \Omega_X \to \Omega_X$; unlike in the previous section, we now assume that this map is explicitly designed to propagate a discrete-time state $x(k-1)$ forward in time, namely:

$$x(k) = f_\theta(x(k-1)) \qquad \forall k \geq 1 \tag{1.2.6}$$

Such update maps naturally arise in the discretization of dynamical systems (see Section 5.2), where they often encode essential information about the underlying physical laws governing state evolution. In this setting, all states are treated on equal footing, and the estimate produced by f_θ at each time step is required to be as accurate as possible. This perspective stands in clear contrast to that of algorithm unrolling. In the latter, primary emphasis is placed on the states *at convergence*, leading to the optimization of the

[1]Compare (1.2.4) to (1.2.2).

algorithm as a whole rather than of individual state updates. In the context of backpropagation through time, the focus instead returns to the update map itself, which is optimized directly with respect to the state estimation problem across time. A schematization of BPTT is shown in Figure 1.2.3.

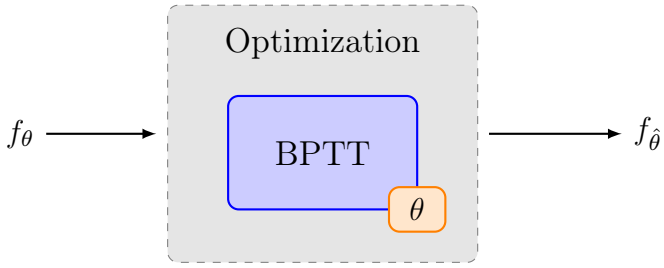


Figure 1.2.3: General backpropagation through time paradigm. The parameters θ entering the definition of the update map f_θ are optimized to a value $\hat{\theta}$ by learning time-dependencies between states.

Assume that a sequence of target states $\{x(k)\}_{k\geq 0}$ is given. If the objective is to optimize θ with respect to the state estimation at a single time step k, a natural and straightforward choice of loss function is:

$$\mathcal{L}_k(\theta) = \frac{1}{2} \left\| f_\theta(x(k-1)) - x(k) \right\|_2^2 . \tag{1.2.7}$$

However, the loss in (1.2.7) is inherently myopic, as it fails to account for long-term dependencies in the evolution of the state sequence. In many dynamical systems, the state $x(k)$ may depend strongly on earlier states $x(k-j)$ for $j \geq 1$. In such cases, restricting the optimization to a single-step prediction can prevent the update map from capturing the underlying dynamics of the system. Backpropagation through time is specifically designed to overcome this limitation by incorporating the influence of parameters across multiple time steps during optimization. By choosing a time-horizon $T \leq k$ and defining the predictions:

$$\begin{cases} \hat{x}(k-T+1) &= f_\theta(x(k-T)) \\ \hat{x}(k-j) &= f_\theta(\hat{x}(k-j-1)) \quad \forall j = 0, \dots, T-2 \end{cases} \tag{1.2.8}$$

we can define the more suitable loss:

$$\mathcal{L}_k(\theta) = \frac{1}{2T} \sum_{j=0}^{T-1} \left\| \hat{x}(k-j) - x(k-j) \right\|_2^2 \tag{1.2.9}$$

Unlike (1.2.7), the K-step prediction loss in (1.2.9) produces parameter updates that explicitly account for temporal dependencies, thereby yielding an update map f_θ capable of providing more informed state estimates at time k. Extending this reasoning to all time steps $k \geq T$, we can either consider a loss function optimizing all possible sub-trajectories of length T:

$$\mathcal{L}(\theta) = \sum_{k \geq T} \mathcal{L}_k(\theta) \tag{1.2.10}$$

or split the optimization into *non-overlapping* sub-trajectories:

$$\mathcal{L}(\theta) = \sum_{j \geq 1} \mathcal{L}_{jT}(\theta) \tag{1.2.11}$$

Among the most widely used black-box architectures trained via backpropagation through time are auto-regressive neural networks (ARNNs) and recurrent neural networks (RNNs). The latter can be viewed as a generalization of the former, with the primary distinction lying in how memory is represented and propagated across time steps. In particular, ARNNs update the state exactly according to (1.2.6). The corresponding unrolled computation graph induced by BPTT is illustrated in Figure 1.2.4.

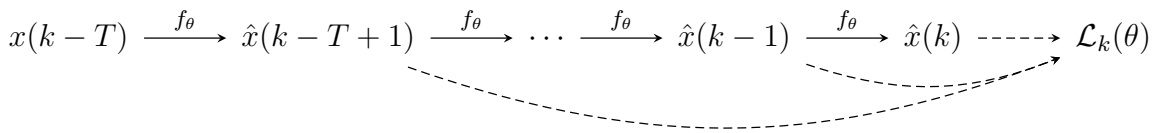


Figure 1.2.4: Unrolled computation graph generated by BPTT for an ARNN.

Note in particular the similarity with the standard unrolled architecture shown in Figure 1.2.2. Recurrent neural networks, on the other hand, interpret $x(k)$ as a hidden state and explicitly introduce input signals $u(k)$ as well as output variables $y(k)$, with the latter typically constituting the primary quantity of interest in the optimization process. In addition to the state-transition map, the observation map also contains learnable parameters. A general RNN architecture can therefore be written in the form:

$$\begin{aligned} x(k) &= f_{\theta_1}(x(k-1), u(k)) \\ y(k) &= g_{\theta_2}(x(k)) \end{aligned} \tag{1.2.12}$$

The unrolled computation graph induced by BPTT is shown in Figure 1.2.5.

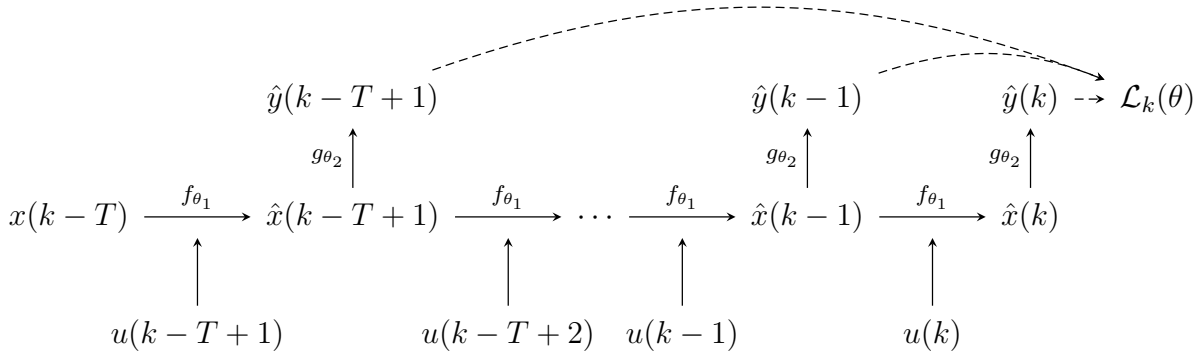


Figure 1.2.5: Unrolled computation graph generated by BPTT for an RNN.

Bilevel optimization

This chapter introduces the bilevel optimization framework and reviews several optimization methodologies proposed in the literature. Special emphasis is placed on bilevel approaches that provide a flexible and rigorous framework for linking unrolled deep networks with more interpretable, model-based paradigms.

2.1 Introduction

Several machine learning problems can be naturally posed as bilevel problems, where the optimal solution of an *inner* (or lower) objective enters the formulation of an *outer* (or upper) problem. Paradigms of this framework, which will be presented in Section 2.2, span a wide variety of topics, including meta-learning [3, 42], recurrent neural networks [78, 106], and, most notably, hyperparameter optimization [88, 86, 44]. With respect to the latter, in particular, popular strategies involve grid and random searches [12], discrepancy methods [43, 87], and L curves [10]. Oftentimes, these approaches are either unsuitable for high-dimensional parameter spaces or suffer from a restricted range of applicability, confining their relevance to specific classes of problems. Gradient-based methods, on the other hand, scale well with problem size and are quite general, thus providing a convincing alternative to the aforementioned approaches. However, in a bilevel optimization setting, computing the gradient of the outer problem, often called *hypergradient*, can be challenging. The root of the problem complexity lies in the dependence of the outer problem on the inner objective optimal solution: in many instances, such a solution either does not admit a closed-form or is simply too costly to compute. Moreover, in the most general setting, the inner solution itself depends on the parameters of the outer problem, adding an extra layer of complexity when dealing with the outer optimization.
Bilevel optimization problems are usually tackled by one of the general strategies briefly described hereafter, each with its strengths and shortcomings: in the following Section 2.3 we will delve more into the details of a few paradigmatic approaches present in the literature, which will serve as an introduction to the main ideas needed to describe and motivate some of the frameworks employed in the later chapters. The first strategy revolves around exploiting the Implicit Function Theorem 2.A.1 to derive a formula for the hypergradient. These methods, based on *Implicit Differentiation* (ID) or *Approximate Implicit Differentiation* (AID) [123, 14, 55], offer a straightforward solution to the bilevel optimization problem, but often require many regularity assumptions to guarantee the applicability of the implicit function theorem and to derive convergence results. Moreover, in order to overcome the need to explicitly compute the inner optimum, AID methods allow one to

relax the optimality requirement but, in turn, need stronger assumptions on the maps involved to control the approximation error of the hypergradient. The second strategy addresses this critical point by replacing the inner objective with a smooth, convergent iterative scheme, which can be truncated to obtain an arbitrary level of precision on the inner optimum. These methods can then exploit *Iterative Differentiation* (ITD) [115, 45] paired with either forward- or reverse-mode automatic differentiation (backpropagation) to compute an approximation of the hypergradient. The ITD approach is not dissimilar to the algorithm unrolling paradigm presented in Section 1.2.1. Remarkably, unlike most AID methods, ITD does not require assumptions on the regularity of the inner objective, as long as a smooth iterative scheme for it can be derived. However, compared to AID, ITD methods are usually more computationally taxing in terms of memory, since all intermediate iterations need to be stored in order to (back)propagate the gradients and assemble the approximate hypergradient.
In later chapters, we will employ ITD-like approaches together with the Deep Unfolding presented in Chapter 1 to tackle bilevel optimization problems. In particular, Section 2.4 explores this idea, which was first introduced in [64], and provides the general notation and backpropagation scheme that will be adapted to specific problem instances later on.

2.2 Inner and outer optimization

In this section, we present the bilevel optimization framework in its most general form. We postpone the discussion on the regularity assumptions required to make the overall problem well-posed in the following Section 2.3, where we will explore different solution approaches. We warn the reader that, in the literature, the notation for this type of framework is not universally agreed upon: the one we chose for this work is general enough to encompass many variants but is ultimately catered towards the specific needs of the later chapters.
We formulate the *inner* optimization problem as the minimization of an objective function $\mathcal{F}^{in} : \Omega_X \times \Omega_\Theta \times \Omega_Y \to \mathbb{R}$ with respect to the $x \in \Omega_X$ variables, namely:

$$\begin{aligned} \min \quad & \mathcal{F}^{in}(x, \theta, y) \\ \text{s.t.} \quad & x \in \Omega_X \end{aligned} \tag{2.2.1}$$

In this general context x is the quantity of interest (or state) to be optimized, while $y \in \Omega_Y$ is a given observation and $\theta \in \Omega_\Theta$ is a fixed parameter. In most cases θ encodes some unknown quantity in the mathematical model of a system that leads to the optimization of (2.2.1). Likewise, y is often thought of as an observation or measurement taken directly from the system and intrinsically related to the optimal solutions of (2.2.1). Examples of this framework include Bayesian inference [19, 109] and data assimilation [41, 40], to cite a few. In the following, we may highlight the dependence on θ of the inner objective by writing $\mathcal{F}_\theta^{in}$. A minimizer of (2.2.1) will be denoted by:

$$\hat{x}(\theta; y) \in \operatorname*{argmin}_{x \in \Omega_X} \mathcal{F}^{in}(x, \theta, y) \tag{2.2.2}$$

where, again, we emphasize the dependence on θ (and y).
Likewise, the *outer* problem is formulated as the minimization of an objective function

$\mathcal{F}^{out} : \Omega_X \times \Omega_\Theta \times \Omega_Z \to \mathbb{R}$ with respect to the parameters $\theta \in \Omega_\Theta$, where the minimizer $\hat{x}(\theta; y)$ enters into the variables x:

$$\begin{aligned} \min \quad & \mathcal{F}^{out}(\hat{x}(\theta; y), \theta, z) \\ \text{s.t.} \quad & \theta \in \Omega_\Theta \end{aligned} \tag{2.2.3}$$

Similarly to y in the inner problem, the variables $z \in \Omega_Z$ are observations or measurements related to the optimal solutions of (2.2.3). In particular, we remark that the nature of y and z can be deterministic [54] as well as stochastic [55]: in the latter, y (resp. z) is a realization of a random variable $\mathcal{Y}$ (resp. $\mathcal{Z}$) with values in Ω_Y (resp. Ω_Z). The distinction between y and z is typical, for example, in hyperparameter tuning involving neural networks: while the model parameters x are learned using a training dataset that samples some random variable $\mathcal{Y}$, the hyperparameters θ are tuned using a validation dataset sampling a different random variable $\mathcal{Z}$ [84, 86]. Of course, one may use the same observation variables for both the inner and outer problem (letting $y = z$ in (2.2.3)), as will be the case for most applications described in later chapters. In some formulations, the inner and outer problem optimize over multiple realizations of the random variables $\mathcal{Y}$ and $\mathcal{Z}$. In most cases this is done simply by considering the expected value of some intermediate maps $\bar{\mathcal{F}}^{in}$ and $\bar{\mathcal{F}}^{out}$:

$$\mathcal{F}^{in}(x, \theta, \mathcal{Y}) = \mathbb{E}_{\mathcal{Y}}\left[\bar{\mathcal{F}}^{in}(x, \theta, \mathcal{Y})\right] \qquad \mathcal{F}^{out}(x, \theta, \mathcal{Z}) = \mathbb{E}_{\mathcal{Z}}\left[\bar{\mathcal{F}}^{out}(x, \theta, \mathcal{Z})\right] \tag{2.2.4}$$

A minimizer of (2.2.3) will be denoted by:

$$\hat{\theta}(y, z) \in \operatorname*{argmin}_{\theta \in \Omega_\Theta} \mathcal{F}^{out}(\hat{x}(\theta; y), \theta, z) \tag{2.2.5}$$

For notational convenience and clarity when computing the hypergradient, especially within bilevel formulations that do not expect observations, we may drop the dependence on y, z (or $\mathcal{Y}, \mathcal{Z}$) and denote:

$$\hat{\mathcal{F}}(\theta) = \hat{\mathcal{F}}(\theta, y, z) = \mathcal{F}^{out}(\hat{x}(\theta; y), \theta, z) \tag{2.2.6}$$

A similar treatment can also be adopted for $\hat{x}(\theta) = \hat{x}(\theta; y)$, $\hat{\theta} = \hat{\theta}(y, z)$ and $\mathcal{F}^{in}(x, \theta) = \mathcal{F}^{in}(x, \theta, y)$, $\mathcal{F}^{out}(x, \theta) = \mathcal{F}^{out}(x, \theta, z)$. In total generality, we stress that minimizers of (2.2.1) and (2.2.3) may not even exist, and if they do, may not be finite or unique: as already mentioned, the well-posedness will be a matter of later discussions.

From a modeling standpoint, bilevel optimization can be quite flexible, allowing one to shift the focus towards either the inner or outer problem depending on the specific task. In fact, in many cases, the main optimization target is the minimizer $\hat{x}(\theta; y)$ of the inner problem, and the outer parameters θ are used to impose additional a priori properties, such as sparsity, smoothness, orthogonality, proximity to prescribed targets, etc. Here, θ resemble degrees of freedom that can be exploited to explore the manifold of optimal solutions in search of one that satisfies the prior. Examples include [122, 57, 64], where learnable dictionaries are optimized in a supervised setting to guide the inner minimizer closer to a desired representation. These works are discussed in a more detailed way in Sections 2.3.1, 1.2.1 and 4.2. In other scenarios, the inner and outer parameters x and θ are placed at similar hierarchical levels, resulting in frameworks in which both minimizers

are required to fully describe the final optimized model. A paradigmatic example can be found in [84], where the inner and outer parameters encode, respectively, the weights and architecture of a neural network. At the other end of the spectrum, some approaches may even concentrate entirely on recovering the optimal parameter $\hat{\theta}(y, z)$ of the outer problem. In this case, $\hat{x}(\theta; y)$ is just a constraint on the outer objective arising from other modeling requirements and is ultimately discarded.

2.3 Different approaches

Despite its seemingly simple formulation, bilevel optimization has become a central topic in mathematical optimization in recent years, as reflected by the growing number of publications proposing new frameworks and refinements of existing solution approaches. A comprehensive review of all these contributions is beyond the scope of this work. Instead, this section introduces a selection of representative frameworks that provide an overview of the main mathematical ideas developed to address bilevel optimization. First, we begin by considering a specific bilevel optimization instance where all minimizers admit closed-forms. This will serve as an ideal reference case for the subsequent general analysis, where the explicit forms of the minimizers may be unknown. Each of the following subsections, on the other hand, presents a slight variation of the general framework introduced previously, progressively leading to the main formulation discussed in Section 2.4, which forms the foundation of the architectures developed in Chapter 4. Throughout this section, the focus gradually shifts from formulations based on strict regularity assumptions and exact analytical relationships to more flexible approaches that allow controlled approximations and incorporate machine learning techniques.

2.3.1 Preliminar toy problem

Let us consider the generalization of a Lasso-type pursuit problem described in [122]. Given some matrices $M_1 \in \mathcal{M}_{m\times k}(\mathbb{R})$, $M_2 \in \mathcal{M}_{m\times n}(\mathbb{R})$, $M_3 \in \mathcal{M}_{r\times n}(\mathbb{R})$ and an observation $y \in \mathbb{R}^k$, the problem is formulated as the following unconstrained optimization:

$$\begin{aligned} \min \quad & \frac{1}{2}\|M_1 y - M_2 x_1\|_2^2 + \lambda_1 \|M_3 x_1\|_1 + \frac{\lambda_2}{2}\|x_1\|_2^2 \\ \text{s.t.} \quad & x_1 \in \mathbb{R}^n \end{aligned} \tag{2.3.1}$$

where λ_2, $\lambda_2 > 0$ are regularization parameters. This general framework encompasses many particular cases. For example, letting $k = m < r = n$, $M_1 = \mathbb{1}_k$, $M_3 = \mathbb{1}_n$ yields a sparse *synthesis* model. Conversely, letting $m = k = n < r$, $M_1 = M_2 = \mathbb{1}_n$ yields a sparse *analysis* model.
Notice that, for a general M_3, the map $x \mapsto \|M_3 x\|_1$ does not admit a closed-form proximal operator. As a consequence, the authors propose to introduce an auxiliary variable $x_2 = M_3 x_1$ and to relax such a constraint by employing a penalization term. For $t > 0$, the resulting optimization problem becomes:

$$\begin{aligned} \min \quad & \frac{1}{2}\|M_1 y - M_2 x_1\|_2^2 + \frac{t}{2}\|M_3 x_1 - x_2\|_2^2 + \lambda_1 \|x_2\|_1 + \frac{\lambda_2}{2}\|x_1\|_2^2 \\ \text{s.t.} \quad & (x_1, x_2) \in \mathbb{R}^{n+r} \end{aligned} \tag{2.3.2}$$

Given the strong convexity of the objective, problem (2.3.2) admits a unique minimizer $\hat{x}(\theta; y, t) = (\hat{x}_1(\theta; y, t), \hat{x}_2(\theta; y, t))$, which depends on the given collection of matrices $\theta = (M_1, M_2, M_3)$ defining the original formulation (2.3.1), as well as the constraint hyperparameter t. In particular, letting $t \to +\infty$ allows one to recover the optimal solution $\hat{x}_1(\theta; y) = \lim_{t\to+\infty} \hat{x}_1(\theta; y, t)$ of (2.3.1), where $\hat{x}_2(\theta; y) = \lim_{t\to+\infty} \hat{x}_2(\theta; y, t) = M_3\hat{x}_1(\theta; y)$. In [122] the matrices within θ are then treated as tunable parameters to optimize a problem-specific objective acting on $\hat{x}_1(\theta; y)$. More precisely, given an observation $z \in \mathbb{R}^p$, they consider the general problem:

$$\begin{aligned} \min \quad & l(\hat{x}_1(\theta; y), z) \\ \text{s.t.} \quad & \theta \in \mathcal{M}_{m\times k}(\mathbb{R}) \times \mathcal{M}_{m\times n}(\mathbb{R}) \times \mathcal{M}_{r\times n}(\mathbb{R}) \end{aligned} \tag{2.3.3}$$

As an example, one could have $p = n$ and let $l(x_1, z) = \frac{1}{2}\|x_1 - z\|_2^2$ penalize the distance to the optimal observation vector z, or even have $z \in \{-1, 1\}$ play the role of a label and let $l(x_1, z) = \log(1 + e^{-z(w^\top x_1 + b)})$ be the logistic regression loss for binary classification. In particular, we remark that the problem we just introduced fits perfectly within the bilevel optimization framework presented in Section 2.2. Indeed, in our previous notation, we have $\Omega_X = \mathbb{R}^n$, $\Omega_Y = \mathbb{R}^k$, $\Omega_Z = \mathbb{R}^p$, $\Omega_\Theta = \mathcal{M}_{m\times k}(\mathbb{R}) \times \mathcal{M}_{m\times n}(\mathbb{R}) \times \mathcal{M}_{r\times n}(\mathbb{R})$ and $\mathcal{F}^{in}(x, \theta, y) = \frac{1}{2}\|M_1 y - M_2 x\|_2^2 + \lambda_1\|M_3 x\|_1 + \frac{\lambda_2}{2}\|x\|_2^2$, $\mathcal{F}^{out}(x, \theta, z) = l(x, z)$.
For this specific instance of bilevel optimization, we will now show how to derive a closed-form for the hypergradient. We will hereon drop the dependence on y and z for notational convenience. The general argument relies on the observation that, for a sufficiently smooth outer objective l:

$$\nabla_\theta \hat{\mathcal{F}}(\theta) = \nabla_\theta l(\hat{x}_1(\theta)) = \lim_{t\to+\infty} \nabla_\theta l(\hat{x}_1(\theta; t)) \tag{2.3.4}$$

First order optimality conditions of problem (2.3.2) yield:

$$\begin{aligned} M_2^\top (\, M_2\hat{x}_1(\theta; t) - M_1 y \,) + tM_3^\top (\, M_3\hat{x}_1(\theta; t) - \hat{x}_2(\theta; t) \,) + \lambda_2\hat{x}_1(\theta; t) &= 0 \\ t\,(\, \hat{x}_2(\theta; t) - M_3\hat{x}_1(\theta; t) \,) + \lambda_1\,(\, \mathrm{sign}(\hat{x}_2(\theta; t)) + \alpha \,) &= 0 \end{aligned} \tag{2.3.5}$$

where $\alpha \in \partial_x\|\hat{x}_2(\theta; y)\|_1$. In particular, denoting $\Lambda = \{i \mid \hat{x}_2(\theta; t)_i \neq 0\}$ the support of $\hat{x}_2(\theta; t)$, we have $\alpha_\Lambda = 0$ and $|\alpha_{\Lambda^c}| \leq 1$. Moreover, if $P_\Lambda = \mathrm{diag}(\,|\mathrm{sign}(\hat{x}_2(\theta; t))|\,)$ we notice that $P_\Lambda\hat{x}_2(\theta; t) = \hat{x}_2(\theta; t)$ and $P_\Lambda\mathrm{sign}(\hat{x}_2(\theta; t)) = \mathrm{sign}(\hat{x}_2(\theta; t))$. As a consequence, pre-multiplying the second equation in (2.3.5) by P_Λ we obtain:

$$\hat{x}_2(\theta; t) = P_\Lambda M_3\hat{x}_1(\theta; t) - \frac{\lambda_1}{t}\mathrm{sign}(\hat{x}_2(\theta; t)) \tag{2.3.6}$$

Substituting (2.3.6) into the first equation of (2.3.5) yields:

$$\hat{x}_1(\theta; t) = Q(t)\,\big(\, M_2^\top M_1 y - \lambda_1 M_3^\top \mathrm{sign}(\hat{x}_2(\theta; t))\,\big) \tag{2.3.7}$$

where $Q(t) = \big(tM_3^\top P_{\Lambda^c} M_3 + B\big)^{-1} = \big(\,t(B^{-1}M_3^\top P_{\Lambda^c} M_3) + \mathbb{1}_n\,\big)^{-1} B^{-1} = (tC + \mathbb{1}_n)^{-1}B^{-1}$, $P_{\Lambda^c} = \mathbb{1}_r - P_\Lambda$, and $B = M_2^\top M_2 + \lambda_2\mathbb{1}_n$. Owing to the properties of Lasso-type problems, it can be shown [89] that small perturbations of the parameters θ do not influence $\mathrm{sign}(\hat{x}_2(\theta; t))$ (or Λ), thus its partial derivatives with respect to the components of θ will

be zero. Finally, by using the explicit form for the minimizer in (2.3.7), we can compute $\nabla_\theta l(\hat{x}_1(\theta;t))$:

$$\begin{aligned}
\nabla_{M_1} l(\hat{x}_1(\theta;t)) &= M_2\beta(t)y^\top \\
\nabla_{M_2} l(\hat{x}_1(\theta;t)) &= M_1 y\beta(t)^\top - M_2\left(\hat{x}_1(\theta;t)\beta(t)^\top + \beta(t)\hat{x}_1(\theta;t)^\top\right) \\
\nabla_{M_3} l(\hat{x}_1(\theta;t)) &= -\lambda_1 \mathrm{sign}(\hat{x}_2(\theta;t))\beta(t)^\top - P_{\Lambda^c} M_3\left(t\hat{x}_1(\theta;t)\beta(t)^\top + t\beta(t)\hat{x}_1(\theta;t)^\top\right)
\end{aligned} \tag{2.3.8}$$

where $\beta(t) = Q(t)\nabla_x l(\hat{x}_1(\theta;t))$. For more details on the derivation, which led to a correction of the relations reported in [122], see Appendix 2.B. Lastly, letting $C = UDU^{-1}$ be the eigen-decomposition of C with $D = \mathrm{diag}(\{d_i\}_i)$, one can easily show that:

$$\begin{aligned}
Q' &= \lim_{t\to+\infty} Q(t) = UD'U^{-1}B^{-1} \quad \text{with } D' = \mathrm{diag}(\{d'_i\}_i), \quad d'_i = \begin{cases} 0 & \text{if } d_i \neq 0 \\ 1 & \text{if } d_i = 0 \end{cases} \\
Q'' &= \lim_{t\to+\infty} tQ(t) = UD''U^{-1}B^{-1} \quad \text{with } D'' = \mathrm{diag}(\{d''_i\}_i), \quad d''_i = \begin{cases} \frac{1}{d_i} & \text{if } d_i \neq 0 \\ 0 & \text{if } d_i = 0 \end{cases}
\end{aligned} \tag{2.3.9}$$

As a consequence, defining:

$$\begin{aligned}
\hat{x}'_1(\theta) &= Q'\left(M_2^\top M_1 y - \lambda_1 M_3^\top \mathrm{sign}(\hat{x}_2(\theta))\right) \ (= \hat{x}_1(\theta)) \\
\hat{x}''_1(\theta) &= Q''\left(M_2^\top M_1 y - \lambda_1 M_3^\top \mathrm{sign}(\hat{x}_2(\theta))\right) \\
\beta' &= Q'\nabla_x l(\hat{x}_1(\theta)) \\
\beta'' &= Q''\nabla_x l(\hat{x}_1(\theta))
\end{aligned} \tag{2.3.10}$$

we obtain the hypergradient by combining (2.3.4), (2.3.8), (2.3.9) and (2.3.10):

$$\begin{aligned}
\nabla_{M_1} l(\hat{x}_1(\theta)) &= M_2\beta' y^\top \\
\nabla_{M_2} l(\hat{x}_1(\theta)) &= M_1 y\beta'^\top - M_2\left(\hat{x}_1(\theta)\beta'^\top + \beta'\hat{x}_1(\theta)^\top\right) \\
\nabla_{M_3} l(\hat{x}_1(\theta)) &= -\lambda_1 \mathrm{sign}(M_3\hat{x}_1(\theta))\beta'^\top - P_{\Lambda^c} M_3\left(\hat{x}''_1(\theta)\beta'^\top + \beta''\hat{x}_1(\theta)^\top\right)
\end{aligned} \tag{2.3.11}$$

2.3.2 Joint optimization

The first general approach we consider roughly follows the work of [123]. For simplicity, we restrict the analysis to finite-dimensional real vector spaces rather than the general Hilbert spaces examined therein. Similarly, we slightly strengthen certain regularity requirements. This will allow us to avoid invoking functional derivatives and subdifferentials, for the sake of brevity and exposition clarity. In particular, in this work we will always assume $\Omega_X \subseteq \mathbb{R}^n$ and $\Omega_\Theta \subseteq \mathbb{R}^m$.
Le us suppose, as a preliminary step, that $\mathcal{F}^{out} \in \mathcal{C}^1(\Omega_X \times \Omega_\Theta)$ and $\theta \mapsto \hat{x}(\theta) \in \mathcal{C}^1(\Omega_\Theta)$. The hypergradient $\nabla_\theta \hat{\mathcal{F}}(\theta)$ can then be expressed as the sum of a direct and indirect component [86]:

$$\begin{aligned} \nabla_\theta \hat{\mathcal{F}}(\theta) &= \nabla_\theta \mathcal{F}^{\,out}(\hat{x}(\theta), \theta) + \nabla_\theta \hat{x}(\theta)^\top \nabla_x \mathcal{F}^{\,out}(\hat{x}(\theta), \theta) \\ &= \nabla_\theta^{dir} \hat{\mathcal{F}}(\theta) + \nabla_\theta^{ind} \hat{\mathcal{F}}(\theta) \end{aligned} \tag{2.3.12}$$

While the direct gradient measures how θ directly influences the outer objective, the indirect gradient takes into account how the inner minimizer $\hat{x}(\theta)$ changes with respect to the parameters. Computing and assuring the well-posedness of $\nabla_\theta \hat{x}(\theta)$ is at the core of most ID and AID methods. In particular, the well-posedness of the Jacobian can be guaranteed *locally* by the Implicit Function Theorem 2.A.1 applied to the map $\nabla_x \mathcal{F}^{\,in}$. Indeed, if $\mathcal{F}^{\,in} \in \mathcal{C}^1(\Omega_X \times \Omega_\Theta)$ and $\hat{x}(\theta) \in \mathring{\Omega}_X$, by first-order optimality conditions on the inner objective, we have:

$$\nabla_x \mathcal{F}^{\,in}(\hat{x}(\theta), \theta) = 0 \qquad \forall \theta \in \Omega_\Theta \tag{2.3.13}$$

Assuming further $\mathcal{F}^{\,in} \in \mathcal{C}^2(\Omega_X \times \Omega_\Theta)$ and that $\nabla_x^2 \mathcal{F}^{\,in}(\hat{x}(\hat{\theta}), \hat{\theta})$ is invertible, the Implicit Function Theorem assures that for a sufficiently small $\delta > 0$, the map $B_\delta(\hat{\theta}) \ni \theta \mapsto \hat{x}(\theta)$ is uniquely defined and continuously differentiable, with the Jacobian given by:

$$\nabla_\theta \hat{x}(\theta) = -\left(\nabla_x^2 \mathcal{F}^{\,in}(\hat{x}(\theta), \theta)\right)^{-1} \nabla_{\theta x}^2 \mathcal{F}^{\,in}(\hat{x}(\theta), \theta) \tag{2.3.14}$$

The combination of (2.3.12) and (2.3.13) results in an explicit[1] yet local representation of the hypergradient:

$$\nabla_\theta \hat{\mathcal{F}}(\theta) = \nabla_\theta \mathcal{F}^{\,out}(\hat{x}(\theta), \theta) - \nabla_{x\theta}^2 \mathcal{F}^{\,in}(\hat{x}(\theta), \theta) \left(\nabla_x^2 \mathcal{F}^{\,in}(\hat{x}(\theta), \theta)\right)^{-1} \nabla_x \mathcal{F}^{\,out}(\hat{x}(\theta), \theta) \tag{2.3.15}$$

where we have used the relation $\left(\nabla_{\theta x}^2 \mathcal{F}^{\,in}(\hat{x}(\theta), \theta)\right)^\top = \nabla_{x\theta}^2 \mathcal{F}^{\,in}(\hat{x}(\theta), \theta)$ as well as the symmetry of the Hessian. Notice, in particular, that first-order optimality conditions applied to the outer objective imply:

$$\nabla_\theta \hat{\mathcal{F}}(\hat{\theta}) = 0 \tag{2.3.16}$$

The main drawback of (2.3.15) is its local validity, which is directly tied to the neighborhood of $\hat{\theta}$ in which $\nabla_x^2 \mathcal{F}^{\,in}(\hat{x}(\theta), \theta)$ allows an inverse. In order to address this limitation, in [123] the authors require fairly strong yet necessary assumptions on the Hessian block, essentially forcing its non-degeneracy on a parametric neighborhood of $(\hat{x}(\hat{\theta}), \hat{\theta})$. We will explore the precise assumption at the end of this section.
Using an adjoint variable $p = \nabla_\theta \hat{x}(\theta)$ for the Jacobian of the inner minimizer, and letting:

$$H(x, p, \theta) = \begin{bmatrix} \nabla_x \mathcal{F}^{\,in}(x, \theta) & \text{(Inner optimality)} \\ \nabla_x^2 \mathcal{F}^{\,in}(x, \theta) p + \nabla_{\theta x}^2 \mathcal{F}^{\,in}(x, \theta) & \text{(Jacobian adjoint equation)} \\ \nabla_\theta \mathcal{F}^{\,out}(x, \theta) + p^\top \nabla_x \mathcal{F}^{\,out}(x, \theta) & \text{(Outer optimality)} \end{bmatrix} \tag{2.3.17}$$

[1] The representation is explicit in the sense that the Jacobian $\nabla_\theta \hat{x}(\theta)$ is expressed as a function of the inner objective, but the dependence on the exact minimizer $\hat{x}(\theta)$ means the hypergradient is still implicitly defined.

we can derive *joint* inner and outer optimality conditions by combining (2.3.13), (2.3.14), (2.3.15) and (2.3.16). Indeed, any $\bar{\nu} = (\bar{x}, \bar{p}, \bar{\theta})$ satisfying:

$$H(\bar{\nu}) = 0 \tag{2.3.18}$$

will be candidate optimal solutions for the bilevel optimization problem, that is, $(\bar{x}, \bar{p}, \bar{\theta}) = (\hat{x}(\hat{\theta}), \nabla_\theta \hat{x}(\hat{\theta}), \hat{\theta})$. The proposed algorithm to tackle (2.3.18) is a quasi-Newton method that iteratively solves:

$$H_{k+1}(\nu_{k+1}) + M(\nu_{k+1} - \nu_k) = 0 \tag{2.3.19}$$

where $M = \text{diag}(\lambda_x^{-1}\mathbb{1}_{1\times n}, 0_{1\times mn}, \lambda_\theta^{-1}\mathbb{1}_{1\times m})$ is a preconditioning operator that preserves the decoupling of the three sets of variables and:

$$H_{k+1}(\nu_{k+1}) = \begin{bmatrix} \nabla_x \mathcal{F}^{\,in}(x_k, \theta_k) \\ \nabla_x^2 \mathcal{F}^{\,in}(x_{k+1}, \theta_k) p_{k+1} + \nabla_{\theta x}^2 \mathcal{F}^{\,in}(x_{k+1}, \theta_k) \\ \nabla_\theta \mathcal{F}^{\,out}(x_{k+1}, \theta_{k+1}) + p_{k+1}^\top \nabla_x \mathcal{F}^{\,out}(x_{k+1}, \theta_{k+1}) \end{bmatrix} \tag{2.3.20}$$

Algorithm 2.3.2A: Forward-Exact-Forward-Backward (FEFB) method [123]

Input: Functions $\mathcal{F}^{\,in}$, $\mathcal{F}^{\,out}$, initial estimates (x_0, p_0, θ_0) satisfying [123, Assumptions 2.2] and step length parameters λ_x, $\lambda_\theta > 0$ satisfying [123, Assumptions 2.2].

Output: An approximate optimal solution of (2.3.18).

1: **for** $k = 1, 2, \ldots$ **do:**

2: $\quad x_{k+1} = x_k - \lambda_x \nabla_x \mathcal{F}^{\,in}(x_k, \theta_k)$

3: $\quad p_{k+1} = -(\,\nabla_x^2 \mathcal{F}^{\,in}(x_{k+1}, \theta_k)\,)^{-1} \nabla_{\theta x}^2 \mathcal{F}^{\,in}(x_{k+1}, \theta_k)$

4: $\quad \theta_{k+1} = \text{prox}_{\lambda_\theta \mathcal{F}_2^{\,out}}\left(\theta_k - \lambda_\theta p_{k+1}^\top \nabla_x \mathcal{F}_1^{\,out}(x_{k+1})\right)$

5: **end for**

Owing to a particular choice for the outer objective, that additively decouples the x and θ variables as $\mathcal{F}^{\,out}(x, \theta) = \mathcal{F}_1^{\,out}(x) + \mathcal{F}_2^{\,out}(\theta)$, updates (2.3.19) give rise to a Forward-Exact-Forward-Backward (FEFB) method, described in Algorithm 2.3.2A. The name stems from the *forward* update of the x variables, the *exact* computation of the adjoint variable p (requiring a matrix inversion), and the *forward-backward* update of the θ variable (which requires a proximal step due to weaker assumptions on $\mathcal{F}_2^{\,out}$). In order to alleviate the computational burden caused by the matrix inversion in step 4 of Algorithm 2.3.2A, the authors also propose a Forward-Inexact-Forward-Backward (FIFB) method, described in Algorithm 2.3.2B. Here, they suggest using the preconditioner $M = \text{diag}(\lambda_x^{-1}\mathbb{1}_{1\times n}, \lambda_p^{-1}\mathbb{1}_{1\times mn}, \lambda_\theta^{-1}\mathbb{1}_{1\times m})$ and replacing $H_{k+1}(\nu_{k+1})$ in (2.3.20) by:

$$H_{k+1}(\nu_{k+1}) = \begin{bmatrix} \nabla_x \mathcal{F}^{\,in}(x_k, \theta_k) \\ \nabla_x^2 \mathcal{F}^{\,in}(x_{k+1}, \theta_k) p_k + \nabla_{\theta x}^2 \mathcal{F}^{\,in}(x_{k+1}, \theta_k) \\ \nabla_\theta \mathcal{F}^{\,out}(x_{k+1}, \theta_{k+1}) + p_{k+1}^\top \nabla_x \mathcal{F}^{\,out}(x_{k+1}, \theta_{k+1}) \end{bmatrix} \tag{2.3.21}$$

Algorithm 2.3.2B: Forward-Inxact-Forward-Backward (FIFB) method [123]

Input: Functions $\mathcal{F}^{in}$, $\mathcal{F}^{out}$, initial estimates (x_0, p_0, θ_0) satisfying [123, Assumptions 2.5] and step length parameters $\lambda_x,\ \lambda_p,\ \lambda_\theta > 0$ satisfying [123, Assumptions 2.5].

Output: An approximate optimal solution of (2.3.18).

1: **for** $k = 1, 2, \ldots$ **do:**

2: $\quad x_{k+1} = x_k - \lambda_x \nabla_x \mathcal{F}^{in}(x_k, \theta_k)$

3: $\quad p_{k+1} = p_k - \lambda_p \left(\nabla_x^2 \mathcal{F}^{in}(x_{k+1}, \theta_k) p_k + \nabla_{\theta x}^2 \mathcal{F}^{in}(x_{k+1}, \theta_k) \right)$

4: $\quad \theta_{k+1} = \mathrm{prox}_{\lambda_\theta \mathcal{F}_2^{out}} \left(\theta_k - \lambda_\theta p_{k+1}^\top \nabla_x \mathcal{F}_1^{out}(x_{k+1}) \right)$

5: **end for**

Let us now focus on some key regularity assumptions required to prove the convergence of the FEFB and FIFB methods. A complete list can be found in [123, Assumptions 2.2, Assumptions 2.5]. As already mentioned, the hypergradient formulation (2.3.15) holds locally. This implies that any convergent scheme relying on that analytical expression must have its initial iterate (x_0, p_0, θ_0) initialized within the local neighborhood of $(\bar{x}, \bar{p}, \bar{\theta}) = (\hat{x}(\hat{\theta}), \nabla_\theta \hat{x}(\hat{\theta}), \hat{\theta})$ in which the expression holds true. Of course, this is quite a stringent assumption that can only be relaxed depending on the specific problem. For this reason, the convergence of the proposed method is proved locally on an (ε, δ) parametric neighborhood $B_\varepsilon(\bar{x}) \times B_\delta(\bar{\theta})$ of $(\bar{x}, \bar{\theta})$. In order to guarantee the existence of the Jacobian (2.3.14), within that neighborhood, the Hessian block $\nabla_x^2 \mathcal{F}^{in}(x, \theta)$ is assumed (uniformly) positive definite, namely, $\gamma \mathbb{1}_n \leq \nabla_x^2 \mathcal{F}^{in}(x, \theta)\ \forall (x, \theta) \in B_\varepsilon(\bar{x}) \times B_\delta(\bar{\theta})$ for some $\gamma > 0$. In specific problem instances where letting $\varepsilon,\ \delta \to +\infty$ is allowed, both methods achieve global convergence, at the expense of a very restrictive global assumption on the Hessian block $\nabla_x^2 \mathcal{F}^{in}(x, \theta)$.

A similar approach can be found in [14], where the update for θ_{k+1} in step 4 of Algorithm 2.3.2B is completed with an inexact Proximal Gradient Line-search (iPGL). Here, the main objective was to establish a framework capable of providing controlled estimates $\tilde{f}_k$ and $\tilde{g}_k$ for the outer objective $\hat{\mathcal{F}}(\theta_k)$ and the indirect gradient $\nabla_\theta^{ind} \hat{\mathcal{F}}(\theta_k)$, respectively. For any given $\eta_k > 0$, they should satisfy:

$$\|\tilde{f}_k - \hat{\mathcal{F}}(\theta_k)\| \leq C_f \eta_k \qquad \|\tilde{g}_k - \nabla_\theta^{ind} \hat{\mathcal{F}}(\theta_k)\| \leq C_g \eta_k \tag{2.3.22}$$

for some $C_f,\ C_g > 0$. These estimates are of interest because $\nabla_\theta^{ind} \hat{\mathcal{F}}(\theta_k)$ defines the descent direction in the proximal gradient step, whereas $\hat{\mathcal{F}}(\theta_k)$ is used in the line-search procedure. The authors show that by computing $\tilde{x}_{k+1}$ and $\tilde{q}_{k+1}$ such that:

$$\|\nabla_x \mathcal{F}^{in}(\tilde{x}_{k+1}, \theta_k)\| \leq \eta_k \qquad \|\nabla_x^2 \mathcal{F}^{in}(\tilde{x}_{k+1}, \theta_k) \tilde{q}_{k+1} - \nabla_x \mathcal{F}_1^{out}(\tilde{x}_{k+1})\| \leq \eta_k \tag{2.3.23}$$

then the following quantities will satisfy (2.3.22):

$$\tilde{f}_k = \mathcal{F}^{out}(\tilde{x}_{k+1}, \theta_k) \qquad \tilde{g}_k = -\nabla_{x\theta}^2 \mathcal{F}^{in}(\tilde{x}_{k+1}, \theta_k) \tilde{q}_{k+1} \tag{2.3.24}$$

We remark on the connection between these conditions and steps 2-3 of Algorithm 2.3.2B. Indeed, step 2 consists of a forward iteration, starting from x_k, of a quasi-Newton method to find zeros of $\nabla_x \mathcal{F}^{in}(\cdot, \theta_k)$. This is consistent with the first requirement in (2.3.23); hence, as a rough approximation, we could write $\tilde{x}_{k+1} \approx x_{k+1} = x_k - \lambda_x \nabla_x \mathcal{F}^{in}(x_k, \theta_k)$. Similarly, both step 3 and the second requirement in (2.3.23) deal with the optimization of the adjoint equation for the inner minimizer Jacobian (although the involved matrix-vector products are computed in different orders). Ultimately, step 3 allows for the computation of an estimated indirect gradient given by $g_k = p_{k+1}^\top \nabla_x \mathcal{F}_1^{out}(x_{k+1})$, which is used within the proximal operator in step 4. Recalling that $\tilde{x}_{k+1} \approx x_{k+1}$, this estimate is again consistent with the definition of $\tilde{g}_k$ provided in the second part of (2.3.24).

2.3.3 Contraction reformulation

The second approach we consider [56, 55], while remaining within the scope of AID-based frameworks, provides a crucial insight that underpins the general strategy employed by ITD methods. Instead of defining $\hat{x}(\theta; y)$ as the optimal solution of an inner problem, here we assume the existence of a θ parametric map $f^{in} : \Omega_X \times \Omega_\Theta \times \Omega_Y \to \Omega_X$ for which $\hat{x}(\theta; y)$ is a fixed point. The inner problem (2.2.1) is therefore replaced by a parametric fixed-point equation in the form:

$$\hat{x}(\theta; y) = f^{in}(\hat{x}(\theta; y), \theta, y) \tag{2.3.25}$$

We note that one may still start by examining problem (2.2.1) and subsequently derive a suitable map f^{in} that satisfies (2.3.25). Although this framework can be adopted both in a deterministic as well as in a stochastic setting, the work in [56, 55] focuses on the latter. As a consequence, in this section we adopt the stochastic definition for the outer objective $\mathcal{F}^{out}$ in (2.2.4) and include the expectation operator in the definition of f^{in}:

$$f^{in}(x, \theta, \mathcal{Y}) = \mathbb{E}_{\mathcal{Y}} \left[\bar{f}^{in}(x, \theta, \mathcal{Y}) \right] \tag{2.3.26}$$

When not needed, we will drop the dependence on the observations.
To guarantee the well-posedness of (2.3.25), f^{in} is assumed to be a contraction. More precisely, the working assumption will be $f^{in} \in \mathcal{C}^1(\Omega_X \times \Omega_\Theta)$ and:

$$\|\nabla_x f^{in}(x, \theta)\| \leq \alpha < 1 \qquad \forall (x, \theta) \in \Omega_X \times \Omega_\Theta \tag{2.3.27}$$

As highlighted in Section 2.3.2, a key limitation of the general inner problem formulation (2.2.1) is the local nature of the hypergradient and its dependence on the inner minimizer $\hat{x}(\theta)$, whose computation or approximation can be particularly demanding. The *global* contraction assumption (2.3.27) resolves both issues. Under this reformulation of the inner problem, as we will see shortly, the hypergradient becomes globally defined, and, by virtue of the (parametric) Fixed-Point Theorem 2.A.2, arbitrarily accurate estimates of $\hat{x}(\theta)$ can be readily obtained. Indeed, it is well-known that for any $x_0 \in \Omega_X$, with Ω_X a Banach space, the fixed-point iterations $x_{k+1}(\theta) = f^{in}(x_k(\theta), \theta)$ converge to $\hat{x}(\theta)$ and satisfy the relation:

$$\|\hat{x}(\theta) - x_{k+1}(\theta)\| \leq \frac{\alpha^k}{1 - \alpha} \|x_1(\theta) - x_0\| \tag{2.3.28}$$

which can be used to control the approximation error.
It should be noted that this alternative formulation of the inner problem may be equivalent to the formulation presented in Section 2.2. Indeed, assuming $\mathcal{F}^{in}$ to be Lipschitz smooth and strongly convex in its first argument, there always exists a sufficiently small constant $\eta > 0$ for which the gradient descent map:

$$f^{in}(x, \theta) = x - \eta \nabla_x \mathcal{F}^{in}(x, \theta) \tag{2.3.29}$$

is a contraction[2]. Moreover, under these assumptions, $\hat{x}(\theta)$ is uniquely defined, and the fixed point iterations of (2.3.29) will converge to this solution.
Let us now focus on the hypergradient representation for this bilevel optimization framework. In particular, since (2.3.12) still holds, it suffices to obtain an expression for $\nabla_\theta \hat{x}(\theta)$ analogous to (2.3.14). To this end, assuming $f^{in} \in \mathcal{C}^1(\Omega_X \times \Omega_\Theta)$, by differentiating both sides of the fixed-point equation (2.3.25) with respect to θ, it can be easily shown that:

$$\nabla_\theta \hat{x}(\theta) = \left(\mathbb{1}_n - \nabla_x f^{in}(\hat{x}(\theta), \theta) \right)^{-1} \nabla_\theta f^{in}(\hat{x}(\theta), \theta) \tag{2.3.30}$$

As a consequence, the complete hypergradient is in the form:

$$\nabla_\theta \hat{\mathcal{F}}(\theta) = \nabla_\theta \mathcal{F}^{out}(\hat{x}(\theta), \theta) + \nabla_\theta f^{in}(\hat{x}(\theta), \theta)^\top \left(\mathbb{1}_n - \nabla_x f^{in}(\hat{x}(\theta), \theta)^\top \right)^{-1} \nabla_x \mathcal{F}^{out}(\hat{x}(\theta), \theta) \tag{2.3.31}$$

Both Jacobian representations (2.3.14) and (2.3.30) require a matrix inversion. The key difference is that, in the former, the existence of the inverse is only *local*, while in the latter, owing to our *global* contraction assumption (2.3.27) and Proposition 2.A.2, the inverse is always well-defined. These two representations are actually equivalent by virtue of the above observation regarding the Lipschitzianity of $\mathcal{F}^{in}$. Indeed, plugging (2.3.29) into (2.3.30) yields precisely (2.3.14). The assumption on the invertibility of the Hessian block $\nabla_x^2 \mathcal{F}^{in}(\hat{x}(\theta), \theta)$ required in the previous Section 2.3.2 is here translated into a contraction assumption on the gradient descent map (2.3.29) applied to $\mathcal{F}^{in}(x, \theta)$.
In the following, we denote $q(x, \theta)$ the solution of the linear system:

$$\left(\mathbb{1}_n - \nabla_x f^{in}(x, \theta)^\top \right) q = \nabla_x \mathcal{F}^{out}(x, \theta) \tag{2.3.32}$$

which is again well-defined on account of the contraction assumption. Note that in order to construct the hypergradient (2.3.31) both $\hat{x}(\theta)$ and $\hat{q}(\theta) = q(\hat{x}(\theta), \theta)$ are required. Unlike the FEFB and FIFB methods described in Section 2.3.2, where the proposed optimization scheme updated the x and p variables only *once* for each new θ iteration, in [56] the authors propose a two-phase algorithm composed of nested loops. In the inner loop, in particular, the variables x and q are updated *multiple* times via stochastic fixed-point iterations to approximate $\hat{x}(\theta)$ and $\hat{q}(\theta)$, thereby enabling an estimation of the hypergradient. These updates form the Stochastic Implicit Differentiation (SID) method, described in Algorithm 2.3.3A. Lastly, in the outer loop the θ variables are updated once using the hypergradient approximation provided by SID. The resulting optimization scheme, Bilevel Stochastic Gradient Method (BSGM), can be found in Algorithm 2.3.3B. Apart from the stochastic setting, we remark the similarity between conditions (2.3.23) proposed in [14] and the fixed point iterations in step 3, 9 of Algorithm 2.3.3A. In [56]

[2]If $\mathcal{F}^{in}$ is Lipschitz smooth with constant $L > 0$, it suffices to choose $\eta < \frac{1}{L}$.

Algorithm 2.3.3A: Stochastic Implicit Differentiation (SID) method [56]

Input: Functions f^{in}, $\mathcal{F}^{out}$, iteration bounds $k_x,\ k_q > 0$, batch size $J > 0$, initial estimates x_0, q_0, current value of θ and step length parameters $\{\lambda_k\}_{k\geq 0} > 0$ satisfying the assumptions of [123, Theorem 8].

Output: An approximation of the hypergradient (2.3.31).

1: Choose $\{y_k^x\}_{k=0}^{k_x-1}$, $\{y_k^q\}_{k=0}^{k_q-1}$, $\{y_j\}_{j=1}^{J}$ realizations of $\mathcal{Y}$ and $\{z_j\}_{j=1}^{J}$ realizations of $\mathcal{Z}$.
2: **for** $k = 0, 1, \ldots, k_x - 1$ **do:**
3: $\quad x_{k+1} = x_k + \lambda_k \left(\bar{f}^{in}(x_k, \theta, y_k^x) - x_k \right)$
4: **end for**
5: Compute $\nabla_x \mathcal{F}_J^{out} = \frac{1}{J} \sum_{j=1}^{J} \nabla_x \bar{\mathcal{F}}^{out}(x_{k_x}, \theta, z_j)$
6: Compute $\nabla_\theta \mathcal{F}_J^{out} = \frac{1}{J} \sum_{j=1}^{J} \nabla_\theta \bar{\mathcal{F}}^{out}(x_{k_x}, \theta, z_j)$
7: Compute $\nabla_\theta f_J^{in} = \frac{1}{J} \sum_{j=1}^{J} \nabla_\theta \bar{f}^{in}(x_{k_x}, \theta, y_j)$
8: **for** $k = 0, 1, \ldots, k_q - 1$ **do:**
9: $\quad q_{k+1} = q_k + \lambda_k \left(\nabla_x \bar{f}^{in}(x_{k_x}, \theta, y_k^q)^\top q_k + \nabla_x \mathcal{F}_J^{out} - q_k \right)$
10: **end for**
11: Compute $\tilde{\nabla}_\theta \hat{\mathcal{F}}(\theta) = \nabla_\theta \mathcal{F}_J^{out} + \left(\nabla_\theta f_J^{in} \right)^\top q_{k_q}$

Algorithm 2.3.3B: Bilevel Stochastic Gradient Method (BSGM) [56]

Input: Functions f^{in}, $\mathcal{F}^{out}$, iteration bounds h_θ, $\{k_x^h\}_{h=0}^{h_\theta-1}$, $\{k_q^h\}_{h=0}^{h_\theta-1} > 0$ satisfying the assumptions of [123, Theorem 16], batch sizes $\{J_h\}_{h=0}^{h_\theta-1} > 0$ satisfying the assumptions of [123, Theorem 16], initial estimates θ_0, x_0 and step length parameters λ_θ, $\{\lambda_k\}_{k\geq 0} > 0$ satisfying the assumptions of [123, Theorem 16, Theorem 8].

Output: An approximation of $\hat{\theta}$.

1: **for** $h = 0, 1, \ldots, h_\theta - 1$ **do:**
2: $\quad$ Compute $\tilde{\nabla}_\theta \hat{\mathcal{F}}(\theta_h)$ using Algorithm 2.3.3A (SID) with $k_x = k_x^h$, $k_q = k_q^h$, $J = J_h$, $x_0 = x_0$, $q_0 = 0$, $\theta = \theta_h$, $\lambda_k = \lambda_k$.
3: $\quad \theta_{h+1} = \mathrm{proj}_{\Omega_\Theta} \left(\theta_h - \lambda_\theta \tilde{\nabla}_\theta \hat{\mathcal{F}}(\theta_h) \right)$
4: **end for**

it is shown that $x_{k_x} = x_{k_x}(\theta)$ and $q_{k_q} = q_{k_q}(\theta)$, as functions of the iteration bounds k_x and k_q, converge (sublinearly) in Mean Squared Error (MSE) to $\hat{x}(\theta)$ and $q_J(x_{k_x}, \theta)$ respectively, where $q_J(x_{k_x}, \theta)$ is the solution of (2.3.32) with the right-hand side replaced by its stochastic approximation $\nabla_x \mathcal{F}_J^{out}$ defined in step 5. More precisely, there exist some constants d_x, d_q, $\gamma > 0$ such that:

$$\mathbb{E}\left[\,\|x_{k_x} - \hat{x}(\theta)\|^2\,\right] \leq \frac{d_x}{\gamma + k_x} \qquad \mathbb{E}\left[\,\|q_{k_q} - q_J(x_{k_x}, \theta)\|^2\,\right] \leq \frac{d_q}{\gamma + k_q} \tag{2.3.33}$$

For some appropriate bounds k_x and k_q, the proposed SID method can therefore be interpreted as an operative way of obtaining estimates satisfying conditions (2.3.23), which are sufficient to approximate the hypergradient to any level of precision (see (2.3.22)). Indeed, the approximation $\tilde{\nabla}\hat{\mathcal{F}}(\theta)$ of the hypergradient computed in step 11 of Algorithm 2.3.3A is shown to satisfy similar convergence properties. For $K = k_x = k_q = J$, there exists a constant $d_\nabla > 0$ such that:

$$\mathbb{E}\left[\,\|\tilde{\nabla}\hat{\mathcal{F}}(\theta) - \nabla\hat{\mathcal{F}}(\theta)\|^2\,\right] \leq \frac{d_\nabla}{K} \tag{2.3.34}$$

2.3.4 Fixed point iterations

As previously discussed in Section 2.3.3, the idea of replacing the inner problem with a fixed-point equation lies at the core of most ITD methods for bilevel optimization. In particular, we examine the work of [115], where this general framework is introduced and compared with AID methods. We assume again the existence of a θ parametric map $f^{\,in}$ satisfying the fixed-point equation (2.3.25), but relax the contraction assumption that drove the previous analysis. Here, we simply require that the fixed point iterations $x_{k+1}(\theta) = f^{\,in}(x_k(\theta), \theta)$ converge to $\hat{x}(\theta)$:

$$\hat{x}(\theta) = \lim_{k \to +\infty} x_k(\theta) \tag{2.3.35}$$

Just like before, one can start by examining problem (2.2.1) in order to derive a suitable map $f^{\,in}$. The advantage of this formulation is that, depending on the inner problem, barely any regularity assumptions may be needed for the inner objective $\mathcal{F}^{\,in}$. In [99], for example, a primal-dual scheme with Bregman distances [18] applied to the inner problem produces a smooth map $f^{\,in}$ without requiring the smoothness of the inner objective[3]. Naturally, if $\mathcal{F}^{\,in}$ is Lipschitz smooth, the gradient descent map (2.3.29) is still a notable option.

Up to this point, the AID methods discussed have followed a common strategy: they approximate specific components of the hypergradient, such as the Jacobian (2.3.14) or the inner minimizer, while maintaining the overall analytic form defined in (2.3.12). In contrast, ITD methods take a different route to bilevel optimization: they approximate the global objective $\hat{\mathcal{F}}$ with a suitable surrogate $\tilde{\mathcal{F}}_K$ and then compute its gradient *exactly*. Recalling the definition of $\hat{\mathcal{F}}$ in (2.2.6), ITD methods *truncate* the fixed-point iterations at a prescribed index $K \geq 0$ and essentially approximate $\hat{x}(\theta) \approx x_K(\theta)$, yielding the surrogate objective:

$$\tilde{\mathcal{F}}_K(\theta) = \mathcal{F}^{\,out}(x_K(\theta), \theta) \tag{2.3.36}$$

A minimizer of (2.3.36) will be denoted by $\tilde{\theta}_K$. This particular choice of surrogate, combined with mild smoothness assumptions on the map $f^{\,in}$, ensures that the surrogate

[3] As we will see in this section, the iteration map smoothness is a core assumption of ITD methods.

hypergradient $\nabla_\theta \tilde{\mathcal{F}}_K(\theta)$ can be computed exactly, without the need for costly matrix inversions.
Intuitively, as $K \to +\infty$, one would expect the minimizers $\tilde{\theta}_K$ to converge to a minimizer of the global objective. The work in [45], under very natural assumptions, shows precisely this. We include the main results in the following Theorem 2.3.1.

Theorem 2.3.1 (Existence and Convergence Theorems [45]). *Under the following assumptions:*

(i) $\Omega_\Theta \subseteq \mathbb{R}^m$ *is compact;*

(ii) $\mathcal{F}^{\,out} \in \mathcal{C}(\Omega_X \times \Omega_\Theta)$*;*

(iii) $\mathcal{F}^{\,in} \in \mathcal{C}(\Omega_X \times \Omega_\Theta)$ *and* $\hat{x}(\theta)$ *is uniquely defined* $\forall \theta \in \Omega_\Theta$*;*

(iv) $\|\hat{x}(\theta)\| \leq B \ \forall \theta \in \Omega_\Theta$ *for some* $B > 0$*.*

The bilevel optimization problem is well-posed. In addition, if:

(v) $\mathcal{F}^{\,out}(\,\cdot\,, \theta)$ *is* θ*-uniformly Lipschitz continuous;*

(vi) The iterates $\{x_K(\theta)\}_{K\geq 1}$ *converge* θ*-uniformly to* $\hat{x}(\theta)$*.*

Then the limit points of $\{\tilde{\theta}_K\}_{K\geq 1}$ *are minimizers of the global objective* $\hat{\mathcal{F}}$[4]*.*

We remark that condition (vi), which is slightly stronger than (2.3.35), constitutes an indirect assumption on the iteration map $f^{\,in}$. In particular, under the global contraction hypothesis (2.3.27) introduced in the previous section, assumptions (iii), (iv), and (vi) are automatically satisfied. This slightly weaker formulation is therefore a generalization of the contraction-based approach.
Let us now derive an expression for the surrogate hypergradient. On top of the usual smoothness assumptions on $\mathcal{F}^{\,out}$, in order for the surrogate objective to be differentiable, it suffices to require that $f \in \mathcal{C}^1(\Omega_X \times \Omega_\Theta)$. Indeed, this implies $x_K \in \mathcal{C}^1(\Omega_\Theta)$ and thus the differentiability of $\tilde{\mathcal{F}}_K$. Since $x_{k+1}(\theta) = f^{\,in}(x_k(\theta), \theta)$, it holds:

$$\nabla_\theta x_{k+1}(\theta) = \nabla_\theta f^{\,in}(x_k(\theta), \theta) + \nabla_x f^{\,in}(x_k(\theta), \theta) \nabla_\theta x_k(\theta) \qquad \forall k \geq 0 \tag{2.3.37}$$

From the chain rule, by iteratively applying (2.3.37) we obtain the following representation for the surrogate hypergradient:

$$\nabla_\theta \tilde{\mathcal{F}}_K(\theta) = \nabla_\theta \mathcal{F}^{\,out}(x_K(\theta), \theta) + \sum_{k=0}^{K} B_k(\theta) A_{k+1}(\theta) \cdots A_K(\theta) \nabla_x \mathcal{F}^{\,out}(x_K(\theta), \theta) \tag{2.3.38}$$

where $B_0(\theta) = \nabla_\theta x_0(\theta)^\top$ and $A_{k+1}(\theta) = \nabla_x f^{\,in}(x_k(\theta), \theta)^\top$, $B_{k+1}(\theta) = \nabla_\theta f^{\,in}(x_k(\theta), \theta)^\top$ for $k \geq 0$. Unlike previous representations, (2.3.38) does not involve matrix inversions nor the knowledge of the inner minimizer $\hat{x}(\theta)$. All its components can therefore be evaluated

[4] In general, we cannot state that the chosen minimizer $\hat{\theta}$ is a limit point of $\{\tilde{\theta}_K\}_{K\geq 1}$ since $\hat{\mathcal{F}}$ may have multiple minimizers.

exactly. In practice, $\nabla_\theta \tilde{\mathcal{F}}_K(\theta)$ is computed using either Forward-Mode Differentiation (FMD):

$$\begin{cases} h_0(\theta) &= B_0(\theta) \\ h_{k+1}(\theta) &= h_k(\theta)A_{k+1}(\theta) + B_{k+1}(\theta) \qquad \forall k = 0, \dots, K-1 \\ \nabla_\theta \tilde{\mathcal{F}}_K(\theta) &= \nabla_\theta \mathcal{F}^{\,out}(x_K(\theta), \theta) + h_K(\theta) \nabla_x \mathcal{F}^{\,out}(x_K(\theta), \theta) \end{cases} \tag{2.3.39}$$

or Reverse-Mode Differentiation (RMD):

$$\begin{cases} h_K(\theta) &= \nabla_\theta \mathcal{F}^{\,out}(x_K(\theta), \theta) \\ \alpha_K(\theta) &= \nabla_x \mathcal{F}^{\,out}(x_K(\theta), \theta) \\ h_{k-1}(\theta) &= h_k(\theta) + B_k(\theta)\alpha_k(\theta) \qquad \forall k = K, \dots, 0 \\ \alpha_{k-1}(\theta) &= A_k(\theta)\alpha_k(\theta) \qquad \forall k = K, \dots, 1 \\ \nabla_\theta \tilde{\mathcal{F}}_K(\theta) &= h_{-1}(\theta) \end{cases} \tag{2.3.40}$$

The two modes differ in both computational and memory complexity. Although FMD is generally more computationally demanding than RMD, its memory usage does not scale with K. In contrast, RMD requires storing all variables $x_k(\theta)$ for $k = 1, \dots, K$ computed during the forward pass, as they are needed for the backward pass.
Regarding the relationship between the surrogate hypergradient (2.3.38) and the exact hypergradient (2.3.31), the study in [115] establishes an insightful connection. Assuming once again the contraction hypothesis (2.3.27), and denoting:

$$\begin{aligned} A_\infty(\theta) &= \lim_{k \to +\infty} A_k(\theta) &&= \lim_{k \to +\infty} \nabla_x f^{\,in}(x_k(\theta), \theta)^\top &&= \nabla_x f^{\,in}(\hat{x}(\theta), \theta)^\top \\ B_\infty(\theta) &= \lim_{k \to +\infty} B_k(\theta) &&= \lim_{k \to +\infty} \nabla_\theta f^{\,in}(x_k(\theta), \theta)^\top &&= \nabla_\theta f^{\,in}(\hat{x}(\theta), \theta)^\top \end{aligned} \tag{2.3.41}$$

by Proposition 2.A.2 we have:

$$\left(\mathbb{1}_n - \nabla_x f^{\,in}(\hat{x}(\theta), \theta)^\top \right)^{-1} = \sum_{k=0}^{\infty} \left(\nabla_x f^{\,in}(\hat{x}(\theta), \theta)^\top \right)^k = \sum_{k=0}^{\infty} A_\infty(\theta)^k \tag{2.3.42}$$

Therefore, the hypergradient in (2.3.31) can be written as:

$$\nabla_\theta \hat{\mathcal{F}}(\theta) = \nabla_\theta \mathcal{F}^{\,out}(\hat{x}(\theta), \theta) + B_\infty(\theta) \sum_{k=0}^{\infty} A_\infty(\theta)^k \nabla_x \mathcal{F}^{\,out}(\hat{x}(\theta), \theta) \tag{2.3.43}$$

The above representation shares many similarities with the surrogate hypergradient in (2.3.38). In the very special case where $x_k(\theta) \equiv \hat{x}(\theta)\ \forall k \geq 0$, which can be achieved by initializing $x_0(\theta) = \hat{x}(\theta)$, we get $A_k(\theta) \equiv A_\infty(\theta)$ and $B_k(\theta) \equiv B_\infty(\theta)\ \forall k \geq 0$. The surrogate hypergradient then reduces to:

$$\nabla_\theta \tilde{\mathcal{F}}_K(\theta) = \nabla_\theta \mathcal{F}^{\,out}(\hat{x}(\theta), \theta) + B_\infty(\theta) \sum_{k=0}^{K} A_\infty(\theta)^k \nabla_x \mathcal{F}^{\,out}(\hat{x}(\theta), \theta) \tag{2.3.44}$$

which is an approximation of (2.3.43) obtained by truncating the series at its K-th term. In particular, as $K \to +\infty$, one recovers the exact hypergradient:

$$\nabla_\theta \hat{\mathcal{F}}(\theta) = \lim_{K \to +\infty} \nabla_\theta \tilde{\mathcal{F}}_K(\theta) \tag{2.3.45}$$

It is also worth mentioning that to further reduce the computational and memory cost of constructing the surrogate hypergradient (2.3.38), a possible strategy is to truncate the first terms in the summation. More precisely, given $\bar{K} \leq K$, one could use:

$$\nabla_\theta \tilde{\mathcal{F}}_{K,\bar{K}}(\theta) = \nabla_\theta \mathcal{F}^{\,out}(x_K(\theta), \theta) + \sum_{k=\bar{K}}^{K} B_k(\theta) A_{k+1}(\theta) \cdots A_K(\theta) \nabla_x \mathcal{F}^{\,out}(x_K(\theta), \theta) \tag{2.3.46}$$

Applying RMD to (2.3.46) is equivalent to performing a $(K - \bar{K} + 1)$-step truncated backpropagation on the original formulation (2.3.38). Under some necessary technical requirements [115, Theorem 3.4], it can be shown that optimizing θ using (2.3.46) (even for $\bar{K} = K$, a single backpropagation step), still produces a sequence whose limit points are exact stationary points of the global objective $\hat{\mathcal{F}}$.

2.4 Unfolding and untying the inner problem

The principle underlying ITD methods for bilevel optimization, namely truncating the iterative process used to solve the inner problem and differentiating the resulting surrogate outer objective, is conceptually aligned with the algorithm unrolling approach discussed in Section 1.2.1. From a mathematical standpoint, the developments that resulted in Section 2.3.4 offer a flexible framework that rigorously connects unrolled deep networks to more interpretable modeling paradigms. By doing so, the framework bridges data-driven and model-based perspectives, promoting interpretable and principled learning architectures. Within this context, the function f_θ, which served as a generic update map in Section 1.2.1, now takes on a *problem-specific* role as the iteration map $f_\theta^{\,in} = f^{\,in}(\,\cdot\,, \theta)$, encoding the chosen optimization strategy for the inner problem. Similarly, the parameters θ, optimized solely with respect to a data-driven loss $\mathcal{L}$, now act as *degrees of freedom* within the bilevel framework, enabling the exploration of the manifold of optimal solutions parameterized by $\hat{x}(\theta)$, as discussed in Section 2.2. Finally, the number of layers in the network, traditionally treated as a tunable hyperparameter, acquires a new interpretation, becoming directly linked to the *approximation quality* of the inner minimizer.

As far as this work is concerned, we present one final extension to the previous ITD approach, which was first introduced in [64]. From an architectural standpoint, the fundamental distinction between the unrolled networks introduced in Section 1.2.1 and standard deep networks lies in the parameterization: in unrolled networks, the parameters θ are usually *shared* among all layers, whereas in conventional deep architectures, they

are *independently learned* for each layer. The process of allowing the parameters θ to vary across layers (or iterations, within a bilevel optimization framework) is referred to as *untying*. As highlighted in [64], the rationale for untying such parameters in a model-based-inspired architecture is to enable the network to represent a richer and more flexible family of inference functions than the original model. Denoting $f_{\theta_k}^{in} = f^{in}(\cdot, \theta_k)$ and $x_{k+1} = x_{k+1}(\theta_k) = f_{\theta_k}^{in}(x_k)$ for a collection of layer-specific parameters $\vartheta = \{\theta_k\}_{k=0}^{K} \subseteq \Omega_\Theta$, the high-level architecture of the network is displayed in Figure 2.4.1.

$$x_0 \xrightarrow{f_{\theta_0}^{in}} x_1 \xrightarrow{f_{\theta_1}^{in}} \cdots \xrightarrow{f_{\theta_{K-2}}^{in}} x_{K-1} \xrightarrow{f_{\theta_{K-1}}^{in}} x_K \dashrightarrow \mathcal{F}^{out}(x_K, \theta_K)$$

Figure 2.4.1: Unrolled iterations for the optimization of the inner problem into a K-layer, untied parameters architecture trainable end-to-end with the outer objective $\mathcal{F}^{out}$ as a loss function.

Let us also denote $\hat{\vartheta}$ a minimizer for the global loss function:

$$\hat{\mathcal{F}}(\vartheta) = \mathcal{F}^{out}(x_K, \theta_K) \tag{2.4.1}$$

where the dependence of the right-hand side on all θ_k is implied. In this configuration, the parameter search space grows from Ω_Θ to Ω_Θ^{K+1}, where the optimal solution $\hat{\theta}$ defined previously corresponds to the collection $\tilde{\vartheta} = (\hat{\theta}, \dots, \hat{\theta}) \in \Omega_\Theta^{K+1}$. In this sense, the architecture in Figure 2.4.1 represents an actual generalization of the tied parameters network. Indeed, shifting the focus exclusively to the minimization of the loss (which is typical of deep learning with neural networks), in the worst case, $\hat{\vartheta}$ achieves the same loss value as $\hat{\theta}$ for the tied parameters case:

$$\hat{\mathcal{F}}(\hat{\vartheta}) \leq \hat{\mathcal{F}}(\tilde{\vartheta}) = \hat{\mathcal{F}}(\hat{\theta}) \tag{2.4.2}$$

Untying the outer parameters also introduces certain drawbacks. From an interpretability standpoint, the final variables x_K no longer correspond directly to an approximate minimizer of the inner problem. Within our framework, this interpretation holds only when the parameters are tied, that is, when the iteration index k does not influence θ. A more general formulation than the one considered in the preceding sections, in which θ is allowed to vary across iterations, could potentially restore this correspondence between x_K and the inner problem minimizer. Conversely, from the perspective of algorithm unrolling independent of bilevel optimization, untying the parameters can often be the more natural choice. This is the case, for instance, when unrolling iterations of a time-dependent dynamical system [23, 110], where the set of parameters to be optimized may itself depend on time, giving rise to time-varying parameters θ_k. Another important consideration is the linear growth of the parameter search space with respect to K. Larger parameter spaces can pose additional challenges during optimization, including increased memory and computational demands, as well as a higher risk of convergence to suboptimal local minima. Taken as a whole, untying in the context of bilevel optimization represents a trade-off between interpretability and the expressive power of the model.

2.4.1 General backpropagation scheme

In this section, we begin by deriving the general backpropagation framework used to train the network illustrated in Figure 2.4.1. We also introduce notation that will be used throughout the subsequent chapters.
We start by establishing a set of general assumptions regarding the key components of the network. These assumptions specialize the general architecture described in Section 2.4 to the specific cases we will examine later. First, we impose minimal smoothness conditions to ensure that the loss function is differentiable with respect to the model parameters:

(i) [Smoothness] $f^{\,in} \in \mathcal{C}^1(\Omega_X \times \Omega_\Theta)$ and $\mathcal{F}^{\,out} \in \mathcal{C}^1(\Omega_X \times \Omega_\Theta)$.

Next, we assume that the outer objective is independent of the optimization parameters. This effectively eliminates the contribution of the direct term in the hypergradient:

(ii) [No direct hypergradient] $\mathcal{F}^{\,out}(x, \theta) \equiv \mathcal{F}^{\,out}(x)$, or, equivalently, $\nabla_\theta \mathcal{F}^{\,out}(x, \theta) \equiv 0$.

Finally, we restrict our analysis to deterministic observations $y \in \Omega_Y$ and $z \in \Omega_Z$ for the inner and outer optimization problems, respectively. Accordingly, we adopt deterministic formulations for both the iteration map and the outer objective, explicitly emphasizing their dependence on the observations:

(iii) [Deterministic observations] $f^{\,in}(x, \theta) = f^{\,in}(x, \theta, y)$ and $\mathcal{F}^{\,out}(x) = \mathcal{F}^{\,out}(x, z)$.

Let us now briefly assume that $\Omega_X \subseteq \mathbb{R}^n$ and $\Omega_\Theta \subseteq \mathbb{R}^m$. By applying the chain rule, the backpropagation of the gradients of the global loss function $\hat{\mathcal{F}}$ with respect to the states x_k leads to the following recursive relations:

$$\begin{cases} \nabla_{x_K} \hat{\mathcal{F}} &= \nabla_x \mathcal{F}^{\,out}(x_K, z) \\ \nabla_{x_k} \hat{\mathcal{F}} &= (\,\nabla_{x_k} x_{k+1}\,)^\top \nabla_{x_{k+1}} \hat{\mathcal{F}} \;=\; \nabla_x f^{\,in}(x_k, \theta_k, y)^\top \nabla_{x_{k+1}} \hat{\mathcal{F}} \qquad \forall k = 0, \dots, K-1 \end{cases} \tag{2.4.3}$$

Using (2.4.3) we can then compute the gradients of $\hat{\mathcal{F}}$ with respect to the parameters at each layer:

$$\nabla_{\theta_k} \hat{\mathcal{F}} = (\,\nabla_{\theta_k} x_{k+1}\,)^\top \nabla_{x_{k+1}} \hat{\mathcal{F}} = \nabla_\theta f^{\,in}(x_k, \theta_k, y)^\top \nabla_{x_{k+1}} \hat{\mathcal{F}} \qquad \forall k = 0, \dots, K-1 \tag{2.4.4}$$

These relations are analogous to (2.3.39) in the case of tied parameters. A limitation of (2.4.4), however, is that it assumes the states x_k and parameters θ_k are vector-valued. In practice, these quantities often consist of multi-dimensional tensors (e.g., matrices). To avoid the cumbersome process of vectorization and de-vectorization, it is therefore preferable to define the corresponding gradients in a manner that preserves their inherent tensor structure. With a slight abuse of notation, the first step in this direction will be to write $\nabla_{x_k} \hat{\mathcal{F}} \in \Omega_X$ and $\nabla_{\theta_k} \hat{\mathcal{F}} \in \Omega_\Theta$, where Ω_X and Ω_Θ are now generic vector (tensor) spaces. Moreover, since the tensor representation of those gradients varies on a case-by-case basis, we simply rewrite (2.4.3) as:

$$\begin{cases} \nabla_{x_K} \hat{\mathcal{F}} &= \Phi_K \left(x_K, z \right) \\ \nabla_{x_k} \hat{\mathcal{F}} &= \Phi_x \left(\nabla_{x_{k+1}} \hat{\mathcal{F}}, x_k, \theta_k, y \right) \qquad \forall k = 0, \dots, K-1 \end{cases} \tag{2.4.5}$$

and (2.4.4) as:

$$\nabla_{\theta_k} \hat{\mathcal{F}} = \Phi_\theta \left(\nabla_{x_{k+1}} \hat{\mathcal{F}}, x_k, \theta_k, y \right) \qquad \forall k = 0, \dots, K-1 \tag{2.4.6}$$

where:

- $\Phi_K : \Omega_X \times \Omega_Z \to \Omega_X$ is the *final state gradient* map, which depends on the explicit form of $\nabla_x \mathcal{F}^{\,out}(x, z)$;
- $\Phi_x : \Omega_X^2 \times \Omega_\Theta \times \Omega_Y \to \Omega_X$ is the *state gradient update* map, which depends on the explicit form of $\nabla_x f^{\,in}(x, \theta, y)$;
- $\Phi_\theta : \Omega_X^2 \times \Omega_\Theta \times \Omega_Y \to \Omega_\Theta$ is the *parameter gradient* map, which depends on the explicit form of $\nabla_\theta f^{\,in}(x, \theta, y)$.

Up to this point, we have treated both parameters $x \in \Omega_X$ and $\theta \in \Omega_\Theta$ as collections of homogeneous variables. However, this assumption is not strictly necessary. In many cases, it is more natural to regard x and θ as being composed of multiple subsets of variables:

$$x = \left\{ x^1, \dots, x^{J_X} \right\} \qquad \theta = \left\{ \theta^1, \dots, \theta^{J_\Theta} \right\} \tag{2.4.7}$$

where $\Omega_X = \Omega_X^1 \times \cdots \times \Omega_X^{J_X}$ and $\Omega_\Theta = \Omega_\Theta^1 \times \cdots \times \Omega_\Theta^{J_\Theta}$. In these cases, the gradient maps defined above can be split into an analogous collection:

$$\Phi_K = \left\{ \Phi_K^1, \dots, \Phi_K^{J_X} \right\} \qquad \Phi_x = \left\{ \Phi_x^1, \dots, \Phi_x^{J_X} \right\} \qquad \Phi_\theta = \left\{ \Phi_\theta^1, \dots, \Phi_\theta^{J_\Theta} \right\} \tag{2.4.8}$$

where each sub-map Φ^j depends on the explicit form of the Jacobian with respect to the corresponding subset of variables x^j or θ^j. A similar split also occurs in the iteration map:

$$f_\theta^{in} = \left\{ \left(f_\theta^{in}\right)^1, \dots, \left(f_\theta^{in}\right)^{J_X} \right\} \tag{2.4.9}$$

Appendix

2.A Supporting results

Proposition 2.A.1. *Let $M : I \subseteq \mathbb{R} \to \mathcal{M}_{n\times n}(\mathbb{R})$, $M \in \mathcal{C}^1(I)$ be a matrix-valued map such that $M(x)^{-1}$ exists $\forall x \in I$ and $M^{-1} \in \mathcal{C}^1(I)$. Then it holds:*

$$\frac{d}{dx}M^{-1}(x) = -M^{-1}(x)\left(\frac{d}{dx}M(x)\right)M^{-1}(x) \quad \forall x \in I$$

Proof: Differentiating the relation $M(x)M^{-1}(x) = \mathbb{1}_n$, we obtain:

$$\left(\frac{d}{dx}M(x)\right)M^{-1}(x) + M(x)\left(\frac{d}{dx}M^{-1}(x)\right) = 0_n$$

Isolating the term $\dfrac{d}{dx}M^{-1}(x)$ then yields the desired result.

Theorem 2.A.1 (Implicit Function Theorem)**.** *Let $f : \mathbb{R}^{n+m} \to \mathbb{R}^n$, $f \in \mathcal{C}^1(\mathbb{R}^{n+m})$ and $(\hat{x}, \hat{y}) \in \mathbb{R}^{n+m}$ such that $f(\hat{x}, \hat{y}) = 0$. If $\nabla_y f(\hat{x}, \hat{y})$ is invertible, then:*

1. *$\exists \delta > 0$ and $\exists!\, y : B_\delta(\hat{x}) \to \mathbb{R}^m$, $y \in \mathcal{C}^1(B_\delta(\hat{x}))$ satisfying $y(\hat{x}) = \hat{y}$ and $f(x, y(x)) = 0$ $\forall x \in B_\delta(\hat{x})$;*
2. *$\nabla_y f(x, y(x))$ is invertible $\forall x \in B_\delta(\hat{x})$.*

Moreover, it holds:

$$\nabla_x y(x) = -\left(\nabla_y f(x, y(x))\right)^{-1} \nabla_x f(x, y(x))$$

Proof: See [77].

Theorem 2.A.2 (Fixed-Point Theorem)**.** *Let (X, d) be a complete metric space and $F : X \to X$ a contraction with Lipschitz constant $0 \leq q < 1$. Then:*

- *$\exists! \hat{x} \in X$ fixed point of F;*
- *Defining $x_{k+1} = F(x_k)$ $\forall k \geq 0$ for any $x_0 \in X$, it holds:*

$$d(\hat{x}, x_{k+1}) \leq \frac{q^k}{1-q} d(x_1, x_0)$$

In particular, $\lim\limits_{k\to+\infty} x_k = \hat{x}$.

Proof: See [91].

Proposition 2.A.2. *Let $A \in \mathcal{M}_{n\times n}(\mathbb{R})$ such that $\|A\| \leq \alpha < 1$. Then:*

1. *$\mathbb{1}_n - A$ is invertible and $(\mathbb{1}_n - A)^{-1} = \sum_{k=0}^{\infty} A^k$;*

2. *$\| (\mathbb{1}_n - A)^{-1} \| \leq \frac{1}{1-\alpha}$.*

Proof: Since $\|A\| \leq \alpha < 1$, we have:

$$\sum_{k=0}^{\infty} \|A\|^k \leq \sum_{k=0}^{\infty} \alpha^k = \frac{1}{1-\alpha} < +\infty$$

therefore the series $\sum_{k=0}^{\infty} A^k$ is convergent. Let $B = \sum_{k=0}^{\infty} A^k$. By noticing that:

$$\lim_{k\to\infty} (\mathbb{1}_n - A)\left(\sum_{i=0}^{k} A^i\right) = \lim_{k\to\infty}\left(\sum_{i=0}^{k} A^i\right)(\mathbb{1}_n - A) = \lim_{k\to\infty}\left(\sum_{i=0}^{k} A^i - \sum_{i=0}^{k+1} A^i + \mathbb{1}_n\right) = \mathbb{1}_n$$

we get $(\mathbb{1}_n - A)\, B = B\,(\mathbb{1}_n - A) = \mathbb{1}_n$, hence the first claim is proved. Lastly:

$$\| (\mathbb{1}_n - A)^{-1} \| = \|B\| = \left\| \sum_{k=0}^{\infty} A^k \right\| \leq \sum_{k=0}^{\infty} \|A\|^k \leq \frac{1}{1-\alpha}$$

2.B Details for Section 2.3.1

Let us show the derivation of the gradients in (2.3.8). For conciseness, we drop the dependence on all arguments, so we will write $\hat{x}_1(\theta;t) = \hat{x}_1$, $\hat{x}_2(\theta;t) = \hat{x}_2$, $\beta(t) = \beta$ and $Q(t) = Q$. Let us also denote $v = M_2^\top M_1 y - \lambda_1 M_3^\top \mathrm{sign}(\hat{x}_2)$ (so that $\hat{x}_1 = Qv$), $P = tP_{\Lambda^c}$ and $W = Q^{-1} = M_3^\top P M_3 + B$. To begin, we have:

$$\begin{aligned} (\hat{x}_1)_k &= \textstyle\sum_a Q_{ka} v_a \\ v_a &= \textstyle\sum_{b,c} (M_2)_{ba}(M_1)_{bc} y_c - \lambda_1 \sum_d (M_3)_{da} \mathrm{sign}(\hat{x}_2)_d \\ W_{ef} &= \textstyle\sum_{gh} (M_3)_{ge} P_{gh} (M_3)_{hf} + \sum_m (M_2)_{me}(M_2)_{mf} + \lambda_2 \delta_{ef} \end{aligned} \tag{2.B.1}$$

In particular:

$$\begin{aligned} \frac{\partial v_a}{\partial (M_1)_{ij}} &= (M_2)_{ia} y_j \\ \frac{\partial v_a}{\partial (M_2)_{ij}} &= \textstyle\sum_c (M_1)_{ic} y_c \delta_{aj} \;=\; (M_1 y)_i \delta_{aj} \\ \frac{\partial v_a}{\partial (M_3)_{ij}} &= -\lambda_1 \mathrm{sign}(\hat{x}_1)_i \delta_{aj} \end{aligned} \tag{2.B.2}$$

Moreover, by applying Proposition 2.A.1 we have:

$$\begin{aligned}
\frac{\partial Q_{ka}}{\partial (M_1)_{ij}} &= -\sum_{ef} Q_{ke} \frac{\partial W_{ef}}{\partial (M_1)_{ij}} Q_{fa} \;=\; 0 \\
\frac{\partial Q_{ka}}{\partial (M_2)_{ij}} &= -\sum_{ef} Q_{ke} \frac{\partial W_{ef}}{\partial (M_2)_{ij}} Q_{fa} \;=\; -\sum_{ef} Q_{ke} \left[(M_2)_{if} \delta_{ej} + (M_2)_{ie} \delta_{fj} \right] Q_{fa} \\
&= -(Q M_2^\top)_{ki} Q_{ja} - Q_{kj} (M_2 Q)_{ia} \\
\frac{\partial Q_{ka}}{\partial (M_3)_{ij}} &= -\sum_{ef} Q_{ke} \frac{\partial W_{ef}}{\partial (M_3)_{ij}} Q_{fa} \;=\; -\sum_{ef} Q_{ke} \left[\sum_h P_{ih} (M_3)_{hf} \delta_{ej} + \sum_g (M_3)_{ge} P_{gi} \delta_{fj} \right] Q_{fa} \\
&= -\sum_{ef} Q_{ke} \left[(P M_3)_{if} \delta_{ej} + (M_3^\top P)_{ei} \delta_{fj} \right] Q_{fa} \;=\; -(Q M_3^\top P)_{ki} Q_{ja} - Q_{kj} (P M_3 Q)_{ia}
\end{aligned} \tag{2.B.3}$$

Combining (2.B.1), (2.B.2), (2.B.3) and observing that $Q^\top = Q$, we obtain:

$$\begin{aligned}
\frac{\partial l(\hat{x}_1)}{\partial (M_1)_{ij}} &= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \frac{\partial (\hat{x}_1)_k}{\partial (M_1)_{ij}} \;=\; \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[\sum_a \left(\frac{\partial Q_{ka}}{\partial (M_1)_{ij}} \right) v_a + Q_{ka} \left(\frac{\partial v_a}{\partial (M_1)_{ij}} \right) \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[\sum_a Q_{ka} (M_2)_{ia} y_j \right] \;=\; \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} (Q M_2^\top)_{ki} y_j \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} (M_2 Q)_{ik} y_j \;=\; (M_2 Q \nabla_x l(\hat{x}_1))_i y_j \;=\; (M_2 \beta)_i y_j \;=\; (M_2 \beta y^\top)_{ij} \\
\frac{\partial l(\hat{x}_1)}{\partial (M_2)_{ij}} &= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \frac{\partial (\hat{x}_1)_k}{\partial (M_2)_{ij}} \;=\; \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[\sum_a \left(\frac{\partial Q_{ka}}{\partial (M_2)_{ij}} \right) v_a + Q_{ka} \left(\frac{\partial v_a}{\partial (M_2)_{ij}} \right) \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[\sum_a \left(-(Q M_2^\top)_{ki} Q_{ja} - Q_{kj} (M_2 Q)_{ia} \right) v_a + Q_{ka} (M_1 y)_i \delta_{aj} \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[-(Q M_2^\top)_{ki} (Q v)_j - Q_{kj} (M_2 Q v)_i + Q_{kj} (M_1 y)_i \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[-(Q M_2^\top)_{ki} (\hat{x}_1)_j - Q_{kj} (M_2 \hat{x}_1)_i + Q_{kj} (M_1 y)_i \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[-(M_2 Q)_{ik} (\hat{x}_1)_j - Q_{kj} (M_2 \hat{x}_1)_i + Q_{kj} (M_1 y)_i \right] \\
&= -(M_2 Q \nabla_x l(\hat{x}_1))_i (\hat{x}_1)_j - (M_2 \hat{x}_1)_i (Q \nabla_x l(\hat{x}_1))_j + (M_1 y)_i (Q \nabla_x l(\hat{x}_1))_j \\
&= -(M_2 \beta)_i (\hat{x}_1)_j - (M_2 \hat{x}_1)_i \beta_j + (M_1 y)_i \beta_j \\
&= -(M_2 \beta (\hat{x}_1)^\top)_{ij} - (M_2 \hat{x}_1 \beta^\top)_{ij} + (M_1 y \beta^\top)_{ij} \\
&= \left[-M_2 \left(\beta (\hat{x}_1)^\top + \hat{x}_1 \beta^\top \right) + M_1 y \beta^\top \right]_{ij}
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial l(\hat{x}_1)}{\partial (M_3)_{ij}} &= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \frac{\partial (\hat{x}_1)_k}{\partial (M_2)_{ij}} = \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[\sum_a \left(\frac{\partial Q_{ka}}{\partial (M_3)_{ij}} \right) v_a + Q_{ka} \left(\frac{\partial v_a}{\partial (M_3)_{ij}} \right) \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[\sum_a \left(-(QM_3^\top P)_{ki} Q_{ja} - Q_{kj} (PM_3 Q)_{ia} \right) v_a - \lambda_1 Q_{ka} \text{sign}(\hat{x}_1)_i \delta_{aj} \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[-(QM_3^\top P)_{ki} (Qv)_j - Q_{kj} (PM_3 Qv)_i - \lambda_1 Q_{kj} \text{sign}(\hat{x}_1)_i \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[-(QM_3^\top P)_{ki} (\hat{x}_1)_j - Q_{kj} (PM_3 \hat{x}_1)_i - \lambda_1 Q_{kj} \text{sign}(\hat{x}_1)_i \right] \\
&= \sum_k \frac{\partial l(\hat{x}_1)}{\partial x_k} \left[-(PM_3 Q)_{ik} (\hat{x}_1)_j - Q_{kj} (PM_3 \hat{x}_1)_i - \lambda_1 Q_{kj} \text{sign}(\hat{x}_1)_i \right] \\
&= -(PM_3 Q \nabla_x l(\hat{x}_1))_i (\hat{x}_1)_j - (PM_3 \hat{x}_1)_i (Q \nabla_x l(\hat{x}_1))_j - \lambda_1 \text{sign}(\hat{x}_1)_i (Q \nabla_x l(\hat{x}_1))_j \\
&= -(PM_3 \beta)_i (\hat{x}_1)_j - (PM_3 \hat{x}_1)_i \beta_j - \lambda_1 \text{sign}(\hat{x}_1)_i \beta_j \\
&= -(PM_3 \beta (\hat{x}_1)^\top)_{ij} - (PM_3 \hat{x}_1 \beta^\top)_{ij} - (\lambda_1 \text{sign}(\hat{x}_1) \beta^\top)_{ij} \\
&= \left[-PM_3 \left(\beta (\hat{x}_1)^\top + \hat{x}_1 \beta^\top \right) - \lambda_1 \text{sign}(\hat{x}_1) \beta^\top \right]_{ij}
\end{aligned}
$$

Nonnegative matrix factorizations

This chapter introduces nonnegative matrix factorizations, covering problem definitions, optimization approaches, and the selection of relevant hyperparameters. We conclude with a brief overview of how these methods serve as a foundational baseline in audio processing.

3.1 Introduction

Nonnegative Matrix Factorization (NMF) is a family of unsupervised learning techniques that aim to approximate a nonnegative data matrix as the product of two lower-rank nonnegative matrices. Introduced in the 1990s by Paatero et al. [103] as an alternative to the established principal component analysis, NMF has since emerged as an effective and flexible data analysis technique. In their influential 1999 paper, Lee and Seung [79], who played a key role in popularizing the method, emphasized how the nonnegativity constraints enable NMF to learn interpretable, observable components while simultaneously inferring latent variables, in a manner reminiscent of neural networks. In particular, the nonnegativity of the modeled quantities is closely related to the fact that neuronal firing rates are inherently nonnegative, motivating the additive and parts-based nature of the representations produced by NMF.

Beyond representation learning, NMF has a well-established connection to clustering and interpretable data decomposition. Early studies demonstrated that NMF can be interpreted as a soft clustering method, where the factor matrices encode cluster centroids and membership degrees [136]. Subsequent work formalized its equivalence to spectral clustering, explaining its effectiveness in partitioning data into meaningful groups [34]. At the same time, the additive, nonnegative nature of the factorization ensures that learned components often correspond to meaningful structures in the data, such as spectral templates in audio processing [118], topics in document analysis [136], or localized features in images [66]. This combination of clustering capability and interpretability makes NMF particularly attractive in domains where the data are naturally nonnegative and compositional, including signal processing, document modeling, bioinformatics, and computer vision. For a comprehensive review of NMF applications in machine learning, we refer the reader to [119] and the references therein.

Over the years, numerous variants of NMF have been proposed to address the limitations of the basic formulation and to better capture specific data characteristics. One major research direction concerns the choice of the cost function or divergence measure, with alternatives such as the Kullback-Leibler divergence [80] and the Itakura-Saito diver-

gence [46] being introduced to better reflect underlying noise models, particularly in audio and speech applications. Further works [47] generalized those measures to the family of β-divergences. Other extensions incorporate regularization terms that promote sparsity [66], smoothness, orthogonality [137], data adherence [83], or temporal continuity [130], thereby improving robustness, interpretability, and task-specific performance.
To explicitly model temporal structure and local dependencies, Nonnegative Matrix Factor Deconvolution (NMFD) [117, 114, 129] has been introduced. This approach is particularly well suited to time-frequency representations, where patterns may repeat over time with local shifts. More recently, deep and multilayer NMF models have been proposed [31, 29, 125], in which multiple factorization layers are stacked to capture hierarchical representations. These models draw conceptual parallels with deep neural networks while retaining the interpretability and optimization transparency of classical NMF formulations.
Overall, NMF and its many extensions constitute a versatile and extensible framework for nonnegative data modeling. Their balance between mathematical simplicity, interpretability, and adaptability has ensured their continued relevance in the literature, where they are widely used both as standalone methods and as baseline models for more advanced approaches.

3.2 Nonnegative Matrix Factorization (NMF)

Let $X \in \mathcal{M}_{m\times n}(\mathbb{R}_{\geq 0})$ be a nonnegative matrix, $D : \mathcal{M}_{m\times n}(\mathbb{R}_{\geq 0}) \times \mathcal{M}_{m\times n}(\mathbb{R}_{\geq 0}) \to \mathbb{R}$ an error measure between matrices, and $r \in \mathbb{N}$ a factorization rank. The problem of computing an NMF of X can then be formulated as the following constrained optimization:

$$\begin{aligned} \min \quad & D(X, WH) \\ \text{s.t.} \quad & W \in \mathcal{M}_{m\times r}(\mathbb{R}_{\geq 0}) \\ & H \in \mathcal{M}_{r\times n}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{3.2.1}$$

In terms of notation, NMF closely resembles traditional dictionary learning. Specifically, the left factor W is commonly referred to as the *dictionary*, whose columns serve as the atoms used to approximate the observation matrix X. Conversely, the right factor H represents the matrix of *activation coefficients*. Due to the nonlinear coupling between the optimization variables and the imposed nonnegativity constraints, problem (3.2.1) is, in general, NP-hard [50]. Polynomial-time solutions can only be computed for specific choices of the error measure D and small factorization ranks. For example, suppose D is the squared Euclidean norm (Frobenius norm), let $r = 1$ and denote $X_1 = \sigma_1 u_1 v_1^\top$ the rank-1 truncated SVD of X. It follows directly from the Eckart-Young Theorem 3.A.1, together with the simple inequality:

$$\|X - wh^\top\|_F \geq \|X - |wh^\top|\|_F = \|X - |w||h|^\top\|_F \tag{3.2.2}$$

that, in this setting, the pair $W = \sqrt{\sigma_1}|u_1|$, $H = \sqrt{\sigma_1}|v_1|^\top$ constitutes an optimal solution to our NMF problem, which can be computed in polynomial time. Interestingly, if the nonnegativity constraints are relaxed and D is again the Euclidean norm, Eckart-Young Theorem immediately provides us with a solution for any r. These constraints are, in

fact, central to the overall NP-hardness of problem (3.2.1).

A commonly used class of error measures in NMF is the β-divergence family. This family forms a β-parameterized set of divergences which, unlike true metrics, are not necessarily symmetric and do not need to satisfy the triangle inequality. Such divergences will be denoted by D_β. In particular, letting:

$$d_\beta(a,b) = \begin{cases} \frac{a}{b} - \log\frac{a}{b} - 1 & \text{if } \beta = 0, \\ a\log\frac{a}{b} - a + b & \text{if } \beta = 1, \\ \frac{1}{\beta(\beta-1)}(a^\beta + (\beta-1)b^\beta - \beta a b^{\beta-1}) & \text{if } \beta \neq 0, 1. \end{cases} \tag{3.2.3}$$

one defines:

$$D_\beta(A,B) = \sum_{i,j} d_\beta(A_{ij}, B_{ij}) \tag{3.2.4}$$

Notable members of the family include the Itakura-Saito divergence ($\beta = 0$), the Kullback-Leibler divergence ($\beta = 1$) and the squared Euclidean norm ($\beta = 2$). Some key properties of the β-divergences, as illustrated in Figure 3.1 below, can be summarized as follows:

- *Convexity.* The map $d_\beta(a,b)$ is convex in the second argument for $\beta \in [1,2]$. This implies that $D_\beta(X, WH)$ is convex in H for W fixed and vice versa, making alternating optimization strategies easier to implement. In general, all β-divergences admit a convex-concave-constant additive split with respect to the second parameter. In the following, this split will be denoted as:

$$d_\beta(a,b) = \check{d}_\beta(a,b) + \hat{d}_\beta(a,b) + \bar{d}_\beta(a,b) \tag{3.2.5}$$

  Table 1 provides a complete overview of each component as a function of β. As we shall see shortly in Section 3.2.1, the derivation of optimization schemes for problem (3.2.1) in the case of β-divergences heavily relies on exploiting this additive decomposition.

- *Positive homogeneity.* All β-divergences are positively homogeneous of degree β, that is, they satisfy:

$$d_\beta(\gamma a, \gamma b) = \gamma^\beta d_\beta(a,b) \quad \forall \gamma > 0,\ \forall a, b \geq 0 \tag{3.2.6}$$

  This implies that as β increases, the β-divergences become more and more sensitive to large input values.

- *Limit behaviour.* For $\beta \leq 1$ it holds $\lim_{b\to 0} d_\beta(a,b) = +\infty$ while for $b \leq a$ we notice that the β-divergences decrease as β increases. This implies that NMF formulations with β-divergences will tend to overapproximate (resp. underapproximate) the entries of X for $\beta \leq 1$ (resp. $\beta \geq 1$). We also remark the slight difference in domain for d_β and $d'_\beta = \frac{\partial d_\beta}{\partial b}$, described in Table 2. An optimization scheme encompassing all β-divergences must take into account that, in some cases, only strictly positive arguments are allowed.

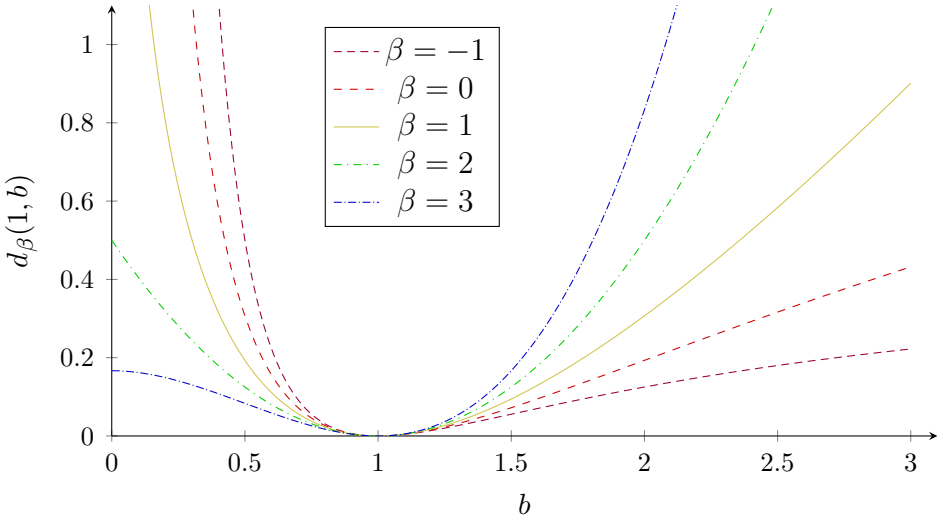


Figure 3.1: Plots of the β-divergences $d_\beta(1, \cdot)$ for $\beta \in \{-1, 0, 1, 2, 3\}$.

Table 1: Convex-concave-constant split of $d_\beta(a, b)$ with respect to b (fixed a) and different values of β.

	$\check{d}_\beta(a, b)$	$\hat{d}_\beta(a, b)$	$\bar{d}_\beta(a)$
$\beta < 1,\ \beta \neq 0$	$-\frac{1}{\beta-1} a b^{\beta-1}$	$\frac{1}{\beta} b^\beta$	$\frac{1}{\beta(\beta-1)} a^\beta$
$\beta = 0$	ab^{-1}	$\log b$	$a(\log a - 1)$
$\beta \in [1, 2]$	$d_\beta(a, b)$	0	0
$\beta > 2$	$\frac{1}{\beta} b^\beta$	$-\frac{1}{\beta-1} a b^{\beta-1}$	$\frac{1}{\beta(\beta-1)} a^\beta$

Table 2: Domains of $d_\beta(a, \cdot)$ (top) and $d'_\beta(a, \cdot)$ (bottom) for different values of a and β.

$\mathbf{dom}(d_\beta)$	$\beta \leq 0$	$\beta \in (0, 1]$	$\beta > 1$
$a = 0$	$\varnothing$	$\mathbb{R}_{\geq 0}$	$\mathbb{R}_{\geq 0}$
$a > 0$	$\mathbb{R}_{>0}$	$\mathbb{R}_{>0}$	$\mathbb{R}_{\geq 0}$

$\mathbf{dom}(d'_\beta)$	$\beta \leq 0$	$\beta \in (0, 1)$	$\beta \in [1, 2)$	$\beta \geq 2$
$a = 0$	$\varnothing$	$\mathbb{R}_{>0}$	$\mathbb{R}_{\geq 0}$	$\mathbb{R}_{\geq 0}$
$a > 0$	$\mathbb{R}_{>0}$	$\mathbb{R}_{>0}$	$\mathbb{R}_{>0}$	$\mathbb{R}_{\geq 0}$

The general NMF formulation in problem (3.2.1) is often adapted to enforce specific *regularizing properties* on the factors W and H. The motivation for introducing such constraints is twofold. First, the standard formulation lacks identifiability: multiple optimal pairs (W, H) may exist, and additional restrictions can help eliminate redundant

solutions. Second, one may wish to design an NMF model where the factors possess distinct physical or analytical interpretations, in which case nonnegativity alone may not be sufficient. A broader class of NMF models that is often used to address these issues is the following:

$$\begin{array}{ll} \min & F(W,H,X) = D(X,WH) + \alpha_W \phi_W(W) + \alpha_H \phi_H(H) \\ \text{s.t.} & W \in \mathcal{M}_{m\times r}(\mathbb{R}_{\geq 0}) \\ & H \in \mathcal{M}_{r\times n}(\mathbb{R}_{\geq 0}) \end{array} \tag{3.2.7}$$

where ϕ_W, ϕ_H are regularizers that promote solutions with the desired properties and $\alpha_W, \alpha_H \geq 0$ are penalty hyperparameters. In [32] the authors propose a bilevel framework for the joint optimization of W, H and α_W, α_H for a particular formulation of problem (3.2.7). We highlight that, when only one of the two factors is regularized, i.e. $\alpha_W = 0$, it is generally advisable to include some other form of regularization on W, and vice versa. Indeed, due to identifiability issues, if (W, H) is a solution pair to (3.2.1), then so is $(\gamma W, \frac{1}{\gamma} H)$ for any $\gamma > 0$. Consequently, without additional regularization, most regularizers tend to drive the penalized factor toward zero while causing the other to grow unbounded in norm. A common strategy to keep both factors bounded during an iterative scheme is to normalize them after each update. For example, one may proceed as follows:

$$W \Leftarrow W\|W\|_F^{-1} \qquad\qquad H \Leftarrow \|W\|_F H \tag{3.2.8}$$

This enforces the dictionary W to maintain unitary norm and, at the same time, the main term in the optimization objective ensures that $\|H\|_F \approx \|X\|_F$. Another approach is to normalize separately each column of W and row of H by a weight vector $w = [w_1, \ldots, w_r]$:

$$W \Leftarrow W\mathrm{diag}(w)^{-1} \qquad\qquad H \Leftarrow \mathrm{diag}(w)H \tag{3.2.9}$$

where $w_s = \|W_{:,s}\|_F$ or $w_s = \|W_{:,s}\|_\infty \ \ \forall s = 1, \ldots, r$. In practice, the numerical stability for this kind of regularization is guaranteed by adding a small positive constant to the entries of w. Some general-purpose regularizers frequently employed in the literature are listed below:

- *Sparsity-promoting.* In order to promote sparse factors it is common practice to exploit the sparsification properties of the 1-norm [93]. Thus, a possible regularizer choice is given by:

$$\begin{array}{rcl} \phi_W(W) & = & \|W\|_1 \\ \phi_H(H) & = & \|H\|_1 \end{array} \tag{3.2.10}$$

- *Smoothness-promoting.* It is not uncommon in applications for the columns of W or rows of H to be discretizations of piecewise smooth functions. In these cases, in order to promote such structure, one may employ the following regularizers:

$$\begin{array}{rcccl} \phi_W(W) & = & \sum\limits_{s=1}^{r}\sum\limits_{i=1}^{m-1}(\,W_{i,s}-W_{i+1,s}\,)^2 & = & \|DW\|_F^2 \\ \phi_H(H) & = & \sum\limits_{s=1}^{r}\sum\limits_{j=1}^{n-1}(\,H_{s,j}-H_{s,j+1}\,)^2 & = & \|DH^\top\|_F^2 \end{array} \tag{3.2.11}$$

where D is a first order finite difference matrix of appropriate dimension.

- *Orthogonality-promoting.* To promote the columns of W or the rows of H to overlap as little as possible [137], orthogonality constraints may be enforced through the regularizers:

$$\begin{array}{rcl} \phi_W(W) & = & \|W^\top W - \mathbb{1}_r\|_F^2 \\ \phi_H(H) & = & \|HH^\top - \mathbb{1}_r\|_F^2 \end{array} \tag{3.2.12}$$

- *Minimum volume-promoting.* In some applications it may be useful to look for factors W as close as possible to the data points contained in X [83]. This can be achieved by enforcing the columns of W to span small volumes. With this in mind, one may use the regularizer:

$$\phi_W(W) = \det(W^\top W) \tag{3.2.13}$$

The choice of the specific value for the parameter β in a β-divergence formulation of problem (3.2.1) often depends on the assumed statistical nature of the observed data distribution [50]. Given $\hat{W} \in \mathcal{M}_{m\times r}(\mathbb{R}_{\geq 0})$ and $\hat{H} \in \mathcal{M}_{r\times n}(\mathbb{R}_{\geq 0})$, assume X to be an observation of a collection of i.i.d. random variables $\widetilde{X}_{ij}$ that depend on the deterministic parameters $(\hat{W}\hat{H})_{ij}$ through the relations $\mathbb{E}[\widetilde{X}_{ij}] = (\hat{W}\hat{H})_{ij}$. For certain distributions satisfying these premises, problem (3.2.1) is equivalent to a Maximum Likelihood Estimation (MLE) of the parameters W and H. More precisely, denoting $p(W, H \mid X)$ the likelihood function, it holds:

$$D_\beta(X, WH) = -\log\, p(W, H \mid X) \tag{3.2.14}$$

In particular:

- *Gamma.* For $\widetilde{X}_{ij} \sim \Gamma\left(k, \frac{(\hat{W}\hat{H})_{ij}}{k}\right)$ (with $k > 0$) we get $\beta = 0$;
- *Poisson.* For $\widetilde{X}_{ij} \sim \mathcal{P}\left((\hat{W}\hat{H})_{ij}\right)$ we get $\beta = 1$;
- *Gaussian.* For $\widetilde{X}_{ij} \sim \mathcal{N}\left((\hat{W}\hat{H})_{ij}, \sigma\right)$ (with $\sigma > 0$) we get $\beta = 2$.

In this setting, the regularized problem (3.2.7) is equivalent to a Maximum a Posteriori (MAP) estimation with exponential priors in the form:

$$p(W) \propto e^{-\alpha_W \phi_W(W)} \qquad\qquad p(H) \propto e^{-\alpha_H \phi_H(H)} \tag{3.2.15}$$

Indeed, since the posterior distribution is obtained by multiplying the likelihood and priors, the objective in (3.2.7) can be written as:

$$F(W, H, X) = -\log\, p(W, H \mid X)p(W)p(H) + C \tag{3.2.16}$$

where C is a normalization constant independent of W, H.

In this work, we will mainly focus on NMF optimization problems where the error distance D is a simple β-divergence. However, in certain applications, the statistical characteristics of the observations may be unknown or susceptible to outliers. In these cases, a noise-robust problem formulation could represent a more appealing option. In [76] the authors propose an $L_{2,1}$ norm NMF formulation as an alternative to the more traditional Frobenius variant. The error measure, in this case, is given by:

$$D(X, WH) = \|X - WH\|_{2,1} = \sum_{j=1}^{n} \|(X - WH)_{:,j}\|_F = \sum_{j=1}^{n} \sqrt{\sum_{i=1}^{m} (X - WH)_{ij}^2} \tag{3.2.17}$$

The presence of the *un*squared Frobenius norm makes the resulting NMF formulation more resistant to outliers in the data points, i.e. the columns of X. From a statistical point of view, the formulation is equivalent to a MLE where the data points are assumed i.i.d. and following a Laplace distribution, that is:

$$\widetilde{X}_{:,j} \sim \mathcal{L}\left((\hat{W}\hat{H})_{:,j}, \lambda \right) \qquad \lambda > 0 \tag{3.2.18}$$

Similarly, by assuming a Cauchy distribution:

$$\widetilde{X}_{:,j} \sim \mathcal{C}\left((\hat{W}\hat{H})_{:,j}, \gamma \right) \qquad \gamma > 0 \tag{3.2.19}$$

one obtains an error measure in the form:

$$D(X, WH) = \|X - WH\|_{2,cau} = \sum_{j=1}^{n} \log\left(\|(X - WH)_{:,j}\|_F^2 + \gamma^2 \right) \tag{3.2.20}$$

which is again more robust to outliers compared to the Frobenius norm. To conclude this section, we highlight the work in [8], where a distributionally robust NMF formulation is proposed. Here, the error measure is a combination of multiple terms accounting for a variety of statistics:

$$D(X, WH) = \max_{\lambda \geq 0,\, \|\lambda\|_1 = 1} \lambda_1 \|X - WH\|_{2,1} + \lambda_2 \|X - WH\|_F^2 + \lambda_{cau} \|X - WH\|_{2,cau} \tag{3.2.21}$$

In the following, we review a general framework for optimizing NMF problems in the form (3.2.7), which also includes all of the aforementioned noise-robust variants. We then provide a detailed derivation of well-known update rules resulting in convergent schemes for β-divergence error measures.

3.2.1 Optimization

A general NMF problem of the form (3.2.7), as noted earlier, is NP-hard. Consequently, algorithms developed for it typically rely on iterative local-optimization schemes aimed at converging to stationary points. A notable family of algorithms used in this setting is the class of *two-block coordinate descent* (2-BCD) methods, in which the variable blocks W and H are optimized alternately, holding one block fixed at each iteration. A high-level overview of this type of approach can be found in Algorithm 3.2.1, where, in particular:

- $\text{UPDATE}_H(W^{(k)}, H^{(k)}, X)$ is an approximate or exact 1-step solution to an iterative subroutine employed to solve:

$$\begin{aligned} \min \quad & F(W^{(k)}, H, X) \\ \text{s.t.} \quad & H \in \mathcal{M}_{r\times n}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{3.2.22}$$

  starting from the initial estimate $H^{(k)}$;

- $\text{UPDATE}_W(W^{(k)}, H^{(k+1)}, X)$ is an approximate or exact 1-step solution to an iterative subroutine employed to solve:

$$\begin{aligned} \min \quad & F(W, H^{(k+1)}, X) \\ \text{s.t.} \quad & W \in \mathcal{M}_{m\times r}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{3.2.23}$$

  starting from the initial estimate $W^{(k)}$.

Algorithm 3.2.1: Two-block coordinate descent scheme for problem (3.2.7)

Input: A matrix $X \in \mathcal{M}_{m\times n}(\mathbb{R}_{\geq 0})$, a factorization rank $r \in \mathbb{N}$ and initial estimates $(W^{(0)}, H^{(0)})$.

Output: An approximate solution to problem (3.2.7).

1: **for** $k = 0, 1, \ldots$ **do:**
2: $\quad H^{(k+1)} = \text{UPDATE}_H(W^{(k)}, H^{(k)}, X)$
3: $\quad W^{(k+1)} = \text{UPDATE}_W(W^{(k)}, H^{(k+1)}, X)$
4: **end for**

If at each iteration both UPDATE_W and UPDATE_H are initialized to the *exact* 1-step solution of their respective subroutines, the resulting algorithm is called *exact* 2-BCD method.
Recalling the structure of the objective function F, it is not unusual for the latter to satisfy some symmetric properties, such as:

$$F(W, H, X) = F(H^\top, W^\top, X^\top) \tag{3.2.24}$$

This is the case, for example, when $F = D_\beta$ or $F = D_\beta + \alpha_W \phi_W + \alpha_H \phi_H$ with $\alpha_W = \alpha_H$ and $\phi_W(\,\cdot\,) = \phi_H(\,\cdot^\top)$. When condition (3.2.24) is satisfied, problems (3.2.23) and (3.2.22) become equivalent. Consequently, Algorithm 3.2.1 can be significantly simplified, since any update rule for H induces an update rule for W, and vice versa. Indeed, given UPDATE_H, step 3 of the algorithm can be replaced by:

$$W^{(k+1)} = \text{UPDATE}_H(H^{(k+1)^\top}, W^{(k)^\top}, X^\top)^\top \tag{3.2.25}$$

Ultimately, such symmetry in the objective function allows us to develop only one update rule for problem (3.2.23) or (3.2.22) instead of two distinct update rules.
A common way [28, 46] to generate iterative updates for the blocks in the 2-BCD framework is to use the following multiplicative rules, which serve as *heuristic* substitutes for UPDATE_H and UPDATE_W:

$$H^{(k+1)} = H^{(k)} \circ \frac{\left[\nabla_H F\left(W^{(k)}, H^{(k)}, X\right)\right]_-}{\left[\nabla_H F\left(W^{(k)}, H^{(k)}, X\right)\right]_+} \tag{3.2.26}$$

$$W^{(k+1)} = W^{(k)} \circ \frac{\left[\nabla_W F\left(W^{(k)}, H^{(k+1)}, X\right)\right]_-}{\left[\nabla_W F\left(W^{(k)}, H^{(k+1)}, X\right)\right]_+} \tag{3.2.27}$$

where $\circ$, $\div$ and $[\,\cdot\,]_\pm$ are the elementwise product, division and positive/negative part. The heuristic nature of (3.2.26)-(3.2.27) is motivated by basic calculus results, namely:

- If $\frac{\partial}{\partial H_{sj}} F\left(W^{(k)}, H^{(k)}, X\right) > 0$ and $H_{sj}^{(k)} > 0$, then a sufficiently small decrease of $H_{sj}^{(k)}$ will decrease the objective. In particular, since:

$$\frac{\left[\frac{\partial}{\partial H_{sj}} F\left(W^{(k)}, H^{(k)}, X\right)\right]_-}{\left[\frac{\partial}{\partial H_{sj}} F\left(W^{(k)}, H^{(k)}, X\right)\right]_+} < 1$$

  we get $H_{sj}^{(k+1)} < H_{sj}^{(k)}$.

- If $\frac{\partial}{\partial H_{sj}} F\left(W^{(k)}, H^{(k)}, X\right) < 0$, then a sufficiently small increase of $H_{sj}^{(k)}$ will decrease the objective. In particular, since:

$$\frac{\left[\frac{\partial}{\partial H_{sj}} F\left(W^{(k)}, H^{(k)}, X\right)\right]_-}{\left[\frac{\partial}{\partial H_{sj}} F\left(W^{(k)}, H^{(k)}, X\right)\right]_+} > 1$$

  we get $H_{sj}^{(k+1)} > H_{sj}^{(k)}$ (if $H_{sj}^{(k)} > 0$).

A similar reasoning holds for the W variables as well. The merit of updates (3.2.26)-(3.2.27) is that they preserve the nonnegativity of the iterates and are compatible with Karush-Kuhn-Tucker (KKT) first order optimality conditions:

$$\begin{aligned} W \geq 0, \qquad & \nabla_W F(W,H,X) \geq 0, \quad W \circ \nabla_W F(W,H,X) = 0 \\ H \geq 0, \qquad & \nabla_H F(W,H,X) \geq 0, \quad H \circ \nabla_H F(W,H,X) = 0 \end{aligned} \tag{3.2.28}$$

Indeed, for a stationary point $(\hat{W}, \hat{H})$ we can have the following configurations:

- $\hat{W}_{is} = 0$ and $\frac{\partial}{\partial W_{is}} F(\hat{W}, \hat{H}, X) \geq 0$;
- $\hat{W}_{is} > 0$ and $\left[\frac{\partial}{\partial W_{is}} F(\hat{W}, \hat{H}, X)\right]_+ = \left[\frac{\partial}{\partial W_{is}} F(\hat{W}, \hat{H}, X)\right]_-$ from the slackness condition in (3.2.28);
- $\hat{H}_{sj} = 0$ and $\frac{\partial}{\partial H_{sj}} F(\hat{W}, \hat{H}, X) \geq 0$;
- $\hat{H}_{sj} > 0$ and $\left[\frac{\partial}{\partial H_{sj}} F(\hat{W}, \hat{H}, X)\right]_+ = \left[\frac{\partial}{\partial H_{sj}} F(\hat{W}, \hat{H}, X)\right]_-$ from the slackness condition in (3.2.28).

As a consequence, we deduce that $(\hat{W}, \hat{H})$ is always a fixed point for (3.2.26)-(3.2.27). Alternatively, the same update rules can also be derived by performing a single iteration of rescaled gradient descent on problems (3.2.23)-(3.2.22), choosing the step size so that the updated estimates remain nonnegative. We remark that a drawback of multiplicative updates such as the ones above is the zero-locking phenomenon. In fact, multiplicative updates cannot modify entries of the blocks that have been set to zero. This seemingly small detail raises major concerns about the convergence of the resulting algorithm to stationary points. Indeed, from the KKT slackness conditions we deduce that some stationary points may have strictly positive entries. Such points can never be reached if, during optimization, those entries become zero. As we will see at the end of the section, this issue is usually addressed by artificially maintaining the entries of new iterates above a fixed threshold.

The heuristic updates described above are particularly appealing when the objective function F is complex, as they provide a simple way to implement 2-BCD schemes with minimal assumptions, namely, the differentiability of F. In the remainder of the section we will briefly present a more mathematically grounded alternative to design the update maps UPDATE_H and UPDATE_W, the properties of which are crucial for the convergence of the overall 2-BCD scheme. In Appendix 3.B we include some of the main convergence theorems for n-BCD and comment on their applicability to our general NMF problem. Here, however, we focus specifically on 2-BCD and the main framework we will use is that of Smooth Majorization-Minimization (SMM) applied to the particular case of $F = D_\beta$. Given the symmetry in the objective considered and our remark leading to relation (3.2.25), it suffices to derive UPDATE_H. We can therefore assume $W^{(k)}$ to be fixed and $H^{(k)}$ to be given. The problem of updating H to time $k+1$ can then be further simplified by splitting the optimization of (3.2.22) between columns[1]. Denoting $x = X_{:,j}$, $h = H_{:,j}$, $h^{(k)} = H^{(k)}_{:,j}$ and $x^{(k)} = (W^{(k)}H^{(k)})_{:,j} = W^{(k)}h^{(k)}$ for any $1 \leq j \leq n$, our optimization problem boils down to:

[1]The β-divergence d_β is applied elementwise to the input matrices and then summed. The resulting objective D_β can be therefore additively split between columns, each term involving only one column of H, which can be optimized independently.

$$\begin{aligned} \min \quad & f(h) = \sum_{i=1}^{m} d_\beta(\, x_i, W_{i,:}^{(k)} h\,) \\ \text{s.t.} \quad & h \in \mathbb{R}^r_{\geq 0} \end{aligned} \tag{3.2.29}$$

In order to tackle it, we employ an SMM framework; see Appendix 3.B and [50, 47] for more details. In particular, we seek a (smooth) majorizer $g(h; h^{(k)})$ for $f(h)$ whose optimum $h = h^{(k+1)}$ can be easily computed. This is the case, for example, when g is convex. Recalling the convex-concave-constant split for β-divergences described in Table 1, one can easily obtain such a function by majorizing each term in the split with a convex function. More precisely:

- *Convex term.* $\check{d}_\beta(\, x_i, W_{i,:}^{(k)} h\,)$ can be majorized using Jensen's inequality:

$$\check{d}_\beta(\, x_i, W_{i,:}^{(k)} h\,) \leq \sum_{s\,\in\,\mathcal{S}} \lambda_{is}^{(k)} \check{d}_\beta\left(\, x_i, \frac{W_{is}^{(k)} h_s}{\lambda_{is}^{(k)}}\,\right) = \sum_{s=1}^{r} \lambda_{is}^{(k)} \check{d}_\beta\left(\, x_i, \frac{W_{is}^{(k)} h_s}{\lambda_{is}^{(k)}}\,\right) \tag{3.2.30}$$

  where $\mathcal{S} = \{s \,:\, W_{is}^{(k)} \neq 0\}$ and $\lambda_{is}^{(k)} = \dfrac{W_{is}^{(k)} h_s^{(k)}}{x_i^{(k)}}$. Notice that $\sum\limits_{s\,\in\,\mathcal{S}} \lambda_{is}^{(k)} = 1$ and $\lambda_{is}^{(k)} = 0$ for $s \notin \mathcal{S}$.

- *Concave term.* $\hat{d}_\beta(\, x_i, W_{i,:}^{(k)} h\,)$ can be majorized by its first order Taylor approximation:

$$\hat{d}_\beta(\, x_i, W_{i,:}^{(k)} h\,) \leq \hat{d}_\beta(\, x_i, x_i^{(k)}\,) + \hat{d}'_\beta(\, x_i, x_i^{(k)}\,) W_{i,:}^{(k)} (h - h^{(k)}) \tag{3.2.31}$$

- *Constant term.* $\bar{d}_\beta(x_i)$ can be majorized by itself.

The overall majorizer for $f(h)$ at $h^{(k)}$ is therefore given by:

$$g(h; h^{(k)}) = \sum_{i=1}^{m} \left[\sum_{s=1}^{r} \lambda_{is}^{(k)} \check{d}_\beta\left(\, x_i, \frac{W_{is}^{(k)} h_s}{\lambda_{is}^{(k)}}\,\right) \right] + \Big[\, \hat{d}_\beta(\, x_i, x_i^{(k)}\,) + \hat{d}'_\beta(\, x_i, x_i^{(k)}\,) W_{i,:}^{(k)} (h - h^{(k)}) \,\Big] + \bar{d}_\beta(x_i) \tag{3.2.32}$$

Since (3.2.32) is convex, the minimizer $h = h^{(k+1)}$ is readily obtained by setting its gradient to zero. Using Table 1 and the derivatives of the maps therein, one obtains the following multiplicative update:

$$h^{(k+1)} = h^{(k)} \circ \left(\frac{W^{(k)\top} \left(x^{(k)\circ(\beta-2)} \circ x \right)}{W^{(k)\top} x^{(k)\circ(\beta-1)}} \right)^{\circ\gamma(\beta)} \tag{3.2.33}$$

where $(\,\cdot\,)^{\circ\alpha}$ denotes the elementwise exponentiation by α and:

$$\gamma(\beta) = \begin{cases} \dfrac{1}{2-\beta} & \text{if} \quad \beta < 1 \\ 1 & \text{if} \quad 1 \leq \beta \leq 2 \\ \dfrac{1}{\beta - 1} & \text{if} \quad \beta > 2 \end{cases} \tag{3.2.34}$$

Joining updates (3.2.33) for all columns finally yields the complete update rule for H:

$$H^{(k+1)} = H^{(k)} \circ \left(\frac{W^{(k)^\top} \left(\left(W^{(k)} H^{(k)} \right)^{\circ(\beta-2)} \circ X \right)}{W^{(k)^\top} \left(W^{(k)} H^{(k)} \right)^{\circ(\beta-1)}} \right)^{\circ\gamma(\beta)} \tag{3.2.35}$$

Similarly, using (3.2.25), the update rule for W reads:

$$W^{(k+1)} = W^{(k)} \circ \left(\frac{\left(\left(W^{(k)} H^{(k+1)} \right)^{\circ(\beta-2)} \circ X \right) H^{(k+1)^\top}}{\left(W^{(k)} H^{(k+1)} \right)^{\circ(\beta-1)} H^{(k+1)^\top}} \right)^{\circ\gamma(\beta)} \tag{3.2.36}$$

Interestingly, it can be shown that these updates differ from the heuristic ones (3.2.26)-(3.2.27) only by the exponent $\gamma(\beta)$. In particular, for $\beta \in [1, 2]$ the two coincide. The remaining issues to resolve for updates (3.2.35)-(3.2.36) concern their range of validity with respect to β and the occurrence of zero-locking. Indeed, as Table 2 shows, some β-divergences require their arguments to be strictly positive. Related to this issue and given the multiplicative nature of these updates, the zero-locking phenomenon remains a troublesome obstacle for the convergence of the iterates to a stationary point. Fortunately, both concerns can be quickly addressed by simply relaxing the nonnegativity assumption [50]. In particular, by choosing a small $\varepsilon > 0$, it suffices to replace our original NMF problem:

$$\begin{array}{ll} \min & D_\beta(X, WH) \\ \text{s.t.} & W \in \mathcal{M}_{m\times r}(\mathbb{R}_{\geq\varepsilon}) \\ & H \in \mathcal{M}_{r\times n}(\mathbb{R}_{\geq\varepsilon}) \end{array} \tag{3.2.37}$$

and to project the updated iterates at each step so that they remain at least ε-away from zero:

$$\begin{aligned} H^{(k+1)} &= \max\left(\varepsilon,\, H^{(k)} \circ \left(\frac{W^{(k)^\top} \left(\left(W^{(k)} H^{(k)} \right)^{\circ(\beta-2)} \circ X \right)}{W^{(k)^\top} \left(W^{(k)} H^{(k)} \right)^{\circ(\beta-1)}} \right)^{\circ\gamma(\beta)} \right) \\ W^{(k+1)} &= \max\left(\varepsilon,\, W^{(k)} \circ \left(\frac{\left(\left(W^{(k)} H^{(k+1)} \right)^{\circ(\beta-2)} \circ X \right) H^{(k+1)^\top}}{\left(W^{(k)} H^{(k+1)} \right)^{\circ(\beta-1)} H^{(k+1)^\top}} \right)^{\circ\gamma(\beta)} \right) \end{aligned} \tag{3.2.38}$$

In applications, it is common practice to let $\varepsilon \approx 10^{-16}$. The convergence of Algorithm 3.2.1 is then guaranteed, in this specific setting, by applying Theorem 3.B.3 in the case of $n = 2$ blocks.

In relation to the contents of this section, we highlight the work in [94], where a joint Majorization-Minimization approach is proposed. Rather than employing a 2-BCD framework, where the H and W subproblems are treated separately, the authors derive a joint optimization framework where the majorizer g acts on both H and W at the same time. The resulting updates are mostly similar to the ones we included above, but slightly more efficient in terms of computation thanks to the caching of certain components during each iteration.

3.2.2 Choice of the rank

Choosing the optimal rank r for a specific NMF problem formulation is a critical step, as the rank affects the resulting model complexity and interpretability. This choice is application dependent since it is contingent on the intrinsic properties of the data matrix X that we intend to factorize. In this setting, expert knowledge is often the most common approach, but it relies on prior data analysis. In general, there is no single consensus method, and in the literature, one can find many different techniques. For instance, heuristics based on the Reconstruction Error [82] or the Cophenetic Correlation Coefficient [16] are both popular options, thanks to their simplicity. The former chooses the rank as the elbow point of the reconstruction error curve, while the latter measures the stability of the clustering results from multiple NMF runs with different random initializations. Other more data-driven approaches involve Cross-Validation [101] or the application of renowned parsimony criteria, such as the Minimum Description Length [61]. In this section, however, we briefly review a famous Bayesian framework introduced in [124], the Automatic Relevance Determination (ARD).
In ARD, the r rank-1 factors obtained by coupling each column of W with the corresponding row of H are associated with a *relevance weight*, which is updated alongside the factors during NMF inference. Let us denote $\lambda_s \geq 0 \ \forall s = 1, \dots, r$ such weights and $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_r)$. As the name suggests, $\boldsymbol{\lambda}$ measures the relevance of each rank-1 factor in the overall reconstruction of the data: if at any point λ_s goes below a given threshold $\tau > 0$, the corresponding factor is pruned. The optimal rank $\hat{r}$ will be the number of rank-1 factors that have survived this pruning. Let us now see this method in more detail.
In order to tie the rank-1 factors to the weights, the authors in [124] propose introducing the priors $p(W_{as}|\lambda_s)$ and $p(H_{sb}|\lambda_s)$ on the entries of W and H. Specifically, they adopt either Half-Normal (ℓ_2-ARD) or Exponential (ℓ_1-ARD) priors in the form:

$$\begin{aligned} p(x|\lambda) &= \mathcal{HN}(x|\lambda) = \left(\tfrac{2}{\pi\lambda}\right)^{\frac{1}{2}} e^{-\frac{x^2}{2\lambda}} \\ p(x|\lambda) &= \mathcal{E}(x|\lambda) = \tfrac{1}{\lambda} e^{-\frac{x}{\lambda}} \end{aligned} \tag{3.2.39}$$

Moreover, they assume the weights to be independent and inverse-Gamma distributed:

$$p(\lambda; a, b) = \mathcal{IG}(\lambda; a, b) = \frac{b^a}{\Gamma(a)} \lambda^{-(a+1)} e^{-\frac{b}{\lambda}} \tag{3.2.40}$$

for some shape and scale hyperparameters $a, b > 0$. In this framework, the MAP objective function is readily given by:

$$F(W, H, X, \boldsymbol{\lambda}) = D_\beta(X, WH) + \sum_{s=1}^{r} \left(\frac{f(W_{:,s}) + f(H_{s,:}) + b}{\lambda_s} + c \log \lambda_s \right) \tag{3.2.41}$$

where $f(x) = \frac{1}{2}\|x\|_2^2$, $c = \frac{n+m}{2} + a + 1$ for the Half-Normal case and $f(x) = \|x\|_1$, $c = n + m + a + 1$ for the Exponential case. In essence, we notice that ARD imposes group sparsity on the rank-1 factors through the *joint* regularizer:

$$R(W, H, \boldsymbol{\lambda}) = \sum_{s=1}^{r} \left(\frac{f(W_{:,s}) + f(H_{s,:}) + b}{\lambda_s} + c \log \lambda_s \right) \tag{3.2.42}$$

By applying an SMM framework to derive updates for W, H, as well as updating $\boldsymbol{\lambda}$ by simply setting to zero the gradient of F, one obtains the following scheme for ℓ_2-ARD:

$$\begin{aligned}
H^{(k+1)} &= H^{(k)} \circ \left(\frac{W^{(k)^\top} \left(\left(W^{(k)} H^{(k)} \right)^{\circ(\beta-2)} \circ X \right)}{W^{(k)^\top} \left(W^{(k)} H^{(k)} \right)^{\circ(\beta-1)} + \dfrac{H^{(k)}}{\mathrm{repmat}(\boldsymbol{\lambda}^{(k)}, 1, n)}} \right)^{\circ\xi(\beta)} \\
W^{(k+1)} &= W^{(k)} \circ \left(\frac{\left(\left(W^{(k)} H^{(k+1)} \right)^{\circ(\beta-2)} \circ X \right) H^{(k+1)^\top}}{\left(W^{(k)} H^{(k+1)} \right)^{\circ(\beta-1)} H^{(k+1)^\top} + \dfrac{W^{(k)}}{\mathrm{repmat}(\boldsymbol{\lambda}^{(k)}, m, 1)}} \right)^{\circ\xi(\beta)} \\
\lambda_s^{(k+1)} &= \frac{\frac{1}{2}\|W_{:,s}^{(k+1)}\|_2^2 + \frac{1}{2}\|H_{s,:}^{(k+1)}\|_2^2 + b}{c} \qquad \forall s = 1, \ldots, r
\end{aligned} \tag{3.2.43}$$

and, similarly, for ℓ_1-ARD:

$$\begin{aligned}
H^{(k+1)} &= H^{(k)} \circ \left(\frac{W^{(k)^\top} \left(\left(W^{(k)} H^{(k)} \right)^{\circ(\beta-2)} \circ X \right)}{W^{(k)^\top} \left(W^{(k)} H^{(k)} \right)^{\circ(\beta-1)} + \dfrac{\mathbb{1}_{r \times n}}{\mathrm{repmat}(\boldsymbol{\lambda}^{(k)}, 1, n)}} \right)^{\circ\gamma(\beta)} \\
W^{(k+1)} &= W^{(k)} \circ \left(\frac{\left(\left(W^{(k)} H^{(k+1)} \right)^{\circ(\beta-2)} \circ X \right) H^{(k+1)^\top}}{\left(W^{(k)} H^{(k+1)} \right)^{\circ(\beta-1)} H^{(k+1)^\top} + \dfrac{\mathbb{1}_{m \times r}}{\mathrm{repmat}(\boldsymbol{\lambda}^{(k)}, m, 1)}} \right)^{\circ\gamma(\beta)} \\
\lambda_s^{(k+1)} &= \frac{\|W_{:,s}^{(k+1)}\|_1 + \|H_{s,:}^{(k+1)}\|_1 + b}{c} \qquad \forall s = 1, \ldots, r
\end{aligned} \tag{3.2.44}$$

where $\gamma(\beta)$ is defined as in (3.2.34) and:

$$\xi(\beta) = \begin{cases} \dfrac{1}{3-\beta} & \text{if} \quad \beta \leq 2 \\ \dfrac{1}{\beta - 1} & \text{if} \quad \beta > 2 \end{cases} \tag{3.2.45}$$

Upon completion of the optimization process, since $\lambda_s^{(k)} \geq B = \dfrac{b}{c}$, the optimal rank will be given by:

$$\hat{r} = \left| \left\{ s \in \{1, \ldots, r\} \, : \, \frac{\lambda_s^{(k)} - B}{B} > \tau \right\} \right| \tag{3.2.46}$$

The authors also propose a rule for selecting the scale hyperparameter $b > 0$. In particular, by the law of large numbers (assuming $mn \gg 1$), it holds:

$$\mu_X = \frac{1}{nm} \sum_{ij} X_{ij} \approx \mathbb{E}[X_{ij}] = \mathbb{E}[(WH)_{ij}] = \sum_{s=1}^{r} \mathbb{E}[W_{is} H_{sj}] \tag{3.2.47}$$

where μ_X can be computed directly from the data matrix X. Similarly, one can derive a closed form for the right-hand side of (3.2.47) by using the moments of the distributions (3.2.39), (3.2.40). Indeed, since:

$$\mathbb{E}[W_{is} H_{sj}] = \mathbb{E}_{\lambda_s}\big[\, \mathbb{E}[W_{is}|\lambda_s]\, \mathbb{E}[H_{sj}|\lambda_s] \,\big] \tag{3.2.48}$$

and:

$$\begin{aligned} \mathbb{E}[H_{sj}|\lambda_s] &= \mathbb{E}[W_{is}|\lambda_s] = \begin{cases} \left(\dfrac{2\lambda_s}{\pi}\right)^{\frac{1}{2}} & \text{for} \quad \ell_2\text{-ARD} \\ \lambda_s & \text{for} \quad \ell_1\text{-ARD} \end{cases} \\ \mathbb{E}[\lambda_s] &= \frac{b}{a-1} \\ \mathbb{E}[\lambda_s^2] &= \frac{b^2}{(a-1)(a-2)} \end{aligned} \tag{3.2.49}$$

by combining (3.2.47), (3.2.48) and (3.2.49) we obtain:

$$\mu_X = \begin{cases} \dfrac{2rb}{\pi(a-1)} & \text{for} \quad \ell_2\text{-ARD} \\ \dfrac{rb^2}{(a-1)(a-2)} & \text{for} \quad \ell_1\text{-ARD} \end{cases} \tag{3.2.50}$$

The relations above can then be transformed to yield the desired rule for b:

$$\hat{b} = \begin{cases} \dfrac{\pi(a-1)\mu_X}{2r} & \text{for} \quad \ell_2\text{-ARD} \\ \sqrt{\dfrac{(a-1)(a-2)\mu_X}{r}} & \text{for} \quad \ell_1\text{-ARD} \end{cases} \tag{3.2.51}$$

Example

The Swimmer dataset [35] consists of $n = 256$ synthetic, vectorized images (20×11 pixels each). Each image shows a stylized swimmer with an invariant torso and four articulated limbs, each of which can take one of four distinct configurations. In Figure 3.2, some of the dataset's elements are shown. The data matrix X containing such vectorized images has an intrinsically low-rank $\hat{r} = 16$.
In Figure 3.3, we show the results obtained by applying ℓ_1-ARD to this dataset. We chose $\beta = 1$ and added Poisson noise to the input data matrix, initial rank $r = 32$, shape hyperparameter $a = 10$ and threshold $\tau = 10^{-6}$.

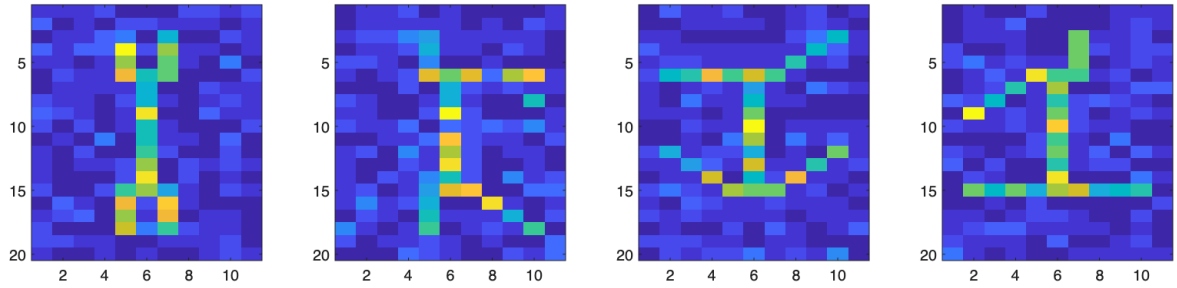


Figure 3.2: Sample of noisy images from the swimmer dataset.

3.3 Nonnegative Matrix Factor Deconvolution (NMFD)

Let $X \in \mathcal{M}_{m\times n}(\mathbb{R}_{\geq 0})$ be a nonnegative matrix, $D : \mathcal{M}_{m\times n}(\mathbb{R}_{\geq 0}) \times \mathcal{M}_{m\times n}(\mathbb{R}_{\geq 0}) \to \mathbb{R}$ an error measure between matrices, and $r_j \in \mathbb{N}$ for $j = 1, \ldots, J$ some factorization ranks. The problem of computing an NMFD of X can then be formulated as the following constrained optimization:

$$\begin{aligned} \min \quad & D(X, W * H) \\ \text{s.t.} \quad & W(j) \in \mathcal{M}_{m\times r_j}(\mathbb{R}_{\geq 0}) \qquad \forall j = 1, \ldots, J \\ & H(j) \in \mathcal{M}_{1\times n}(\mathbb{R}_{\geq 0}) \qquad \forall j = 1, \ldots, J \end{aligned} \tag{3.3.1}$$

where the convolution product $*$ is defined by:

$$W * H = \sum_{j=1}^{J} \sum_{r=0}^{r_j - 1} W(j)_{:,r} \overrightarrow{H(j)}^{r} \tag{3.3.2}$$

and $\overrightarrow{\cdot}^{r}$ denotes the column-wise right-shift operator of r columns, padding the r left-most columns with zeroes. An analogous notation will be used for the left-shift operator $\overleftarrow{\cdot}^{r}$. Unlike NMF, where each row of H is responsible for the activation of a single column of W, in NMFD, the rows $H(j)$ of the right factor are associated with an entire dictionary $W(j)$. The columns of $W(j)$ are then activated in succession, one after the other, whenever a coefficient in $H(j)$ is non-zero. Each $W(j)$ is therefore a collection of r_j temporally-correlated columns[2]. The appeal of NMFD resides in its ability to model *correlation* between data points (i.e. the columns of X). As we will briefly see in Section 3.4, this property is especially relevant when the data points form a time-series of features. We remark that in this particular NMFD formulation, each dictionary $W(j)$ has its own rank r_j. This idea was first proposed in [9] as a generalization of the work in [117], which introduced NMFD. We also note that the NMFD framework described in (3.3.1), (3.3.2) can be extended to handle convolutions along both input dimensions, i.e. features and data points. Indeed, given some maximal feature shifts $s_j \in \mathbb{N}$ for $j = 1, \ldots, J$, one could also consider the optimization:

[2]For notational simplicity, in NMFD we label the columns of each $W(j)$ starting from zero.

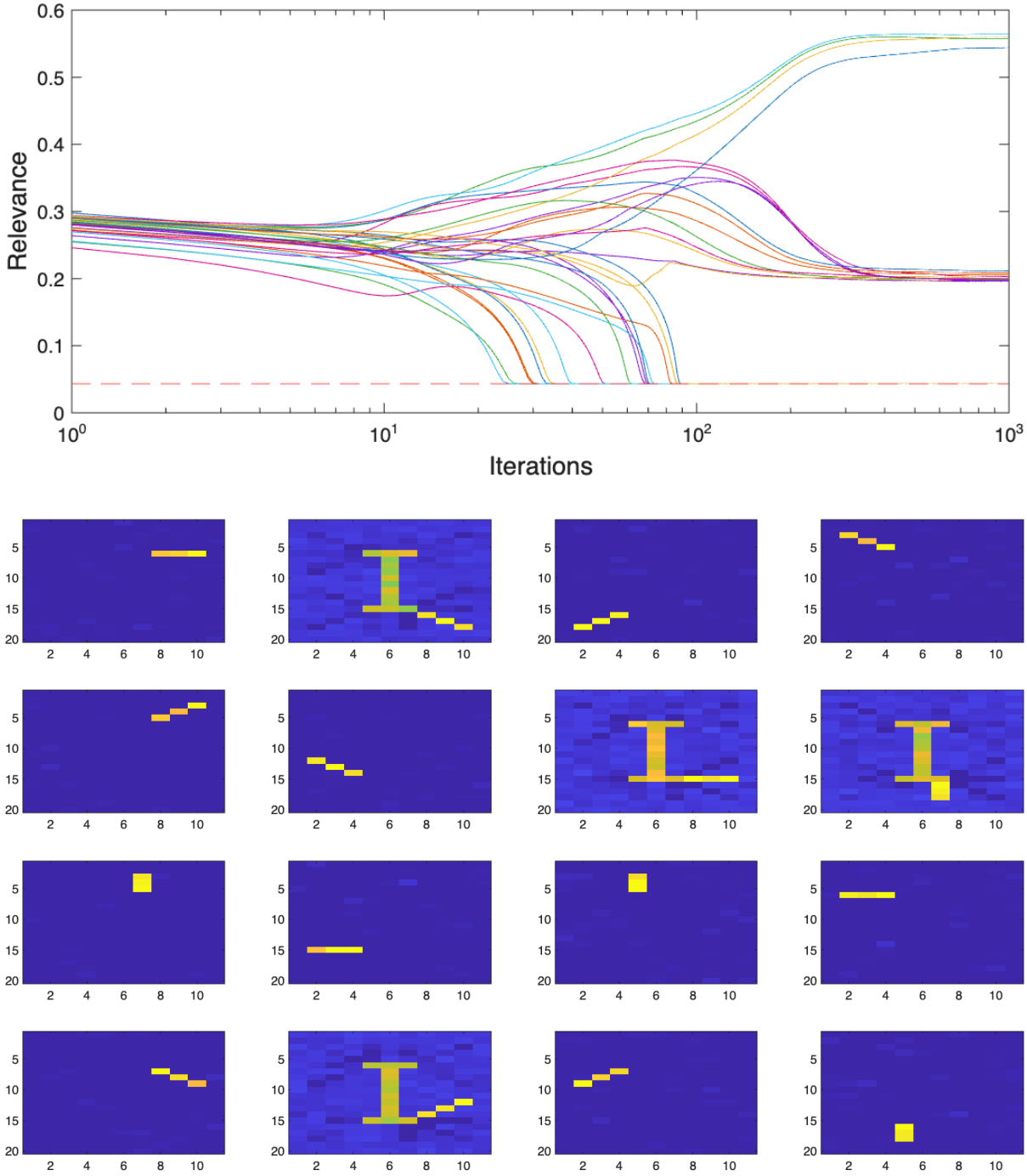


Figure 3.3: Top: values of $\lambda_s^{(k)}$ along iterations. The theoretical lower bound B is the dashed red line. Bottom: unvectorized columns of W that survived the pruning.

$$\begin{aligned} \min \quad & D(X, W * H) \\ \text{s.t.} \quad & W(j) \in \mathcal{M}_{m \times r_j}(\mathbb{R}_{\geq 0}) \qquad \forall j = 1, \ldots, J \\ & H(j) \in \mathcal{M}_{s_j \times n}(\mathbb{R}_{\geq 0}) \qquad \forall j = 1, \ldots, J \end{aligned} \tag{3.3.3}$$

where now $*$ also cycles along the features:

$$W * H = \sum_{j=1}^{J} \sum_{r=0}^{r_j-1} \sum_{s=0}^{s_j-1} \overset{\downarrow s}{W(j)}_{:,r} \overset{\longrightarrow r}{H(j)}_{s,:} \tag{3.3.4}$$

Here, $\overset{\downarrow s}{\cdot}$ denotes the row-wise down-shift operator of s rows, padding the s top-most rows with zeroes. Similar NMFD models [114, 129] have been explored in the case of both constant ranks $r_j \equiv r$ and constant shifts $s_j \equiv s$. To the authors' knowledge, the specific custom-sized formulation in (3.3.3), (3.3.4) has not yet been explored and is left for future work.
Just like NMF, in NMFD the family of β-divergences is again among the most popular choices for the error measure D. In fact, everything that was presented in Section 3.2 still holds true with minor adjustments. Although less common compared to NMF, regularization of the factors is also an option in NMFD, leading to a general problem in the form:

$$\begin{aligned} \min \quad & F(W,H,X) = D(X, W*H) + \alpha_W \phi_W(W) + \alpha_H \phi_H(H) \\ \text{s.t.} \quad & W(j) \in \mathcal{M}_{m\times r_j}(\mathbb{R}_{\geq 0}) \qquad \forall j = 1,\dots,J \\ & H(j) \in \mathcal{M}_{1\times n}(\mathbb{R}_{\geq 0}) \qquad \forall j = 1,\dots,J \end{aligned} \tag{3.3.5}$$

With a slight abuse of notation, throughout this section we may write $W = \{W(j)\}_{j=1,\dots,J}$ and $H = \{H(j)\}_{j=1,\dots,J}$.

3.3.1 Optimization

Generalizing Section 3.2.1, NMFD problems in the form (3.3.5) can be tackled by $2J$-BCD methods, where $H(j)$ and $W(j)$ are treated as individual blocks and updated in succession. A high-level overview of this type of approach can be found in Algorithm 3.3.1, where, in particular:

- $\text{UPDATE}_{H(j)}(W^{(k)}, H^{(k)}, X)$ is an approximate or exact 1-step solution to an iterative subroutine employed to solve:

$$\begin{aligned} \min \quad & F(W^{(k)}, H, X) \\ \text{s.t.} \quad & H(j) \in \mathcal{M}_{1\times n}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{3.3.6}$$

  starting from the initial estimate $H^{(k)}$;

- $\text{UPDATE}_{W(j)}(W^{(k)}, H^{(k+1)}, X)$ is an approximate or exact 1-step solution to an iterative subroutine employed to solve:

$$\begin{aligned} \min \quad & F(W, H^{(k+1)}, X) \\ \text{s.t.} \quad & W(j) \in \mathcal{M}_{m\times r_j}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{3.3.7}$$

  starting from the initial estimate $W^{(k)}$.

Algorithm 3.3.1: $2J$-block coordinate descent scheme for problem (3.3.5)

Input: A matrix $X \in \mathcal{M}_{m\times n}(\mathbb{R}_{\geq 0})$, factorization ranks $r_j \in \mathbb{N}$ for $j = 1, \dots, J$ and initial estimates $(W^{(0)}, H^{(0)})$.

Output: An approximate solution to problem (3.3.5).

```
for k = 0, 1, ... do:
    for j = 1, ..., J do:
        H(j)^(k+1) = UPDATE_H(j)(W^(k), H^(k), X)
    end for
    for j = 1, ..., J do:
        W(j)^(k+1) = UPDATE_W(j)(W^(k), H^(k+1), X)
    end for
end for
```

We highlight that Algorithm 3.3.1, as reported here, slightly deviates from standard $2J$-BCD schemes. Indeed, in a traditional $2J$-BCD scheme, one should update $H(j)$ (resp. $W(j)$) to time $k+1$ starting from $\{H(1)^{(k+1)}, \dots, H(j-1)^{(k+1)}, H(j)^{(k)}, \dots, H(J)^{(k)}\}$ (resp. $\{W(1)^{(k+1)}, \dots, W(j-1)^{(k+1)}, W(j)^{(k)}, \dots, W(J)^{(k)}\}$), while here we use $H^{(k)}$ (resp. $W^{(k)}$). This is commonplace in practical implementations. In fact, most update rules require computing the reconstructed data matrix $\hat{X} = W * H$ after each block update. This slight variation allows the matrix $\hat{X}$ to be reused across multiple steps, significantly reducing the computational burden of the convolution operation.
Perhaps the most significant difference compared to NMF lies in the choice of update rules. Whereas the SMM framework is by far the most common approach for NMF, NMFD most often relies on heuristic update rules analogous to (3.2.26) and (3.2.27):

$$H(j)^{(k+1)} = H(j)^{(k)} \circ \frac{\left[\nabla_{H(j)} F\left(W^{(k)}, H^{(k)}, X\right)\right]_-}{\left[\nabla_{H(j)} F\left(W^{(k)}, H^{(k)}, X\right)\right]_+} \qquad \forall j = 1, \dots, J \tag{3.3.8}$$

$$W(j)^{(k+1)} = W(j)^{(k)} \circ \frac{\left[\nabla_{W(j)} F\left(W^{(k)}, H^{(k+1)}, X\right)\right]_-}{\left[\nabla_{W(j)} F\left(W^{(k)}, H^{(k+1)}, X\right)\right]_+} \qquad \forall j = 1, \dots, J \tag{3.3.9}$$

In the case $F = D_\beta$, one can easily obtain [129] the following:

$$H(j)^{(k+1)} = H(j)^{(k)} \circ \frac{\sum_{r=0}^{r_j-1} W(j)_{:,r}^{(k)\top} \overleftarrow{\left(\left(W^{(k)} * H^{(k)}\right)^{\circ(\beta-2)} \circ X\right)}^{r}}{\sum_{r=0}^{r_j-1} W(j)_{:,r}^{(k)\top} \overleftarrow{\left(\left(W^{(k)} * H^{(k)}\right)^{\circ(\beta-1)}\right)}^{r}} \qquad \forall j = 1, \ldots, J \tag{3.3.10}$$

$$W(j)^{(k+1)} = W(j)^{(k)} \circ \left[\ldots \quad \frac{\left(\left(W^{(k)} * H^{(k)}\right)^{\circ(\beta-2)} \circ X\right) \overrightarrow{H(j)^{(k)}}^{r\top}}{\left(W^{(k)} * H^{(k)}\right)^{\circ(\beta-1)} \overrightarrow{H(j)^{(k)}}^{r\top}} \quad \ldots \right]_{r=0,\ldots,r_j-1} \qquad \forall j = 1, \ldots, J \tag{3.3.11}$$

3.3.2 Choice of the rank and number of dictionaries

Choosing the correct number of dictionaries, as well as their ranks, is crucial for NMFD to produce the desired separation of the input data. For specific applications, expert knowledge is often sufficient to identify the correct number of dictionaries J, as it involves estimating the number of different patterns (or sources) present in X. Estimating the ranks r_j for each pattern, on the other hand, is still a delicate matter. In this context, NMFD carries an intrinsic ambiguity: a given pattern can either be inserted as a whole into a dictionary $W(j)$ and reconstructed in $\hat{X} = W * H$ using a single coefficient of $H(j)$, or it can be encoded as a low-rank pattern that requires multiple coefficient activations. Both cases have their merits: the former prioritizes data fidelity and the sparsity of the right factor, while the latter trades such sparsity for a more parsimonious representation of the data. If data fidelity is the main concern, however, one can still recover it even in the latter case by postprocessing the final dictionary and activations. For example, one may compress consecutive activations into a single activation and modify the dictionary accordingly.

Although most techniques valid in an NMF setting can still be applied to NMFD, the problem of estimating r_j and, in particular, J, remains mostly unexplored. The closest work on this topic can be found in [75], where either J is estimated given $r_j \equiv r$, or vice versa. In this section, we propose a possible approach for the *joint* estimation of J and r_j by generalizing the ARD method presented in Section 3.2.2 to an NMFD setting.

Just like in ARD, we assume a given upper bound J on the number of dictionaries, as well as upper bounds $r_j \ \forall j = 1, \ldots, J$ for their ranks. The goal is to find the optimal $\hat{J}$ and $\hat{r}_j$. We then assign some dictionary-specific weights to each column of $W(j)$, namely, $\boldsymbol{\lambda}(j) = (\lambda_1(j), \ldots, \lambda_{r_j}(j)) \ \forall j = 1, \ldots, J$. Unlike before, each row $H(j)$ is paired to a whole dictionary $W(j)$, rather than just a column; hence, the priors we place on the entries of H must reflect this difference. We choose priors:

$$\begin{aligned} p(W(j)_{ar}|\boldsymbol{\lambda}(j)) &= p\left(W(j)_{ar} \mid \lambda(j)_r\right) \\ p(H(j)_b|\boldsymbol{\lambda}(j)) &= p\left(H(j)_b \,\middle|\, \tfrac{1}{r_j} \sum_{r=1}^{r_j} \lambda(j)_r\right) \end{aligned} \tag{3.3.12}$$

where p is again either Half-Normal (ℓ_2-ARD) or Exponential (ℓ_1-ARD), as in (3.2.39). Similarly, we use the same prior as in (3.2.40) for the weights; namely, $p(\lambda(j)_r; a, b)$ is an inverse-Gamma distribution. For conciseness, we will write $\mu_{\boldsymbol{\lambda}(j)} = \frac{1}{r_j}\sum_{r=1}^{r_j}\lambda(j)_r$ and $\boldsymbol{\lambda} = \{\boldsymbol{\lambda}(j)\}_{j=1,\dots,J}$. In this framework, the MAP objective function is given by:

$$\begin{aligned} F(W,H,X,\boldsymbol{\lambda}) &= D_\beta(X, W*H) \\ &\quad + \sum_{j=1}^{J}\left[\sum_{r=1}^{r_j}\left(\frac{f(W(j)_{:,r})+b}{\lambda(j)_r} + c_1\log\lambda(j)_r\right) + \frac{f(H(j))}{\mu_{\boldsymbol{\lambda}(j)}} + c_2\log\mu_{\boldsymbol{\lambda}(j)}\right] \end{aligned} \tag{3.3.13}$$

where $f(x) = \frac{1}{2}\|x\|_2^2$, $c_1 = \frac{m}{2} + a + 1$, $c_2 = \frac{n}{2}$ for the Half-Normal case and $f(x) = \|x\|_1$, $c_1 = m + a + 1$, $c_2 = n$ for the Exponential case. Notice that when $r_j \equiv 1$, NMFD is equivalent to NMF, and the objective (3.3.13) simplifies to (3.2.41). By employing the heuristic updates for W, H, as well as updating $\boldsymbol{\lambda}(j)$ by setting to zero the gradient of F, one obtains the following scheme for ℓ_2-ARD:

$$H(j)^{(k+1)} = H(j)^{(k)} \circ \frac{\sum_{r=0}^{r_j-1} W(j)_{:,r}^{(k)\top}\ \overleftarrow{\left(\left(W^{(k)}*H^{(k)}\right)^{\circ(\beta-2)}\circ X\right)}^{r}}{\sum_{r=0}^{r_j-1} W(j)_{:,r}^{(k)\top}\ \overleftarrow{\left(\left(W^{(k)}*H^{(k)}\right)^{\circ(\beta-1)}\right)}^{r} + \dfrac{H(j)^{(k)}}{\mu_{\boldsymbol{\lambda}(j)}}} \qquad \forall j = 1,\dots,J$$

$$W(j)^{(k+1)} = W(j)^{(k)} \circ \left[\cdots\ \frac{\left(\left(W^{(k)}*H^{(k)}\right)^{\circ(\beta-2)}\circ X\right)\ \overrightarrow{H(j)^{(k)}}^{r\,\top}}{\left(W^{(k)}*H^{(k)}\right)^{\circ(\beta-1)}\ \overrightarrow{H(j)^{(k)}}^{r\,\top} + \dfrac{W(j)_{:,r}^{(k)}}{\lambda(j)_r^{(k)}}}\ \cdots\right]_{r=0,\dots,r_j-1} \qquad \forall j = 1,\dots,J$$

$$0 = \frac{1}{r_j\mu_{\boldsymbol{\lambda}(j)^{(k+1)}}}\left(c_2 - \frac{\|H(j)^{(k+1)}\|_2^2}{2\mu_{\boldsymbol{\lambda}(j)^{(k+1)}}}\right)\boldsymbol{\lambda}(j)^{(k+1)^{\circ 2}} + c_1\boldsymbol{\lambda}(j)^{(k+1)} - \frac{1}{2}\left(W(j)^{(k+1)^{\circ 2}}\right)^\top \mathbb{1}_{m\times 1} - b \tag{3.3.14}$$

and, similarly, for ℓ_1-ARD:

$$
\begin{aligned}
H(j)^{(k+1)} &= H(j)^{(k)} \circ \frac{\sum_{r=0}^{r_j-1} W(j)_{:,r}^{(k)\top} \overleftarrow{\left(\left(W^{(k)} * H^{(k)}\right)^{\circ(\beta-2)} \circ X\right)}^{r}}{\sum_{r=0}^{r_j-1} W(j)_{:,r}^{(k)\top} \overleftarrow{\left(\left(W^{(k)} * H^{(k)}\right)^{\circ(\beta-1)}\right)}^{r} + \frac{\mathbb{1}_{1\times n}}{\mu_{\boldsymbol{\lambda}(j)}}} \qquad \forall j = 1,\ldots,J \\
W(j)^{(k+1)} &= W(j)^{(k)} \circ \left[\cdots \quad \frac{\left(\left(W^{(k)} * H^{(k)}\right)^{\circ(\beta-2)} \circ X\right) \overrightarrow{H(j)^{(k)}}^{r\top}}{\left(W^{(k)} * H^{(k)}\right)^{\circ(\beta-1)} \overrightarrow{H(j)^{(k)}}^{r\top} + \frac{\mathbb{1}_{m\times 1}}{\lambda(j)_r^{(k)}}} \quad \cdots\right]_{r=0,\ldots,r_j-1} \qquad \forall j = 1,\ldots,J \\
0 &= \frac{1}{r_j \mu_{\boldsymbol{\lambda}(j)^{(k+1)}}} \left(c_2 - \frac{\|H(j)^{(k+1)}\|_1}{\mu_{\boldsymbol{\lambda}(j)^{(k+1)}}}\right) \boldsymbol{\lambda}(j)^{(k+1)^{\circ 2}} + c_1 \boldsymbol{\lambda}(j)^{(k+1)} - \left(W(j)^{(k+1)}\right)^\top \mathbb{1}_{m\times 1} - b
\end{aligned}
\tag{3.3.15}
$$

Unlike the analogous relations (3.2.44) and (3.2.43) for NMF, updating $\boldsymbol{\lambda}(j)$ in this newly proposed framework requires solving a quadratic system of coupled equations. In practice, we use a non-linear solver with $\boldsymbol{\lambda}(j)^{(k)}$ as the starting point. Notice again that when $r_j \equiv 1$, the system contains only one equation, and we recover the updates presented in Section 3.2.2. Let us now discuss the lower bound $B(j)$ needed for pruning the dictionaries $W(j)$. Assuming that such a value is reached when both $H(j)$ and $W(j)$ are driven to zero by the multiplicative updates, one obtains the equation:

$$
\frac{c_2}{r_j \mu_{\boldsymbol{\lambda}(j)}} \boldsymbol{\lambda}(j)^{\circ 2} + c_1 \boldsymbol{\lambda}(j) - b = 0 \tag{3.3.16}
$$

For the dictionary $W(j)$ to be pruned, the mean $\mu_{\boldsymbol{\lambda}(j)}$ must reach the lower bound $B(j)$. At the same time, when that is the case, we expect all weights $\lambda(j)_r$ to be equal. We can therefore obtain $B(j)$ by simply setting $\lambda(j)_r \equiv B(j)$ in (3.3.16). With this configuration, the system rank drops to 1, and we obtain:

$$
B(j) = \frac{b}{c_1 + \dfrac{c_2}{r_j}} \qquad \forall j = 1,\ldots,J \tag{3.3.17}
$$

which is again consistent with the previous lower bound for NMF. For a threshold $\tau > 0$, the optimal number of dictionaries will be given by:

$$
\hat{J} = \left| \left\{ j \in \{1,\ldots,J\} \,:\, \frac{\mu_{\boldsymbol{\lambda}(j)^{(k)}} - B(j)}{B(j)} > \tau \right\} \right| \tag{3.3.18}
$$

Due to the coupling of the equations, deriving lower bounds for the single weights $\lambda(j)_r$ within dictionaries $W(j)$ that survive the pruning is not straightforward. In general, such a lower bound will depend on the total number of columns of $W(j)$ that have been driven to zero by the multiplicative updates: in our notation, this number is $r_j - \hat{r}_j$. As a result, if we knew the correct lower bound, we would already know $\hat{r}_j$, hence defeating the purpose of applying this lower bound-based pruning. Nevertheless, $\hat{r}_j$ can still be estimated empirically by either inspecting which columns of $W(j)$ have been driven to

zero or simply by inspecting the weight vector $\boldsymbol{\lambda}(j)$. In the latter option, the weights $\lambda(j)_r$ associated with zero-columns will be much lower than those associated with non-zero-columns.
When it comes to the selection of the hyperparameter $b > 0$, only ℓ_1-ARD admits a closed form solution. In particular, letting:

$$\mu_X = \frac{1}{nm}\sum_{ab} X_{ab} \approx \mathbb{E}[X_{ab}] = \mathbb{E}[(W * H)_{ab}] \tag{3.3.19}$$

we can expand the right-hand side as a function of b:

$$\mu_X = b^2\left(\frac{J}{(a-1)(a-2)} + \frac{\sum_{j=1}^{J} r_j - J}{(a-1)^2}\right) \qquad \text{for} \quad \ell_1\text{-ARD} \tag{3.3.20}$$

The desired rule for b is, therefore, given by:

$$\hat{b} = \sqrt{\frac{\mu_X}{\frac{J}{(a-1)(a-2)} + \frac{\sum_{j=1}^{J} r_j - J}{(a-1)^2}}} \qquad \text{for} \quad \ell_1\text{-ARD} \tag{3.3.21}$$

Consistency with (3.2.51) is preserved when $r_j \equiv 1$. We refer the reader to Appendix 3.C for more details regarding the derivation of (3.3.20).

Example

We employ the swimmer dataset [35] again; however, instead of vectorizing the images, we construct the data matrix X by concatenating them. The resulting matrix has $n = 256\times 11$ columns. Unlike before, the optimal number of dictionaries J to use with NMFD is not necessarily 16. Indeed, we notice that the patterns "left arm up" and "right arm up" can be compressed into a single one, "arm up", which is then activated either on the left or on the right of the torso. A similar reasoning also holds for the pairs ("left arm out", "right arm out"), ("left leg down", "right leg down"), and ("left leg out", "right leg out"). Assuming that single patterns are not split into multiple dictionaries during the optimization, the optimal number of dictionaries is $\hat{J} = 13$. As for the ranks, we notice that the patterns "leg out" and "arm out" can be described either with a rank-1 pattern that is activated 3 times in succession or by a single activation of a rank-3 pattern.
In Figure 3.4, we show the results obtained by applying ℓ_1-ARD to this dataset. We chose $\beta = 1$ and added Poisson noise to the input data matrix, initial number of dictionaries $J = 26$, ranks $r_j \equiv 6$, shape hyperparameter $a = 5 \times 10^4$ and threshold $\tau = 10^{-3}$. In particular, we see that the patterns "leg down", "left leg at 45°" and "right leg at 45°" have been split into 3 patterns encoding individual pixels (dictionaries $8, 9, 13$). Nevertheless, this does not affect the recovered optimal number of dictionaries. Lastly, Figure 3.5 shows the weights $\boldsymbol{\lambda}(j)^{(k)}$ for a selection of dictionaries.

3.4 NMF-based audio analysis

Single-channel audio processing and analysis are among the most significant application domains in which NMF is widely used. In fact, the spectrogram of a discrete audio

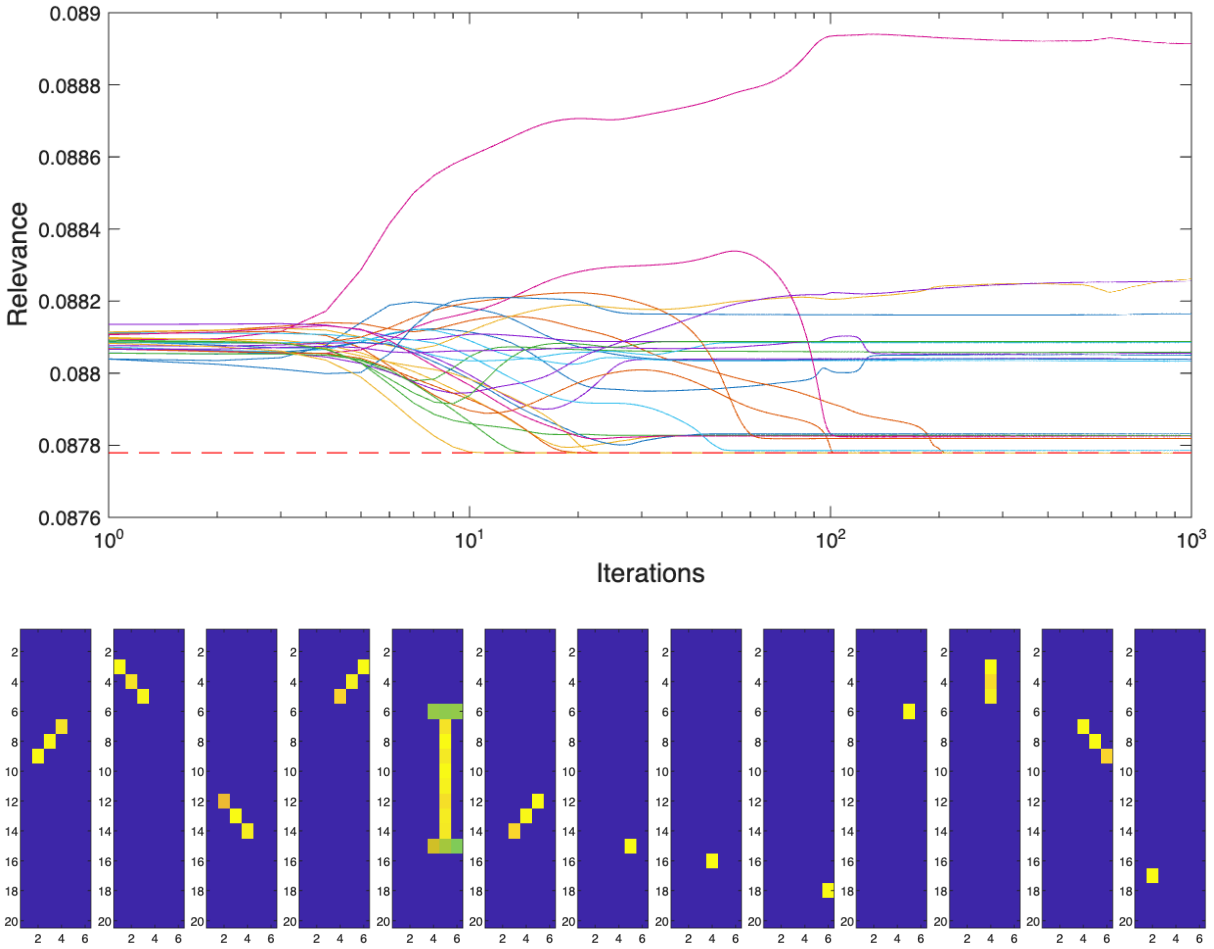


Figure 3.4: Top: values of $\mu_{\boldsymbol{\lambda}^{(k)}}$ along iterations. The theoretical lower bound $B(j) \equiv B$ is the dashed red line. Bottom: dictionaries $W(j)$ that survived the pruning.

signal is a frequency-by-time *nonnegative* matrix that captures the signal's frequency evolution over time. In this context, time frames are seen as data points and frequencies as features. By applying NMF to spectrograms, one can therefore extract relevant frequency patterns, monitor their appearance within the signal, and relate them to their sources. As a high-level approach, NMF can therefore be regarded as a solid and versatile baseline for spectrogram-based audio processing, allowing customization to specific application requirements and objectives, thanks to its many variants. In particular, its relatively inexpensive update rules and straightforward algorithmic implementation make it well suited for real-time applications where computational resources are limited.

In the following sections, we will denote $x \in \mathbb{R}^N$ a discrete-time audio signal, $\check{X} = \mathrm{STFT}(x) \in \mathcal{M}_{m\times n}(\mathbb{C})$ its (complex) Short-Time Fourier Transform (STFT)[3] and $X = |\check{X}|^2$ its spectrogram, computed by taking the elementwise squared norm of $\check{X}$. In general, the size of X depends on the chosen STFT parameters, such as window length and hop size.

[3]For audio signals, the check $\check{\cdot}$ will always denote the STFT operator.

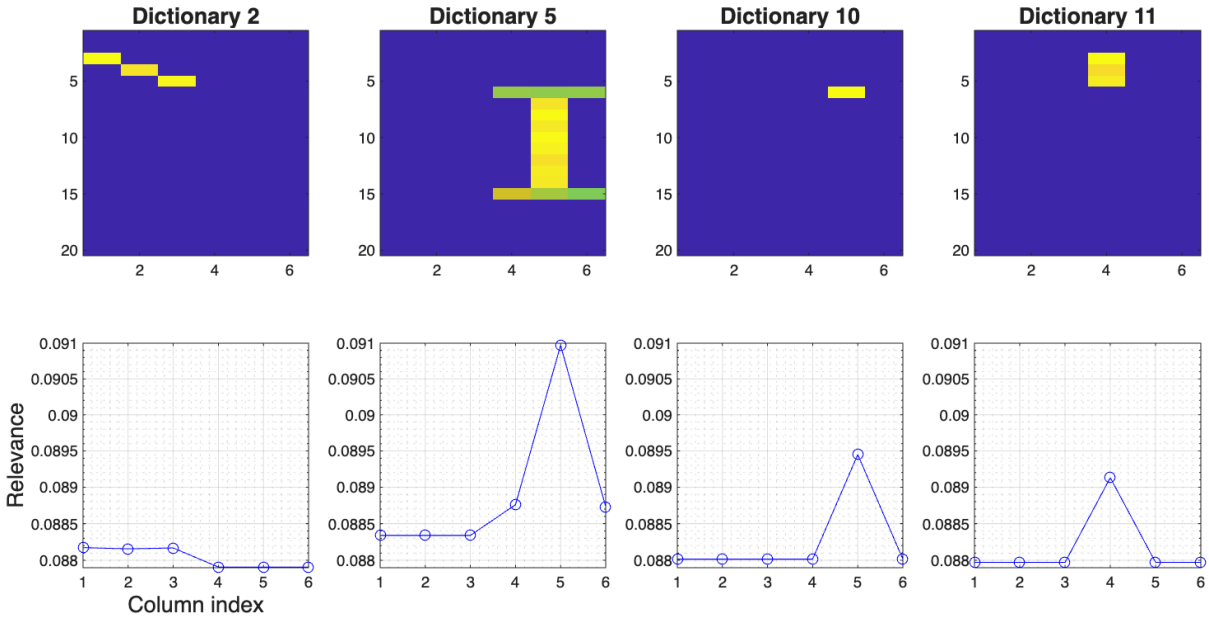


Figure 3.5: Top: a selection of dictionaries $W(j)$ that survived the pruning. Bottom: the corresponding weight vectors $\boldsymbol{\lambda}(j)^{(k)}$

3.4.1 Noise reduction

Audio signals are frequently affected by a wide range of noise sources, which can significantly degrade signal quality and hinder subsequent analysis. Consequently, denoising represents a fundamental step in audio processing, as it aims to suppress unwanted noise components while preserving the underlying clean (or target) signal, thereby enabling more accurate and reliable analysis. In this context, the recorded audio mixture x can be additively decomposed into a target component $s \in \mathbb{R}^N$ and a noise component $n \in \mathbb{R}^N$:

$$x = s + n \tag{3.4.1}$$

and the goal is to recover s given x. Spectrogram-based denoising and separation are based on the observation that the spectrogram X can be expressed through a comparable *approximate* decomposition. Namely, denoting S and N the spectrograms of s and n respectively, it holds:

$$X \approx S + N \tag{3.4.2}$$

This property, commonly referred to as spectrogram additivity, holds only approximately, as the phase information of the signals is discarded. In general, the greater the overlap between s and n in the frequency domain, the more pronounced the discrepancy between X and $S + N$ becomes. Nevertheless, spectrogram additivity is typically assumed in practice, and in most applications (3.4.2) provides a sufficiently accurate approximation. The key insight linking spectrogram-based audio processing with NMF is that NMF provides a naturally additive decomposition of the data X. In particular, given an NMF approximation $X \approx WH$, the factors can be partitioned into two distinct blocks:

$$W = \begin{bmatrix} W_S & W_N \end{bmatrix} \qquad H = \begin{bmatrix} H_S \\ H_N \end{bmatrix} \tag{3.4.3}$$

which leads to the additive decomposition:

$$X \approx W_S H_S + W_N H_N = \hat{X} \tag{3.4.4}$$

The estimated target and noise spectrograms are therefore given by $\hat{S} = W_S H_S$ and $\hat{N} = W_N H_N$. This general framework can be fine-tuned in countless ways, depending on the specific task [134, 74, 60]. Here and in Chapter 4, however, we will focus on a particular approach that takes advantage of Wiener filtering. In order to recover the desired target signal s from the estimated spectrograms $\hat{S}$ and $\hat{N}$ is to apply a Wiener filter $F \in \mathcal{M}_{m \times n}(\mathbb{R}_{\geq 0})$ to the complex STFT $\check{X}$:

$$\hat{s} = \text{ISTFT}\left(F^{\circ \frac{1}{2}} \circ \check{X}\right) = \text{ISTFT}\left(F^{\circ \frac{1}{2}} \circ \text{STFT}(x)\right) \tag{3.4.5}$$

where ISTFT denotes the inverse STFT operator. The Wiener filter, in particular, can be constructed as:

$$F = \frac{\hat{S}}{\hat{X}} = \frac{W_S H_S}{WH} \tag{3.4.6}$$

The objective of F is to selectively attenuate the time-frequency patterns corresponding to noise while preserving those that constitute the target signal. Observe, in fact, that the spectrogram of the complex matrix to which we apply the ISTFT in (3.4.5) provides an approximation of the estimated clean spectrogram:

$$|F^{\circ \frac{1}{2}} \circ \check{X}|^2 = F \circ X = \frac{\hat{S}}{\hat{X}} \circ X \approx \hat{S} \tag{3.4.7}$$

The main challenge in this framework is obtaining an NMF of X with a suitable two-block split of the factors. In (3.4.4), we implicitly assumed that the frequency patterns in W_S characterize the target signal, while those in W_N characterize the noise. In general, applying NMF to X without prior knowledge of the problem will not yield such a factorization. Accordingly, one often assumes a given *optimal* dictionary $\hat{W}$ that inherently possesses these properties. Depending on the application, $\hat{W}$ can be constructed from available data as a preprocessing step. By optimizing only the right factor while keeping $\hat{W}$ fixed, we obtain the desired NMF of X:

$$\begin{aligned} \min \quad & D_1(X, \hat{W}H) + \alpha_H \|H\|_1 \\ \text{s.t.} \quad & H \in \mathcal{M}_{r \times n}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{3.4.8}$$

When working with audio spectrograms, the Kullback-Leibler divergence ($\beta = 1$) is one of the most popular choices. In (3.4.8), we also added a sparsity regularizer for H.

3.4.2 Source separation and detection

Audio source separation involves decomposing an input mixture x into its additive parts. Unlike conventional noise reduction, this process can account for several target components $s_i \in \mathbb{R}^N$ $(i = 1, \ldots, I)$ and multiple noise components $n_j \in \mathbb{R}^N$ $(j = 1, \ldots, J)$:

$$x = s_1 + \cdots + s_I + n_1 + \cdots + n_J \tag{3.4.9}$$

The goal is to recover each s_i given x. As before, approximate spectrogram additivity yields:

$$X \approx S_1 + \cdots + S_I + N_1 + \cdots + N_J \tag{3.4.10}$$

and with an appropriate split of the NMF factors:

$$\begin{aligned} W &= \begin{bmatrix} W_{S_1} & \cdots & W_{S_I} & W_{N_1} & \cdots & W_{N_J} \end{bmatrix} \\ H &= \begin{bmatrix} H_{S_1}^\top & \cdots & H_{S_I}^\top & H_{N_1}^\top & \cdots & H_{N_J}^\top \end{bmatrix}^\top \end{aligned} \tag{3.4.11}$$

one possible approach is to proceed as in Section 3.4.1, treating each s_i as the target and all remaining components as the noise [102]:

$$\begin{aligned} \hat{s}^i &= \text{ISTFT}\left(F_i^{\circ\frac{1}{2}} \circ \check{X}\right) = \text{ISTFT}\left(F_i^{\circ\frac{1}{2}} \circ \text{STFT}(x)\right) \\ F_i &= \frac{\hat{S}_i}{\hat{X}} = \frac{W_{S_i} H_{S_i}}{WH} \end{aligned} \tag{3.4.12}$$

Again, one often assumes a given *optimal* dictionary $\hat{W}$ that inherently allows for the desired splitting of the factors. The right factor H is then optimized as in (3.4.8) keeping W fixed.

An effective and elegant strategy for source separation, especially with sources displaying distinct time-frequency signatures, is to use NMFD. The method's ability to incorporate multiple dictionaries enables the allocation of one dictionary per source for accurate reconstruction. In this setting, we can let $\bar{J} = J_S \dot{\cup} J_N$ with $J_S = \{1, \ldots, I\}$, $J_N = \{1, \ldots, J\}$ and replace the Wiener filter in (3.4.12) with:

$$F_i = \frac{\hat{S}_i}{\hat{X}} = \frac{W(i) * H(i)}{W * H} \tag{3.4.13}$$

where $W(i)$ is the dictionary associated with s_i. The optimization problem for the right factor becomes:

$$\begin{aligned} \min \quad & D_1(X, \hat{W} * H) \\ \text{s.t.} \quad & H(j) \in \mathcal{M}_{1\times n}(\mathbb{R}_{\geq 0}) \qquad \forall j \in \bar{J} \end{aligned} \tag{3.4.14}$$

In the following Chapter 4, we will focus on a variant of the source separation problem, namely source detection. Unlike source separation, where the goal is to reconstruct the full target s_i, source detection is concerned solely with determining its presence or absence. The mixture is therefore in the form:

$$x = \alpha_1^S s_1 + \cdots + \alpha_I^S s_I + n_1 + \cdots + n_J \tag{3.4.15}$$

where $\alpha_i^S \in \{0, 1\}$ are binary coefficients that model the presence or absence of the corresponding target component.

Appendix

3.A Supporting results

Theorem 3.A.1 (Eckart-Young). *Let $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ and let (U, Σ, V) be its SVD. Let also $r \leq \min(m, n)$ and $X_r = \sum_{s=1}^{r} \sigma_s U_{:,s} V_{:,s}^\top$ be the rank-r truncated SVD of X. Then it holds:*

$$\min_{\text{rank}(B)\leq r} \|X - B\|_F^2 = \|X - X_r\|_F^2 = \sum_{s=r+1}^{\min(m,n)} \sigma_s^2$$

Proof: See [38].

3.B Convergence results for n-BCD schemes

Algorithm 3.2.1 relies on two iterative subroutines applied to problems (3.2.23) and (3.2.22) to produce update rules for each factor. In this context, Majorization-Minimization (MM) schemes form a class of iterative methods that can be applied to the following, more general optimization problem:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & x \in \Omega \end{aligned} \tag{3.B.1}$$

Denoting $x^{(k)} \in \Omega$ the current iterate, such schemes have a two-step structure:

1. *Majorization step.* Construct a majorizer of f at $x^{(k)}$, that is, a function $g(x^{(k)}; \cdot)$ satisfying:
   (i) $g(x^{(k)}; x) \geq f(x)\ \forall x \in \Omega$;
   (ii) $g(x^{(k)}; x^{(k)}) = f(x^{(k)})$.
2. *Minimization step.* Define $x^{(k+1)} \in \operatorname*{argmin}_{x\in\Omega} g(x^{(k)}; x)$.

The simple conditions satisfied by the majorizer g are enough to guarantee a monotone decrease of the objective function. Indeed, it is trivial to check that:

$$f(x^{(k+1)}) \ \leq\ g(x^{(k)}; x^{(k+1)}) \ \leq\ g(x^{(k)}; x^{(k)}) \ =\ f(x^{(k)})$$

As a consequence, the only hurdle in applying these schemes is constructing a majorizer *simple enough* so that a global minimizer of $g(x^{(k)}; \cdot)$ can be computed in closed form at every iteration. One way of achieving this, for example, is choosing g in the form:

$$g(x^{(k)}; x) = \sum_i g_i(x^{(k)}; x_i)$$

where the g_i's are univariate functions. In this case computing a global minimizer of g is equivalent to working out global minimizers for every g_i.
Problem (3.B.1), although more general than (3.2.23) and (3.2.22), does not take into account that in our NMF setting the objective function depends also on external parameters, namely the $H^{(k)}$ variables for problem (3.2.23) and the $W^{(k+1)}$ variables for problem (3.2.22). The majorization-minimization framework presented so far imposes no regularity on the majorizer with respects to those parameters. As a consequence, it will be useful to introduce a more encompassing framework which we will refer to as Smooth Majorization-Minimization (SMM). Given a map $f : \mathcal{X} = \Omega_1 \times \cdots \times \Omega_n \to \mathbb{R}$ where $\Omega_i \subset \mathbb{R}^{n_i}$ and a current iterate $(x_1^{(k)}, \ldots, x_n^{(k)}) \in \mathcal{X}$, such schemes can be applied to optimization problems in the form:

$$\begin{aligned} \min \quad & f(x_1, x_2^{(k)}, \ldots, x_n^{(k)}) \\ \text{s.t.} \quad & x_1 \in \Omega_1 \end{aligned} \tag{3.B.2}$$

Much like in the MM framework, SMM schemes follow a similar two-step structure:

1. *Majorization step.* Construct a smooth, uniform majorizer of f, that is, a function $g : \Omega_1 \times \mathcal{X} \to \mathbb{R}$ satisfying:

   (i) $g(x, x_1, \ldots, x_n) \geq f(x, x_2, \ldots, x_n) \;\; \forall x \in \Omega_1, \;\; \forall (x_1, \ldots, x_n) \in \mathcal{X}$;

   (ii) $g(x_1, x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \;\; \forall (x_1, \ldots, x_n) \in \mathcal{X}$;

   (iii) $g \in \mathcal{C}^1(\Omega_1 \times \mathcal{X})$;

   (iv) $d \bullet (\nabla_x g(x_1, x_1, \ldots, x_n) - \nabla_{x_1} f(x_1, \ldots, x_n)) = 0 \;\; \forall (x_1, \ldots, x_n) \in \mathcal{X}$ and $\forall d \in \mathbb{R}^{n_1}$ such that $x_1 + d \in \Omega_1$.

2. *Minimization step.* Define $x_1^{(k+1)} \in \operatorname*{argmin}_{x \in \Omega_1} g(x, x_1^{(k)}, \ldots, x_n^{(k)})$.

Simply put, SMM schemes require additional regularity on the majorizer. Indeed, at the current parameter iterate, the chosen map g must be a majorizer and its partial derivatives must coincide with those of the objective function. Moreover, these two properties must be satisfied uniformly for all such iterates.
In the following, we will provide some convergence results for n-Block Coordinate Descent (n-BCD) methods. In particular, given the problem:

$$\begin{aligned} \min \quad & f(x_1, \ldots, x_n) \\ \text{s.t.} \quad & (x_1, \ldots, x_n) \in \mathcal{X} = \Omega_1 \times \cdots \times \Omega_n \end{aligned} \tag{3.B.3}$$

all n-BCD methods follow the scheme presented in Algorithm 3.B below:

Algorithm 3.B Block coordinate descent scheme for problem (3.B.3)

Input: An initial point $x^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)}) \in \mathcal{X}$

Output: An approximate solution to problem (3.B.3).

```
for k = 0, 1, ... do:
  for i = 1, 2, ... do:
    x_i^(k+1) = UPDATE_{x_i}(x_1^(k+1), ..., x_{i-1}^(k+1), x_i^(k), ..., x_n^(k))
  end for
end for
```

where, in particular, $\text{UPDATE}_{x_i}(x_1^{(k+1)}, \ldots, x_{i-1}^{(k+1)}, x_i^{(k)}, \ldots, x_n^{(k)})$ is either an approximate or exact 1-step solution to an iterative subroutine employed to solve:

$$\begin{aligned} \min \quad & f(x_1^{(k+1)}, \ldots, x_{i-1}^{(k+1)}, x_i, x_{i+1}^{(k)}, \ldots, x_n^{(k)}) \\ \text{s.t.} \quad & x_i \in \Omega_i \end{aligned} \tag{3.B.4}$$

starting from the initial estimate $x_i^{(k)}$.
Again, if at each iteration all UPDATE_{x_i}'s are initialized to the *exact* 1-step solution of their respective subroutines, the resulting algorithm is called *exact* n-BCD method.
With respect to the above notation, we are now ready to state three convergence results:

Theorem 3.B.1. *The limit points of the iterates of an exact 2-BCD algorithm are stationary points of problem* (3.B.3) *provided that the following conditions hold:*

1. $f \in \mathcal{C}^1(\mathcal{X})$*;*
2. Ω_1 *and* Ω_2 *are closed convex sets.*

Proof: See [59, Corollary 2].

As far as Algorithm 3.2.1 is concerned, the second condition is always satisfied since the nonnegative orthant is a closed convex set. On the other hand, when working with β-divergences, the validity of the first condition must be cross-checked using Table 2.

Theorem 3.B.2. *The limit points of the iterates of an exact n-BCD algorithm are stationary points of problem* (3.B.3) *provided that the following conditions hold:*

1. $f \in \mathcal{C}^1(\mathcal{X})$*;*
2. Ω_i *is a closed convex set* $\forall i = 1, \ldots, n$*;*
3. *Problem* (3.B.4) *admits a unique global solution independently of the variables* $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n) \in \Omega_1 \times \cdots \times \Omega_{i-1} \times \Omega_{i+1} \times \cdots \times \Omega_n$*,* $\forall i = 1, \ldots, n$*;*
4. UPDATE_{x_i} *must return the exact global solution* $\forall i = 1, \ldots, n$*.*

Proof: See [11, Proposition 2.7.1].

The main drawback of this result lies in condition 4, which requires solving problems (3.B.4) exactly rather than simply returning a 1-step solution of an iterative subroutine. The following theorem addresses precisely this inconvenience:

Theorem 3.B.3. *The limit points of the iterates of an exact n-BCD algorithm are stationary points of problem (3.B.3) provided that the following conditions hold:*

1. $f \in \mathcal{C}^1(\mathcal{X})$;
2. Ω_i *is a closed convex set* $\forall i = 1, \dots, n$;
3. *The subroutines employed for problems (3.B.4) follow a SMM framework and either one of the following is satisfied:*
   - *(i) the majorizers are quasi-convex in the first variables and their minimum is uniquely attained;*
   - *(ii) the level set* $\mathcal{X}_0 = \{x \in \mathcal{X} : f(x) \leq f(x^{(0)})\}$ *is compact and at least* $n-1$ *of the majorizers have an uniquely attained minimum.*

Proof: See [107, Theorem 2].

In particular, an exact n-BCD algorithm that satisfies the hypothesis of Theorem (3.B.3) is part of the so called Block Successive Upper-bound Minimization (BSUM) framework.

3.C Details for Section 3.3.2

Let us denote $J \in \mathcal{M}_{n\times n}(\mathbb{R})$ the following nilpotent matrix and rewrite the shift operators for any $A \in \mathcal{M}_{m\times n}(\mathbb{R})$ as:

$$J = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ \vdots & & & \ddots & 1 \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix} \qquad \overrightarrow{A}^r = AJ^r \qquad \overleftarrow{A}^r = A\left(J^r\right)^\top$$

We can therefore write:

$$(W * H)_{ab} = \sum_{j=1}^{J} \sum_{r=0}^{r_j-1} W(j)_{ar} \sum_{q=1}^{n} H(j)_q (J^r)_{qb} \tag{3.C.1}$$

Taking the expectation and conditioning over $\boldsymbol{\lambda}(j)$, we get:

$$\begin{aligned}\mathbb{E}[(W*H)_{ab}] &= \sum_{j=1}^{J}\sum_{r=0}^{r_j-1}\sum_{q=1}^{n}(J^r)_{qb}\mathbb{E}[W(j)_{ar}H(j)_q] \\ &= \sum_{j=1}^{J}\sum_{r=0}^{r_j-1}\sum_{q=1}^{n}(J^r)_{qb}\mathbb{E}_{\boldsymbol{\lambda}(j)}[\,\mathbb{E}[W(j)_{ar}|\boldsymbol{\lambda}(j)]\,\mathbb{E}[H(j)_q|\boldsymbol{\lambda}(j)]\,]\end{aligned} \tag{3.C.2}$$

For ℓ_1-ARD, recalling (3.2.49) and (3.3.12), we get:

$$\begin{aligned}\mathbb{E}[(W*H)_{ab}] &= \sum_{j=1}^{J}\sum_{r=0}^{r_j-1}\sum_{q=1}^{n}(J^r)_{qb}\mathbb{E}_{\boldsymbol{\lambda}(j)}[\,\lambda(j)_r\mu_{\boldsymbol{\lambda}(j)}\,] \\ &= \sum_{j=1}^{J}\sum_{r=0}^{r_j-1}\mathbb{E}_{\boldsymbol{\lambda}(j)}[\,\lambda(j)_r\mu_{\boldsymbol{\lambda}(j)}\,] = \sum_{j=1}^{J} r_j\mathbb{E}_{\boldsymbol{\lambda}(j)}[\,\lambda(j)_r\mu_{\boldsymbol{\lambda}(j)}\,] \\ &= \sum_{j=1}^{J}\mathbb{E}_{\boldsymbol{\lambda}(j)}[\,\lambda(j)_r\sum_{s=1}^{r_j}\lambda(j)_s\,] = \sum_{j=1}^{J}\mathbb{E}_{\boldsymbol{\lambda}(j)}[\,\lambda(j)_r^2+\sum_{r\neq s=1}^{r_j}\lambda(j)_r\lambda(j)_s\,] \\ &= \sum_{j=1}^{J}\Big(\tfrac{b^2}{(a-1)(a-2)}+\tfrac{b^2}{(a-1)^2}(r_j-1)\Big) = b^2\left(\tfrac{J}{(a-1)(a-2)}+\tfrac{\sum_{j=1}^{J}r_j-J}{(a-1)^2}\right)\end{aligned} \tag{3.C.3}$$

where, in the second step, we assume $b \geq \max_j r_j$ so that $\sum_{q=1}^{n}(J^r)_{qb} \equiv 1$. For ℓ_2-ARD, on the other hand, we get:

$$\begin{aligned}\mathbb{E}[(W*H)_{ab}] &= \sum_{j=1}^{J}\sum_{r=0}^{r_j-1}\sum_{q=1}^{n}(J^r)_{qb}\mathbb{E}_{\boldsymbol{\lambda}(j)}\left[\left(\tfrac{2\lambda(j)_r}{\pi}\right)^{\frac{1}{2}}\left(\tfrac{2\mu_{\boldsymbol{\lambda}(j)}}{\pi}\right)^{\frac{1}{2}}\right] \\ &= \sum_{j=1}^{J}\sum_{r=0}^{r_j-1}\mathbb{E}_{\boldsymbol{\lambda}(j)}\left[\left(\tfrac{2\lambda(j)_r}{\pi}\right)^{\frac{1}{2}}\left(\tfrac{2\mu_{\boldsymbol{\lambda}(j)}}{\pi}\right)^{\frac{1}{2}}\right] = \sum_{j=1}^{J}\sum_{r=0}^{r_j-1}\tfrac{2}{\pi}\mathbb{E}_{\boldsymbol{\lambda}(j)}\left[\sqrt{\lambda(j)_r\mu_{\boldsymbol{\lambda}(j)}}\right] \\ &= \sum_{j=1}^{J}\tfrac{2\sqrt{r_j}}{\pi}\mathbb{E}_{\boldsymbol{\lambda}(j)}\left[\sqrt{\lambda(j)_r\sum_{s=1}^{r_j}\lambda(j)_s}\right]\end{aligned} \tag{3.C.4}$$

In this case, $\mathbb{E}[(W*H)_{ab}]$ does not admit a simple closed form.

Deep nonnegative matrix factorizations

This chapter investigates several deep architectures derived from NMF and their application to audio tasks, including enhancement and hit detection. We detail the distinctions between these architectures, explain the rationale behind key design choices, and provide a comprehensive overview of the training procedure, covering all backpropagation relations. The chapter concludes with a comparison of computational and memory costs, focusing on real-time deployment on resource-constrained microcontrollers.

4.1 Introduction

While Nonnegative Matrix Factorization (NMF) provides a powerful framework for decomposing signals into additive, interpretable components, it is inherently limited by its unsupervised nature and the simplicity of its factorization model. Standard NMF often produces factorizations that are generic and may not align with specific application goals, such as emphasizing particular features in the data or capturing domain-specific structure. Moreover, in complex signals, such as audio mixtures or mechanical vibrations, classical NMF can struggle to exploit temporal correlations or prior knowledge about the sources, leading to suboptimal separations. These limitations motivate the use of supervised and structured machine learning approaches to guide the factorization towards solutions that are both meaningful and task-specific. By incorporating training data, prior information, or architectural constraints, such methods can enforce desirable properties in the learned components, improve robustness to noise, and enable the detection of subtle patterns that would otherwise be obscured. In this chapter, we focus on deep extensions of NMF that leverage the idea of unrolling iterative NMF updates into multi-layer architectures. These frameworks integrate data-driven learning with the interpretability and constraints of NMF, effectively combining model-based and learning-based paradigms. In particular, we study three architectures in detail: Deep-NMF [64], which unrolls standard NMF into a deep network to enable discriminative training; PAD-NMF (Physics-Aware Deep-NMF) [22], which incorporates structured priors and masks to embed physical knowledge and temporal correlations; and Deep-NMFD, which extends the NMF framework to convolutional dictionaries to capture long-range temporal dependencies. Across these architectures, we analyze their design choices, training procedures, and applications to audio processing tasks such as source separation, enhancement, and hit detection, highlighting how deep NMF variants can overcome the limitations of classical NMF while remaining interpretable and computationally efficient.

4.2 Deep-NMF (DNMF)

This section briefly introduces the Deep-NMF architecture, originally proposed in [64] as a canonical example of the Deep Unfolding paradigm. We consider the same general setting described in Section 3.4.1, in which NMF-based methods are employed for noise reduction, with a particular focus on speech enhancement. Within this framework, the observed spectrogram X represents a mixture of target speech and environmental noise. As discussed in Section 3.4.1, denoising is achieved by constructing an appropriate Wiener filter from the resulting NMF factors. The key idea explored in [64] is to leverage machine learning techniques to optimize the NMF factorization *explicitly* for Wiener filter estimation. This is accomplished by embedding the traditional NMF optimization problem (3.4.8) into a bilevel optimization framework and subsequently applying the Deep Unfolding methodology described in Section 2.4. The resulting network will be used as a baseline for the proposed methodology of Section 4.3.

4.2.1 Network architecture

To employ the bilevel optimization framework, the original optimization problem must first be reformulated. Specifically, *among the NMF factorizations that are optimal in the sense of* (3.4.8)*, we aim to identify those that yield an optimal Wiener filter F as defined in* (3.4.6). Under this perspective, the *inner* optimization problem is given by (3.4.8). Using the notation introduced in Chapter 2, the matrix X corresponds to the observation, while the task-specific optimal dictionary $\hat{W}$ plays the role of the parameter vector θ. Similarly, the admissible search space for the optimization variable H is the nonnegative orthant, namely $\Omega_H = \mathcal{M}_{r\times m}(\mathbb{R}_{\geq 0})$. Note that the factorization rank r is not treated as a hyperparameter in this setting, as it is implicitly determined by the dictionary $\hat{W}$. The resulting inner objective function is therefore given by:

$$\mathcal{F}^{\,in}(H, W, X) = D_1(X, WH) + \alpha_H \|H\|_1 \tag{4.2.1}$$

Moreover, as discussed in Section 3.2.1, the update rule (3.2.26) defines an optimization scheme for the inner problem (4.2.1). In the particular case $\beta = 1$ with an additional sparsity regularization term, this leads to the following inner map:

$$f^{\,in}(H, W, X) = f_W^{in}(H) = H \circ \frac{W^\top \left(\frac{X}{WH}\right)}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \tag{4.2.2}$$

As for the *outer* optimization problem, given the target spectrogram S (which we consider part of the observation (X, S) for the inner problem), we define the following objective:

$$\mathcal{F}^{\,out}(H, W, (X, S)) = \mathcal{F}^{\,out}(H, W) = \frac{1}{2}\|S - F \circ X\|_2^2 = \frac{1}{2}\left\|S - \frac{W_S H_S}{WH} \circ X\right\|_2^2 \tag{4.2.3}$$

where we assume a two-block split of the factors, as in (3.4.3). As can be seen from (4.2.3), the objective of the resulting network architecture is to optimize the construction of the Wiener filter F to more effectively extract the target spectrogram S from the noisy

mixture X.
The baseline network is constructed by unrolling the inner mapping (4.2.2) for a fixed number of iterations K and by untying the parameters W across layers. However, such an architecture is generally insufficient to recover the desired NMF factors and, consequently, to produce an effective Wiener filter F. This limitation arises because the network, in its current form, imposes no constraints on the types of spectral patterns it should preferentially learn, nor on how these patterns should be distributed between the two dictionary blocks $W_S^{(k)}$ and $W_N^{(k)}$ at each layer. To address these issues, in [64], the authors propose distinguishing between two types of network layers: *non-discriminative* and *discriminative*. In a non-discriminative layer k, the dictionary is fixed to the optimal one, namely $W^{(k)} = \hat{W}$, and those parameters are not modified by backpropagation. In a discriminative layer k, on the other hand, the network is free to update the dictionary $W^{(k)}$ by backpropagation. For a given $C \leq K$, the first $K - C$ layers are set as non-discriminative. The remaining C layers, with the exception of the final reconstruction layer, are discriminative. The last dictionary $W^{(K)}$, which is used only for reconstruction within the outer objective (i.e. the loss function), is also fixed and set equal to $\hat{W}$. An overview of the architecture is shown in Figure 4.1.

$$\underbrace{H^{(0)} \xrightarrow{f^{in}_{W^{(0)}}} \cdots \xrightarrow{f^{in}_{W^{(K-C-1)}}} H^{(K-C)}}_{K-C \text{ non-discriminative layers}} \underbrace{\xrightarrow{f^{in}_{W^{(K-C)}}} \cdots \xrightarrow{f^{in}_{W^{(K-1)}}} H^{(K)} \dashrightarrow \mathcal{F}^{out}(H^{(K)}, W^{(K)})}_{C \text{ discriminative layers}}$$

Figure 4.1: Deep-NMF network architecture [64]. The last C discriminative layers have trainable dictionaries $W^{(k)}$ while the first $K-C$ non-discriminative layers have fixed dictionaries $W^{(k)} \equiv \hat{W}$.

The motivation for adopting this architecture lies in the interpretability preserved by the unfolded NMF iterations:

- The first $K - C$ layers of the network correspond to the initial $K - C$ iterations of the update scheme (3.2.26). Their role is to drive the coefficient matrix $H^{(K-C)}$ sufficiently close to a minimizer of Problem (3.4.8), using a dictionary $\hat{W}$ that is optimal for the task under consideration.

- The subsequent C intermediate layers employ learned dictionaries to further refine the representation, guiding $H^{(K-C)}$ toward the final coefficient matrix $H^{(K)}$ used to produce the network output. Unlike the preceding layers, these dictionaries are trained and therefore can both help escape poor local minima and enforce task-specific structure on $H^{(K)}$, as dictated by the training loss $\mathcal{F}^{out}$.

- Finally, the last layer again makes use of the optimal dictionary $\hat{W}$ together with the optimized coefficient matrix $H^{(K)}$ to reconstruct and extract the features of interest from the input spectrogram X.

As a general rule of thumb, the greater the complexity of the dataset to be modeled by the Deep-NMF architecture, the larger the value of C should be. Furthermore, since in

the present implementation the first $K - C$ layers employ a fixed dictionary $\hat{W}$, the total number of layers K may be chosen according to standard NMF practice for selecting the maximum number of iterations. In particular, K can be set such that, after $K - C$ coefficient updates, the relative residual $\|H^{(K-C)} - H^{(K-C-1)}\|_2$ falls below a predefined threshold. This criterion guarantees the convergence of $H^{(K-C)}$ to a stationary point of (3.4.8).
We also note an additional technical modification aimed at incorporating temporal correlations into the network architecture. In the non-discriminative layers, the observed spectrogram matrix X (together with the optimal dictionary $\hat{W}$) is extended to form a block-Hankel–augmented spectrogram. The extent of this augmentation is governed by a parameter $T \in \mathbb{N}$, which determines the number of consecutive spectrogram frames assumed to be temporally correlated. Further discussion of this modification is deferred to the next section.

4.2.2 Backpropagation

The general backpropagation scheme for unfolded networks derived from bilevel optimization was introduced in Section 2.4.1. We emphasize, however, that standard gradient descent updates applied to the network parameters $W^{(k)}$ do not preserve their nonnegativity, a property that is essential for maintaining interpretability within the NMF framework. As a result, the standard backpropagation relations require a slight modification. Motivated by the multiplicative NMF update rule (3.2.27), and adopting the notation introduced in Section 2.4.1, in [64], the authors propose updating each $W^{(k)}$ according to:

$$W^{(k)} \Leftarrow W^{(k)} \circ \frac{\left[\nabla_{W^{(k)}} \hat{\mathcal{F}}\right]_-}{\left[\nabla_{W^{(k)}} \hat{\mathcal{F}}\right]_+} \qquad \forall k = K - C, \ldots, K - 1 \tag{4.2.4}$$

This modification requires backpropagating both the positive and negative components of the gradient, rather than the gradient as a whole, as is typically done in conventional neural networks. Specifically, (4.2.4) involves $[\nabla_{W^{(k)}} \hat{\mathcal{F}}]_\pm$, which in turn depend on the parameter gradient map Φ_W through the terms $[\Phi_W(\nabla_H, H, W, X)]_\pm$. These terms are defined as follows:

$$\begin{aligned}
[\Phi_W(\nabla_H, H, W, X)]_\pm \;=\; & \frac{X}{WH}\left(\frac{H}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ [\nabla_H]_\pm\right)^\top \\
+ \; & \left[\frac{X}{(WH)^{\circ 2}} \circ \left[W\left(\frac{H}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ [\nabla_H]_\mp\right)\right]\right] H^\top \\
+ \; & \mathbb{1}_{m\times n}\left[\frac{H}{(W^\top \mathbb{1}_{m\times n} + \alpha_H)^{\circ 2}} \circ \left(W^\top \frac{X}{WH}\right) \circ [\nabla_H]_\mp\right]^\top \\
- \; & W \circ \left[\frac{X}{(WH)^{\circ 2}}\left[\frac{H^{\circ 2}}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ \left([\nabla_H]_\pm + [\nabla_H]_\mp\right)\right]^\top\right]
\end{aligned} \tag{4.2.5}$$

Similarly, (4.2.5) applied to (2.4.6) involves $[\nabla_H]_\pm = [\nabla_{H^{(k)}}\hat{\mathcal{F}}]_\pm$, which in turn depend on the state gradient map Φ_H and the final state gradient map Φ_K through the terms $[\Phi_H(\nabla_H, H, W, X)]_\pm$ and $[\Phi_K(H, X)]_\pm$. The latter are in the form:

$$\begin{aligned}[\Phi_H(\nabla_H, H, W, X)]_\pm &= \frac{W^\top\left(\frac{X}{WH}\right)}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ [\nabla_H]_\pm \\ &+ W^\top\left[\frac{X}{(WH)^{\circ 2}} \circ \left[W\left(\frac{H}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ [\nabla_H]_\mp\right)\right]\right]\end{aligned} \tag{4.2.6}$$

$$\begin{aligned}[\Phi_K(H, X)]_+ &= \begin{bmatrix} \hat{W}_S^\top\left(\frac{X^{\circ 2} \circ \hat{W}_S H_S \circ \hat{W}_N H_N}{\left(\hat{W}H\right)^{\circ 3}}\right) \\ \hat{W}_N^\top\left(\frac{X \circ S \circ \hat{W}_S H_S}{\left(\hat{W}H\right)^{\circ 2}}\right) \end{bmatrix} \\ [\Phi_K(H, X)]_- &= \begin{bmatrix} \hat{W}_S^\top\left(\frac{X \circ S \circ \hat{W}_N H_N}{\left(\hat{W}H\right)^{\circ 2}}\right) \\ \hat{W}_N^\top\left(\frac{X^{\circ 2} \circ \left(\hat{W}_S H_S\right)^{\circ 2}}{\left(\hat{W}H\right)^{\circ 3}}\right) \end{bmatrix}\end{aligned} \tag{4.2.7}$$

For a detailed derivation of these relations, refer to [64].

4.3 Physics-Aware Deep-NMF (PAD-NMF)

The Deep-NMF architecture presented in the previous section should not be regarded as a general-purpose solution for audio processing. It was specifically designed for speech denoising, and therefore there is no guarantee that it will perform effectively on other audio-related tasks, even when such tasks exhibit apparent similarities. In this section, we revisit the work presented in [22], in which the Deep-NMF architecture is adapted to address a source detection problem. More specifically, we focus on hit detection in mechanical systems. In this context, the recorded audio signals arise from the response of a mechanical system to impulsive excitations, referred to as hits. The objective of the architecture described in the following sections is to accurately detect the presence or absence of such responses, whose characteristics may vary depending on the component of the system involved. This setting naturally leads to a multiple-target source detection problem analogous to the one discussed in Section 3.4.2. A key advantage of working with mechanical systems lies in the possibility of mathematically modeling their responses, which facilitates the generation of synthetic datasets and, in particular, provides access to clean target responses that can be used either for training or for the construction

of optimal dictionaries. Moreover, as is typical of causal physical systems, hit responses exhibit strong temporal correlations. As will be shown, explicitly embedding this property into the Deep-NMF architecture is crucial to the success of the hit detection task. The resulting optimized architecture is referred to as the Physics-Aware Deep-NMF (PAD-NMF).

4.3.1 Physics-aware enhancements

Since our objective differs substantially from that of audio enhancement, we now describe how the Deep-NMF architecture was modified and optimized to incorporate a form of physics awareness tailored to the applications considered here.

·*Hankel-extended matrices for temporal correlation.* The spectrogram matrices produced by the physico-mathematical models considered in this section exhibit a distinctive temporal structure. In particular, for any mixture $\tilde{X}$ containing a target source spectrogram $\tilde{S}$, the columns of $\tilde{S}$ necessarily appear in $\tilde{X}$ in the same temporal order. Consequently, embedding this temporal correlation into the Deep-NMF architecture is essential for developing an algorithm capable of reliably identifying such signals, even in the presence of strong noise. With this in mind, given a (nonnegative) matrix $A \in \mathcal{M}_{m' \times n}(\mathbb{R}_{\geq 0})$ with a column-wise representation $A = \{A_j\}_{j=1}^{n}$ and a correlation index $T \in \mathbb{N}_{\leq n}$, we will denote $\mathcal{H}_T(A) \in \mathcal{M}_{m'T \times n}(\mathbb{R}_{\geq 0})$ the column-wise-Hankel extension of A:

$$\mathcal{H}_T(A) = \begin{bmatrix} 0 & \cdots & 0 & A_1 & \cdots & A_{n-T+1} \\ \vdots & \iddots & \iddots & A_2 & \iddots & \vdots \\ 0 & A_1 & \iddots & \vdots & \iddots & A_{n-1} \\ A_1 & A_2 & \cdots & A_T & \cdots & A_n \end{bmatrix} \tag{4.3.1}$$

Consequently, given a mixture spectrogram $\tilde{X} \in \mathcal{M}_{m' \times n}(\mathbb{R}_{\geq 0})$ to be processed by the Deep-NMF network, we instead propagate its augmented representation $X = \mathcal{H}_T(\tilde{X})$. With this construction, the feature dimension increases to $m = m'T$, where the hyperparameter T controls the extent of temporal context and must be chosen sufficiently large to capture long-range temporal correlations. As a result of this augmentation, the network weights partially inherit a Hankel-like structure, although additional mechanisms are required to preserve this structure throughout training, as discussed in the following paragraphs. More importantly, since each dictionary atom encodes a feature present in the mixture X, the Hankel augmentation ensures that these atoms represent features spanning consecutive frames of the original spectrogram $\tilde{X}$. This, in turn, enables the Deep-NMF architecture to learn the temporal correlations that characterize the signals of interest. We emphasize that in [64] the Hankel structure was introduced only in the non-discriminative layers, thereby preventing backpropagation from acting on augmented dictionaries. In contrast, our implementation extends this construction to the discriminative layers as well, where the dictionaries are learned. The following sections further investigate this design choice, with particular emphasis on preserving the structure of both the learned weights and the coefficient matrices.

·*Discriminative dictionary initialization.* Let $\{\tilde{S}_i\}_{i=1}^{I}$ and $\{\tilde{N}_j\}_{j=1}^{J}$ denote sufficiently

rich collections of target source spectrograms and noise spectrograms, respectively. We refer to these collections as Initialization Sets, and they are assumed to contain all the signal features that the Deep-NMF algorithm is expected to detect. Our objective is to construct an optimal dictionary $\hat{W}$ capable of encoding both the characteristic features of the underlying physico-mathematical model and their temporal structure across all instances $\tilde{S}_i$ and $\tilde{N}_j$. To this end, the dictionary $\hat{W}$ must include specialized atoms that enable discriminative reconstruction of both source and noise spectrograms. Accordingly, we define $\hat{W}$ as follows:

$$\hat{W} = \begin{bmatrix} \hat{W}_{S_1} & \cdots & \hat{W}_{S_I} & \hat{W}_{N_1} & \cdots & \hat{W}_{N_J} \end{bmatrix} \tag{4.3.2}$$

where $\hat{W}_{S_i}$ (resp. $\hat{W}_{N_j}$) is an NMF solution to the following problem (4.3.3) (resp. (4.3.4))

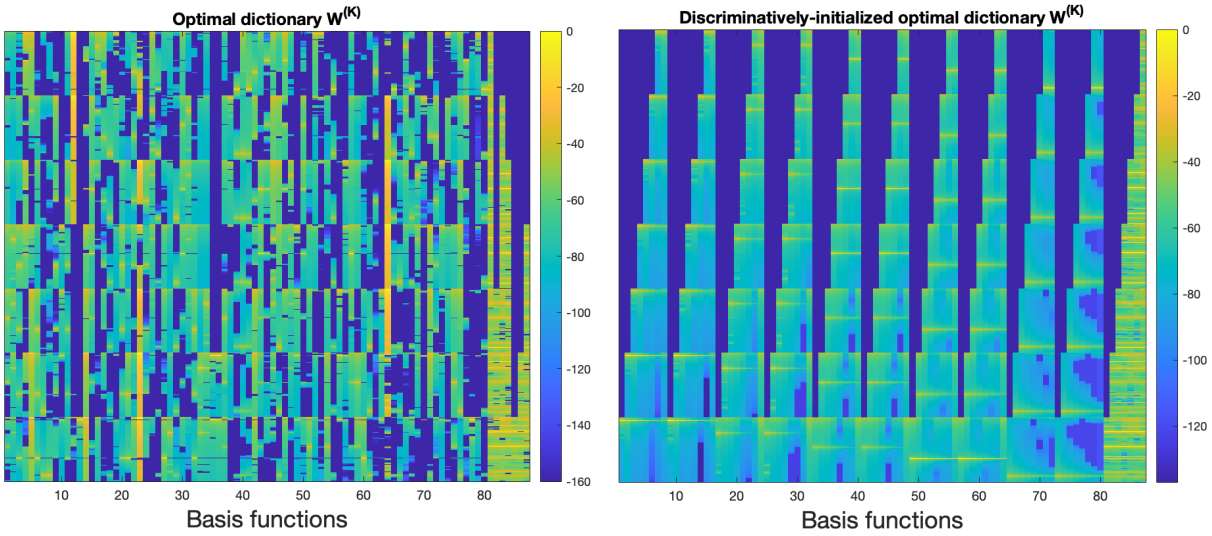
$$\begin{aligned} \min \quad & D_1(S_i, \hat{W}_{S_i}\hat{H}_{S_i}) \\ \text{s.t.} \quad & \hat{W}_{S_i} \in \mathcal{M}_{m\times r_{S_i}}(\mathbb{R}_{\geq 0}) \\ & \hat{H}_{S_i} \in \mathcal{M}_{r_{S_i}\times n}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{4.3.3}$$

$$\begin{aligned} \min \quad & D_1(N_j, \hat{W}_{N_j}\hat{H}_{N_j}) \\ \text{s.t.} \quad & \hat{W}_{N_j} \in \mathcal{M}_{m\times r_{N_j}}(\mathbb{R}^+) \\ & \hat{H}_{N_j} \in \mathcal{M}_{r_{N_j}\times n}(\mathbb{R}^+) \end{aligned} \tag{4.3.4}$$

and we let:

- T_{S_i} (resp. T_{N_j}) the number of temporally correlated consecutive frames in $\tilde{S}_i$ (resp. $\tilde{N}_j$);
- $T = \max\{T_{S_i}, T_{N_j} \,|\, i = 1, \ldots, I,\ j = 1, \ldots, J\}$;
- $r_{S_i} = \min\{n, T_{S_i} + T - 1\}$ (resp. $r_{N_j} = \min\{n, T_{N_j} + T - 1\}$);
- $S_i = \mathcal{H}_T(\tilde{S}_i)$ (resp. $N_j = \mathcal{H}_T(\tilde{N}_j)$).

Moreover, whenever necessary, the atoms in each subdictionary $\hat{W}_{S_i}$ and $\hat{W}_{N_j}$ are permuted to recover the column-wise Hankel structure. Once $\hat{W}$ is constructed and before the training process begins, all weights in the network are initialized to $\hat{W}$. Lastly, in the notation introduced earlier, the overall factorization rank is given by $r = \sum_i r_{S_i} + \sum_j r_{N_j}$. Figure 4.2 illustrates the effect of the discriminative initialization procedure on the optimal dictionary. Instead of subdividing the initialization process into $I + J$ independent NMF problems, the dictionary shown in Figure 4.2a is constructed using only two factorizations: one for the target source spectrograms and one for the noise spectrograms. More precisely, the augmented spectrograms corresponding to each category (target and noise) are concatenated sequentially, and two separate subdictionaries are obtained via NMF. These subdictionaries are then combined to form the resulting dictionary.
The primary motivation for employing a discriminative dictionary initialization in our proposed physics-aware Deep-NMF algorithm is that it imposes structure on the coefficient matrices. Indeed, as illustrated in Figure 4.3, the combination of discriminative initialization with Hankel-structured discriminative layers causes the final output matrix $H^{(K)}$ to inherit a block-wise upper-diagonal layout that mirrors the structure of $\hat{W}$. As will be discussed in the next section, this property is central to our hit-detection procedure, as it enables a direct assessment of the contribution of each spectrogram in the Initialization Sets to the reconstruction of the propagated mixture. Furthermore, comparison of Figures 4.3c and 4.3d highlights the importance of preserving the Hankel structure



(a) (b)

Figure 4.2: Effect of discriminative initialization on the optimal dictionary $\hat{W} = W^{(K)}$. Shown are the optimal dictionary (a) without and (b) with discriminative initialization. Parameters are set to $T = 7$, $I = 10$, $J = 1$, $r_{S_i} \equiv 8$, and $r_{N_j} \equiv 7$.

throughout the discriminative layers, a choice not made in the original algorithm proposed by Hershey et al. [64]. Specifically, Figure 4.3c shows a clear deterioration of detected features, evidenced by the loss of the block-wise diagonal behavior characteristic of Hankel-augmented dictionaries. In contrast, Figure 4.3d demonstrates improved performance, with the block-wise structure preserved and features absent from the mixture appropriately attenuated.

·*Preservation of Hankel structure by projection.* To preserve the Hankel-augmented structure of the dictionaries, which naturally arises from the models under consideration, it is necessary to project the coefficient matrices $H^{(k)}$ for all $k \in \{1, \ldots, K\}$ during the forward propagation, and the trainable weights $W^{(k)}$ for all $k \in \{K - C, \ldots, K - 1\}$ during backpropagation, onto the appropriate structured space. By decomposing the weights into blocks as in (4.3.2):

$$W^{(k)} = \begin{bmatrix} W_{S_1}^{(k)} & \cdots & W_{S_I}^{(k)} & W_{N_1}^{(k)} & \cdots & W_{N_J}^{(k)} \end{bmatrix} \tag{4.3.5}$$

after each backpropagation update, the elementwise binary masks $M_{S_i}^W \in \mathcal{M}_{m \times r S_i}(0, 1)$ and $M_{N_j}^W \in \mathcal{M}_{m \times r N_j}(0, 1)$ are applied to the subdictionaries $W_{S_i}^{(k)}$ and $W_{N_j}^{(k)}$, respectively, as defined by:

$$M_{S_i}^W = \mathcal{H}_T(\mathbb{1}_{m' \times r_{S_i}}) \tag{4.3.6}$$

$$M_{N_j}^W = \mathcal{H}_T(\mathbb{1}_{m' \times r_{N_j}}) \tag{4.3.7}$$

More concretely, denoting the complete W mask by:

$$M^W = \begin{bmatrix} M_{S_1}^W & \cdots & M_{S_I}^W & M_{N_1}^W & \cdots & M_{N_J}^W \end{bmatrix} \tag{4.3.8}$$

The update rule for the weights will be:

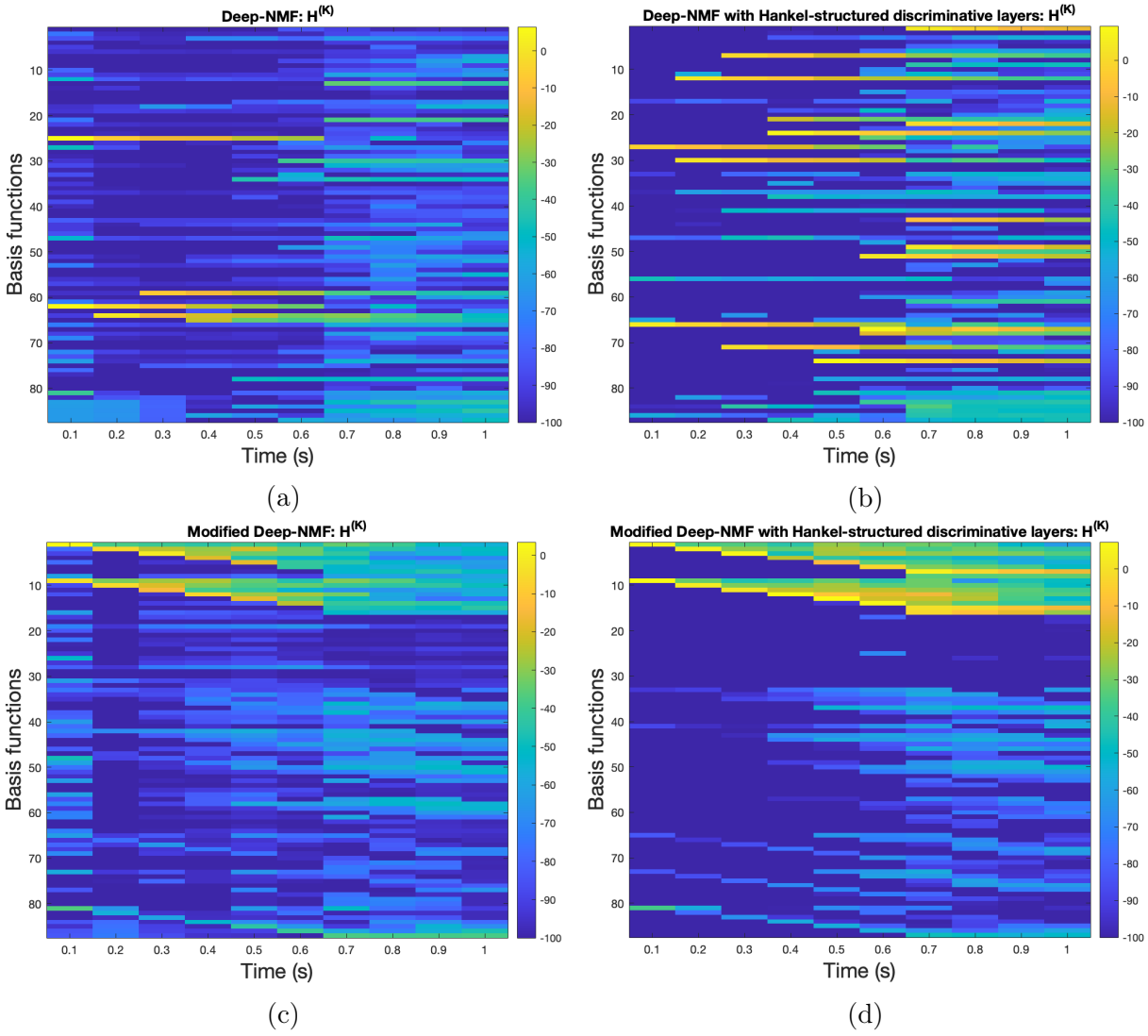


Figure 4.3: Effect of discriminative dictionary initialization and Hankel-structured discriminative layers on the coefficient matrix $H^{(K)}$. Shown are: (a) $H^{(K)}$ from the standard Deep-NMF algorithm, (b) $H^{(K)}$ from a standard Deep-NMF with Hankel-structured discriminative layers, (c) $H^{(K)}$ from a modified Deep-NMF using discriminative dictionary initialization without Hankel-structured discriminative layers, and (d) $H^{(K)}$ from a modified Deep-NMF with both discriminative dictionary initialization and Hankel-structured discriminative layers. Here, 'modified' indicates the use of discriminative dictionary initialization. Parameters are set to $K = 1000$, $C = 4$, and $T = 7$. All matrices are displayed on a log scale.

$$W^{(k)} \Leftarrow W^{(k)} \circ \frac{\left[\nabla_{W^{(k)}} \hat{\mathcal{F}}\right]_-}{\left[\nabla_{W^{(k)}} \hat{\mathcal{F}}\right]_+} \circ M^W \qquad \forall k = K - C, \ldots, K - 1 \tag{4.3.9}$$

Similarly, by decomposing the coefficient matrices in a symmetric way:

$$H^{(k)} = \left[H_{S_1}^{(k)^\top} \quad \cdots \quad H_{S_I}^{(k)^\top} \quad H_{N_1}^{(k)^\top} \quad \cdots \quad H_{N_J}^{(k)^\top} \right]^\top \tag{4.3.10}$$

after each propagation update, the elementwise binary masks $M_{S_i}^H \in \mathcal{M}_{r_{S_i} \times n}(\{0,1\})$ and $M_{N_j}^H \in \mathcal{M}_{r_{N_j} \times n}(\{0,1\})$ are applied to the submatrices $H_{S_i}^{(k)}$ and $H_{N_j}^{(k)}$, respectively, as defined by:

$$\left(M_{S_i}^H\right)_{s,j} = \begin{cases} 1 & \text{for } (s=j) \vee (j > s = r_{S_i}) \\ 0 & \text{otherwise} \end{cases} \tag{4.3.11}$$

$$\left(M_{N_j}^H\right)_{s,j} = \begin{cases} 1 & \text{for } (s=j) \vee \left(j > s = r_{N_j}\right) \\ 0 & \text{otherwise} \end{cases} \tag{4.3.12}$$

Once again, denoting the complete H mask by:

$$M^H = \left[M_{S_1}^{H^\top} \quad \cdots \quad M_{S_I}^{H^\top} \quad M_{N_1}^{H^\top} \quad \cdots \quad M_{N_J}^{H^\top} \right]^\top \tag{4.3.13}$$

the update rule for the coefficient matrices will be:

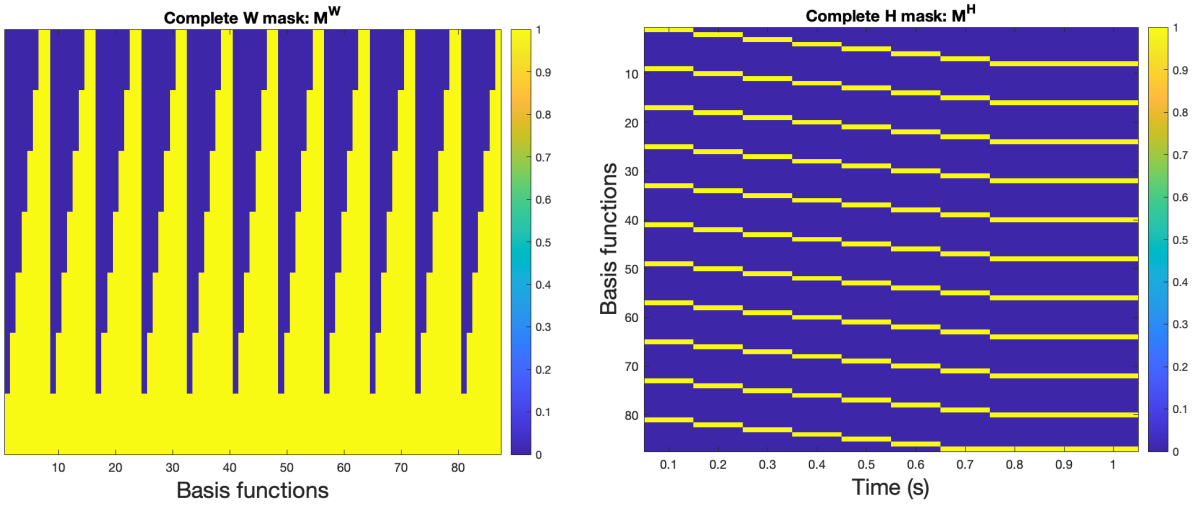
$$H^{(k+1)} = f_{W^{(k)}}^{in}(H^{(k)}) \circ M^H \qquad \forall k = 0, \ldots, K-1 \tag{4.3.14}$$

Within the context of our bilevel optimization framework, implementing (4.3.14) requires replacing (4.2.2) with:

$$f^{in}(H, W, X) = f_W^{in}(H) = H \circ \frac{W^\top \left(\frac{X}{WH}\right)}{W^\top \mathbb{1}_{m \times n} + \alpha_H} \circ M^H \tag{4.3.15}$$

We emphasize, however, that as a consequence of this projection, the backpropagation updates encoded by the maps (4.2.5), (4.2.6) and (4.2.7) need to be modified accordingly. The adjusted maps are provided in the following Section 4.3.2.
Furthermore, to obtain an optimal dictionary $\hat{W}$ that is more compatible with the masked H matrices, all matrices involved in the discriminative dictionary initialization problems (4.3.3) and (4.3.4) can be projected using their corresponding masks after each update. As illustrated in Figure 4.4, the proposed masks preserve both the Hankel structure of the learned dictionaries and the temporal correlation between their columns by allowing only synchronous activations within each block. At any given time, only the corresponding column in each subdictionary can be active. We also explored an alternative approach in which the total energy of the coefficient matrix is preserved rather than partially discarded. However, due to its higher computational cost and comparable results, we ultimately adopted the simpler projection-based approach described above..

·*Loss function.* The loss function at the last layer (or outer objective, in our bilevel optimization framework) plays the key role of imposing some desirable structure on the output coefficient matrix $H^{(K)}$. In particular, given a training triplet $(\tilde{X}, \tilde{S}, \tilde{N})$ in which $\tilde{S}$ is the target signal spectrogram we are interested in detecting, we want to design a loss



(a) (b)

Figure 4.4: Complete masks for Hankel structure preservation. Shown are the mask for the W matrices (a) and the mask for the H matrices (b). Parameters are set to $T = 7$, $I = 10$, $J = 1$, $r_{S_i} \equiv 8$, and $r_{N_j} \equiv 7$.

function that prioritizes an accurate reconstruction of $\tilde{S}$ using the available atoms in the final dictionary $W^{(K)} = \hat{W}$. Denoting:

$$W_S = [W_{S_1} \cdots W_{S_I}] \qquad (4.3.16) \qquad W_N = [W_{N_1} \cdots W_{N_J}] \qquad (4.3.17)$$

$$H_S = \begin{bmatrix} H_{S_1} \\ \vdots \\ H_{S_I} \end{bmatrix} \qquad (4.3.18) \qquad H_N = \begin{bmatrix} H_{N_1} \\ \vdots \\ H_{N_J} \end{bmatrix} \qquad (4.3.19)$$

the outer objective we adopted is:

$$\mathcal{F}^{\,out}(H, W) = \frac{1}{2}\|S - F \circ X\|_2^2 + \frac{\alpha_S}{2}\|S - W_S H_S\|_2^2 \qquad (4.3.20)$$

where $F = \frac{W_S H_S}{WH}$ is a Wiener filter defined as in the noise reduction setting of Section 3.4.1 and $X = \mathcal{H}_T(\tilde{X})$, $S = \mathcal{H}_T(\tilde{S})$.
During inference, the two terms in (4.3.20) serve complementary purposes. The first term, involving the Wiener filter, is responsible for identifying the correct target spectrogram S within the mixture and for approximating the noise spectrogram via the term $W_N H_N$ in the denominator of the filter. The second term, in contrast, is specialized in reconstructing S with an accuracy controlled by the hyperparameter α_S. Although the loss function itself is not present during inference, the complementary roles of these terms are reflected in the way the discriminative dictionaries propagate the coefficient matrices as a result of the training phase. It is important to emphasize that an accurate reconstruction of S is crucial for the subsequent hit detection process, and therefore α_S should be set to a sufficiently high value. However, because the penalty term in (4.3.20) does not involve the noise-related portions of the coefficient matrix H_N or the dictionary W_N, the Wiener filter term remains the only mechanism providing a descent direction for these blocks. Consequently, the Wiener filter term cannot be omitted, as it plays a fundamental role

in reconstructing the noise spectrogram, especially in cases where certain target features could be mistaken for deterministic noise components. As will be illustrated in the numerical experiments section, omitting the Wiener filter leads to target features present in the noise being represented using W_S rather than W_N, which would compromise the hit detection procedure. Therefore, while the Wiener filter term is essential for accurate noise reconstruction, its influence on the reconstruction of S is secondary due to the presence of the specialized penalty. Regarding the hyperparameter α_S, it can be set sufficiently high to suppress the influence of the Wiener filter on the descent directions of W_S and H_S. In our experiments, results were largely insensitive to the specific value chosen for α_S, provided it was high enough to achieve this goal. Nevertheless, excessively large values should be avoided, as they may introduce bias toward the last instance used during network training.

·*Mini-batches.* For a training set $\{(\tilde{X}_q, \tilde{S}_q, \tilde{N}_q)\}_{q=1}^{Q}$ the complete loss function for our proposed architecture will be:

$$\mathcal{F}^{\,out}(H, W) = \sum_{q=1}^{Q} \frac{1}{2}\|S_q - F \circ X_q\|_2^2 + \frac{\alpha_S}{2}\|S_q - W_S H_S\|_2^2 \tag{4.3.21}$$

where $X_q = \mathcal{H}_T(\tilde{X}_q)$ and $S_q = \mathcal{H}_T(\tilde{S}_q)$. Consequently, the final aspect to discuss regarding the training process of our physics-aware Deep-NMF is the use of stochastic gradients and mini-batch size. Algorithmically, the relations involving the final state gradient map Φ_K presented so far correspond to mini-batches of size 1. However, in Section 4.3.2, we provide an overview of the complete training scheme, which generalizes naturally to any batch size by leveraging the linear dependence of the recursively updated gradients on their previous iterate. During training, we employed mini-batches of varying size for our physics-aware Deep-NMF, up to 50% of the available training set. In contrast, the standard Deep-NMF proposed by [64] relied on pure stochastic gradient updates (mini-batches of size 1). Our experiments indicate that larger mini-batches result in a more effective and robust training process, enabling the network to learn our physics-based datasets more efficiently and accurately. Figure 4.5 illustrates the Deep-NMF outputs for different mini-batch sizes. To emphasize differences in reconstruction accuracy, a discriminative dictionary initialization was employed. A plausible explanation for this behavior is that accurately retrieving features from a dictionary constructed from deterministic, physically modeled target sources requires higher precision than a pure stochastic gradient can provide. Conversely, in [64], the objective was simply to attenuate noise in speech signals, for which mini-batches of size 1 were sufficient to learn a sufficiently general Wiener filter. In comparison, our datasets demand a more precise learning process, motivating the use of larger mini-batches.

4.3.2 Backpropagation

Compared to the standard Deep-NMF architecture, PAD-NMF exhibits two principal differences. The first is the introduction of binary masks M^W and M^H, which enforce the Hankel-augmented structure of the dictionary and coefficient matrices, respectively. The second is the modification of the outer objective function, which now includes an additional regularization term. Both changes affect the backpropagation mappings defined

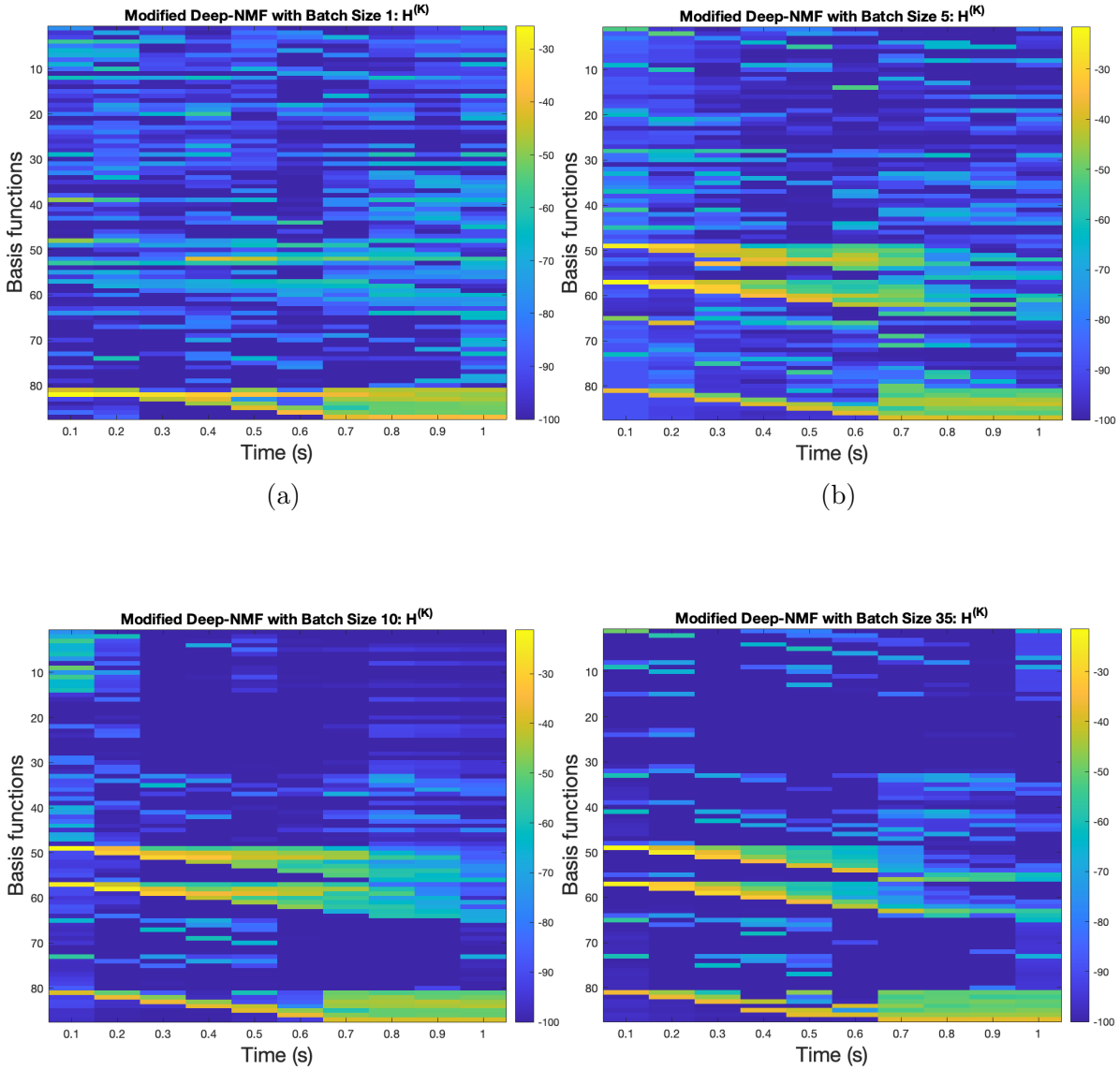


Figure 4.5: Effect of mini-batch size on the final output matrices $H^{(K)}$ of a modified Deep-NMF algorithm. Shown are results for mini-batch sizes of (a) 1, (b) 5, (c) 10, and (d) 35 on a dataset containing 70 instances. Here, 'modified' refers to the use of discriminative dictionary initialization. Parameters are set to $K = 1000$, $C = 10$, and $T = 7$. All matrices are displayed on a log scale.

for the Deep-NMF architecture in Section 4.2.2. In addition, [22] introduces a third, more technical modification concerning the parameter gradient map Φ_W. By slightly altering the decomposition of the gradients into positive and negative components, a simplified expression for Φ_W is obtained, involving only three terms instead of the four originally proposed in [64]. Details of this derivation are provided in Appendix 4.A, and the resulting modified mappings are reported below.
The parameter gradient map for the PAD-NMF architecture is given by:

$$
\begin{aligned}
[\Phi_W(\nabla_H, H, W, X)]_{\pm} &= \frac{X}{WH}\left(\frac{H}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ [\nabla_H]_{\pm} \circ M^H\right)^\top \\
&+ \left[\frac{X}{(WH)^{\circ 2}} \circ \left[W\left(\frac{H}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ [\nabla_H]_{\mp} \circ M^H\right)\right]\right] H^\top \\
&+ \mathbb{1}_{m\times n}\left[\frac{H}{(W^\top \mathbb{1}_{m\times n} + \alpha_H)^{\circ 2}} \circ \left(W^\top \frac{X}{WH}\right) \circ [\nabla_H]_{\mp} \circ M^H\right]^\top
\end{aligned}
\tag{4.3.22}
$$

The state gradient update map is also modified by taking into account M^H:

$$
\begin{aligned}
[\Phi_H(\nabla_H, H, W, X)]_{\pm} &= \frac{W^\top\left(\frac{X}{WH}\right)}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ [\nabla_H]_{\pm} \circ M^H \\
&+ W^\top\left[\frac{X}{(WH)^{\circ 2}} \circ \left[W\left(\frac{H}{W^\top \mathbb{1}_{m\times n} + \alpha_H} \circ [\nabla_H]_{\mp} \circ M^H\right)\right]\right]
\end{aligned}
\tag{4.3.23}
$$

Lastly, the final state gradient map shows the contribution of the added penalty:

$$
\begin{aligned}
[\Phi_K(H, X)]_{+} &= \begin{bmatrix} \hat{W}_S^\top\left(\frac{X^{\circ 2} \circ \hat{W}_S H_S \circ \hat{W}_N H_N}{\left(\hat{W}H\right)^{\circ 3}} + \alpha_S \hat{W}_S H_S\right) \\ \hat{W}_N^\top\left(\frac{X \circ S \circ \hat{W}_S H_S}{\left(\hat{W}H\right)^{\circ 2}}\right) \end{bmatrix} \\
[\Phi_K(H, X)]_{-} &= \begin{bmatrix} \hat{W}_S^\top\left(\frac{X \circ S \circ \hat{W}_N H_N}{\left(\hat{W}H\right)^{\circ 2}} + \alpha_S S\right) \\ \hat{W}_N^\top\left(\frac{X^{\circ 2} \circ \left(\hat{W}_S H_S\right)^{\circ 2}}{\left(\hat{W}H\right)^{\circ 3}}\right) \end{bmatrix}
\end{aligned}
\tag{4.3.24}
$$

Algorithm 4.3.2 shows the complete training scheme for the PAD-NMF architecture.

4.3.3 Detection indices

Let us assume that the PAD-NMF network has been initialized, as described in Section 4.3.1, using the Initialization Sets $\{\tilde{S}_i\}_{i=1}^I$ and $\{\tilde{N}_j\}_{j=1}^J$, and subsequently trained on a collection of triplets $\{(\tilde{X}_q, \tilde{S}_q, \tilde{N}_q)\}_{q=1}^Q$. Moreover, suppose that we are given a partition of the set:

Algorithm 4.3.2: Physics-aware Deep-NMF training

Input: A discriminatively initialized set of weights $\{W^{(k)}\}_{k=0}^{K}$ and a training set $\{(\tilde{X}_q, \tilde{S}_q, \tilde{N}_q)\}_{q=1}^{Q}$.

Output: A set of trained weights $\{W^{(k)}\}_{k=0}^{K}$.

for Epoch $= 1 :$ EpochsNum **do:**
  Permute training set
  for BatchBegin $= 1 :$ BatchSize $: Q$
    Initialize $[\Sigma_{\nabla^{(k)}}]_{\pm} = 0 \;\; \forall k = K - C, \ldots, K - 1$
    for $q =$ BatchBegin $:$ BatchBegin+BatchSize-1 **do:**
      $X = \mathcal{H}_T(\tilde{X}_q), \quad S = \mathcal{H}_T(\tilde{S}_q) \qquad (4.3.1)$
      Randomly initialize $H^{(0)} \in \mathcal{M}_{r \times n}(\mathbb{R}^+)$
      for $k = 0 : K - 1$ **do:**
        $H^{(k+1)} = f^{\,in}(H^{(k)}, W^{(k)}, X) \qquad (4.3.15)$
      end for
      Compute $\left[\nabla_{H^{(K)}} \hat{\mathcal{F}}\right]_{\pm}$ using $\Phi_K \qquad (4.3.24)$
      for $k = K - 1 : -1 : K - C$ **do:**
        Compute $\left[\nabla_{W^{(k)}} \hat{\mathcal{F}}\right]_{\pm}$ using $\Phi_W \qquad (4.3.22)$
        Update $\left[\nabla_{H^{(k)}} \hat{\mathcal{F}}\right]_{\pm}$ using $\Phi_H \qquad (4.3.23)$
        $[\Sigma_{\nabla^{(k)}}]_{\pm} = [\Sigma_{\nabla^{(k)}}]_{\pm} + [\nabla_{W^{(k)}} \mathcal{E}]_{\pm}$
      end for
      for $k = K - C : K - 1$ **do:**
        $W^{(k)} = W^{(k)} \circ \frac{[\Sigma_{\nabla^{(k)}}]_{-}}{[\Sigma_{\nabla^{(k)}}]_{+}} \circ M^W \qquad (4.3.8)$
      end for
    end for
  end for
end for

$$\{1, \ldots, I\} = \bigcup_{l=1}^{L} F_l \qquad (4.3.25)$$

into L subsets of *feature indices* F_l, which induce an analogous partition of the Initialization Set's target spectrograms into *feature sets*:

$$\{\tilde{S}_i\}_{i=1,\dots,I} = \bigcup_{l=1}^{L} \{\tilde{S}_i\}_{i\in F_l} \tag{4.3.26}$$

This partition should reflect the distinct features of the signals generated by the physico-mathematical model that we aim to detect and subsequently discriminate. As an illustrative example, consider an n-mass system composed of multiple macro-components that induce a partition of the masses. If the objective is to identify which macro-component has been excited by an impulse, the Initialization Set may be constructed by probing the system response to impulses applied at each individual mass and then partitioning the resulting spectrograms accordingly. Numerical results for this type of scenario are presented in Section 4.3.5. Given a new mixture $\tilde{X}$ whose decomposition into a target spectrogram $\tilde{S}$ and noise spectrogram $\tilde{N}$ is unknown, our goal is to determine which feature among those represented in the Initialization Set is most dominant within $\tilde{S}$. To ensure that such a determination is meaningful, we assume prior knowledge of the set of possible features to be sufficiently comprehensive. This requirement motivates the need for a rich Initialization Set. Nevertheless, there is no unique procedure for constructing such a set, as it depends both on the intrinsic complexity of the underlying model and on the specific detection task of interest.
Once we have propagated the mixture $\tilde{X}$ through the net, we obtain a final coefficient matrix $H^{(K)}$ and the associated optimal dictionary $W^{(K)}$, which we recall can be decomposed into blocks as in (4.3.10) and (4.3.5), respectively. By grouping those blocks based on the feature indices $F_l = \{l_1, \dots, l_{|F_l|}\}$, we can define:

$$\begin{aligned} W^{(K)}_{S_{F_l}} &= \begin{bmatrix} W^{(K)}_{S_{l_1}} & \cdots & W^{(K)}_{S_{l_{|F_l|}}} \end{bmatrix} \\ H^{(K)}_{S_{F_l}} &= \begin{bmatrix} H^{(K)}_{S_{l_1}} \\ \vdots \\ H^{(K)}_{S_{l_{|F_l|}}} \end{bmatrix} \end{aligned} \qquad \forall l = 1, \dots, L \tag{4.3.27}$$

In particular, the matrix $H^{(K)}_{S_{F_l}}$ provides a quantitative measure of the relative contribution of the feature-specific dictionary $W^{(K)}_{S_{F_l}}$ to the reconstruction of the estimated augmented spectrogram $S = \mathcal{H}_T(\tilde{S})$[1]. Owing to the Hankel structure of each dictionary sub-block $W^{(K)}_{S_{l_j}}$, the presence of the corresponding feature in the mixture $\tilde{X}$ manifests itself as nonzero coefficients along the main diagonal of the associated coefficient sub-block $H^{(K)}_{S_{l_j}}$, with magnitudes that reflect the relevance of that feature. Consequently, by denoting the main diagonals of $H^{(K)}_{S_{l_j}}$ as row vectors:

$$d^{(K)}_{l_j} = \text{diag } H^{(K)}_{S_{l_j}} \qquad \forall l = 1, \dots, L,\ \forall j = 1, \dots, |F_l| \tag{4.3.28}$$

and their feature-wise concatenations:

[1]Since $\tilde{X} \approx \tilde{S} + \tilde{N}$, by linearity of the Hankel extension operator we obtain $X = \mathcal{H}_T(\tilde{X}) \approx \mathcal{H}_T(\tilde{S}) + \mathcal{H}_T(\tilde{N}) = S + N$.

$$D_l^{(K)} = \begin{bmatrix} d_{l_1}^{(K)} & \cdots & d_{l_{|F_l|}}^{(K)} \end{bmatrix} \qquad \forall l = 1, \ldots, L \tag{4.3.29}$$

we employ the geometric mean of the concatenated diagonals (4.3.29) as the feature detection index $\mathcal{I}(l)$:

$$\mathcal{I}(l) = \mathrm{GM}\,(D_l^{(K)}) \qquad \forall l = 1, \ldots, L \tag{4.3.30}$$

We emphasize that, due to the nonnegative thresholding inherent in the multiplicative update rules for the coefficient matrices, as discussed in Section 3.2.1, none of the sub-block diagonals can be exactly zero. When a feature is absent and therefore not detected, the corresponding sub-block diagonal consists of small thresholded values, typically close to machine precision. This property ensures that, even in cases where only one diagonal $d_{l_j}^{(K)}$ within a given set of feature indices F_l is significantly positive, the geometric mean $\mathrm{GM}(D_l^{(K)})$ remains nonzero and thus defines a well-posed detection index. In its simplest form, the detection procedure therefore concludes that the feature most strongly present in the clean signal spectrogram $\tilde{S}$ is the one associated with the feature indices $F_{\bar{l}}$, where:

$$\bar{l} = \underset{l \in \{1,\ldots,L\}}{\mathrm{argmax}}\ \mathcal{I}(l) \tag{4.3.31}$$

Moreover, we can also render the above-described procedure more robust by introducing a set of feature-specific thresholds $\{\tau_l\}_{l=1,\ldots,L}$, one for each feature set, and letting:

$$\bar{l} = \underset{l \in \mathcal{L}}{\mathrm{argmax}}\ \mathcal{I}(l), \qquad \mathcal{L} = \{l \in \{1, \ldots, L\} \,:\, \mathcal{I}(l) > \tau_l\} \tag{4.3.32}$$

In Sections 4.3.5 and 4.3.6 we will provide an example on how such a threshold set can be chosen in order to attenuate some bias towards one or more feature set due to the presence of clean signal features within the noise component of the mixture. Lastly, it will be useful to define the partial index $\mathcal{I}(l_j) = \mathrm{GM}\,(d_{l_j}^{(K)})$.
Algorithm 4.3.3 shows the complete hit detection procedure.

4.3.4 Uncertainty reduction

The hit detection procedure can be further enhanced by incorporating an uncertainty reduction module. In certain scenarios, particularly when the target sources and the noise share a substantial portion of their spectral content, the sub-blocks of $\hat{W}_S$ may partially account for noise components. This can lead to a contamination of the corresponding coefficient blocks in $H_S^{(K)}$, which in turn may degrade the reliability of the hit detection indices $\mathcal{I}(l)$. To mitigate this issue, we now outline the approach proposed in [25].
The idea is to form the concatenation of the detection indices:

$$\mathcal{I} = \begin{bmatrix} \mathcal{I}(1) & \cdots & \mathcal{I}(L) \end{bmatrix}^\top \tag{4.3.33}$$

and interpret it as a possibly noisy, *active* detection vector. To carry out the actual detection, we introduce a *latent* detection vector $\hat{\mathcal{I}} \in \mathbb{R}^L_{\geq 0}$, which should be interpreted as a denoised version of $\mathcal{I}$. These two vectors are then connected, within a Bayesian framework, through a linear operator $\hat{A} \in \mathcal{M}_{L\times L}(\mathbb{R}_{\geq 0})$:

Algorithm 4.3.3 Hit detection

Input: A set of trained weights $\{W^{(k)}\}_{k=0}^{K}$, a set of feature-specific thresholds $\{\tau_l\}_{l=1,\dots,L}$, and a mixture spectrogram $\tilde{X}$.

Output: A feature index $\bar{l} \in \{1, \dots, L\}$ associated to the recognized feature within $\tilde{X}$.

1: $X = \mathcal{H}_T(\tilde{X})$ (4.3.1)

2: Randomly initialize $H^{(0)} \in \mathcal{M}_{r\times n}(\mathbb{R}_{\geq 0})$

3: **for** $k = 0 : K-1$ **do:**

4: $H^{(k+1)} = f^{in}(H^{(k)}, W^{(k)}, X)$ (4.3.15)

5: **end for**

6: Extract $d_{l_j}^{(K)}$ $\forall l = 1, \dots, L$ $\forall j = 1, \dots, |F_l|$ (4.3.10)-(4.3.27)-(4.3.28)

7: Assemble $D_l^{(K)}$ and compute $\mathcal{I}(l)$ $\forall l = 1, \dots, L$ (4.3.29)-(4.3.30)

8: Compute $\bar{l}$ (4.3.32)

$$\hat{A}\hat{\mathcal{I}} = \mathcal{I} \tag{4.3.34}$$

Given $\hat{A}$ and once $\mathcal{I}$ has been constructed from the PAD-NMF output, the latent vector $\hat{\mathcal{I}}$ is obtained by matrix inversion within this Bayesian framework. Let us now describe how to construct $\hat{A}$ and how to recover $\hat{\mathcal{I}}$. A visual description of the process is shown in Figures 4.6 and 4.7.

From the training set $\{(\tilde{X}_q, \tilde{S}_q, \tilde{N}_q)\}_{q=1}^{Q}$ used to train the PAD-NMF architecture, we construct, for each training instance, the active vector $\mathcal{I}_q$ as defined in Step 7 of Algorithm 4.3.3, together with a corresponding target latent vector (or label) $\hat{\mathcal{I}}_q$. The latter can, for instance, be chosen as a binary vector encoding the specific feature present in the training triplet $(\tilde{X}_q, \tilde{S}_q, \tilde{N}_q)$. We then concatenate these vectors in two matrices:

$$\hat{I}_Q = \begin{bmatrix} \hat{\mathcal{I}}_1 & \cdots & \hat{\mathcal{I}}_Q \end{bmatrix} \qquad I_Q = \begin{bmatrix} \mathcal{I}_1 & \cdots & \mathcal{I}_Q \end{bmatrix} \tag{4.3.35}$$

Given the nonnegativity of both matrices, the linear operator $\hat{A}$ is obtained by solving an NMF optimization problem over the left factor:

$$\begin{aligned} \min \quad & D_\beta(I_Q, \hat{A}\hat{I}_Q) \\ \text{s.t.} \quad & \hat{A} \in \mathcal{M}_{L\times L}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{4.3.36}$$

where β implicitly governs the noise assumptions on the active vectors $\mathcal{I}_q$, as discussed in Section 3.2. Since the indices $\mathcal{I}_q(l)$ are constructed from geometric means, the multiplicative (Gamma) noise model corresponding to $\beta = 0$ constitutes a particularly natural and well-suited choice.

During inference, $\hat{A}$ is assumed to be known, and a new vector of active indices $\mathcal{I}$ is computed from the coefficient matrix $H^{(K)}$ using Algorithm 4.3.3. To recover the estimated latent vector $\hat{\mathcal{I}}$, we then employ a similar regularized NMF model, this time optimizing only with respect to the right factor:

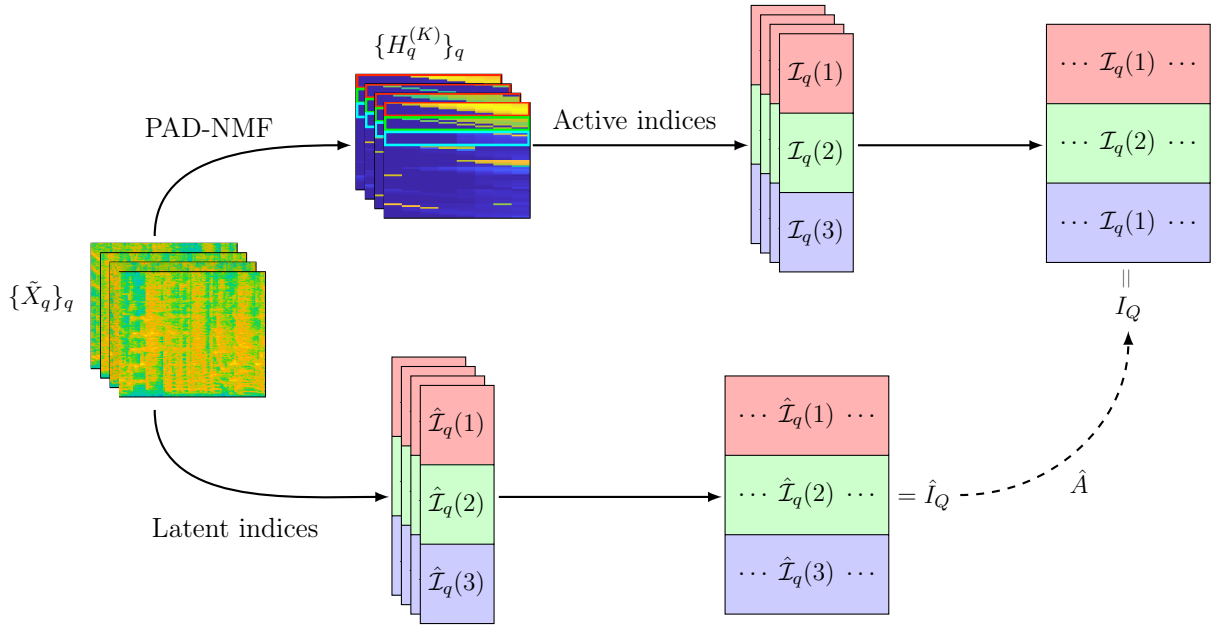


Figure 4.6: Construction of the linear operator $\hat{A}$ mapping latent indices to active indices in the case of $L = 3$ features.

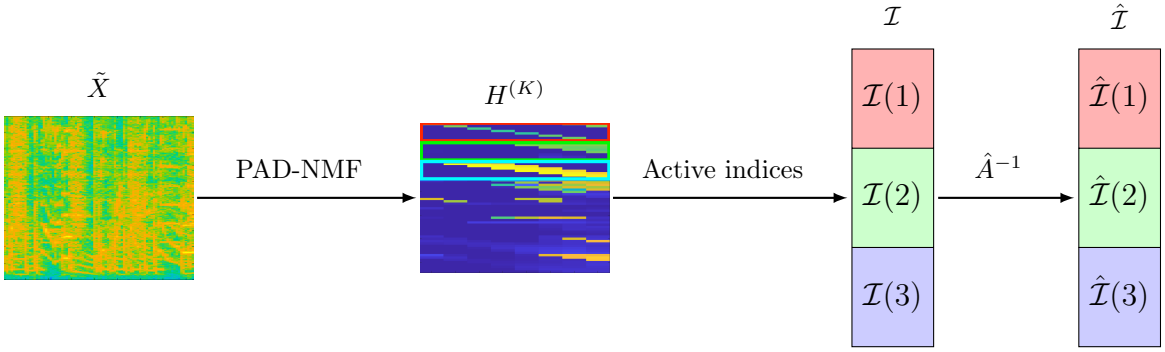


Figure 4.7: Inference of the latent index vector in the case of $L = 3$ features.

$$\begin{aligned} \min \quad & D_\beta(\mathcal{I}, \hat{A}\hat{\mathcal{I}}) + \alpha_{\hat{\mathcal{I}}}\|\hat{\mathcal{I}}\|_p^p \\ \text{s.t.} \quad & \hat{\mathcal{I}} \in \mathcal{M}_{L\times 1}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{4.3.37}$$

Given the desired sparsity of the latent detection vectors, we employ an aggressive sparsity regularizer in the form:

$$R(x) = \|x\|_p^p = \sum_i |x_i|^p \tag{4.3.38}$$

with $p \in (0, 1]$. We note that for $p < 1$ this regularizer is concave; nevertheless, (4.3.37) can still be optimized in practice using the heuristic update rule (3.2.26). Moreover, as discussed in Section 3.2, problem (4.3.37) is equivalent, within a Bayesian framework, to a MAP estimate with a sparsity-promoting exponential prior [19] given by:

$$p(\hat{\mathcal{I}}) \propto e^{-\alpha_{\hat{\mathcal{I}}}\|\hat{\mathcal{I}}\|_p^p} \tag{4.3.39}$$

We also remark that, although (4.3.37) is essentially equivalent to solving the linear system (4.3.34), it can be implemented far more easily on resource-constrained hardware, thanks to the low computational cost of the NMF updates.

4.3.5 Numerical experiments: hit detection in dynamical systems

We evaluated the hit detection procedure described in Algorithm 4.3.3 on a general and paradigmatic model, namely the multiple degrees-of-freedom system illustrated in Figure (4.8). As is well known, this model encompasses a wide range of lumped-parameter systems and serves as a discretized representation of continuum models arising in linear elastodynamics. We considered several system configurations and report here results obtained from two representative cases: one consisting of $n = 23$ masses, springs, and dampers, and another with $n = 28$ such elements. External forces were applied to arbitrary subsets of the masses, and the observed mixture was taken as the sum of their displacements. This modeling choice was made to ensure that the resulting mixtures could be meaningfully interpreted as audio signals captured by a microphone.

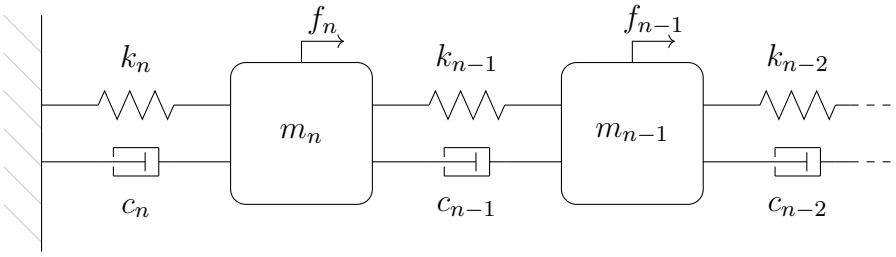


Figure 4.8: General n-degrees-of-freedom model used for the generation of audio mixture datasets.

Datasets

Let us denote by *Dataset1* and *Dataset2* the two synthetic datasets used to evaluate the proposed PAD-NMF–based hit detection procedure. As detailed below, each dataset consists of mixtures generated from a specific instance of the n-mass model introduced above. In particular, each mass is subjected either to a fixed, fully deterministic periodic excitation, giving rise to the noise component of the mixture, or to an impulsive excitation, which constitutes the target signal to be detected. We emphasize that the use of deterministic noise excitations capable of producing system responses similar to those induced by impulsive inputs is a deliberate design choice, intended to rigorously stress-test the proposed hit detection algorithm. The specific forcing function employed in our experiments is given by:

$$f(t) = \left(1 - e^{-\frac{1}{2}t^2}\right) \sum_{i=1}^{25} \sin(i\omega t) \; = A(t) \sum_{i=1}^{25} \sin(i\omega t), \qquad \omega = 2800\,\mathrm{rad/s} \tag{4.3.40}$$

The forcing described above is representative of a typical *engine order excitation* encountered in rotating machinery. Note that the transient behavior modeled by $A(t)$ is identical

for all harmonics. This assumption is not restrictive for our experiments, as each 1 s mixture acquisition begins at least 3 s after the deterministic forcing in (4.3.40) is activated, by which time the system has reached steady state. Likewise, neglecting phase shifts in the harmonic components of the forcing does not impose a limitation. Indeed, the natural frequencies of the n-mass models considered here do not coincide with the excitation harmonics. In practical applications, such a coincidence would be undesirable and would instead motivate a different choice of ω, aside from its low statistical likelihood. Moreover, due to the spectrogram-based formulation of PAD-NMF, the algorithm applied to *Dataset1* and *Dataset2* is inherently insensitive to phase shifts. These simplifications allow us to restrict the number of experimental configurations to the minimum necessary. In the following, we compare the proposed PAD-NMF approach with the standard Deep-NMF algorithm. To ensure a fair comparison, both networks employ identical architectures.

·*Dataset1*. The first dataset is generated by setting $n = 23$ and adopting the following model parameters: $M = [m_{1:8} = 2.00 \times 10^{-4}\,, m_{9:12} = 1.05\,, m_{13:23} = 7.04 \times 10^{-2}\,]$ kg, $K = [k_{1:8} = 1.36 \times 10^{7}\,, k_{9:13} = 2.00 \times 10^{6}\,, k_{14:19} = 4.00 \times 10^{8}\,, k_{20:23} = 8.00 \times 10^{8}\,]$ N/m, $C = [c_{1:7} = 2.14 \times 10^{-1}\,, c_{8} = 1.10\,, c_{9:12} = 3.00 \times 10^{3}\,, c_{13} = 1.50\,, c_{14:23} = 1.50 \times 10^{-1}\,]$ N s/m. Table 1 and Figure 4.9 provide a modal representation of such model. The noise forcing (4.3.40) is applied to mass m_{10}. Six impulses at varying intensities are applied to each of the masses $m_{1:8}$ and $m_{14:23}$. As a consequence, for any of the latter, *Dataset1* will contain six mixtures at different SNRs, with an impulse response at the beginning of each mixture. The value for the impulses' intensities are chosen in order to produce mixtures at SNRs ranging from -40 dB to 10 dB. Moreover, we also include 18 mixtures containing no impulses.

·*Dataset2*. The second dataset is generated by setting $n = 28$ and adopting the following model parameters: $M = [m_{1:4} = 5.25 \times 10^{-2}\,, m_{5:6} = 1.00 \times 10^{3}\,, m_{7:10} = 1.50 \times 10^{-1}\,, m_{11:12} = 1.00 \times 10^{3}\,, m_{13:16} = 2.00 \times 10^{-1}\,, m_{17:18} = 1.00 \times 10^{3}\,, m_{19:22} = 2.50 \times 10^{-1}\,, m_{23:24} = 1.00 \times 10^{3}\,, m_{25:28} = 3.50 \times 10^{-1}\,]$ kg, $K = [k_{1:4} = 1.27 \times 10^{6}\,, k_{5} = 1.00 \times 10^{5}\,, k_{6:10} = 2.20 \times 10^{7}\,, k_{11} = 1.00 \times 10^{5}\,, k_{12:16} = 9.60 \times 10^{7}\,, k_{17} = 1.00 \times 10^{5}\,, k_{18:22} = 2.25 \times 10^{8}\,, k_{23} = 1.00 \times 10^{5}\,, k_{24:18} = 6.30 \times 10^{8}\,]$ N/m, $C = [c_{1:4} = 5.00\,, c_{5} = 5.00 \times 10^{1}\,, c_{6:10} = 7.00\,, c_{11} = 5.00\,, c_{12:16} = 5.00\,, c_{17} = 5.00 \times 10^{1}\,, c_{18:22} = 5.00\,, c_{23} = 5.00 \times 10^{1}\,, c_{24:28} = 7.00\,]$ N s/m. Table (2) and Figure (4.10) provide a modal representation of such model. The noise forcing (4.3.40) is applied to masses m_4, m_{10}, m_{16}, m_{22} and m_{27}. Six impulses at varying intensities are applied to each of the masses $m_{2:3}$, $m_{8:9}$, $m_{14:15}$, $m_{20:21}$, m_{26} and m_{28}. As a consequence, for any of the latter, *Dataset2* will contain six mixtures at different SNRs, with an impulse response at the beginning of each mixture. The value for the impulses' intensities are chosen in order to produce mixtures at SNRs ranging from -40 dB to 20 dB. Moreover, we also include 10 mixtures containing no impulses.

Table 1: Modal representation of the 23-DOF model for *Dataset1*, showing each mode's damped frequency f and damping ratio ξ.

Mode	f(Hz)	ξ
1	0	1.0
2	0	1.0
3	0	1.0
4	64	1.9×10^{-1}
5	180	6.6×10^{-1}
6	2229	9.0×10^{-4}
7	3402	1.1×10^{-3}
8	6086	1.0×10^{-4}
9	9622	4.5×10^{-5}
10	10080	1.4×10^{-3}
11	13504	3.6×10^{-5}
12	16686	2.1×10^{-3}
13	16966	2.8×10^{-5}
14	19414	2.5×10^{-5}
15	20096	1.5×10^{-3}
16	21556	2.7×10^{-5}
17	23315	2.8×10^{-5}
18	24334	1.2×10^{-3}
19	26638	1.5×10^{-3}
20	26895	1.8×10^{-5}
21	32066	1.9×10^{-5}
22	57662	1.9×10^{-3}
23	78012	2.6×10^{-3}

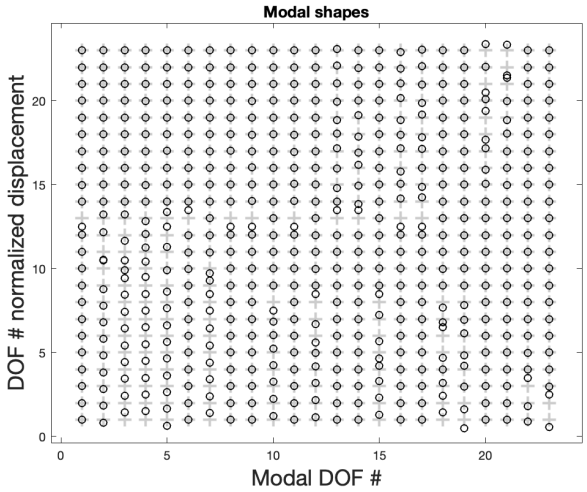


Figure 4.9: Modal shapes of the 23-DOF model used for *Dataset1*, shown as normalized displacements. Black circles indicate mass displacements from equilibrium, marked by equidistant gray crosses.

Table 2: Modal representation of the 28-DOF model for *Dataset2*, showing each mode's damped frequency f and damping ratio ξ.

Mode	f(Hz)	ξ
1	1	7.0×10^{-4}
2	1	1.9×10^{-3}
3	2	2.9×10^{-3}
4	2	3.5×10^{-3}
5	15	3.0×10^{-4}
6	31	1.0×10^{-4}
7	48	1.0×10^{-4}
8	57	1.0×10^{-4}
9	282	3.7×10^{-3}
10	785	1.0×10^{-2}
11	1170	1.1×10^{-3}
12	1185	1.2×10^{-2}
13	1561	1.9×10^{-2}
14	2155	4.0×10^{-4}
15	2169	2.0×10^{-3}
16	2961	2.0×10^{-4}
17	3417	3.7×10^{-3}
18	3928	3.9×10^{-3}
19	4108	8.0×10^{-4}
20	4169	2.0×10^{-4}
21	5439	5.0×10^{-4}
22	5642	8.0×10^{-4}
23	6646	1.1×10^{-3}
24	7853	3.0×10^{-4}
25	8090	5.0×10^{-4}
26	9204	6.0×10^{-4}
27	11159	4.0×10^{-4}
28	12992	5.0×10^{-4}

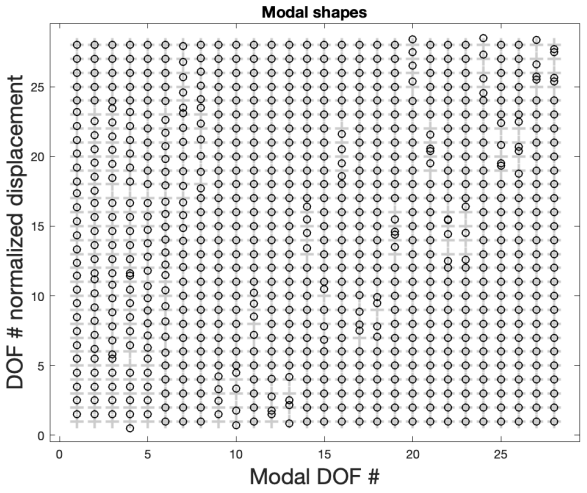
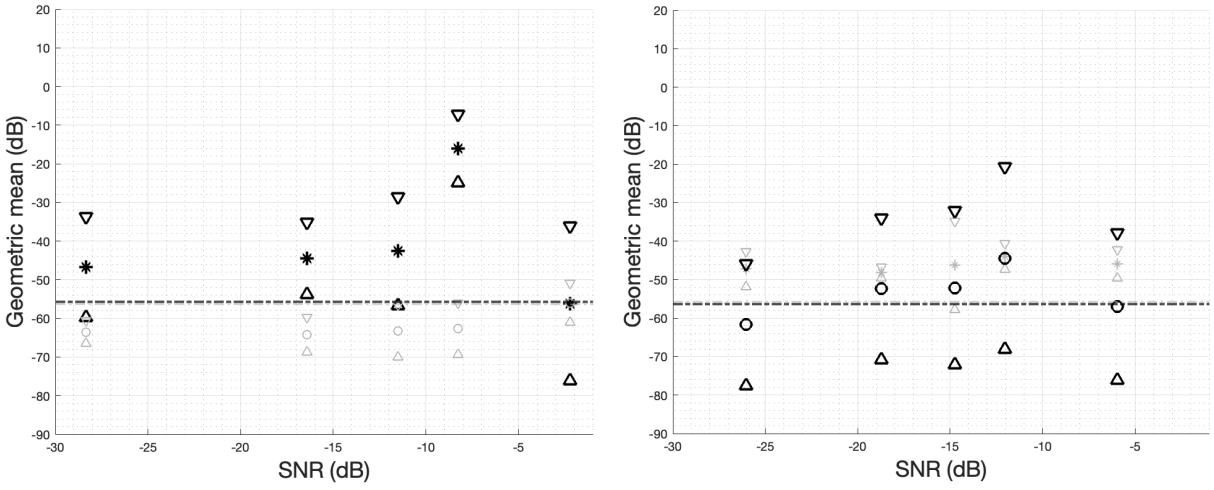


Figure 4.10: Modal shapes of the 28-DOF model used for *Dataset2*, shown as normalized displacements. Black circles indicate mass displacements from equilibrium, marked by equidistant gray crosses.

Two macro-components experiment

For this experiment, we consider Dataset1, which is generated from a model that, by construction, exhibits two dominant frequency responses. The first response arises when exciting masses $m_1, \ldots, m_8$ (first macro-component), while the second corresponds to excitations applied to masses $m_{14}, \ldots, m_{23}$ (second macro-component). Consequently, the dataset contains two distinct features that can be targeted for detection. The Initialization Sets for the PAD-NMF algorithm are constructed using target spectrograms obtained from impulsive excitations applied to the same masses at a fixed intensity, together with a single isolated forcing–response spectrogram to model the noise. The training set is formed by selecting, for each probed mass, the mixtures corresponding to the second-highest and second-lowest signal-to-noise ratios (SNRs), as well as all 18 mixtures containing no impulsive excitations. The testing set coincides with Dataset1 itself. The network parameters are set to $I = 18$, $J = 1$, $r_{S_i} \equiv 8$, $r_{N_j} \equiv 7$, $K = 1000$, $C = 20$, $T = 7$, $\alpha_H = 0$, and $\alpha_S = 10^6$. Training is performed over 3 epochs using mini-batches of size 27. Figure (4.11) illustrates selected detection indices computed from the outputs of both PAD-NMF and standard Deep-NMF. The thresholds τ_l are determined by taking the geometric mean of the corresponding detection indices calculated from all mixtures containing no impulses. This procedure effectively reduces any bias toward the target features that could be introduced by the deterministic noise forcing.

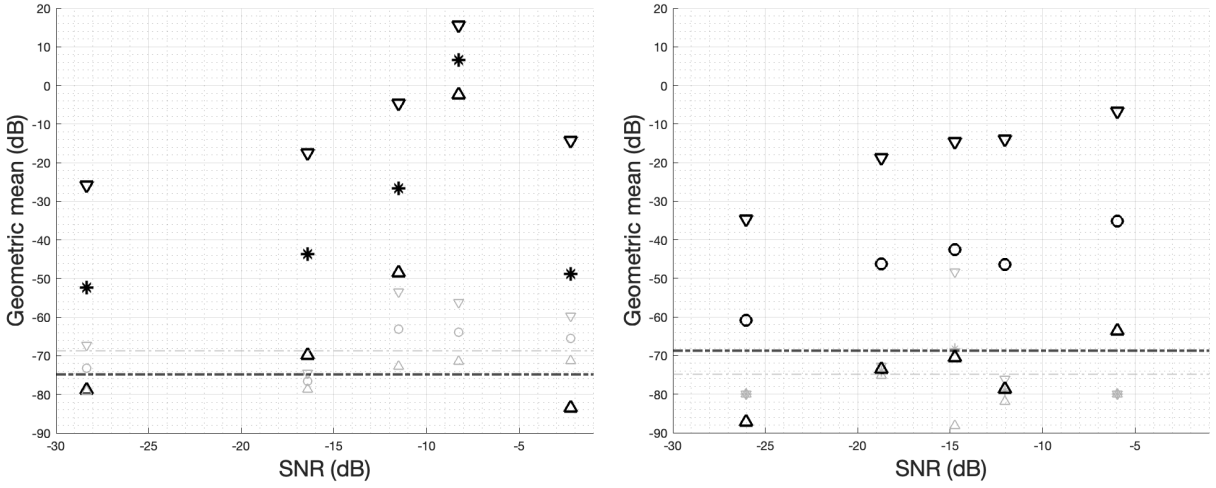


(a) (b)

In particular, Figure4.11b shows that, when applied to the Deep-NMF output, the detection algorithm fails to correctly identify any of the five displayed hits. In contrast, Figure4.11d demonstrates that the PAD-NMF output enables the detection procedure to correctly recognize the target feature in each test mixture.

Five macro-components experiment

For this experiment, we consider Dataset2, which is generated from a model that, by construction, exhibits five dominant frequency responses. These correspond to excitations applied to the following masses: m_2, m_3 (first macro-component), m_8, m_9 (second macro-component), m_{14}, m_{15} (third macro-component), m_{20}, m_{21} (fourth macro-component), and m_{26}, m_{28} (fifth macro-component). Consequently, the dataset contains five distinct



(c) (d)

Figure 4.11: Hit detection indices computed from Deep-NMF outputs (a), (b) and PAD-NMF outputs (c), (d) on *Dataset1* across five different SNRs. Hits correspond to the first macro-component (asterisk) in (a) and (c), and the second macro-component (circle) in (b) and (d). Triangular markers denote a 1-standard-deviation interval for each detection index[2]. Thresholds τ_l are displayed as dashed lines matching the thickness of the corresponding symbol. Symbols, standard deviation markers, and thresholds associated with the correct feature present in each test mixture are shown in bold.

features that can be detected. The Initialization Sets for the PAD-NMF algorithm are built using clean spectrograms obtained from impulsive excitations applied to the same masses at a fixed intensity, along with a single isolated forcing–response spectrogram to represent the noise. The training set is formed by selecting, for each probed mass, the mixtures corresponding to the second-highest and second-lowest signal-to-noise ratios (SNRs), in addition to all 10 mixtures containing no impulses. The testing set consists of Dataset2 itself. The network parameters are set as follows: $I = 10$, $J = 1$, $r_{S_i} \equiv 8$, $r_{N_j} \equiv 7$, $K = 1000$, $C = 40$, $T = 7$, $\alpha_H = 0$, and $\alpha_S = 10^6$. Training is performed over 3 epochs using mini-batches of size 30. Figure 4.12 shows some of the detection indices computed from the PAD-NMF output.
Despite the increased number of possible features to be detected, our proposed algorithm performs effectively up to -20 dB SNR test mixtures and, in some particular cases, namely those in Figures 4.12a and 4.12d, it was able to recognize features closing in on the -25 dB SNR mark.

Multiple hits experiment

We conclude this section by extending the detection procedure to handle multiple hits, i.e., cases where a single mixture contains two or more distinct features to be recognized.

[2]Since we chose $r_{S_i} \equiv 8$, we observe that $\mathcal{I}(l) = \mathrm{GM}(D_l^{(K)}) = \mathrm{GM}(\,\mathcal{I}(l_1), \ldots, \mathcal{I}(l_{|F_l|})\,)$, thus in a (dB) log-scale plot we get $\log \mathcal{I}(l) = \mathrm{mean}(\,\log \mathcal{I}(l_1), \ldots, \log \mathcal{I}(l_{|F_l|})\,)$ and we can similarly compute the standard deviation of the set $\{\log \mathcal{I}(l_j)\}_{j=1,\ldots,|F_l|}$

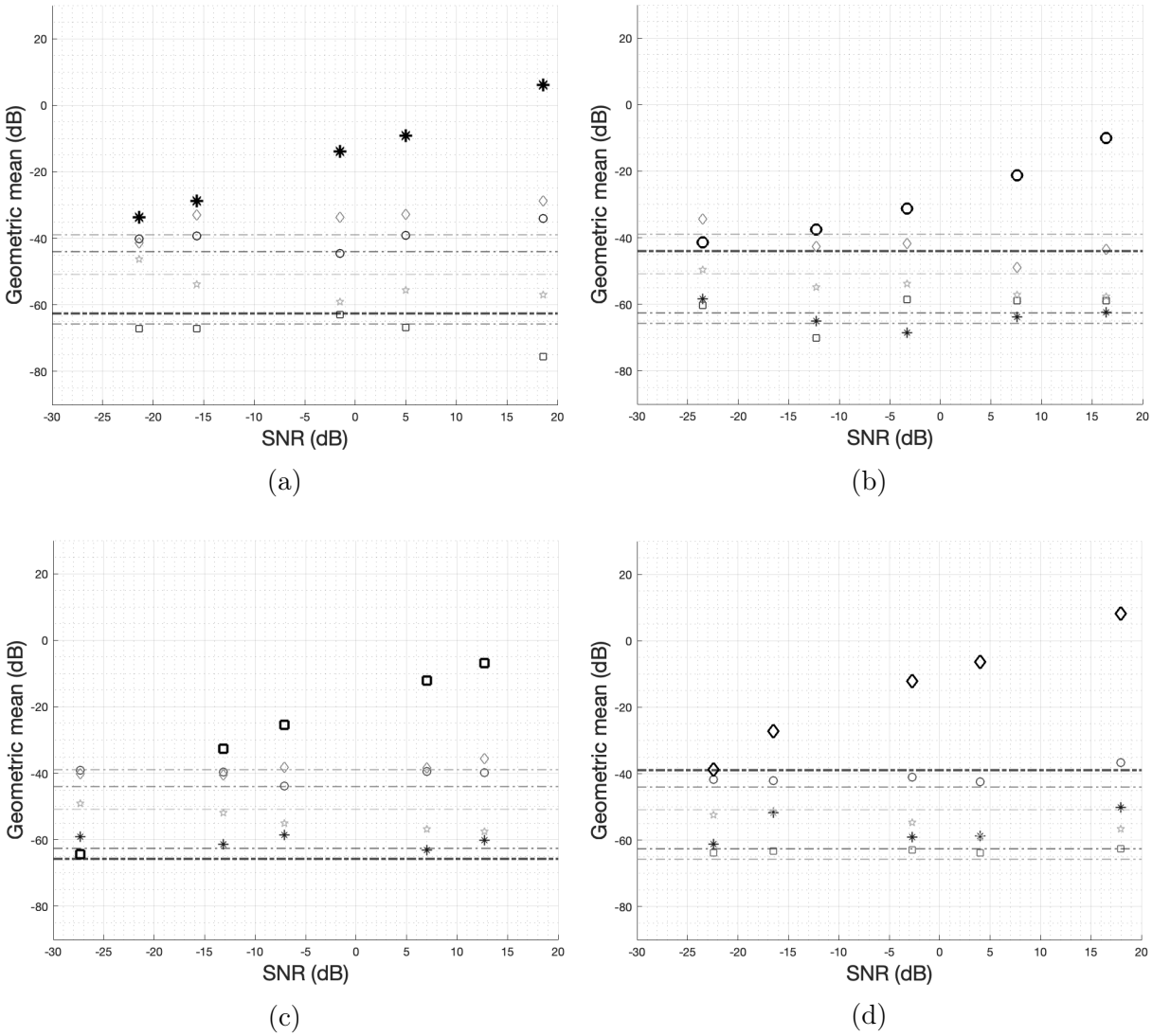
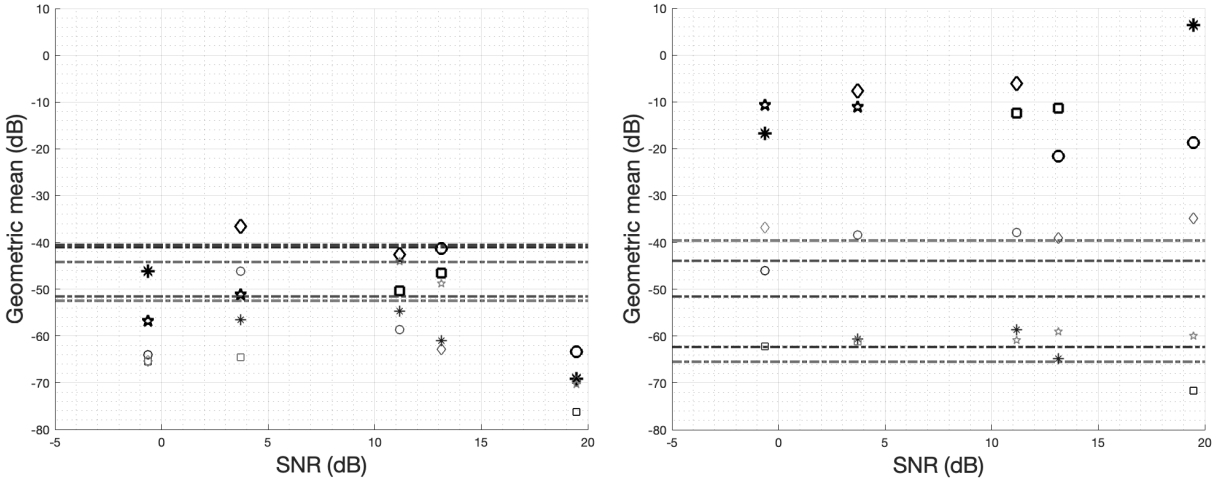


Figure 4.12: Hit detection indices computed from PAD-NMF outputs on Dataset2 across five different SNRs. Hits correspond to the first macro-component (asterisk) in (a), second macro-component (circle) in (b), third macro-component (square) in (c), and fourth macro-component (diamond) in (d). Thresholds τ_l are shown as dashed lines matching the thickness of the corresponding symbol. The symbol and threshold associated with the correct feature present in the test mixture are highlighted in bold.

Once again, we use Dataset2 and initialize and train the PAD-NMF network as described in the previous section. For testing, we consider five mixtures, each containing exactly two impulses applied to different macro-components, along with five mixtures containing no impulses. Figure 4.13 presents the detection indices computed from the outputs of both PAD-NMF and standard Deep-NMF.

Once again, Figure 4.13a shows that the source reconstruction provided by Deep-NMF is not accurate enough to correctly detect both features present in the mixtures. In all five test cases, at least one feature is missed, either because its corresponding detection index falls below the computed threshold or because an index associated with another, absent



(a) (b)

Figure 4.13: Hit detection indices computed from Deep-NMF output (a) and PAD-NMF output (b) on mixtures containing two distinct features. From high to low SNR: hits on the first (asterisk) and second (circle) macro-components, second and third (square) macro-components, third and fourth (diamond) macro-components, fourth and fifth (star) macro-components, and fifth and first macro-components. Thresholds τ_l are shown as dashed lines matching the thickness of the corresponding symbol. Symbols associated with the correct features present in each test mixture are highlighted in bold.

feature is larger. In contrast, Figure 4.13b demonstrates that the PAD-NMF algorithm successfully identifies all pairs of features, even though these specific mixtures were not included in the training set. Here, we assume prior knowledge that two features are present, so (4.3.31) is modified to return the features corresponding to the two largest detection indices.

4.3.6 Numerical experiments: detection of piano notes

We conclude this experimental section by evaluating the PAD-NMF algorithm on real-world audio signals. Specifically, we consider mixtures in which the hit-responses correspond to piano notes. Each piano note is generated by a hammer striking the piano strings, which can be modeled as a mechanical system and, when discretized, resembles the multiple degrees-of-freedom system shown in Figure 4.8. Compared to the previous synthetic examples, these mixtures are significantly more complex: the noise consists of a poly-instrumental music track, producing highly non-stationary and overlapping spectral patterns that partially coincide with the piano notes. The resulting audio signals are recorded at a sampling frequency of $f_s = 48\,\text{kHz}$ and compiled into *Dataset3*, which will be described in the following.

Dataset

Let us denote *Dataset3* as the real-world dataset used to evaluate the PAD-NMF algorithm. This dataset contains mixtures derived from a single 277 s poly-instrumental

music track, which includes a total of 84 piano notes, the hit-responses we aim to detect. The track is segmented into 277 consecutive 1 s mixtures. Unlike the previous synthetic datasets, the transient amplitudes here are roughly uniform, partially determined by the pianist's playing intensity. The diversity of the mixtures in *Dataset3* arises from the background music, which features frequency sweeps, drum beats, and other instruments, making the recognition of piano notes particularly challenging. Figure 4.14 illustrates the spectrogram of a segment containing one piano note embedded within the music track.

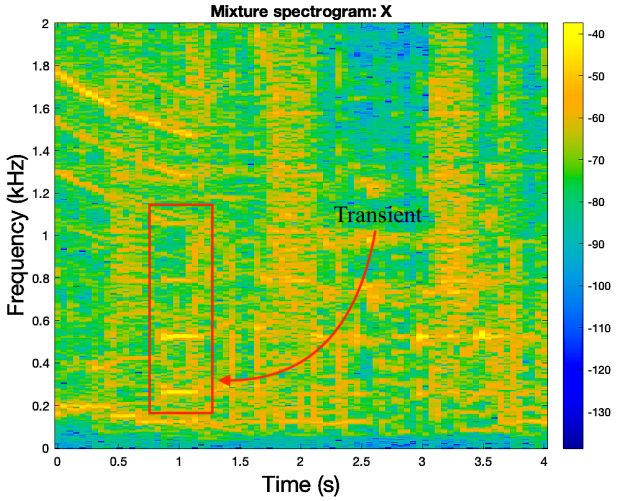


Figure 4.14: Spectrogram of a 4 s segment from the audio signal in *Dataset3*. The piano note (transient) within the segment is indicated by the red box.

·*Dataset3*. Of the 277 mixtures, only 84 contain an actual transient, while the remaining 193 consist solely of background noise. The piano notes in this dataset are C4, E4, and G4, played sequentially with at least one second between each note to ensure that each mixture contains at most one transient. The dataset is then divided into a training set and a testing set, as described in the following paragraph.

Three-notes experiment

For this experiment, we use *Dataset3*, which contains three distinct detectable features corresponding to the piano notes C4, E4, and G4. The Initialization Sets for the PAD-NMF algorithm are built using target spectrograms of these notes along with multiple isolated noise spectrograms capturing some of the repetitive instrumental patterns present in the background track. The training set consists of 12 mixtures: 9 containing piano notes (three instances each of C, E, and G) and 3 containing only background noise. The testing set includes all 277 mixtures. The network parameters are $I = 3$, $J = 55$, $r_{S_i} \equiv 8$, $r_{N_j} \equiv 8$, $K = 1000$, $C = 3$, $T = 8$, $\alpha_H = 0.05$, and $\alpha_S = 10^6$, and training is performed over 15 epochs with mini-batches of size 12. Figure 4.15 displays the detection indices computed from the PAD-NMF output for a subset of the testing set. The thresholds τ_l are obtained by taking the maximum detection index from the network's output across all training-set mixtures containing only background noise. Unlike in the previous experiments, detection indices for mixtures without any piano notes are also shown.
We note that the mixtures in *Dataset3* differ substantially in structure from those in the previous datasets. In particular, the clean sources are no longer constrained to appear at

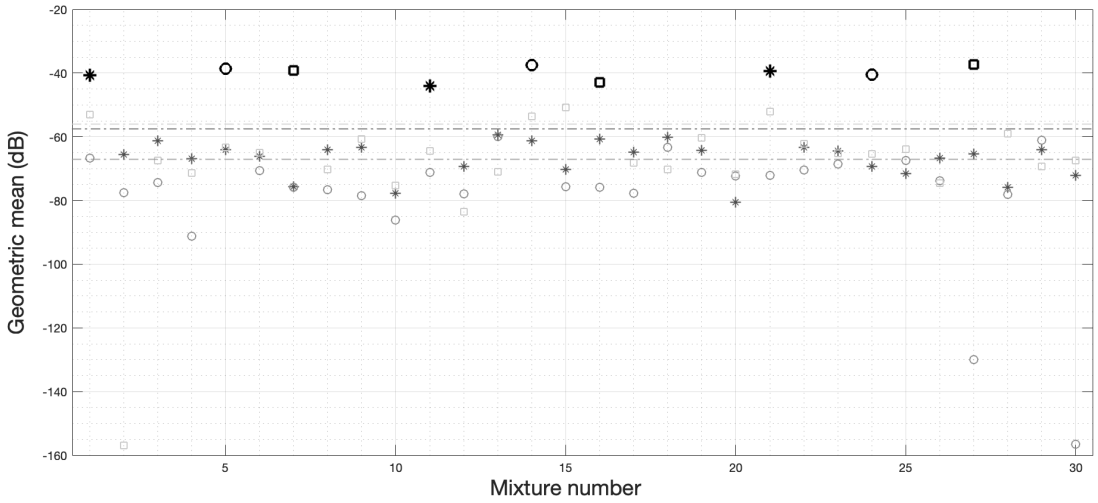


Figure 4.15: Hit detection indices computed from the PAD-NMF output on a selection of 30 consecutive mixtures. The first (C4), second (E4), and third (G4) features are represented by an asterisk, circle, and square, respectively. Thresholds τ_l are shown as dashed lines matching the thickness of the corresponding symbol. The symbol corresponding to the correct feature present in each test mixture is highlighted in bold.

the beginning of the signal, as was the case for the synthetic mixtures generated from the model in Figure 4.8. As a result, both the forward pass of the network and the detection procedure require slight modifications. During forward passes, since each target source can occur at any point within the mixture, we do not apply the mask M^H; equivalently, the PAD-NMF algorithm can be run with $M^H = \mathbb{1}_{r\times n}$. For the detection indices $\mathcal{I}(l)$, the procedure is largely unchanged, except that the computations in (4.3.28)-(4.3.30) must be repeated for all off-diagonals of the sub-blocks, with the feature index then taken as the maximum. This effectively generalizes the detection to handle clean sources that are time-shifted within the mixture.
Finally, Table 3 reports the detection results obtained using the uncertainty reduction method described in Section 4.3.4, tested under various noise assumptions. In all cases, a sparsity regularizer (4.3.38) with $p = 0.25$ was applied.

Table 3: Detection results (%) under normal ($\beta = 2$), Poisson ($\beta = 1$), and Gamma ($\beta = 0$) noise assumptions. Slashes "/" separate results for each note (C4/E4/G4); overall percentage is shown in brackets.

%	True positives	False negatives	True negatives	False positives
Normal	68/89/**100** (86)	32/11/**0** (14)	96	**1**/1/3 (4)
Poisson	54/89/75 (73)	46/11/25 (27)	96	3/0/**1** (4)
Gamma	**82**/**96**/**100** (**93**)	**18**/**4**/**0** (**7**)	**97**	2/**0**/**1** (**3**)

4.4 Deep-NMFD (DNMFD)

Most of the modeling efforts presented in Section 4.3.1 to adapt the Deep-NMF architecture to the hit detection task focused on embedding temporal correlation into the network. This was achieved through the use of Hankel-augmented matrices and binary masks to preserve the desired structure. An alternative way to overcome this limitation of Deep-NMF is provided by NMFD, a family of NMF-like algorithms that, as discussed in Section 3.3, naturally enforces temporal correlation within dictionary atoms via convolution. In this section, we introduce the Deep-NMFD architecture, originally proposed in , as another instance of the Deep Unfolding paradigm.

4.4.1 Network architecture

At a high level, the Deep-NMFD architecture closely resembles Deep-NMF, with only minor modifications. We again consider the hit detection task introduced in Section 3.4.2 and define $\bar{J} = J_S \dot{\cup} J_N$, where $J_S = 1, \dots, I$ and $J_N = 1, \dots, J$ index the target and noise components, respectively. The corresponding bilevel optimization reformulation can then be stated as follows: *among the NMFD factorizations that are optimal in the sense of* (3.4.14)*, we seek those that yield an optimal Wiener filter* F:

$$F = \frac{W_S * H_S}{W * H} = \frac{\sum_{j \in J_S} W(j) * H(j)}{W * H} \tag{4.4.1}$$

The inner objective function is therefore given by:

$$\mathcal{F}^{in}(H, W, X) = D_1(X, W * H) \tag{4.4.2}$$

As discussed in Section 3.3.1, the update rule (3.3.10) constitutes an optimization scheme for the inner problem (4.4.2). In the specific case $\beta = 1$, and using the notation introduced in Section 2.4.1, this yields the following inner map:

$$(f^{in})^j(H, W, X) = (f_W^{in})^j(H) = H(j) \circ \frac{\sum_{r=0}^{r_j - 1} W(j)_{:,r}^\top \left(\frac{\overleftarrow{X}^{r}}{W * H} \right)}{\sum_{r=0}^{r_j - 1} W(j)_{:,r}^\top \, \overleftarrow{\mathbb{1}_{m \times n}}^{r}} \qquad \forall j = 1, \dots, J \tag{4.4.3}$$

A minor difference with respect to the architecture proposed in [24] concerns the denominator of (4.4.3). While [24] adopts the multiplicative update rule originally introduced in [117], here we instead use the variant proposed in [129], which includes an additional shift operation applied to the all-ones matrix $\mathbb{1}_{m \times n}$. As for the outer optimization problem, given the target spectrogram S (which we consider part of the observation (X, S) for the inner problem), we define the following objective:

$$\mathcal{F}^{out}(H, W, (X, S)) = \mathcal{F}^{out}(H, W) = \frac{1}{2} \|S - F \circ X\|_2^2 + \frac{\alpha_S}{2} \|S - W_S * H_S\|_2^2 \tag{4.4.4}$$

where F is defined as in (4.4.1). The outer objective (4.4.4) is equivalent to PAD-NMF's (4.3.20).

As in the PAD-NMF case, we distinguish between non-discriminative and discriminative layers in the unrolled architecture generated by (4.4.3). For a given $C \leq K$, where K denotes the total number of layers, the first $K-C$ layers are chosen to be non-discriminative, while the remaining C layers, excluding the final reconstruction layer, are discriminative. This distinction implies that the dictionaries $W^{(k)}$ for $k \in 0, \ldots, K-C-1, K$ are not treated as network parameters and are instead fixed to the optimal dictionary $\hat{W}$. More specifically, since NMFD involves $\bar{J}$ dictionaries, in these layers we impose $W^{(k)}(j) = \hat{W}(j)$ for all $j \in \bar{J}$. An overview of the architecture is shown in Figure 4.16

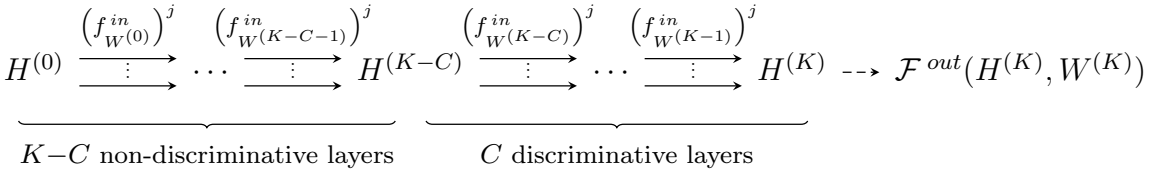


Figure 4.16: Deep-NMFD network architecture [24]. The last C discriminative layers have trainable dictionaries $W(j)^{(k)}$ while the first $K-C$ non-discriminative layers have fixed dictionaries $W(j)^{(k)} = \hat{W}(j)$.

Unlike the previous two architectures, no Hankel-augmented matrices are employed here, as temporal correlation is naturally captured by the convolution operation.

4.4.2 Backpropagation

Using the notation introduced in Section 2.4.1, we now present the maps required to perform backpropagation in the proposed Deep-NMFD architecture. Further details on their derivation are provided in Appendix 4.B. As in the Deep-NMF and PAD-NMF cases, the trainable dictionaries $W^{(k)}(j)$ are updated using standard split-gradient relations:

$$W(j)^{(k)} \Leftarrow W(j)^{(k)} \circ \frac{\left[\nabla_{W(j)^{(k)}} \hat{\mathcal{F}}\right]_-}{\left[\nabla_{W(j)^{(k)}} \hat{\mathcal{F}}\right]_+} \qquad \forall k = K-C, \ldots, K-1, \quad \forall j \in \bar{J} \tag{4.4.5}$$

In particular, (4.4.5) involves $[\nabla_W(j)^{(k)} \hat{\mathcal{F}}]_\pm$, which in turn depend on the parameter gradient maps $\Phi_{W(j)}$ through the terms $[\Phi_{W(j)}(\nabla_H, H, W, X)]_\pm$. These terms are defined as follows:

$$
\begin{aligned}
\left[\Phi_{W(j)}(\nabla_H, H, W, X)\right]_{\pm} = \left(\tfrac{X}{W*H}\right) & \left[\begin{array}{c} \vdots \\ \left(\left[\nabla_{H(j)}\right]_{\pm} \circ \frac{H(j)}{\sum_{t=0}^{r_j-1} W(j)_t^\top \overleftarrow{\mathbb{1}}_{m\times n}^{\,t}}\right)\overrightarrow{}^{\,r} \\ \vdots \end{array}\right]_{r=0,\ldots,r_j-1}^{\top} \\
+ \left[\left(\tfrac{X}{(W*H)^{\circ 2}}\right) \circ \left[\sum_{i\in\bar{J}}\sum_{s=0}^{r_i-1} W(i)_s \left(\left[\nabla_{H(i)}\right]_{\mp} \circ \frac{H(i)}{\sum_{t=0}^{r_i-1} W(i)_t^\top \overleftarrow{\mathbb{1}}_{m\times n}^{\,t}}\right)\overrightarrow{}^{\,s}\right]\right] & \left[\begin{array}{c} \vdots \\ \overrightarrow{H(j)}^{\,r} \\ \vdots \end{array}\right]_{r=0,\ldots,r_j-1}^{\top} \\
+ \left[\begin{array}{c|c} \cdots \quad \mathbb{1}_{m\times 1} & \left(\tfrac{X}{W*H}\right) \circ \left[\sum_{s=0}^{r_j-1} W(j)_s \left(\left[\nabla_{H(j)}\right]_{\mp} \circ \frac{H(j)}{\left(\sum_{t=0}^{r_j-1} W(j)_t^\top \overleftarrow{\mathbb{1}}_{m\times n}^{\,t}\right)^{\circ 2}} \circ \overleftarrow{\mathbb{1}}_{1\times n}^{\,r}\right)\overrightarrow{}^{\,s}\right] \end{array}\right\|_1 \cdots \Bigg]_{r=0,\ldots,r_j-1}
\end{aligned}
\tag{4.4.6}
$$

Similarly, (4.4.6) applied to (2.4.6) involves $[\nabla_{H(j)}]_{\pm} = [\nabla_{H(j)^{(k)}}\hat{\mathcal{F}}]_{\pm}$, which in turn depend on the state gradient maps $\Phi_{H(j)}$ and the final state gradient maps Φ_K^j through the terms $[\Phi_{H(j)}(\nabla_H, H, W, X)]_{\pm}$ and $[\Phi_K^j(H, X)]_{\pm}$. The latter are in the form:

$$
\begin{aligned}
\left[\Phi_{H(j)}(\nabla_H, H, W, X)\right]_{\pm} = & \frac{\left[\nabla_{H(j)}\right]_{\pm}}{\sum_{t=0}^{r_j-1} W(j)_t^\top \overleftarrow{\mathbb{1}}_{m\times n}^{\,t}} \circ \left(\sum_{r=0}^{r_j-1} W(j)_r^\top \left(\tfrac{\overleftarrow{X}^{\,r}}{W*H}\right)\right) \\
& + \sum_{r=0}^{r_j-1} W(j)_r^\top \left[\overleftarrow{\left(\tfrac{X}{(W*H)^{\circ 2}}\right) \circ \left[\sum_{i\in\bar{J}}\sum_{s=0}^{r_i-1} W(i)_s \left(\left[\nabla_{H(i)}\right]_{\mp} \circ \frac{H(i)}{\sum_{t=0}^{r_i-1} W(i)_t^\top \overleftarrow{\mathbb{1}}_{m\times n}^{\,t}}\right)\overrightarrow{}^{\,s}\right]}^{\,r}\right]
\end{aligned}
\tag{4.4.7}
$$

$$
\begin{aligned}
\left[\Phi_K^j(H,X)\right]_+ &= \sum_{r=0}^{r_j-1} \hat{W}(j)_r^\top \left[\overleftarrow{\frac{\left(\hat{W}_S*H_S\right)\circ X^{\circ 2} + S\circ X\circ\left(\hat{W}_S*H_S\right)}{\left(\hat{W}*H\right)^{\circ 2}} + \alpha_S\left(\hat{W}_S * H_S\right)}^{\,r}\right] \quad \forall j\in J_S \\
\left[\Phi_K^j(H,X)\right]_- &= \sum_{r=0}^{r_j-1} \hat{W}(j)_r^\top \left[\overleftarrow{\frac{S\circ X}{\hat{W}*H} + \frac{\left(\hat{W}_S*H_S\right)^{\circ 2}\circ X^{\circ 2}}{\left(\hat{W}*H\right)^{\circ 3}} + \alpha_S S}^{\,r}\right] \quad \forall j\in J_S \\
\left[\Phi_K^j(H,X)\right]_+ &= \sum_{r=0}^{r_j-1} \hat{W}(j)_r^\top \left[\overleftarrow{\frac{S\circ X\circ\left(\hat{W}_S*H_S\right)}{\left(\hat{W}*H\right)^{\circ 2}}}^{\,r}\right] \quad \forall j\in J_N \\
\left[\Phi_K^j(H,X)\right]_- &= \sum_{r=0}^{r_j-1} \hat{W}(j)_r^\top \left[\frac{\left(\hat{W}_S*H_S\right)^{\circ 2}\circ X^{\circ 2}}{\left(\hat{W}*H\right)^{\circ 3}}\right] \quad \forall j\in J_N
\end{aligned}
\tag{4.4.8}
$$

4.5 Computational complexity comparison

In this section, we compare the computational and memory costs of the three architectures introduced in this chapter: Deep-NMF, PAD-NMF, and Deep-NMFD. This analysis is crucial for the development of real-time applications based on these models, particularly when targeting resource-constrained microcontrollers, where a careful trade-off between computational load and memory usage is required. We report the computational cost $\mathcal{C}$, measured as the number of basic floating-point operations for a single forward pass, and the memory cost $\mathcal{M}$, expressed as the number of floating-point scalars, for each architecture.

$$\mathcal{C}_{\text{Deep-NMF}} = rn\left[\,(K-C)\,(4mT+1) + C\,(4m+1)\,\right] \tag{4.5.1}$$

$$\mathcal{M}_{\text{Deep-NMF}} = mTr + Cmr + \{(C+1)\,r\} \tag{4.5.2}$$

$$\mathcal{C}_{\text{PAD-NMF}} = rnK\,(4mT+1) \tag{4.5.3}$$

$$\mathcal{M}_{\text{PAD-NMF}} = (C+1)\,mTr + \{(C+1)\,r\} \tag{4.5.4}$$

$$\mathcal{C}_{\text{Deep-NMFD}} = K\left[\bar{J}n\left(m+\bar{J}-1\right) + \left(\bar{J}+1\right)m\left((2n+1)\sum_{j\in\bar{J}} r_j - \sum_{j\in\bar{J}} r_j^2 - n\bar{J}\right)\right] \tag{4.5.5}$$

$$\mathcal{M}_{\text{Deep-NMFD}} = (C+1)\,m\sum_{j\in\bar{J}} r_j + \left\{(C+1)\,\bar{J}\right\} \tag{4.5.6}$$

We note that the reported computational costs assume that certain quantities (specifically, the denominators appearing in the update maps f^{in}) are precomputed and stored in memory to improve efficiency. The corresponding increase in memory usage, relative to that required to store the network parameters, is indicated in curly brackets for each algorithm; in practice, this overhead is relatively small.
We now focus on a direct comparison between PAD-NMF and Deep-NMFD. For fairness, Deep-NMF is excluded from this analysis, as it lacks the ability to embed strong temporal correlations within the nonnegative factors. Table 4 reports the costs associated with the following three representative scenarios:

1. *Musical notes detection*: $|J_S|(=I) = 88$, $|J_N|(=J) = 12$, $m = 2|J_S|$, $T = 3$, $n = 2T$, $K = 10$, $C = 5$, $r_j \equiv T$, $r = \sum_{j\in\bar{J}} r_j$;

2. *Natural phenomena monitoring*: $|J_S| = 3$, $|J_N| = 12$, $m = 3|J_S|$, $T = 20$, $n = 2T$, $K = 10$, $C = 5$, $r_j \equiv T$, $r = \sum_{j\in\bar{J}} r_j$;

3. *Machine monitoring*: $|J_S| = 6$, $|J_N| = 6$, $m = 3|J_S|$, $T = 6$, $n = 2T$, $K = 40$, $C = 10$, $r_j \equiv T$, $r = \sum_{j\in\bar{J}} r_j$;

Table 4: Comparison of computational cost (MFLOPS) and memory usage (MB) for PAD-NMF and Deep-NMFD across the three scenarios described above.

	$\mathcal{C}_{\text{PAD-NMF}}$	$\mathcal{C}_{\text{Deep-NMFD}}$	$\mathcal{M}_{\text{PAD-NMF}}$	$\mathcal{M}_{\text{Deep-NMFD}}$
Scenario 1	25.94	437.99	3.71	1.24
Scenario 2	45.48	25.63	1.24	0.06
Scenario 3	11.85	11.62	0.09	0.01

In terms of memory, Deep-NMFD is roughly T times more efficient. Regarding computational cost, both Deep-NMFD and PAD-NMF can be competitive depending on the parameter configuration. Specifically, for a small number of sources $\bar{J}$ (Scenarios 2 and 3), Deep-NMFD performs better, but its advantage diminishes as $\bar{J}$ increases (Scenario 1). This trade-off can be inferred from Equations (4.5.3) and (4.5.5): when $\bar{J} \lesssim 2T$, Deep-NMFD generally outperforms PAD-NMF, especially for larger r. Conversely, for larger $\bar{J}$ ($\bar{J} \gtrsim 2T$), PAD-NMF is more efficient for smaller r, though its performance degrades as r grows. Consequently, PAD-NMF is preferable for scenarios with short temporal correlations, while Deep-NMFD is better suited for longer correlations. Additionally, unlike Deep-NMFD, where each source can have its own correlation length r_j, PAD-NMF enforces a uniform correlation length T across all sources. The values reported in Table 4 indicate that both PAD-NMF and Deep-NMFD are suitable for implementation on low-cost embedded microprocessors, such as ARM4 or ARM7, achieving roughly 20 MFLOPS.

Appendix

4.A PAD-NMF backpropagation details

The goal of this section is to derive the backpropagation maps for the PAD-NMF architecture presented in Section 4.3.2.
Denote $W^{(k)} = \{w^{(k)}_{m,r}\}_{m,r}$, $H^{(k)} = \{h^{(k)}_{r,n}\}_{r,n}$, $M^H = \{m^H_{r,n}\}_{r,n}$, and $X = \{x_{m,n}\}_{m,n}$. The update rule for $H^{(k)}$ (4.3.15) written elementwise then becomes:

$$h^{(k+1)}_{r,n} = m^H_{r,n} \frac{h^{(k)}_{r,n}}{\sum\limits_m w^{(k)}_{m,r} + \alpha_H} \left[\sum_m w^{(k)}_{m,r} \frac{x_{m,n}}{\sum\limits_s w^{(k)}_{m,s} h^{(k)}_{s,n}} \right] \tag{4.A.1}$$

Deriving (4.A.1) with respect to $w^{(k)}_{\overline{m},\bar{r}}$, we obtain:

$$\begin{aligned}
\frac{\partial h^{(k+1)}_{r,n}}{\partial w^{(k)}_{\overline{m},\bar{r}}} &= m^H_{\bar{r},n} \frac{h^{(k)}_{\bar{r},n}}{\sum\limits_m w^{(k)}_{m,\bar{r}} + \alpha_H} \left[\frac{x_{\overline{m},n}}{\sum\limits_s w^{(k)}_{\overline{m},s} h^{(k)}_{s,n}} \right] \delta_{r,\bar{r}} \\
&- m^H_{r,n} \frac{h^{(k)}_{r,n}}{\sum\limits_m w^{(k)}_{m,r} + \alpha_H} \left[w^{(k)}_{\overline{m},r} \frac{x_{\overline{m},n} h^{(k)}_{\bar{r},n}}{\left(\sum\limits_s w^{(k)}_{\overline{m},s} h^{(k)}_{s,n} \right)^2} \right] \\
&- m^H_{\bar{r},n} \frac{h^{(k)}_{\bar{r},n}}{\left(\sum\limits_m w^{(k)}_{m,\bar{r}} + \alpha_H \right)^2} \left[\sum_m w^{(k)}_{m,\bar{r}} \frac{x_{m,n}}{\sum\limits_s w^{(k)}_{m,s} h^{(k)}_{s,n}} \right] \delta_{r,\bar{r}} \\
&= \left(\frac{X}{W^{(k)} H^{(k)}} \right)_{\overline{m},n} \left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \right)_{\bar{r},n} \left(M^H \right)_{\bar{r},n} \delta_{r,\bar{r}} \\
&- \left(\frac{X}{\left(W^{(k)} H^{(k)} \right)^{\circ 2}} \right)_{\overline{m},n} \left(W^{(k)} \right)_{\overline{m},r} \left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \right)_{r,n} \left(M^H \right)_{r,n} \left(H^{(k)} \right)_{\bar{r},n} \\
&- \left(\frac{H^{(k)}}{\left({W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H \right)^{\circ 2}} \right)_{\bar{r},n} \left({W^{(k)}}^\top \frac{X}{W^{(k)} H^{(k)}} \right)_{\bar{r},n} \left(M^H \right)_{\bar{r},n} \delta_{r,\bar{r}}
\end{aligned} \tag{4.A.2}$$

Similarly, deriving (4.A.1) with respect to $h^{(k)}_{\bar{r},\bar{n}}$, we obtain:

$$
\begin{aligned}
\frac{\partial h_{r,n}^{(k+1)}}{\partial h_{\bar{r},\bar{n}}^{(k)}} &= m_{\bar{r},\bar{n}}^{H} \frac{1}{\sum_m w_{m,\bar{r}}^{(k)} + \alpha_H} \left[\sum_m w_{m,\bar{r}}^{(k)} \frac{x_{m,\bar{n}}}{\sum_s w_{m,s}^{(k)} h_{s,\bar{n}}^{(k)}} \right] \delta_{\bar{r},r} \delta_{\bar{n},n} \\
&- m_{r,\bar{n}}^{H} \frac{h_{r,\bar{n}}^{(k)}}{\sum_m w_{m,r}^{(k)} + \alpha_H} \left[\sum_m w_{m,r}^{(k)} \frac{x_{m,\bar{n}} w_{m,\bar{r}}^{(k)}}{\left(\sum_s w_{m,s}^{(k)} h_{s,\bar{n}}^{(k)} \right)^2} \right] \delta_{\bar{n},n} \\
&= \left(\frac{{W^{(k)}}^\top \frac{X}{W^{(k)} H^{(k)}}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \right)_{\bar{r},\bar{n}} \left(M^H \right)_{\bar{r},\bar{n}} \delta_{\bar{r},r} \delta_{\bar{n},n} \\
&- \sum_m \left(W^{(k)} \right)_{m,\bar{r}} \left(\frac{X}{\left(W^{(k)} H^{(k)} \right)^{\circ 2}} \right)_{m,\bar{n}} \left(W^{(k)} \right)_{m,r} \left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \right)_{r,\bar{n}} \left(M^H \right)_{r,\bar{n}} \delta_{\bar{n},n}
\end{aligned}
\tag{4.A.3}
$$

Therefore, since:

$$
\frac{\partial \hat{\mathcal{F}}}{\partial w_{\overline{m},\bar{r}}^{(k)}} = \sum_{r,n} \frac{\partial \hat{\mathcal{F}}}{\partial h_{r,n}^{(k+1)}} \frac{\partial h_{r,n}^{(k+1)}}{\partial w_{\overline{m},\bar{r}}^{(k)}}
$$

from (4.A.2) we get the following split gradient relations:

$$
\begin{aligned}
&\left[\frac{\partial \hat{\mathcal{F}}}{\partial w_{\overline{m},\bar{r}}^{(k)}} \right]_{\pm} = \sum_{r,n} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h_{r,n}^{(k+1)}} \right]_{\pm} \left[\frac{\partial h_{r,n}^{(k+1)}}{\partial w_{\overline{m},\bar{r}}^{(k)}} \right]_{+} + \left[\frac{\partial \hat{\mathcal{F}}}{\partial h_{r,n}^{(k+1)}} \right]_{\mp} \left[\frac{\partial h_{r,n}^{(k+1)}}{\partial w_{\overline{m},\bar{r}}^{(k)}} \right]_{-} \\
&= \sum_{r,n} \left[\left(\frac{X}{W^{(k)} H^{(k)}} \right)_{\overline{m},n} \left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \right)_{\bar{r},n} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h_{r,n}^{(k+1)}} \right]_{\pm} \left(M^H \right)_{\bar{r},n} \delta_{r,\bar{r}} \right] \\
&+ \sum_{r,n} \left[\left(\frac{X}{\left(W^{(k)} H^{(k)} \right)^{\circ 2}} \right)_{\overline{m},n} \left(W^{(k)} \right)_{\overline{m},r} \left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \right)_{r,n} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h_{r,n}^{(k+1)}} \right]_{\mp} \left(M^H \right)_{r,n} \left(H^{(k)} \right)_{\bar{r},n} \right] \\
&+ \sum_{r,n} \left[\left(\frac{H^{(k)}}{\left({W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H \right)^{\circ 2}} \right)_{\bar{r},n} \left({W^{(k)}}^\top \frac{X}{W^{(k)} H^{(k)}} \right)_{\bar{r},n} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h_{r,n}^{(k+1)}} \right]_{\mp} \left(M^H \right)_{\bar{r},n} \delta_{r,\bar{r}} \right] \\
&= \sum_{n} \left(\frac{X}{W^{(k)} H^{(k)}} \right)_{\overline{m},n} \left[\left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \right)_{\bar{r},n} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h_{\bar{r},n}^{(k+1)}} \right]_{\pm} \left(M^H \right)_{\bar{r},n} \right] \\
&+ \sum_{n} \left[\left(\frac{X}{\left(W^{(k)} H^{(k)} \right)^{\circ 2}} \right)_{\overline{m},n} \sum_r \left(W^{(k)} \right)_{\overline{m},r} \left[\left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \right)_{r,n} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h_{r,n}^{(k+1)}} \right]_{\mp} \left(M^H \right)_{r,n} \right] \right] \left(H^{(k)} \right)_{\bar{r},n} \\
&+ \sum_{n} \left[\left(\frac{H^{(k)}}{\left({W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H \right)^{\circ 2}} \right)_{\bar{r},n} \left({W^{(k)}}^\top \frac{X}{W^{(k)} H^{(k)}} \right)_{\bar{r},n} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h_{\bar{r},n}^{(k+1)}} \right]_{\mp} \left(M^H \right)_{\bar{r},n} \right] \\
&= \left[\frac{X}{W^{(k)} H^{(k)}} \left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \circ \left[\nabla_{H^{(k+1)}} \hat{\mathcal{F}} \right]_{\pm} \circ M^H \right)^\top \right]_{\overline{m},\bar{r}} \\
&+ \left[\left[\frac{X}{\left(W^{(k)} H^{(k)} \right)^{\circ 2}} \circ \left[W^{(k)} \left(\frac{H^{(k)}}{{W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H} \circ \left[\nabla_{H^{(k+1)}} \hat{\mathcal{F}} \right]_{\mp} \circ M^H \right) \right] \right] {H^{(k)}}^\top \right]_{\overline{m},\bar{r}} \\
&+ \left[\mathbb{1}_{m \times n} \left[\frac{H^{(k)}}{\left({W^{(k)}}^\top \mathbb{1}_{m \times n} + \alpha_H \right)^{\circ 2}} \circ \left({W^{(k)}}^\top \frac{X}{W^{(k)} H^{(k)}} \right) \circ \left[\nabla_{H^{(k+1)}} \hat{\mathcal{F}} \right]_{\mp} \circ M^H \right] \right]_{\overline{m},\bar{r}} \\
&= \left[\Phi_W(\nabla_{H^{(k+1)}} \hat{\mathcal{F}}, H^{(k)}, W^{(k)}, X)_{\overline{m},\bar{r}} \right]_{\pm}
\end{aligned}
\tag{4.A.4}
$$

Similarly, since:

$$\frac{\partial \hat{\mathcal{F}}}{\partial h^{(k)}_{\bar{r},\bar{n}}} = \sum_{r,n} \frac{\partial \hat{\mathcal{F}}}{\partial h^{(k+1)}_{r,n}} \frac{\partial h^{(k+1)}_{r,n}}{\partial h^{(k)}_{\bar{r},\bar{n}}}$$

from (4.A.3), we get the following split gradient relations:

$$\begin{aligned}
&\left[\frac{\partial \hat{\mathcal{F}}}{\partial h^{(k)}_{\bar{r},\bar{n}}}\right]_{\pm} = \sum_{r,n} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h^{(k+1)}_{r,n}}\right]_{\pm} \left[\frac{\partial h^{(k+1)}_{r,n}}{\partial h^{(k)}_{\bar{r},\bar{n}}}\right]_{+} + \left[\frac{\partial \hat{\mathcal{F}}}{\partial h^{(k+1)}_{r,n}}\right]_{\mp} \left[\frac{\partial h^{(k+1)}_{r,n}}{\partial h^{(k)}_{\bar{r},\bar{n}}}\right]_{-} \\
&= \sum_{r,n} \left[\left(\frac{{W^{(k)}}^{\top} \frac{X}{W^{(k)}H^{(k)}}}{{W^{(k)}}^{\top} \mathbb{1}_{m\times n} + \alpha_H} \right)_{\bar{r},\bar{n}} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h^{(k+1)}_{r,n}}\right]_{\pm} \left(M^H\right)_{\bar{r},\bar{n}} \delta_{\bar{r},r} \delta_{\bar{n},n} \right] \\
&+ \sum_{r,n} \left[\sum_m \left(W^{(k)}\right)_{m,\bar{r}} \left(\frac{X}{\left(W^{(k)}H^{(k)}\right)^{\circ 2}} \right)_{m,\bar{n}} \left(W^{(k)}\right)_{m,r} \left(\frac{H^{(k)}}{{W^{(k)}}^{\top} \mathbb{1}_{m\times n} + \alpha_H} \right)_{r,\bar{n}} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h^{(k+1)}_{r,n}}\right]_{\mp} \left(M^H\right)_{r,\bar{n}} \delta_{\bar{n},n} \right] \\
&= \left(\frac{{W^{(k)}}^{\top} \frac{X}{W^{(k)}H^{(k)}}}{{W^{(k)}}^{\top} \mathbb{1}_{m\times n} + \alpha_H} \right)_{\bar{r},\bar{n}} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h^{(k+1)}_{\bar{r},\bar{n}}}\right]_{\pm} \left(M^H\right)_{\bar{r},\bar{n}} \\
&+ \sum_m \left(W^{(k)}\right)_{m,\bar{r}} \left[\left(\frac{X}{\left(W^{(k)}H^{(k)}\right)^{\circ 2}} \right)_{m,\bar{n}} \sum_r \left(W^{(k)}\right)_{m,r} \left[\left(\frac{H^{(k)}}{{W^{(k)}}^{\top} \mathbb{1}_{m\times n} + \alpha_H} \right)_{r,\bar{n}} \left[\frac{\partial \hat{\mathcal{F}}}{\partial h^{(k+1)}_{r,\bar{n}}}\right]_{\mp} \left(M^H\right)_{r,\bar{n}} \right] \right] \\
&= \left[\frac{{W^{(k)}}^{\top} \frac{X}{W^{(k)}H^{(k)}}}{{W^{(k)}}^{\top} \mathbb{1}_{m\times n} + \alpha_H} \circ \left[\nabla_{H^{(k+1)}} \hat{\mathcal{F}}\right]_{\pm} \circ M^H \right]_{\bar{r},\bar{n}} \\
&+ \left[{W^{(k)}}^{\top} \left[\frac{X}{\left(W^{(k)}H^{(k)}\right)^{\circ 2}} \circ \left[W^{(k)} \left(\frac{H^{(k)}}{{W^{(k)}}^{\top} \mathbb{1}_{m\times n} + \alpha_H} \circ \left[\nabla_{H^{(k+1)}} \hat{\mathcal{F}}\right]_{\mp} \circ M^H \right) \right] \right] \right]_{\bar{r},\bar{n}} \\
&= \left[\Phi_H(\nabla_{H^{(k+1)}} \hat{\mathcal{F}}, H^{(k)}, W^{(k)}, X)_{\bar{r},\bar{n}} \right]_{\pm}
\end{aligned} \tag{4.A.5}$$

Denote $\hat{W}_S = \{(\hat{w}_S)_{m,r}\}_{m,r}$, $\hat{W}_N = \{(\hat{w}_N)_{m,r}\}_{m,r}$, $H^{(K)}_S = \{(h^{(K)}_S)_{r,n}\}_{r,n}$, $H^{(K)}_N = \{(h^{(K)}_N)_{r,n}\}_{r,n}$ and $S = \{s_{m,n}\}_{m,n}$. The loss-function (4.3.20) written elementwise (for a single training instance) is of the form:

$$\begin{aligned}
\hat{\mathcal{F}} &= \sum_{m,n} \frac{1}{2} \left(\frac{\sum_r (\hat{w}_S)_{m,r} (h^{(K)}_S)_{r,n}}{\sum_r (\hat{w}_S)_{m,r} (h^{(K)}_S)_{r,n} + \sum_r (\hat{w}_N)_{m,r} (h^{(K)}_N)_{r,n}} x_{m,n} - s_{m,n} \right)^2 \\
&+ \sum_{m,n} \frac{\alpha_S}{2} \left(\sum_r (\hat{w}_S)_{m,r} (h^{(K)}_S)_{r,n} - s_{m,n} \right)^2
\end{aligned} \tag{4.A.6}$$

Deriving (4.A.6) with respect to $(h^{(K)}_S)_{\bar{r},\bar{n}}$, we obtain:

$$
\begin{aligned}
&\frac{\partial \hat{\mathcal{F}}}{\partial (h_S^{(K)})_{\bar{r},\bar{n}}} = \sum_m \left(\frac{\hat{W}_S H_S^{(K)}}{\hat{W} H^{(K)}} \circ X - S \right)_{m,\bar{n}} \left(\frac{X}{\hat{W} H^{(K)}} \right)_{m,\bar{n}} (\hat{w}_S)_{m,\bar{r}} \\
&+ \sum_m \alpha_S \left(\hat{W}_S H_S^{(K)} - S \right)_{m,\bar{n}} (\hat{w}_S)_{m,\bar{r}} \\
&- \sum_m \left(\frac{\hat{W}_S H_S^{(K)}}{\hat{W} H^{(K)}} \circ X - S \right)_{m,\bar{n}} \left(\frac{\hat{W}_S H_S^{(K)}}{\left(\hat{W} H^{(K)}\right)^{\circ 2}} \circ X \right)_{m,\bar{n}} (\hat{w}_S)_{m,\bar{r}} \\
&= \sum_m (\hat{w}_S)_{m,\bar{r}} \left(\frac{X^{\circ 2} \circ \hat{W}_S H_S^{(K)} \circ \hat{W}_N H_N^{(K)}}{\left(\hat{W} H^{(K)}\right)^{\circ 3}} + \alpha_S \hat{W}_S H_S^{(K)} \right)_{m,\bar{n}} \\
&- \sum_m (\hat{w}_S)_{m,\bar{r}} \left(\frac{X \circ S \circ \hat{W}_N H_N^{(K)}}{\left(\hat{W} H^{(K)}\right)^{\circ 2}} + \alpha_S S \right)_{m,\bar{n}} \\
&= \left[\hat{W}_S^\top \left(\frac{X^{\circ 2} \circ \hat{W}_S H_S^{(K)} \circ \hat{W}_N H_N^{(K)}}{\left(\hat{W} H^{(K)}\right)^{\circ 3}} + \alpha_S \hat{W}_S H_S^{(K)} \right) \right]_{\bar{r},\bar{n}} - \left[\hat{W}_S^\top \left(\frac{X \circ S \circ \hat{W}_N H_N^{(K)}}{\left(\hat{W} H^{(K)}\right)^{\circ 2}} + \alpha_S S \right) \right]_{\bar{r},\bar{n}} \\
&= \left[\frac{\partial \hat{\mathcal{F}}}{\partial (h_S^{(K)})_{\bar{r},\bar{n}}} \right]_+ - \left[\frac{\partial \hat{\mathcal{F}}}{\partial (h_S^{(K)})_{\bar{r},\bar{n}}} \right]_- = \left[\Phi_K(H^{(K)}, X)_S \right]_+ - \left[\Phi_K(H^{(K)}, X)_S \right]_-
\end{aligned}
\tag{4.A.7}
$$

Similarly, deriving (4.A.6) with respect to $(h_N^{(K)})_{\bar{r},\bar{n}}$, we obtain:

$$
\begin{aligned}
&\frac{\partial \hat{\mathcal{F}}}{\partial (h_N^{(K)})_{\bar{r},\bar{n}}} = \sum_m \left(\frac{\hat{W}_S H_S^{(K)}}{\hat{W} H^{(K)}} \circ X - S \right)_{m,\bar{n}} \left(\frac{\hat{W}_S H_S^{(k)}}{\left(\hat{W} H^{(K)}\right)^{\circ 2}} \circ X \right)_{m,\bar{n}} (\hat{w}_N)_{m,\bar{r}} \\
&= \sum_m (\hat{w}_N)_{m,\bar{r}} \left(\frac{X \circ S \circ \hat{W}_N H_N^{(K)}}{\left(\hat{W} H^{(K)}\right)^{\circ 2}} \right)_{m,\bar{n}} - \sum_m (\hat{w}_N)_{m,\bar{r}} \left(\frac{X^{\circ 2} \circ \left(\hat{W}_S H_S^{(K)}\right)^{\circ 2}}{\left(\hat{W} H^{(K)}\right)^{\circ 3}} \right)_{m,\bar{n}} \\
&= \left[\hat{W}_N^\top \left(\frac{X \circ S \circ \hat{W}_N H_N^{(K)}}{\left(\hat{W} H^{(K)}\right)^{\circ 2}} \right) \right]_{\bar{r},\bar{n}} - \left[\hat{W}_N^\top \left(\frac{X^{\circ 2} \circ \left(\hat{W}_S H_S^{(K)}\right)^{\circ 2}}{\left(\hat{W} H^{(K)}\right)^{\circ 3}} \right) \right]_{\bar{r},\bar{n}} \\
&= \left[\frac{\partial \hat{\mathcal{F}}}{\partial (h_N^{(K)})_{\bar{r},\bar{n}}} \right]_+ - \left[\frac{\partial \hat{\mathcal{F}}}{\partial (h_N^{(K)})_{\bar{r},\bar{n}}} \right]_- = \left[\Phi_K(H^{(K)}, X)_N \right]_+ - \left[\Phi_K(H^{(K)}, X)_N \right]_-
\end{aligned}
\tag{4.A.8}
$$

4.B DNMFD backpropagation details

The goal of this section is to derive the backpropagation maps for the Deep-NMFD architecture presented in Section 4.4.2.
Denote $W(i)^{(k)} = \{w(i)_{m,r}^{(k)}\}_{m,r}$, $H(i)^{(k)} = \{h(i)_n^{(k)}\}_m$, and $X = \{x_{m,n}\}_{m,n}$. The update rule for $H(i)^{(k)}$ (4.4.3) written elementwise then becomes:

$$
h(i)_n^{(k+1)} = \frac{h(i)_n^{(k)}}{\sum_{t,m} w(i)_{m,t}^{(k)} \sum_q \left(J^t\right)_{n,q}} \left[\sum_{s,m} w(i)_{m,s}^{(k)} \sum_p \frac{x_{m,p}}{\sum_{l,u} w(l)_{m,u}^{(k)} \sum_q h(l)_q^{(k)} \left(J^u\right)_{q,p}} \left(J^s\right)_{n,p} \right]
\tag{4.B.1}
$$

Deriving (4.B.1) with respect to $w(j)_{\bar{m},\bar{r}}^{(k)}$, we obtain:

$$
\begin{aligned}
\frac{\partial h(i)_n^{(k+1)}}{\partial w(j)_{\overline{m},\bar{r}}^{(k)}} &= \frac{h(j)_n^{(k)}}{\sum\limits_{t,m} w(j)_{m,t}^{(k)} \sum\limits_q (J^t)_{n,q}} \left[\sum_p \frac{x_{\overline{m},p}}{\sum\limits_{l,u} w(l)_{\overline{m},u}^{(k)} \sum\limits_q h(l)_q^{(k)} (J^u)_{q,p}} (J^{\bar{r}})_{n,p} \right] \delta_{i,j} \\
&- \frac{h(i)_n^{(k)}}{\sum\limits_t w(i)_{m,t}^{(k)} \sum\limits_q (J^t)_{n,q}} \left[\sum_s w(i)_{\overline{m},s}^{(k)} \sum_p \frac{x_{\overline{m},p} \sum\limits_q h(j)_q^{(k)} (J^{\bar{r}})_{q,p}}{\left(\sum\limits_{l,u} w(l)_{\overline{m},u}^{(k)} \sum\limits_q h(l)_q^{(k)} (J^u)_{q,p} \right)^2} (J^s)_{n,p} \right] \\
&- \frac{h(j)_n^{(k)} \sum\limits_q \left(J^{\bar{r}}\right)_{n,q}}{\left(\sum\limits_{t,m} w(j)_{m,t}^{(k)} \sum\limits_q (J^t)_{n,q} \right)^2} \left[\sum_{s,m} w(j)_{m,s}^{(k)} \sum_p \frac{x_{m,p}}{\sum\limits_{l,u} w(l)_{m,u}^{(k)} \sum\limits_q h(l)_q^{(k)} (J^u)_{q,p}} (J^s)_{n,p} \right] \delta_{i,j} \\
&= \sum_p \left(\frac{X}{W^{(k)} * H^{(k)}} \right)_{\overline{m},p} \left(\frac{H(j)^{(k)}}{\sum\limits_t W(j)_t^{(k)^\top} \overleftarrow{\mathbb{1}}_{m \times n}^{t}} \right)_n (J^{\bar{r}})_{n,p} \delta_{i,j} \\
&- \sum_p \left(\frac{X}{\left(W^{(k)} * H^{(k)}\right)^{\circ 2}} \right)_{\overline{m},p} \left[\sum_s w(i)_{\overline{m},s}^{(k)} \left(\frac{H(i)^{(k)}}{\sum\limits_t W(i)_t^{(k)^\top} \overleftarrow{\mathbb{1}}_{m \times n}^{t}} \right)_n (J^s)_{n,p} \right] \left(\overrightarrow{H(j)^{(k)}}^{\bar{r}} \right)_p \\
&- \sum_{m,p} \left(\frac{X}{W^{(k)} * H^{(k)}} \right)_{m,p} \left[\sum_s w(j)_{m,s}^{(k)} \left(\frac{H(j)^{(k)}}{\left(\sum\limits_t W(j)_t^{(k)^\top} \overleftarrow{\mathbb{1}}_{m \times n}^{t} \right)^{\circ 2}} \right)_n \left(\overleftarrow{\mathbb{1}_{1 \times n}}^{\bar{r}} \right)_n (J^s)_{n,p} \right] \delta_{i,j}
\end{aligned}
\tag{4.B.2}
$$

Similarly, deriving (4.B.1) with respect to $h(j)_{\overline{n}}^{(k)}$, we obtain:

$$
\begin{aligned}
\frac{\partial h(i)_n^{(k+1)}}{\partial h(j)_{\overline{n}}^{(k)}} &= \frac{1}{\sum\limits_{t,m} w(j)_{m,t}^{(k)} \sum\limits_q (J^t)_{\overline{n},q}} \left[\sum_{s,m} w(j)_{m,s}^{(k)} \sum_p \frac{x_{m,p}}{\sum\limits_{l,u} w(l)_{m,u}^{(k)} \sum\limits_q h(l)_q^{(k)} (J^u)_{q,p}} (J^s)_{\overline{n},p} \right] \delta_{i,j} \delta_{n,\overline{n}} \\
&- \frac{h(i)_n^{(k)}}{\sum\limits_{t,m} w(i)_{m,t}^{(k)} \sum\limits_q (J^t)_{n,q}} \left[\sum_{s,m} w(i)_{m,s}^{(k)} \sum_p \frac{x_{m,p} \sum\limits_u w(j)_{m,u}^{(k)} (J^u)_{\overline{n},p}}{\left(\sum\limits_{l,u} w(l)_{m,u}^{(k)} \sum\limits_q h(l)_q^{(k)} (J^u)_{q,p} \right)^2} (J^s)_{n,p} \right] \\
&= \left(\frac{\mathbb{1}_{1 \times n}}{\sum\limits_t W(j)_t^{(k)^\top} \overleftarrow{\mathbb{1}}_{m \times n}^{t}} \right)_{\overline{n}} \left(\sum_s W(j)_s^{(k)^\top} \left(\frac{\overleftarrow{X}^{s}}{W^{(k)} * H^{(k)}} \right) \right)_{\overline{n}} \delta_{i,j} \delta_{n,\overline{n}} \\
&- \sum_{u,m} w(j)_{m,u}^{(k)} \left[\sum_p \left(\frac{X}{\left(W^{(k)} * H^{(k)}\right)^{\circ 2}} \right)_{m,p} \left[\sum_s w(i)_{m,s}^{(k)} \left(\frac{H(i)^{(k)}}{\sum\limits_t W(i)_t^{(k)^\top} \overleftarrow{\mathbb{1}}_{m \times n}^{t}} \right)_n (J^s)_{n,p} \right] (J^u)_{\overline{n},p} \right]
\end{aligned}
\tag{4.B.3}
$$

Therefore, since:

$$
\frac{\partial \hat{\mathcal{F}}}{\partial w(j)_{\overline{m},\bar{r}}^{(k)}} = \sum_{i,n} \frac{\partial \hat{\mathcal{F}}}{\partial h(i)_n^{(k+1)}} \frac{\partial h(i)_n^{(k+1)}}{\partial w(j)_{\overline{m},\bar{r}}^{(k)}}
$$

from (4.B.2), we get the following split gradient relations:

$$
\begin{aligned}
\left[\frac{\partial\hat{\mathcal{F}}}{\partial w(j)^{(k)}_{\overline{m},\bar{r}}}\right]_{\pm} &= \sum_{i,n}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\pm}\left[\frac{\partial h(i)^{(k+1)}_{n}}{\partial w(j)^{(k)}_{\overline{m},\bar{r}}}\right]_{+} + \left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\mp}\left[\frac{\partial h(i)^{(k+1)}_{n}}{\partial w(j)^{(k)}_{\overline{m},\bar{r}}}\right]_{-} \\
&= \sum_{i,n}\left[\sum_{p}\left(\frac{X}{W^{(k)}*H^{(k)}}\right)_{\overline{m},p}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\pm}\left(\frac{H(j)^{(k)}}{\sum_{t} W(j)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}}\right)_{n}(J^{\bar{r}})_{n,p}\delta_{i,j}\right] \\
&+ \sum_{i,n}\left[\sum_{p}\left(\frac{X}{\left(W^{(k)}*H^{(k)}\right)^{\circ 2}}\right)_{\overline{m},p}\left[\sum_{s} w(i)^{(k)}_{\overline{m},s}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\mp}\left(\frac{H(i)^{(k)}}{\sum_{t} W(i)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}}\right)_{n}(J^{s})_{n,p}\right]\left(\overrightarrow{H(j)^{(k)}}^{\bar{r}}\right)_{p}\right] \\
&+ \sum_{i,n}\left[\sum_{m,p}\left(\frac{X}{W^{(k)}*H^{(k)}}\right)_{m,p}\left[\sum_{s} w(j)^{(k)}_{m,s}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\mp}\left(\frac{H(j)^{(k)}}{\left(\sum_{t} W(j)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}\right)^{\circ 2}}\right)_{n}\left(\overleftarrow{\mathbb{1}}^{\bar{r}}_{1\times n}\right)_{n}(J^{s})_{n,p}\right]\delta_{i,j}\right] \\
&= \sum_{p}\left(\frac{X}{W^{(k)}*H^{(k)}}\right)_{\overline{m},p}\left[\sum_{n}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(j)^{(k+1)}_{n}}\right]_{\pm}\left(\frac{H(j)^{(k)}}{\sum_{t} W(j)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}}\right)_{n}(J^{\bar{r}})_{n,p}\right] \\
&+ \sum_{p}\left(\frac{X}{\left(W^{(k)}*H^{(k)}\right)^{\circ 2}}\right)_{\overline{m},p}\left[\sum_{i,s} w(i)^{(k)}_{\overline{m},s}\left[\sum_{n}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\mp}\left(\frac{H(i)^{(k)}}{\sum_{t} W(i)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}}\right)_{n}(J^{s})_{n,p}\right]\right]\left(\overrightarrow{H(j)^{(k)}}^{\bar{r}}\right)_{p} \\
&+ \sum_{m,p}\left(\frac{X}{W^{(k)}*H^{(k)}}\right)_{m,p}\left[\sum_{s} w(j)^{(k)}_{m,s}\left[\sum_{n}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(j)^{(k+1)}_{n}}\right]_{\mp}\left(\frac{H(j)^{(k)}}{\left(\sum_{t} W(j)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}\right)^{\circ 2}}\right)_{n}\left(\overleftarrow{\mathbb{1}}^{\bar{r}}_{1\times n}\right)_{n}(J^{s})_{n,p}\right]\right] \\
&= \left[\left(\frac{X}{W^{(k)}*H^{(k)}}\right)\left(\left[\overrightarrow{\nabla_{H(j)^{(k+1)}}\hat{\mathcal{F}}}^{\bar{r}}\right]_{\pm}\circ\frac{H(j)^{(k)}}{\sum_{t} W(j)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}}\right)^{\top}\right]_{\overline{m}} \\
&+ \left[\left[\left(\frac{X}{\left(W^{(k)}*H^{(k)}\right)^{\circ 2}}\right)\circ\left[\sum_{i,s} W(i)^{(k)}_{s}\left(\left[\overrightarrow{\nabla_{H(i)^{(k+1)}}\hat{\mathcal{F}}}^{s}\right]_{\pm}\circ\frac{H(i)^{(k)}}{\sum_{t} W(i)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}}\right)\right]\right]\left(\overrightarrow{H(j)^{(k)}}^{\bar{r}}\right)^{\top}\right]_{\overline{m}} \\
&+ \left\|\left(\frac{X}{W^{(k)}*H^{(k)}}\right)\circ\left[\sum_{s} W(j)^{(k)}_{s}\left(\left[\overrightarrow{\nabla_{H(j)^{(k+1)}}\hat{\mathcal{F}}}^{s}\right]_{\mp}\circ\frac{H(j)^{(k)}}{\left(\sum_{t} W(j)^{(k)\top}_{t}\overleftarrow{\mathbb{1}}_{m\times n}\right)^{\circ 2}}\circ\overleftarrow{\mathbb{1}}^{\bar{r}}_{1\times n}\right)\right]\right\|_{1} \\
&= \left[\Phi_{W(j)}\left(\nabla_{H^{(k+1)}}\hat{\mathcal{F}}, H^{(k)}, W^{(k)}, X\right)_{\overline{m},\bar{r}}\right]_{\pm}
\end{aligned}
\tag{4.B.4}
$$

Similarly, since:

$$
\frac{\partial\hat{\mathcal{F}}}{\partial h(j)^{(k)}_{\bar{n}}} = \sum_{i,n}\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\frac{\partial h(i)^{(k+1)}_{n}}{\partial h(j)^{(k)}_{\bar{n}}}
$$

from (4.B.3), we get the following split gradient relations:

$$
\begin{aligned}
\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(j)^{(k)}_{\overline{n}}}\right]_{\pm} &= \sum_{i,n}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\pm}\left[\frac{\partial h(i)^{(k+1)}_{n}}{\partial h(j)^{(k)}_{\overline{n}}}\right]_{+} + \left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\mp}\left[\frac{\partial h(i)^{(k+1)}_{n}}{\partial h(j)^{(k)}_{\overline{n}}}\right]_{-} \\
&= \sum_{i,n}\left[\left(\frac{\mathbb{1}_{1\times n}}{\sum_t W(j)^{(k)\top}_t \overleftarrow{\mathbb{1}_{m\times n}}^{\,t}}\right)_{\overline{n}}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\pm}\left(\sum_s W(j)^{(k)\top}_s \left(\frac{\overleftarrow{X}^{\,s}}{W^{(k)}*H^{(k)}}\right)\right)_{\overline{n}}\delta_{i,j}\delta_{n,\overline{n}}\right] \\
&+ \sum_{i,n}\left[\sum_{u,m} w(j)^{(k)}_{m,u}\left[\sum_p\left(\frac{X}{\left(W^{(k)}*H^{(k)}\right)^{\circ 2}}\right)_{m,p}\left[\sum_s w(i)^{(k)}_{m,s}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\mp}\left(\frac{H(i)^{(k)}}{\sum_t W(i)^{(k)\top}_t \overleftarrow{\mathbb{1}_{m\times n}}^{\,t}}\right)_{n}(J^s)_{n,p}\right](J^u)_{\overline{n},p}\right]\right] \\
&= \left(\frac{\mathbb{1}_{1\times n}}{\sum_t W(j)^{(k)\top}_t \overleftarrow{\mathbb{1}_{m\times n}}^{\,t}}\right)_{\overline{n}}\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(j)^{(k+1)}_{\overline{n}}}\right]_{\pm}\left(\sum_s W(j)^{(k)\top}_s \left(\frac{\overleftarrow{X}^{\,s}}{W^{(k)}*H^{(k)}}\right)\right)_{\overline{n}} \\
&+ \sum_{u,m} w(j)^{(k)}_{m,u}\left[\sum_p\left(\frac{X}{\left(W^{(k)}*H^{(k)}\right)^{\circ 2}}\right)_{m,p}\left[\sum_{i,s} w(i)^{(k)}_{m,s}\left[\sum_m\left[\frac{\partial\hat{\mathcal{F}}}{\partial h(i)^{(k+1)}_{n}}\right]_{\mp}\left(\frac{H(i)^{(k)}}{\sum_t W(i)^{(k)\top}_t \overleftarrow{\mathbb{1}_{m\times n}}^{\,t}}\right)_{n}(J^s)_{n,p}\right]\right](J^u)_{\overline{n},p}\right] \\
&= \left[\left(\frac{\left[\nabla_{H(j)^{(k+1)}}\hat{\mathcal{F}}\right]_{\pm}}{\sum_t W(j)^{(k)\top}_t \overleftarrow{\mathbb{1}_{m\times n}}^{\,t}}\right)\circ\left(\sum_s W(j)^{(k)\top}_s \left(\frac{\overleftarrow{X}^{\,s}}{W^{(k)}*H^{(k)}}\right)\right)\right]_{\overline{n}} \\
&+ \left[\sum_u W(j)^{(k)\top}_u \left[\left(\frac{X}{\left(W^{(k)}*H^{(k)}\right)^{\circ 2}}\right)\circ\left[\sum_{i,s} W(i)^{(k)}_s\left(\left[\nabla_{H(i)^{(k+1)}}\hat{\mathcal{F}}\right]_{\mp}\circ\frac{H(i)^{(k)}}{\sum_t W(i)^{(k)\top}_t \overleftarrow{\mathbb{1}_{m\times n}}^{\,t}}\right)\right]\right]\right]_{\overline{n}}^{\overleftarrow{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} u \overrightarrow{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} s} \\
&= \left[\Phi_{H(j)}\left(\nabla_{H^{(k+1)}}\hat{\mathcal{F}}, H^{(k)}, W^{(k)}, X\right)_{\overline{n}}\right]_{\pm}
\end{aligned}
\tag{4.B.5}
$$

Denote $\hat{W}(i) = \{\hat{w}(i)_{m,r}\}_{m,r}$ and $S = \{s_{m,n}\}_{m,n}$. The loss-function written elementwise (for a single training instance) is of the form:

$$
\begin{aligned}
\hat{\mathcal{F}} &= \sum_{m,n}\frac{1}{2}\left(x_{m,n}\frac{\sum_{i\,\in\,J_S}\sum_r \hat{w}(i)_{m,r}\sum_q h(i)^{(K)}_q (J^r)_{q,n}}{\sum_{i\,\in\,J_S}\sum_s \hat{w}(i)_{m,s}\sum_q h(i)^{(K)}_q (J^s)_{q,n} + \sum_{i\,\in\,J_N}\sum_t \hat{w}(i)_{m,t}\sum_q h(i)^{(K)}_q (J^t)_{q,n}} - s_{m,n}\right)^2 \\
&\quad + \sum_{m,n}\frac{\alpha_S}{2}\left(\sum_{i\,\in\,J_S}\sum_u \hat{w}(i)_{m,u}\sum_q h(i)^{(K)}_q (J^u)_{q,n} - s_{m,n}\right)^2
\end{aligned}
\tag{4.B.6}
$$

Deriving (4.B.6) with respect to $h(j)^{(K)}_{\overline{n}}$ for $j \in J_S$, we obtain:

$$
\begin{aligned}
&\frac{\partial \hat{\mathcal{F}}}{\partial h(j)_{\overline{n}}^{(K)}} = \sum_{m,n} \left(\frac{X \circ \left(\hat{W}_S * H_S^{(K)} \right)}{\hat{W} * H^{(K)}} - S \right)_{m,n} \left(\frac{X}{\hat{W} * H^{(K)}} \right)_{m,n} \sum_r \hat{w}(j)_{m,r} (J^r)_{\overline{n},n} \\
&+ \sum_{m,n} \alpha_S \left(\hat{W}_S * H_S^{(K)} - S \right)_{m,n} \sum_u \hat{w}(j)_{m,u} (J^u)_{\overline{n},n} \\
&- \sum_{m,n} \left(\frac{X \circ \left(\hat{W}_S * H_S^{(K)} \right)}{\hat{W} * H^{(K)}} - S \right)_{m,n} \left(\frac{X \circ \left(\hat{W}_S * H_S^{(K)} \right)}{\left(\hat{W} * H^{(K)} \right)^{\circ 2}} \right)_{m,n} \sum_s \hat{w}(j)_{m,s} (J^s)_{\overline{n},n} \\
&= \sum_{r,m} \hat{w}(j)_{m,r} \left[\sum_n \left(\frac{\left(\hat{W}_S * H_S^{(K)} \right) \circ X^{\circ 2} + S \circ X \circ \left(\hat{W}_S * H_S^{(K)} \right)}{\left(\hat{W} * H^{(K)} \right)^{\circ 2}} + \alpha_S \left(\hat{W}_S * H_S^{(K)} \right) \right)_{m,n} (J^r)_{\overline{n},n} \right] \\
&- \sum_{r,m} \hat{w}(j)_{m,r} \left[\sum_n \left(\frac{S \circ X}{\hat{W} * H^{(K)}} + \frac{\left(\hat{W}_S * H_S^{(K)} \right)^{\circ 2} \circ X^{\circ 2}}{\left(\hat{W} * H^{(K)} \right)^{\circ 3}} + \alpha_S S \right)_{m,n} (J^r)_{\overline{n},n} \right] \\
&= \left[\sum_r \hat{W}(j)_r^\top \left[\frac{\overleftarrow{\left(\hat{W}_S * H_S^{(K)} \right) \circ X^{\circ 2} + S \circ X \circ \left(\hat{W}_S * H_S^{(K)} \right)}^{r}}{\left(\hat{W} * H^{(K)} \right)^{\circ 2}} + \alpha_S \left(\hat{W}_S * H_S^{(K)} \right) \right] \right]_{\overline{n}} \\
&- \left[\sum_r \hat{W}(j)_r^\top \left[\frac{\overleftarrow{S \circ X}^{r}}{\hat{W} * H^{(K)}} + \frac{\left(\hat{W}_S * H_S^{(K)} \right)^{\circ 2} \circ X^{\circ 2}}{\left(\hat{W} * H^{(K)} \right)^{\circ 3}} + \alpha_S S \right] \right]_{\overline{n}} \\
&= \left[\frac{\partial \hat{\mathcal{F}}}{\partial h(j)_{\overline{n}}^{(K)}} \right]_+ - \left[\frac{\partial \hat{\mathcal{F}}}{\partial h(j)_{\overline{n}}^{(K)}} \right]_- = \left[\Phi_K^j (H^{(K)}, X)_{\overline{n}} \right]_+ - \left[\Phi_K^j (H^{(K)}, X)_{\overline{n}} \right]_-
\end{aligned}
\tag{4.B.7}
$$

Similarly, deriving (4.B.6) with respect to $h(j)_{\overline{n}}^{(K)}$ for $j \in J_N$, we obtain:

$$
\begin{aligned}
&\frac{\partial \hat{\mathcal{F}}}{\partial h(j)_{\overline{n}}^{(K)}} = - \sum_{m,n} \left(\frac{X \circ \left(\hat{W}_S * H_S^{(K)} \right)}{\hat{W} * H^{(K)}} - S \right)_{m,n} \left(\frac{X \circ \left(\hat{W}_S * H_S^{(K)} \right)}{\left(\hat{W} * H^{(K)} \right)^{\circ 2}} \right)_{m,n} \sum_t \hat{w}(j)_{m,t} (J^t)_{\overline{n},n} \\
&= \sum_{r,m} \hat{w}(j)_{m,r} \left[\sum_n \left(\frac{S \circ X \circ \left(\hat{W}_S * H_S^{(K)} \right)}{\left(\hat{W} * H^{(K)} \right)^{\circ 2}} \right)_{m,n} (J^r)_{\overline{n},n} \right] - \sum_{r,m} \hat{w}(j)_{m,r} \left[\sum_n \left(\frac{\left(\hat{W}_S * H_S^{(K)} \right)^{\circ 2} \circ X^{\circ 2}}{\left(\hat{W} * H^{(K)} \right)^{\circ 3}} \right)_{m,n} (J^r)_{\overline{n},n} \right] \\
&= \left[\sum_r \hat{W}(j)_r^\top \left[\frac{\overleftarrow{S \circ X \circ \left(\hat{W}_S * H_S^{(K)} \right)}^{r}}{\left(\hat{W} * H^{(K)} \right)^{\circ 2}} \right] \right]_{\overline{n}} - \left[\sum_r \hat{W}(j)_r^\top \left[\frac{\overleftarrow{\left(\hat{W}_S * H_S^{(K)} \right)^{\circ 2} \circ X^{\circ 2}}^{r}}{\left(\hat{W} * H^{(K)} \right)^{\circ 3}} \right] \right]_{\overline{n}} \\
&= \left[\frac{\partial \hat{\mathcal{F}}}{\partial h(j)_{\overline{n}}^{(K)}} \right]_+ - \left[\frac{\partial \hat{\mathcal{F}}}{\partial h(j)_{\overline{n}}^{(K)}} \right]_- = \left[\Phi_K^j (H^{(K)}, X)_{\overline{n}} \right]_+ - \left[\Phi_K^j (H^{(K)}, X)_{\overline{n}} \right]_-
\end{aligned}
\tag{4.B.8}
$$

Kalman filtering

In this chapter, we review the Kalman filter and a set of ensemble-based extensions, including the Ensemble Kalman filter and particle filters. We present a unified probabilistic perspective that highlights the common structure underlying these algorithms, while emphasizing their key differences in assumptions and objectives. Each method is described in detail, with particular attention to its prediction and update mechanisms.

5.1 Introduction

State estimation from incomplete, noisy, and heterogeneous information sources is a fundamental problem in science and engineering. In many applications, ranging from control systems and signal processing to geophysics, robotics, and finance, the goal is to infer the latent state of a dynamical system by combining a mathematical model of its evolution with partial and noisy observations. This general inference task is commonly referred to as data assimilation, a paradigm that formalizes the fusion of prior model-based predictions with measurement data in a principled probabilistic framework [41, 40, 109].
Within this context, the Kalman filter (KF) stands as one of the most influential and widely used data assimilation techniques. Originally introduced in the early 1960s for aerospace navigation problems [72], the Kalman filter provides an optimal recursive solution, under linearity and Gaussian noise assumptions, to the problem of estimating the state of a dynamical system from sequential observations. Its appeal lies in its recursive predictor-corrector structure, which enables real-time operation, and in its strong optimality guarantees: among all linear unbiased estimators, it minimizes the estimation error covariance [95, 71]. From a data assimilation perspective, the Kalman filter can be interpreted as a sequential Bayesian estimator that propagates and updates probability distributions over the system state. At each time step, prior information obtained from the system dynamics is combined with new observational evidence to produce a posterior estimate. Under the assumptions of linear dynamics and additive Gaussian noise, all relevant distributions remain Gaussian and are therefore fully characterized by their first- and second-order statistics, namely the mean and covariance. This observation explains both the computational efficiency and the limitations of the classical Kalman filter [116].
Despite its success, the standard Kalman filter relies on assumptions that are often violated in practical scenarios. Real-world systems frequently exhibit nonlinear dynamics, non-Gaussian noise, partial observability, or model uncertainty. To address these challenges, a broad family of Kalman filter extensions has been developed. Deterministic approximations such as the Extended Kalman Filter (EKF) and the Unscented Kalman

Filter (UKF) relax the linearity assumption at the cost of additional approximations [70]. In parallel, ensemble-based methods, most notably the Ensemble Kalman Filter (EnKF), reinterpret the Kalman update in a Monte Carlo setting, enabling scalable state estimation for high-dimensional systems commonly encountered in data assimilation problems in meteorology and oceanography [39, 40]. Beyond ensemble Kalman methods, particle filters provide a fully nonparametric alternative by representing probability distributions through weighted samples. These methods can, in principle, approximate arbitrary posterior distributions and thus overcome both linearity and Gaussianity assumptions. However, this generality comes at a significant computational cost and introduces challenges such as sample degeneracy, especially in high-dimensional settings [36, 5].
In this chapter, we review the Kalman filter and its ensemble-based extensions from a unified data assimilation viewpoint. We begin by revisiting discrete-time linear dynamical systems and their state-space representations, which form the foundation of Kalman filtering theory. We then present the classical Kalman filter, emphasizing its probabilistic interpretation and recursive structure. Building upon this framework, we introduce ensemble Kalman methods and particle filters, highlighting their connections, differences, and respective trade-offs. This progression sets the stage for the following chapter, where modern learning-based extensions of Kalman filtering are discussed.

5.2 DLTI systems

Throughout this section, we denote by $x(:)$ a $\mathbb{Z}$-indexed sequence, namely $x(:) = \{x(k)\}_{k\in\mathbb{Z}}$. A discrete linear time-invariant (DLTI) dynamical system is an operator T that maps discrete-time input sequences $u(:)$ to discrete-time output sequences $y(:) = T[u(:)]$ and satisfies the following properties:

1. *Linearity.* $T[\alpha_1 u_1(:) + \alpha_2 u_2(:)] = \alpha_1 T[u_1(:)] + \alpha_2 T[u_2(:)]$ $\forall \alpha_1, \alpha_2$, $\forall u_1(:), u_2(:)$;
2. *Time-invariance.* $T[u(:-j)] = T[u(:)](:-j)$ $\forall j \in \mathbb{Z}$.

Interestingly, by exploiting the Z-transform [100], DLTI systems can be expressed in three equivalent ways:

- **Convolution sum**. By introducing the discrete impulse $\delta(:)$, defined by $\delta(k) = \delta_{0,k}$ with $\delta_{0,k}$ denoting the Kronecker delta, a DLTI system is uniquely characterized by its discrete impulse response $h(:) = T[\delta(:)]$. Indeed:

$$y(:) = (h * u)(:) = \sum_{j\in\mathbb{Z}} u(j)h(:-j) \tag{5.2.1}$$

  This is a direct consequence of the linearity and time-invariance assumptions applied to the input $u(:) = \sum_{j\in\mathbb{Z}} u(j)\delta(:-j)$.

- **Auto-regressive moving-average (ARMA)**. For suitable coefficient sets $\{a_i\}_{i=1,\dots,n_a}$ and $\{b_j\}_{j=1,\dots,n_b}$, the following relation holds:

$$\sum_{i=0}^{n_a} a_i y(k-i) = \sum_{j=0}^{n_b} b_j u(k-j) \tag{5.2.2}$$

In this formulation, n_a denotes the model order. Furthermore, to ensure a one-to-one correspondence between the model parameters and the resulting input–output behavior, it is customary to impose the normalization condition $a_0 = 1$.

- **State-space**. For some suitable matrices $A \in \mathcal{M}_{n\times n}(\mathbb{R})$, $B \in \mathcal{M}_{n\times q}(\mathbb{R})$, $C \in \mathcal{M}_{m\times n}(\mathbb{R})$ and $D \in \mathcal{M}_{m\times q}(\mathbb{R})$, it holds:

$$\begin{aligned} x(k) &= Ax(k-1) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \tag{5.2.3}$$

In this formulation, n denotes the model order, and the first equation in (5.2.3) is known as the state equation. In particular, we remark that systems in the form (5.2.3) are often obtained by discretizing a Linear Time-Invariant (LTI) system in continuous time:

$$\begin{aligned} \dot{x}(t) &= \tilde{A}(t)x(t) + \tilde{B}(t)u(t) \\ y(t) &= \tilde{C}(t)x(t) + \tilde{D}(t)u(t) \end{aligned} \tag{5.2.4}$$

State-space representations are particularly attractive because they naturally enable the analysis of fundamental system properties such as *reachability* and *observability*. Given an initial condition $x_0 \in \mathbb{R}^n$ and an input sequence $u(:)$, we denote by:

$$x(k; u, x_0) = A^k x_0 + \sum_{j=0}^{k-1} A^{k-1-j} B u(j) \tag{5.2.5}$$

the solution of the state equation at time k, and by:

$$y(k; u, x_0) = Cx(k; u, x_0) + Du(k) \tag{5.2.6}$$

the corresponding output. A state $\bar{x} \in \mathbb{R}^n$ is:

- *Reachable* if $\exists \bar{u}(:)$ and $\exists \bar{k}$ such that $\bar{x} = x(\bar{k}; \bar{u}, 0)$. In particular, by letting:

$$\chi_R = \{x \in \mathbb{R}^n \mid x \text{ is reachable}\}$$

it holds $\chi_R = \mathrm{Im}(\mathcal{R})$, where $\mathcal{R} \in \mathcal{M}_{n\times nq}(\mathbb{R})$ is the reachability matrix:

$$\mathcal{R} = \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix} \tag{5.2.7}$$

If $\mathrm{rank}(\mathcal{R}) = n$, the state-space system (5.2.3) is reachable.

- *Not observable* if $\forall u(:)$ and $\forall k$ it holds $y(k; u, \bar{x}) = y(k; u, 0)$. In particular, by letting:

$$\chi_{NO} = \{x \in \mathbb{R}^n \mid x \text{ is not observable}\}$$

it holds $\chi_{NO} = \ker(\mathcal{O})$, where $\mathcal{O} \in \mathcal{M}_{nm\times n}(\mathbb{R})$ is the observability matrix:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{5.2.8}$$

If $\text{rank}(\mathcal{O}) = n$, the state-space system (5.2.3) is observable.

Moreover, one can also define the following reachability and observability Gramians:

$$G_R = \mathcal{R}\mathcal{R}^\top \qquad G_O = \mathcal{O}^\top\mathcal{O} \tag{5.2.9}$$

In this context, the reachability and observability Gramians provide quantitative measures of how easily states can be controlled or observed. The reachability Gramian G_R characterizes the ability to steer the system from the origin to a given state using admissible inputs, while the observability Gramian G_O quantifies how well internal states can be inferred from output measurements. These matrices are fundamental tools in control theory, as they allow the assessment of controllability and observability, support model reduction techniques, and are central to energy-based analyses of system behavior [71, 4].

5.3 Kalman filter

In this section, we present a comprehensive overview of the Kalman filter, largely following the exposition in [67]. Let us consider a *noisy* state-space system[1] in the form:

$$\begin{aligned} x(k) &= A(k)x(k-1) + B(k)u(k) + w(k) \\ y(k) &= C(k)x(k) + v(k) \end{aligned} \tag{5.3.1}$$

where $w(k) \sim \mathcal{N}(0, Q(k))$ and $v(k) \sim \mathcal{N}(0, R(k))$ are uncorrelated, zero-mean normal random vectors with positive-definite covariances $Q(k) \in \mathcal{M}_{n\times n}(\mathbb{R})$ and $R(k) \in \mathcal{M}_{m\times m}(\mathbb{R})$, respectively. Let us also assume the initial state of the system to be given by $x(0) = \mu_0 + w(0)$, for some known $\mu_0 \in \mathbb{R}^n$.
Given l known observations $y(1), \dots, y(l)$ and k known inputs $u(1), \dots, u(k)$, with $k \geq l$, we consider the problem of finding the best linear unbiased estimate of the states $x(1), \dots, x(k)$. A possible approach is to rewrite (5.3.1) as a linear system:

[1] In the particular case of constant matrices $A(k) \equiv A$, $B(k) \equiv B$ and $C(k) \equiv C$ we obtain a DLTI system as in (5.2.3).

$$\begin{array}{rcccc} \mu_0 & = & x(0) & - & w(0) \\ B(1)u(1) & = & x(1) - A(1)x(0) & - & w(1) \\ y(1) & = & C(1)x(1) & + & v(1) \\ & \vdots & & \vdots & \\ B(l)u(l) & = & x(l) - A(l)x(l-1) & - & w(l) \\ y(l) & = & C(l)x(l) & + & v(l) \\ B(l+1)u(l+1) & = & x(l+1) - A(l+1)x(l) & - & w(l+1) \\ & \vdots & & \vdots & \\ B(k)u(k) & = & x(k) - A(k)x(k-1) & - & w(k) \end{array} \tag{5.3.2}$$

which admits a compact representation in the form:

$$b(k \mid l) = \mathcal{A}(k \mid l)z(k) + \varepsilon(k \mid l), \qquad z(k) = \begin{bmatrix} x(0) \\ \vdots \\ x(k) \end{bmatrix} \tag{5.3.3}$$

Since $\varepsilon(k \mid l)$ is a zero-mean random vector with block-diagonal inverse covariance:

$$W(k \mid l) = \operatorname{diag}(Q(0)^{-1}, Q(1)^{-1}, R(1)^{-1}, \ldots, Q(l)^{-1}, R(l)^{-1}, Q(l+1)^{-1}, \ldots, Q(k)^{-1}) \tag{5.3.4}$$

and since $\mathcal{A}(k \mid l)$ is full rank by construction, the desired estimate $\hat{z}(k \mid l)$ of $z(k)$ is obtained by solving (5.3.3) in a weighted least-squares sense:

$$\hat{z}(k \mid l) = \left[\mathcal{A}(k \mid l)^\top W(k \mid l)\mathcal{A}(k \mid l)\right]^{-1} \mathcal{A}(k \mid l)^\top W(k \mid l)b(k \mid l), \qquad \hat{z}(k \mid l) = \begin{bmatrix} \hat{x}(0 \mid l) \\ \vdots \\ \hat{x}(k \mid l) \end{bmatrix} \tag{5.3.5}$$

This general state estimation framework can be specialized in several ways, depending on the choice of l and the intended objective. In particular, we distinguish three well-known scenarios:

• *Kalman prediction.* For $k > l$, the framework specializes to state prediction, where the states $x(l+1), \ldots, x(k)$ are estimated using only the prior estimate $\hat{x}(l \mid l)$ and the inputs $u(l+1), \ldots, u(k)$:

$$\hat{x}(l \mid l),\, u(l+1), \ldots, u(k) \quad \underset{\text{prediction}}{\Longrightarrow} \quad \hat{x}(l+1 \mid l), \ldots, \hat{x}(k \mid l)$$

• *Kalman filtering.* For $k = l$, the framework yields the standard Kalman filtering problem, in which the state $x(k)$ is estimated from the previous estimate $\hat{x}(k-1 \mid k-1)$ using the input $u(k)$ and the measurement $y(k)$:

$$\hat{x}(k-1 \mid k-1),\, u(k),\, y(k) \quad \underset{\text{filtering}}{\Longrightarrow} \quad \hat{x}(k \mid k)$$

• *Kalman smoothing.* In the smoothing setting, the entire state trajectory $z(k)$ is estimated from the prior estimate $\hat{z}(k \mid l-1)$ after incorporating the measurement $y(l)$:

$$\hat{z}(k \mid l-1),\, y(l) \quad \underset{\text{smoothing}}{\Longrightarrow} \quad \hat{z}(k \mid l)$$

In this chapter, and in the following one, we focus on Kalman filtering. The next section, in particular, provides an overview of the classical two-step Kalman filter algorithm, while a more detailed derivation of the associated relations and additional technical insights are deferred to Appendix 5.B.

5.3.1 Predictor-corrector state estimation

The Kalman filter links the optimal state estimate at discrete time k, $\hat{x}(k) = \hat{x}(k \mid k)$, to the previous estimate $\hat{x}(k-1) = \hat{x}(k-1 \mid k-1)$. By definition, these vectors correspond to the last components of $\hat{z}(k) = \hat{z}(k \mid k)$ and $\hat{z}(k-1) = \hat{z}(k-1 \mid k-1)$, respectively, which solve the normal equations associated with (5.3.3) for $l = k-1$ and $l = k$. The strength of the Kalman filter lies in its ability to compute $\hat{x}(k)$ recursively from $\hat{x}(k-1)$, without explicitly solving these systems.
The recursive updates proceed via a two-step procedure, initialized with $\hat{x}(0) = \mu_0$ and $P(0) = P(0 \mid 0) = Q(0)$. In the *prediction* step, both the state estimate and its error covariance are propagated forward according to the DLTI system (5.3.1):

$$P(k \mid k-1) = A(k)P(k-1)A(k)^\top + Q(k) \tag{5.3.6}$$
$$\hat{x}(k \mid k-1) = A(k)\hat{x}(k-1) + B(k)u(k) \tag{5.3.7}$$

This step requires the input $u(k)$. In the *correction* step, these quantities are then updated using the observation $y(k)$. In particular, by defining the prediction error (or innovation):

$$e_y(k) = y(k) - C(k)\hat{x}(k \mid k-1) \tag{5.3.8}$$

one obtains:

$$P(k) = \left[P(k \mid k-1)^{-1} + C(k)^\top R(k)^{-1} C(k)\right]^{-1} \tag{5.3.9}$$
$$\hat{x}(k) = \hat{x}(k \mid k-1) + P(k)C(k)^\top R(k)^{-1} e_y(k) \tag{5.3.10}$$

Equation (5.3.10) shows that the Kalman filter applies a proportional feedback on the prediction error $e_y(k)$, scaled by the (Kalman) gain:

$$\widetilde{\mathcal{K}}_G^{(k)} = P(k)\, C(k)^\top R(k)^{-1} \tag{5.3.11}$$

By invoking Proposition 5.A.1, one can further derive the following equivalent expressions for $\widetilde{\mathcal{K}}_G^{(k)}$ and $P(k)$:

$$\widetilde{\mathcal{K}}_G^{(k)} = P(k \mid k-1)C(k)^\top \left[C(k)P(k \mid k-1)C(k)^\top + R(k)\right]^{-1} \tag{5.3.12}$$
$$P(k) = \left[\mathbb{1}_n - \widetilde{\mathcal{K}}_G^{(k)} C(k)\right] P(k \mid k-1) \tag{5.3.13}$$

In particular, (5.3.12) shows that the Kalman gain can be expressed as the product of two covariance matrices:

$$\begin{aligned} \widetilde{\mathcal{K}}_G^{(k)} &= \operatorname{cov}[x(k), y(k) \mid k-1] \operatorname{cov}[y(k), y(k) \mid k-1]^{-1} \\ \operatorname{cov}[x(k), y(k) \mid k-1] &= P(k \mid k-1) C(k)^\top \\ \operatorname{cov}[y(k), y(k) \mid k-1] &= C(k) P(k \mid k-1) C(k)^\top + R(k) \end{aligned} \tag{5.3.14}$$

where we use $k-1$ as shorthand for the observation sequence $y(1), \ldots, y(k-1)$. Equivalently, the Kalman gain is the solution to the linear system:

$$\widetilde{\mathcal{K}}_G^{(k)} \operatorname{cov}[y(k), y(k) \mid k-1] = \operatorname{cov}[x(k), y(k) \mid k-1] \tag{5.3.15}$$

As will be discussed in the next Section 5.4.1, this formulation naturally enables a generalization of the Kalman filter to particle ensembles, where the covariance matrices are replaced by their empirical counterparts.

When the system matrices are constant, i.e., $A(k) \equiv A$, $B(k) \equiv B$ and $C(k) \equiv C$, the notions of reachability and observability introduced in Section 5.2 can be directly applied to the Kalman filter. This connection provides valuable insight into the filter's ability to predict and correct state estimates. Indeed, the reachability and observability Gramians (5.2.9) quantify how easily states can be influenced by inputs and inferred from outputs, respectively. From this perspective, the reachability Gramian G_R determines how process noise propagates through the system, influencing the predicted uncertainty $P(k \mid k-1)$. Similarly, the observability Gramian G_O governs how effectively measurements reduce uncertainty in the updated covariance $P(k)$. States with low observability energy (small eigenvalues of G_O) are inherently harder to estimate, leading to higher posterior uncertainty, whereas states with low reachability energy (small eigenvalues of G_R) are more sensitive to process noise, potentially degrading prediction accuracy. The Kalman filter's performance can therefore be directly interpreted in terms of system structure: the Gramians provide a theoretical foundation for understanding which states are well-estimated, which are inherently uncertain, and how the choice of $Q(k)$ and $R(k)$ interacts with system dynamics to shape estimation accuracy. A practical demonstration of this interplay is provided in Section 6.4.4.

5.3.2 Limitations

While the Kalman filter provides optimal state estimates for linear systems with Gaussian noise, it suffers from several important limitations that must be considered in practical applications.

- *Known dynamics.* Its performance critically depends on accurate knowledge of the system dynamics and the noise covariances; any mismatch in the system model or misestimation of the process and measurement noise can lead to biased estimates or even divergence of the filter [116];
- *Linearity.* The standard Kalman filter is inherently limited to linear systems. For nonlinear dynamics, extensions such as the Extended Kalman Filter (EKF) or the Unscented

Kalman Filter (UKF) are required, but these approaches introduce approximations that may compromise accuracy and increase computational cost [70];

- *Gaussianity.* The Kalman filter assumes Gaussian noise. In the presence of heavy-tailed, impulsive, or otherwise non-Gaussian disturbances, its estimates are no longer optimal and may exhibit instability [5];

- *Cost.* The computational and memory requirements grow rapidly with the system dimension, which can pose challenges for real-time implementation on resource-constrained platforms [58].

These limitations are partially addressed in the next section, 5.4, which presents ensemble-based extensions of the Kalman filter. By contrast, Chapter 6 tackles all of the aforementioned challenges using recent Kalman filter extensions that integrate machine learning techniques.

5.4 Ensemble extensions to the Kalman filter

In this section, we briefly review the mathematical foundations of the Bayes filter [5, 113], a general framework that encompasses the Kalman filter and several of its ensemble extensions. At a high level, the Bayes filter recursively estimates $p(x(k) \mid k)$ from the filtered distribution at the previous time step $p(x(k-1) \mid k-1)$, where, as before, $k-1$ and k are used as shorthand for the corresponding sequences of observations.
Similarly to the Kalman filter, the Bayes filter updates the state distribution through a two-step procedure. In the *prediction* step, the distribution $p(x(k-1) \mid k-1)$ is propagated forward using only the process model, which specifies the conditional distribution $p(x(k) \mid x(k-1))$:

$$p(x(k) \mid k-1) = \int p(x(k) \mid x(k-1)) \cdot p(x(k-1) \mid k-1) \, \mathrm{d}x(k) \tag{5.4.1}$$

In the *update* step, Bayes' theorem is applied to compute the posterior distribution, taking $p(x(k) \mid k-1)$ as the prior and $p(y(k) \mid x(k))$ as the likelihood, the latter being specified by the observation model. The resulting posterior is thus given by:

$$p(x(k) \mid k) \propto p(y(k) \mid x(k)) \cdot p(x(k) \mid k-1) \tag{5.4.2}$$

Let us now relate the Bayes filter to the Kalman filter described in Section 5.3. Under linear dynamics and Gaussian noise, all relevant distributions are Gaussian, and thus fully determined by their mean (first-order statistic) and covariance (second-order statistic). At each iteration, the Kalman filter implicitly estimates $p(x(k) \mid k)$ by propagating the state estimate $\hat{x}(k)$ and its covariance $P(k)$ to time k. Specifically, in the prediction step, the prior distribution (5.4.1) is characterized by computing its mean $\hat{x}(k \mid k-1)$ and covariance $P(k \mid k-1)$ according to (5.3.7) and (5.3.6):

$$x(k) \mid k-1 \sim \mathcal{N}(\hat{x}(k \mid k-1), P(k \mid k-1)) \tag{5.4.3}$$

By exploiting the fact that the likelihood takes the form:

$$y(k) \mid x(k) \sim \mathcal{N}(C(k)x(k), R(k)) \tag{5.4.4}$$

with a slight abuse of notation, the posterior (5.4.2) reduces to:

$$x(k) \mid k \sim \mathcal{N}(C(k)x(k), R(k)) \cdot \mathcal{N}(\hat{x}(k \mid k-1), P(k \mid k-1)) = \mathcal{N}(\hat{x}(k), P(k)) \tag{5.4.5}$$

where $\hat{x}(k)$ and $P(k)$ are computed using (5.3.10) and (5.3.9). Interestingly, the MAP estimate of $\hat{x}(k)$ provided by (5.4.5) reads[2]:

$$\hat{x}(k) = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2}\|y(k) - C(k)x\|^2_{R(k)^{-1}} + \frac{1}{2}\|x - \hat{x}(k \mid k-1)\|^2_{P(k|k-1)^{-1}} \tag{5.4.6}$$

From this perspective, the Kalman filter can be viewed as a highly specialized instance of the Bayes filter.

5.4.1 Ensemble Kalman filter

The Ensemble Kalman Filter [40] extends the classical Kalman filter to alleviate several of its well-known limitations, most notably the assumption of linear system dynamics and the requirement of Gaussian process and observation noise. In particular, let us now consider a general state-space system in the form:

$$\begin{aligned} x(k) &= f(x(k-1), w(k), k) \\ y(k) &= h(x(k), v(k), k) \end{aligned} \tag{5.4.7}$$

where f is the system's forward operator and h is the observation model, both of which may be nonlinear. As in (5.3.1), $w(k)$ and $v(k)$ represent uncorrelated process and observation noise; however, the Gaussianity assumption is no longer imposed. As we will now detail, the EnKF provides a Monte Carlo approximation to the states' first- and second-order statistics $\hat{x}(k)$ and $P(k)$, respectively.
Suppose that, at time $k-1$, we are given an ensemble of N particles $\{x^i(k-1)\}_{i=1}^N$ acting as surrogate samples from the distribution $p(x(k-1) \mid k-1)$. The EnKF provides a procedure to propagate and update this ensemble to time k, yielding $\{x^i(k)\}_{i=1}^N$. First, each particle is propagated through the forward operator and observation model to obtain:

$$x^i(k \mid k-1) = f(x^i(k-1), w^i(k), k) \tag{5.4.8}$$

$$y^i(k \mid k-1) = h(x^i(k \mid k-1), v^i(k), k) \tag{5.4.9}$$

where $v^i(k)$ and $w^i(k)$ are sampled from $p(v(k))$ and $p(w(k))$, respectively. This ensures that the ensembles $\{x^i(k \mid k-1)\}_{i=1}^N$ and $\{y^i(k \mid k-1)\}_{i=1}^N$ can be regarded as surrogate samples from $p(x(k) \mid k-1)$ and $p(y(k) \mid k-1)$. Next, the propagated ensemble is corrected using a relation analogous to (5.3.10):

$$x^i(k) = x^i(k \mid k-1) + \widetilde{\mathcal{K}}^{(k)}_{\mathrm{En}} \left(y(k) - y^i(k \mid k-1)\right) \tag{5.4.10}$$

[2]We denote $\|v\|^2_W = v^\top W v$ the 2-norm squared of a vector v weighted by W.

where $\widetilde{\mathcal{K}}_{\text{En}}^{(k)}$ is an ensemble Kalman gain matrix obtained by replacing the covariances in (5.3.14) with their empirical counterparts, computed from (5.4.8) and (5.4.9). Denoting:

$$\bar{x}(k \mid k-1) = \frac{1}{N}\sum_{i=1}^{N} x^i(k \mid k-1) \qquad \bar{y}(k \mid k-1) = \frac{1}{N}\sum_{i=1}^{N} y^i(k \mid k-1) \tag{5.4.11}$$

the ensemble Kalman gain is given by:

$$\begin{aligned}
\widetilde{\mathcal{K}}_{\text{En}}^{(k)} &= \text{cov}_{\text{En}}\left[x(k), y(k) \mid k-1\right] \text{cov}_{\text{En}}\left[y(k), y(k) \mid k-1\right]^{-1} \\
\text{cov}_{\text{En}}\left[x(k), y(k) \mid k-1\right] &= \frac{1}{N}\sum_{i=1}^{N} \left(x^i(k \mid k-1) - \bar{x}(k \mid k-1)\right)\left(y^i(k \mid k-1) - \bar{y}(k \mid k-1)\right)^\top \\
\text{cov}_{\text{En}}\left[y(k), y(k) \mid k-1\right] &= \frac{1}{N}\sum_{i=1}^{N} \left(y^i(k \mid k-1) - \bar{y}(k \mid k-1)\right)\left(y^i(k \mid k-1) - \bar{y}(k \mid k-1)\right)^\top
\end{aligned} \tag{5.4.12}$$

The update (5.4.10) ensures that the ensemble $\{x^i(k)\}_{i=1}^N$ represents surrogate samples from the posterior distribution $p(x(k) \mid k)$. As a consequence, the first and second order statistics can be recovered as follows:

$$\hat{x}_{\text{En}}(k) = \frac{1}{N}\sum_{i=1}^{N} x^i(k) \tag{5.4.13}$$

$$P_{\text{En}}(k) = \frac{1}{N}\sum_{i=1}^{N} \left(x^i(k) - \hat{x}_{\text{En}}(k)\right)\left(x^i(k) - \hat{x}_{\text{En}}(k)\right)^\top \tag{5.4.14}$$

Before moving to the next section, it is important to clarify an implicit convention used in our presentation of the EnKF. Throughout the exposition, we treated the ensembles generated at each stage of the algorithm as *surrogate* samples for the *true* distributions governing the system, as discussed in the Bayes filter in Section 5.4. This notational caveat arises because the ensembles represent samples from the true distributions only in the case of linear systems with additive Gaussian noise, where the EnKF converges to the Kalman filter solution as the ensemble size $N \to +\infty$. For nonlinear systems or non-Gaussian noise, this property does not hold, since the update step (5.4.10) inherently assumes linearity and Gaussianity. As a result, the EnKF primarily provides estimates of the first- (5.4.13) and second- (5.4.14) order statistics, which are exact only under the standard Kalman filter assumptions.

5.4.2 Particle filters

In general non-Gaussian noise settings, the first- and second-order statistics provided by the EnKF are not sufficient to fully characterize the state distributions generated by systems of the form (5.4.7). Particle filters [36, 68], on the other hand, offer a more general inference framework by explicitly representing and propagating approximations of the full state distributions through the prediction and update steps. Unlike the EnKF, whose ensembles primarily encode low-order moments, particle filters aim to produce

ensembles that asymptotically approximate the true posterior distributions as the number of particles increases.
Suppose that, at time $k-1$, we are given an ensemble of N weighted particles $\{x^i(k-1), \lambda^i(k-1)\}_{i=1}^N$ representing the distribution $p(x(k-1) \mid k-1)$. In particular, this allows us to approximate the distribution by its corresponding weighted empirical measure:

$$p(x(k-1) \mid k-1) \approx \frac{1}{N} \sum_{i=1}^{N} \lambda^i(k-1) \delta \left(x(k-1) - x^i(k-1) \right) \tag{5.4.15}$$

Particle filters provide a procedure to propagate and update this weighted ensemble to time k, yielding $\{x^i(k), \lambda^i(k)\}_{i=1}^N$. First, each particle is propagated through the forward operator and observation model to obtain:

$$x^i(k \mid k-1) = f(x^i(k-1), w^i(k), k) \tag{5.4.16}$$
$$y^i(k \mid k-1) = h(x^i(k \mid k-1), v^i(k), k) \tag{5.4.17}$$

This step remains identical to that of the EnKF. Next, the particle weights are updated according to the likelihood of the observation $y(k)$ given each predicted observation $y^i(k \mid k-1)$, namely:

$$\tilde{\lambda}^i(k) = \lambda^i(k-1) \cdot p(y(k) \mid y^i(k \mid k-1)) = \lambda^i(k-1) \cdot p(y(k) \mid x^i(k \mid k-1)) \tag{5.4.18}$$

The resulting weights are then self-normalized:

$$\lambda^i(k) = \frac{\tilde{\lambda}^i(k)}{\sum_{i=1}^N \tilde{\lambda}^i(k)} \tag{5.4.19}$$

Finally, particle filters based on Sequential Importance Resampling (SIR) construct the ensemble $\{x^i(k)\}_{i=1}^N$ at time k by drawing samples from:

$$\sum_{i=1}^{N} \lambda^i(k) \delta \left(x(k) - x^i(k \mid k-1) \right) \approx p(x(k) \mid k) \tag{5.4.20}$$

The corresponding weights are then set to be uniform, i.e. $\lambda^i(k) = \frac{1}{N}\ \forall i = 1, \ldots, N$. By comparing (5.4.20) with (5.4.5), we observe that the weights $\lambda^i(k)$ encode information about the likelihood $p(y(k) \mid x(k))$, while the Dirac delta centered at $x^i(k \mid k-1)$ captures the prior information $p(x(k) \mid k-1)$. Given the weighted ensemble $\{x^i(k), \lambda^i(k)\}_{i=1}^N$, the first- and second-order statistics can be estimated analogously to the EnKF:

$$\hat{x}_{\mathrm{PF}}(k) = \sum_{i=1}^{N} w^i(k) x^i(k) \tag{5.4.21}$$
$$P_{\mathrm{PF}}(k) = \sum_{i=1}^{N} w^i(k) \left(x^i(k) - \hat{x}_{\mathrm{PF}}(k) \right) \left(x^i(k) - \hat{x}_{\mathrm{PF}}(k) \right)^\top \tag{5.4.22}$$

Audio tracker

We now present an application of particle filters to a tracking problem. The general algorithm was originally introduced in [135, 131], where a standard visual tracker employs an NMF-based likelihood to track and recursively update a low-rank signature of a target object. Here, we adapt this approach to real-time tracking in audio signals. The key idea is to interpret consecutive column batches of a spectrogram as video frames. Before presenting the proposed algorithm, we briefly introduce the general tracking problem and review the original NMF tracker.
In visual tracking, we are given a real-time stream of video frames $F(1), \dots, F(k), \dots$ where each $F(k) \in \mathcal{M}_{w\times h}(\mathbb{R}_{\geq 0})$ is a *grayscale* image containing the target object. We adopt a state-space formulation of the form (5.4.7), with the following components:

- the state $x(k) \in \mathbb{R}^4$ encodes the object xy-position and its size (width and height) within the frame;
- the forward model f propagates the state by additive Gaussian perturbations, i.e.:

$$x(k) = f(x(k-1), w(k), k) = x(k-1) + w(k) \tag{5.4.23}$$

  with $w(k) \sim \mathcal{N}(0, \sigma^2 \mathbb{1}_4)$;
- the observation model h is deterministic and extracts from the current frame $F(k)$ the *vectorized* image patch $y(k) \in \mathbb{R}^m_{\geq 0}$ corresponding to the state $x(k)$, i.e.:

$$y(k) = h(x(k), k) \tag{5.4.24}$$

With a slight abuse of notation, we will also denote the observed frames $F(k)$ by $y(k)$, even though they are $w \times h$ matrices and not outputs of the observation model h. This notational simplification does not affect the particle filtering procedure and will be clarified in the following discussion.
With respect to the general particle filter framework described in the previous section, the only remaining component to be specified is the likelihood associated with a given particle observation, namely the quantity $p\big(y(k) \mid y^i(k \mid k-1)\big)$ used to update the weights in (5.4.18). Let us suppose to be at time $k-1$ and denote:

$$\hat{y}_{\mathrm{PF}}(l) = h(\hat{x}_{\mathrm{PF}}(l), l) \qquad \forall l = 1, \dots, k-1 \tag{5.4.25}$$

the vectorized patches computed at all previous times by the filter. Assembling these observations within a nonnegative matrix:

$$\hat{Y}(k-1) = \begin{bmatrix} \hat{y}_{\mathrm{PF}}(1) & \cdots & \hat{y}_{\mathrm{PF}}(k-1) \end{bmatrix} \tag{5.4.26}$$

we extract a nonnegative, low-rank signature dictionary $W(k-1)$ by solving the following NMF optimization problem:

$$\begin{aligned} \min \quad & \|\hat{Y}(k-1) - W(k-1)H\|_2^2 \\ \text{s.t.} \quad & W(k-1) \in \mathcal{M}_{m\times r}(\mathbb{R}_{\geq 0}) \\ & H \in \mathcal{M}_{r\times(k-1)}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{5.4.27}$$

The underlying rationale is that $W(k-1)$ contains feature vectors encoding the appearance history of the tracked object until frame $F(k-1)$. Under the assumption of a smooth temporal evolution of the object's appearance, a standard and well-justified assumption in visual tracking, these features can be leveraged to localize and track the object in the subsequent frame $y(k) = F(k)$. More precisely, given $y(k)$, we measure the fitness of the i-th particle observation $y^i(k \mid k-1)$[3] by solving:

$$\begin{aligned} \min \quad & \|y^i(k \mid k-1) - W(k-1)H\|_2^2 \\ \text{s.t.} \quad & H \in \mathcal{M}_{r\times 1}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{5.4.28}$$

Denoting $H^i(k \mid k-1)$ the optimal solution of (5.4.28), the residue:

$$r^i(k \mid k-1) = \|y^i(k \mid k-1) - W(k-1)H^i(k \mid k-1)\|_2^2 \tag{5.4.29}$$

is inversely proportional to the fitness of the particle. As a consequence, for a constant $C > 0$, we define the likelihood as:

$$p(y(k) \mid y^i(k \mid k-1)) \propto e^{-Cr^i(k|k-1)} \tag{5.4.30}$$

In practice, the dictionary $W(k)$ at the next filter iteration can be efficiently obtained by updating $W(k-1)$, without explicitly forming the observation matrix $\hat{Y}(k)$. This trick ensures that the memory footprint of the algorithm remains constant over time. Finally, for clarity of presentation, the objective functions defining the two NMF problems in (5.4.27) and (5.4.30) have been simplified. A complete description of the regularized NMF formulations used in the actual tracker implementation can be found in [135].
We now describe how this general framework can be adapted to tracking tasks in audio signals. The objective is to modify the particle filter so as to follow continuous time-frequency patterns in the spectrogram of an audio signal. Such patterns typically arise from nonstationary fundamental frequencies associated with excited systems, such as rotating machinery, car engines, or musical instruments. Figure 5.1 shows a typical time-frequency pattern observed from the spectrogram of an accelerating car engine. Moreover, the particle filter should be able to track an arbitrary number of harmonics simultaneously, as these often carry valuable information about the underlying system. As discussed earlier, an audio signal acquired in real time is represented by treating consecutive batches of columns of its spectrogram as video frames, with the batch size chosen a priori. Within each batch, the objective is to track n frequency components: the fundamental frequency and its first $n-1$ harmonics. We again adopt the state-space formulation (5.4.7), where:

- the state $x(k) \in \mathbb{R}^n$ represents the n target frequencies. Unlike the visual tracking case, no bounding box width or height is estimated: the width is fixed by the number of columns in each spectrogram batch, while the height (in pixels) associated with each tracked frequency is selected during initialization and kept constant. This reflects the assumption that the shape of the system's frequency response remains approximately constant or, at least, bounded;

- the forward model f propagates the state while taking into account that the information it carries is intrinsically one-dimensional. In fact, given a perfect state estimate $x(k-1)$,

[3]Notice that computing $y^i(k \mid k-1)$ by (5.4.24) requires $F(k) = y(k)$.

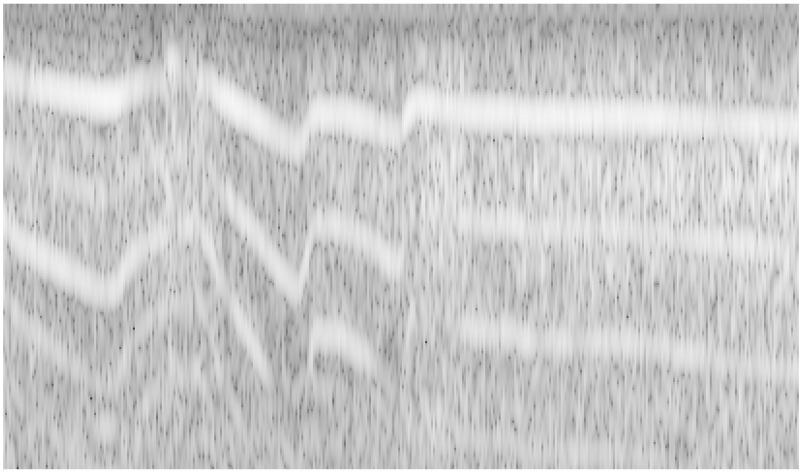


Figure 5.1: Spectrogram of an accelerating car engine. The fundamental and two of its harmonics can be easily recognized.

any entry can be recovered from any other by multiplication with a suitable rational factor. The proposed forward model is constructed by pre-multiplying (5.4.23) with the following rank-1 matrix A:

$$x(k) = f(x(k-1), w(k), k) = A\,(\,x(k-1) + w(k)\,) \qquad A = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ n \end{bmatrix} \begin{bmatrix} \alpha_1 & \frac{\alpha_2}{2} & \cdots & \frac{\alpha_n}{n} \end{bmatrix} \tag{5.4.31}$$

where $\alpha_1 + \cdots + \alpha_n = 1$ and each $\alpha_j \geq 0$ represents the confidence in the estimate of the $(j-1)$-th harmonic frequency at time $k-1$;

- the observation model h is deterministic and extracts from the current frame $F(k)$ the set of n *vectorized* image patches $y(k) = \{y_j(k)\}_{j=1}^n \subseteq \mathbb{R}^m$ corresponding to the entries of $x(k)$, i.e.:

$$y_j(k) = h(x_j(k), k) \qquad \forall j = 1, \ldots, n \tag{5.4.32}$$

The likelihood for the filter is constructed by generalizing the above procedure to n distinct objects, each having its own NMF dictionary. In particular, let us suppose to be at time $k-1$ and denote, for $j = 1, \ldots, n$:

$$(\hat{y}_{\mathrm{PF}})_j(l) = h\big((\hat{x}_{\mathrm{PF}})_j(l), l\big) \qquad \forall l = 1, \ldots, k-1 \tag{5.4.33}$$

the vectorized patches computed at all previous times by the filter. We then assemble these observations within n nonnegative matrices:

$$\hat{Y}_j(k-1) = \big[\,(\hat{y}_{\mathrm{PF}})_j(1) \quad \cdots \quad (\hat{y}_{\mathrm{PF}})_j(k-1)\,\big] \qquad \forall j = 1, \ldots, n \tag{5.4.34}$$

and extract the corresponding dictionaries $W_j(k-1)$ by solving the NMF optimization problems:

$$\begin{aligned} \min \quad & \|\hat{Y}_j(k-1) - W_j(k-1)H\|_2^2 \\ \text{s.t.} \quad & W_j(k-1) \in \mathcal{M}_{m\times r}(\mathbb{R}_{\geq 0}) \\ & H \in \mathcal{M}_{r\times(k-1)}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{5.4.35}$$

As before, given $y(k)$, we measure the fitness of the i-th particle observation $y^i(k \mid k-1) = \{y_j^i(k \mid k-1)\}_{j=1}^n$ by solving, for $j = 1, \ldots, n$:

$$\begin{aligned} \min \quad & \|y_j^i(k \mid k-1) - W_j(k-1)H\|_2^2 \\ \text{s.t.} \quad & H \in \mathcal{M}_{r\times 1}(\mathbb{R}_{\geq 0}) \end{aligned} \tag{5.4.36}$$

Denoting $H_j^i(k \mid k-1)$ the optimal solution of (5.4.36) and the residue:

$$r_j^i(k \mid k-1) = \|y_j^i(k \mid k-1) - W_j(k-1)H_j^i(k \mid k-1)\|_2^2 \qquad \forall j = 1, \ldots, n \tag{5.4.37}$$

we define the likelihood as:

$$p(y(k) \mid y^i(k \mid k-1)) \propto e^{-\frac{C}{n}\sum_{j=1}^{n} r_j^i(k|k-1)} \tag{5.4.38}$$

In particular, the confidence weights $\alpha_1, \ldots, \alpha_n$ to be used in the next filter iteration can be constructed by normalizing $\{e^{-C\hat{r}_j(k)}\}_{j=1}^n$, where $\{\hat{r}_j(k)\}_{j=1}^n$ are analogous residues to (5.4.37) computed from $(\hat{y}_{\text{PF}})_j(k) = h\big((\hat{x}_{\text{PF}})_j(k), k\big)$ using the updated dictionaries $W_j(k)$. To conclude, Figure 5.2 shows the results obtained by applying the proposed audio tracker on the spectrogram of an accelerating car. We chose to track 2 harmonics ($n = 3$) using $N = 50$ particles. Each column batch has a size of 3, and we set $r = 15$ and $C = 3 \times 10^{-1}$.

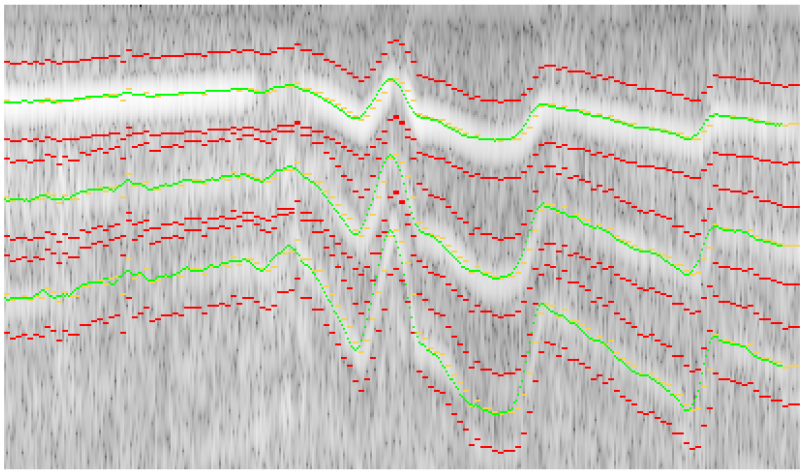


Figure 5.2: Output of the proposed audio tracker on a spectrogram segment of an accelerating car. Red boxes indicate the fixed-size regions associated with each tracked frequency, the yellow lines show the estimated frequencies, and the green line corresponds to a smoothed version of the frequency estimate.

Appendix

5.A Supporting results

Proposition 5.A.1. *Let $A \in \mathcal{M}_{n\times n}(\mathbb{R})$, $B \in \mathcal{M}_{m\times n}(\mathbb{R})$ and $C \in \mathcal{M}_{m\times m}(\mathbb{R})$ with A, C positive definite. Denoting:*

$$S = BAB^\top + C^{-1} \qquad G = AB^\top S^{-1}$$

it holds:

1. $\left(A^{-1} + B^\top CB\right)^{-1} = (\mathbb{1}_n - GB)A$;
2. $\left(A^{-1} + B^\top CB\right)^{-1} = (\mathbb{1}_n - GB)A(\mathbb{1}_n - B^\top G^\top) + GC^{-1}G^\top$;
3. $G = \left(A^{-1} + B^\top CB\right)^{-1} B^\top C$.

Proof: See [67].

Proposition 5.A.2 (Schur)**.** *Let $A \in \mathcal{M}_{n\times n}(\mathbb{R})$, $D \in \mathcal{M}_{m\times m}(\mathbb{R})$, $B \in \mathcal{M}_{n\times m}(\mathbb{R})$, $C \in \mathcal{M}_{m\times n}(\mathbb{R})$ and denote M the matrix with block form:*

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

1. If D and the Schur complement $M_A = A - BD^{-1}C$ are invertible, then it holds:

$$M^{-1} = \begin{bmatrix} M_A^{-1} & -M_A^{-1}BD^{-1} \\ -D^{-1}CM_A^{-1} & D^{-1} + D^{-1}CM_A^{-1}BD^{-1} \end{bmatrix}$$

2. If A and the Schur complement $M_D = D - CA^{-1}B$ are invertible, then it holds:

$$M^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BM_D^{-1}CA^{-1} & -A^{-1}BM_D^{-1} \\ -M_D^{-1}CA^{-1} & M_D^{-1} \end{bmatrix}$$

Proof: It can be easily checked by direct computation.

Proposition 5.A.3 (Woodbury)**.** *Let $A \in \mathcal{M}_{n\times n}(\mathbb{R})$, $D \in \mathcal{M}_{m\times m}(\mathbb{R})$, $B \in \mathcal{M}_{n\times m}(\mathbb{R})$ and $C \in \mathcal{M}_{m\times n}(\mathbb{R})$. If A, D, $A - BD^{-1}C$ and $D - CA^{-1}B$ are invertible, then it holds:*

$$\left(A - BD^{-1}C\right)^{-1} = A^{-1} + A^{-1}B\left(D - CA^{-1}B\right)^{-1}CA^{-1}$$

Proof: Follows by equating the upper-left blocks of M^{-1} in 5.A.2.

5.B Details for Section 5.3.1

Following [67], we now show how to obtain the recursive predictor-corrector update rule typical of the Kalman filter. Let us begin by noticing that the objective defining the least squares problem with solution $\hat{z}(k \mid l)$ (5.3.5) can be written as:

$$\begin{aligned} F_{k|l}(z(k)) &= \frac{1}{2}\left\|x(0)-\mu_0\right\|^2_{Q(0)^{-1}} + \frac{1}{2}\sum_{i=1}^{k}\left\|y(i)-C(i)x(i)\right\|^2_{R(i)^{-1}} \\ &+ \frac{1}{2}\sum_{i=1}^{k}\left\|x(i)-A(i)x(i-1)-B(i)u(i)\right\|^2_{Q(i)^{-1}} \end{aligned} \tag{5.B.1}$$

In the prediction step, we seek $\hat{x}(k \mid k-1)$, which is defined as the last component of $\hat{z}(k \mid k-1)$. The latter, is associated to the objective in (5.B.1) for $l = k-1$, which we can recursively write in function of $F_{k-1} = F_{k-1|k-1}$:

$$F_{k|k-1}(z(k)) = F_{k-1}(z(k-1)) + \frac{1}{2}\left\|x(k)-\mathcal{A}(k)z(k-1)-B(k)u(k)\right\|^2_{Q(k)^{-1}} \tag{5.B.2}$$

where $\mathcal{A}(k) = \begin{bmatrix} 0 & \cdots & 0 & A(k) \end{bmatrix} \in \mathcal{M}_{n\times kn}(\mathbb{R})$, so that $\mathcal{A}(k)z(k-1) = A(k)x(k-1)$. In particular, the gradient and Hessian for $F_{k|k-1}$ are in the form:

$$\begin{aligned} \nabla_{z(k)}F_{k|k-1}(z(k)) &= \begin{bmatrix} \nabla_{z(k-1)}F_{k-1}(z(k-1)) + \mathcal{A}(k)^\top Q(k)^{-1}\left(\mathcal{A}(k)z(k-1)-x(k)+B(k)u(k)\right) \\ -Q(k)^{-1}\left(\mathcal{A}(k)z(k-1)-x(k)+B(k)u(k)\right) \end{bmatrix} \\ \nabla^2_{z(k)}F_{k|k-1}(z(k)) &= \begin{bmatrix} \nabla^2_{z(k-1)}F_{k-1}(z(k-1)) + \mathcal{A}(k)^\top Q(k)^{-1}\mathcal{A}(k) & -\mathcal{A}(k)^\top Q(k)^{-1} \\ -Q(k)^{-1}\mathcal{A}(k) & Q(k)^{-1} \end{bmatrix} \end{aligned} \tag{5.B.3}$$

Since the Hessian is positive definite, we can obtain $\hat{z}(k \mid k-1)$ by applying a single step of Newton's method with any starting point $z(k)$:

$$\hat{z}(k \mid k-1) = z(k) - \left(\nabla^2_{z(k)}F_{k|k-1}(z(k))\right)^{-1}\nabla_{z(k)}F_{k|k-1}(z(k)) \tag{5.B.4}$$

Choosing:

$$z(k) = \begin{bmatrix} \hat{z}(k-1) \\ A(k)\hat{x}(k)+B(k)u(k) \end{bmatrix} \tag{5.B.5}$$

we observe that $\nabla_{z(k)}F_{k|k-1}(z(k)) = 0$. By virtue of (5.B.4), we get $\hat{z}(k \mid k-1) = z(k)$, and $\hat{x}(k \mid k-1)$ is then given by the last component of (5.B.5), recovering the Kalman filter state prediction in (5.3.7). If we then recall that the inverse of the Hessian in (5.B.3) defines the state covariances through the following relation:

$$\left(\nabla^2_{z(k)}F_{k|k-1}(z(k))\right)^{-1} = \begin{bmatrix} \operatorname{cov}\left[z(k-1), z(k-1) \mid k-1\right] & * \\ * & P(k \mid k-1) \end{bmatrix} \tag{5.B.6}$$

by observing that:

$$\mathcal{A}(k)\,\mathrm{cov}\left[z(k-1), z(k-1) \mid k-1\right]\mathcal{A}(k)^\top = A(k)P(k-1)A(k)^\top \tag{5.B.7}$$

Proposition 5.A.2-1 applied to $\nabla^2_{z(k)} F_{k|k-1}(z(k))$ yields the predicted state covariance in (5.3.6).

In the correction step, we seek $\hat{x}(k)$, which is defined as the last component of $\hat{z}(k) = \hat{z}(k \mid k)$. Its associated objective is obtained by letting $l = k$ in (5.B.1), and can be recursively written as a function of $F_{k|k-1}$:

$$F_k(z(k)) = F_{k|k-1}(z(k)) + \frac{1}{2}\left\|y(k) - \mathcal{C}(k)z(k)\right\|^2_{R(k)^{-1}} \tag{5.B.8}$$

where $\mathcal{C}(k) = \begin{bmatrix}0 & \cdots & 0 & C(k)\end{bmatrix} \in \mathcal{M}_{m\times(k+1)n}(\mathbb{R})$, so that $\mathcal{C}(k)z(k) = C(k)x(k)$. The gradient and Hessian for F_k are in the form:

$$\begin{aligned}
\nabla_{z(k)} F_k(z(k)) &= \nabla_{z(k)} F_{k|k-1}(z(k)) + \mathcal{C}(k)^\top R(k)^{-1}\left(\mathcal{C}(k)z(k) - y(k)\right) \\
&= \nabla_{z(k)} F_{k|k-1}(z(k)) + \begin{bmatrix} 0 \\ C(k)^\top R(k)^{-1}\left(C(k)x(k) - y(k)\right)\end{bmatrix} \\
\nabla^2_{z(k)} F_k(z(k)) &= \nabla^2_{z(k)} F_{k|k-1}(z(k)) + \mathcal{C}(k)^\top R(k)^{-1}\mathcal{C}(k) \\
&= \nabla^2_{z(k)} F_{k|k-1}(z(k)) + \begin{bmatrix} 0 & 0 \\ 0 & C(k)^\top R(k)^{-1}C(k)\end{bmatrix}
\end{aligned} \tag{5.B.9}$$

Since the Hessian is positive definite, we again apply a single step of Newton's method:

$$\hat{z}(k) = z(k) - \left(\nabla^2_{z(k)} F_k(z(k))\right)^{-1} \nabla_{z(k)} F_k(z(k)) \tag{5.B.10}$$

Choosing again $z(k) = \hat{z}(k \mid k-1)$ as in (5.B.5), the gradient reduces to:

$$\nabla_{z(k)} F_k(z(k)) = \begin{bmatrix} 0 \\ C(k)^\top R(k)^{-1}\left(C(k)\hat{x}(k \mid k-1) - y(k)\right)\end{bmatrix} \tag{5.B.11}$$

As before, since:

$$\left(\nabla^2_{z(k)} F_k(z(k))\right)^{-1} = \begin{bmatrix} \mathrm{cov}\left[z(k-1), z(k-1) \mid k\right] & * \\ * & P(k)\end{bmatrix} \tag{5.B.12}$$

by combining (5.B.10), (5.B.11) and (5.B.12), we recover the Kalman filter state correction in (5.3.10). Lastly, by Proposition 5.A.3, one obtains the following relation:

$$\mathcal{A}(k)\left(\nabla^2_{z(k-1)} F_{k-1}(z(k-1)) + \mathcal{A}(k)^\top Q(k)^{-1}\mathcal{A}(k)\right)^{-1}\mathcal{A}(k)^\top = Q(k) - Q(k)P(k \mid k-1)^{-1}Q(k) \tag{5.B.13}$$

Applying Proposition 5.A.2-2 to $\nabla^2_{z(k)} F_k(z(k))$, together with (5.B.13), yields the explicit form for the corrected state covariance in (5.3.9).

Deep Kalman filtering

This chapter introduces learning-enhanced data assimilation, focusing on deep Kalman filtering as a unifying framework. We review conditional diffusion models and their role within filtering algorithms, and present KalmanNet and the Deep Kalman Filter as structured, deep-unfolded generalizations of classical Kalman filtering.

6.1 Introduction

Data assimilation addresses the problem of estimating the evolving state of a dynamical system by combining partial, noisy observations with a predictive model of the system dynamics. Classical filtering methods, such as the Kalman filter and its nonlinear and ensemble-based extensions, provide principled Bayesian solutions to this problem under specific assumptions on the system structure and uncertainty statistics. As reviewed in Chapter 5, the Kalman filter yields an optimal recursive estimator for linear systems with Gaussian noise, while extensions such as the Extended Kalman Filter, the Unscented Kalman Filter, the Ensemble Kalman Filter, and particle filters progressively relax assumptions on linearity, Gaussianity, and dimensionality at the cost of additional approximations or computational burden [72, 40, 36, 109]. Despite their success across a wide range of scientific and engineering applications, these methods remain fundamentally model-driven and rely on accurate knowledge of the system dynamics, observation operators, and noise statistics. In many real-world scenarios, however, these assumptions are difficult to satisfy. Forward models may be only partially known, affected by modeling errors, or computationally expensive to evaluate; observation operators may be nonlinear or implicitly defined; and noise statistics may be unknown, time-varying, or strongly non-Gaussian. These challenges have motivated increasing interest in learning-enhanced data assimilation, a paradigm that augments classical filtering methods with data-driven components learned from observations or simulations. Rather than replacing model-based approaches altogether, learning-enhanced methods aim to complement and extend them, preserving their recursive structure and interpretability while leveraging the expressive power of modern machine learning to compensate for model inadequacies.

The integration of learning into data assimilation raises a number of conceptual and practical questions. From a probabilistic perspective, one must determine which components of the filtering pipeline should be learned and how learned models interact with uncertainty propagation. From an algorithmic standpoint, it is crucial to retain stability, robustness, and scalability, particularly in sequential and high-dimensional settings. From a methodological viewpoint, learning-enhanced filtering methods must strike a balance between

data efficiency, generalization, and adherence to physical or statistical constraints. These considerations have led to a broad spectrum of hybrid approaches, ranging from learning parametric corrections to classical filters, to fully data-driven inference mechanisms embedded within recursive estimation frameworks [13, 109]. Within this broader landscape, learning-enhanced data assimilation can be understood as part of a more general trend that blurs the traditional boundary between model-based and data-driven methods. Rather than viewing learning and filtering as separate stages, modern approaches increasingly treat filtering itself as a trainable computational process, optimized end-to-end using data. In this sense, learning-enhanced filtering can be seen as a natural extension of classical data assimilation, adapting its core principles to the realities of modern data-rich but model-imperfect environments.

This chapter focuses on Deep Kalman Filtering, understood broadly as the incorporation of learned components into Kalman-style recursive estimators. Building on the classical filtering framework introduced in Chapter 5, we examine how learning can be used to relax key assumptions underlying traditional methods, while maintaining their recursive structure and probabilistic interpretation. Rather than advocating a single universal solution, the goal of this chapter is to highlight different design philosophies and illustrate how learning-enhanced data assimilation can be realized in practice through complementary approaches. We begin by considering learning-based generative models for filtering, with particular emphasis on diffusion models [30, 121, 120]. In contrast to particle filters, which require explicit evaluation of the likelihood to update particle weights, diffusion-based approaches enable sampling from posterior distributions without requiring a closed-form likelihood. By learning to represent conditional distributions directly, these models provide a scalable and flexible alternative for state estimation in complex, high-dimensional, or implicitly defined systems. From a data assimilation perspective, diffusion models can be interpreted as learning-enhanced filtering algorithms that replace explicit Bayesian updates with learned generative mechanisms. We then turn to KalmanNet [110], a neural-network-aided filtering architecture that extends the Kalman filter to settings with nonlinear dynamics and unknown noise statistics. KalmanNet replaces analytically derived gain computations with learned recurrent modules, enabling the filter to adapt to complex system behavior while retaining the predictor-corrector structure of classical Kalman filtering. This approach exemplifies how learning can be introduced selectively into the filtering pipeline, targeting the most restrictive modeling assumptions without abandoning the underlying recursive estimation framework. Finally, we explore the Deep Kalman Filter [21, 23], which further generalizes Kalman filtering by allowing the forward operator itself to be learned from data. In this setting, learning is not limited to auxiliary components such as gains or covariances, but extends to the system dynamics, which are embedded within a structured filtering architecture. The training process is guided by statistical principles inherited from classical filtering theory, in particular through objectives that enforce the whiteness of the prediction error. This careful balance between learned representations and model-based regularization exemplifies the core philosophy of learning-enhanced data assimilation: combining flexibility with structure to achieve robust and interpretable state estimation. As will be shown, KalmanNet and the Deep Kalman Filter constitute representative examples of the deep unfolding paradigm presented in Chapter 1, which underpins the conceptual framework of this dissertation.

6.2 Conditional diffusion models

One of the main limitations of particle filters, presented in Section 5.4.2, is their reliance on a known likelihood function $p(y(k) \mid x(k))$ to update the particle weights in (5.4.18). When such a likelihood is unavailable or intractable, conditional diffusion models provide a principled alternative to classical particle filtering. By learning to generate samples directly from the conditional posterior $p(x(k) \mid k)$, these models eliminate the need for explicit likelihood evaluation while preserving the ability to represent complex, high-dimensional, and potentially multimodal uncertainty. The purpose of this section is to briefly review the mathematical foundations of conditional diffusion models, with particular emphasis on denoising score matching, and to illustrate how these methods can be integrated within filtering iterations. For the theoretical background, we loosely follow [30], which builds upon earlier seminal works [121, 120].

Let $p_{\text{tgt}}(x \mid y)$ denote a target conditional distribution of interest. A key observation behind diffusion models is that, independently of the specific form of p_{tgt}, propagating this distribution through a drift-diffusion process for a sufficiently long time results in a distribution that is arbitrarily close to a Gaussian. More precisely, denoting $p_t(x \mid y)$ a time-dependent distribution satisfying:

$$\begin{cases} \dfrac{\partial p_t(x \mid y)}{\partial t} &= \dfrac{b(t)}{2}\nabla_x \cdot (x p_t(x \mid y)) + \dfrac{g(t)}{2}\Delta_x p_t(x \mid y) \qquad \forall t > 0 \\ p_0(x \mid y) &= p_{\text{tgt}}(x \mid y) \end{cases} \tag{6.2.1}$$

for $T \gg 0$, it holds $p_T(x \mid y) \approx \mathcal{N}(0, \sigma^2(T)\mathbb{1}_n)$. We note that (6.2.1) encompasses both the variance-exploding and variance-preserving diffusion formulations [121]. For a concise summary of the corresponding definitions of $b(t)$, $g(t)$, $\sigma^2(t)$, and $m(t)$, the reader is referred to Table 1. As their names suggest, the primary distinction between the two formulations lies in the asymptotic behavior of $\sigma^2(t)$, which denotes the variance of the Gaussian distribution approximating $p_t(x \mid y)$ in the limit. As can be seen from Table 1, in the variance-exploding formulation one has $\lim_{t\to+\infty} \sigma^2(t) = +\infty$, whereas in the variance-preserving case the variance remains bounded, i.e., $\lim_{t\to+\infty} \sigma^2(t) < +\infty$.

The system in (6.2.1) defines the *forward* diffusion process, which progressively transforms an arbitrary target distribution into a Gaussian. To generate samples from p_{tgt}, the key idea is to reverse this diffusion in time, starting from samples drawn from a simple Gaussian distribution and mapping them back to the more complex target distribution. Introducing the time-reversal variable $\tau = T - t$ and defining $\tilde{p}_\tau(x \mid y) = p_{T-\tau}(x \mid y)$, the corresponding *reverse* process can be readily derived [30]:

$$\begin{cases} \dfrac{\partial \tilde{p}_\tau(x \mid y)}{\partial \tau} &= -\nabla_x \cdot \left(\left(\dfrac{b(t)}{2}x + \dfrac{(1+\alpha)g(t)}{2} s_t(x \mid y) \right) \tilde{p}_\tau(x \mid y) \right) + \dfrac{\alpha g(t)}{2}\Delta_x \tilde{p}_\tau(x \mid y) \quad \forall \tau \in (0, T] \\ \tilde{p}_0(x \mid y) &= \mathcal{N}(0, \sigma^2(T)) \end{cases} \tag{6.2.2}$$

where $\alpha \geq 0$ and:

$$s_t(x \mid y) = \nabla_x \log p_t(x \mid y) \tag{6.2.3}$$

Table 1: Definitions of the scalar quantities associated to the drift-diffusion equation (6.2.1). In the variance-preserving formulation, $\mu > 0$ is an hyperparameter that controls $\sigma^2(t)$, and is usually set to $\mu = 2$ [30].

Quantity	Variance exploding	Variance preserving
$b(t)$	0	$\beta(t)$
$g(t)$	$\gamma(t)$	$\beta(t)\left(1 - e^{-\frac{\mu}{2}\int_0^t \beta(s)\,\mathrm{d}s}\right)^{\frac{2}{\mu}-1}$
$m(t)$	1	$e^{-\frac{1}{2}\int_0^t \beta(s)\,\mathrm{d}s}$
$\sigma^2(t)$	$\int_0^t \gamma(s)\,\mathrm{d}s$	$(1 - m^\mu(t))^{\frac{2}{\mu}}$
User inputs	$\gamma(t) > 0$	$\beta(t) > 0$

is the *score function*. This quantity effectively links the reverse diffusion process in (6.2.2) to the forward process in (6.2.1), ensuring that at $\tau = T$ the solution of the reverse process satisfies $\tilde{p}_T(x \mid y) = p_{\mathrm{tgt}}(x \mid y)$.
Let us now assume that the score function $s_t(x \mid y)$ is known. By inspection of (6.2.2), the corresponding drift $\mu(x, t; y)$ and diffusion coefficient $D(t)$ are given by

$$\begin{aligned} \mu(x,t;y) &= \frac{b(t)}{2}x + \frac{(1+\alpha)g(t)}{2}s_t(x \mid y) \\ D(t) &= \frac{\alpha g(t)}{2} \end{aligned} \tag{6.2.4}$$

As a result, equation (6.2.2) can be interpreted as the Fokker-Planck equation associated with the following Itô stochastic differential equation (SDE) [111, 127]:

$$\begin{cases} \mathrm{d}x_\tau &= \mu(x,t;y)\,\mathrm{d}\tau + \sqrt{2D(t)}\,\mathrm{d}w_\tau \qquad \forall \tau \in (0,T] \\ x_0 &\sim \mathcal{N}\left(0, \sigma^2(T)\right) \end{cases} \tag{6.2.5}$$

where w_τ denotes an n-dimensional Wiener process. This particle-level formulation of the drift-diffusion equation in (6.2.2) guarantees that at $\tau = T$ the stochastic process satisfies $x_T \sim p_{\mathrm{tgt}}(x \mid y)$. To generate samples from $p_{\mathrm{tgt}}(x \mid y)$, the Itô SDE in (6.2.5) can be discretized using, for example, the Euler-Maruyama method:

$$x_{\tau+\Delta\tau} = x_\tau + \mu(x_\tau, t; y)\Delta\tau + \sqrt{2D(t)\Delta\tau}z \qquad \forall \tau = 0, \Delta\tau, \ldots, T - \Delta\tau \tag{6.2.6}$$

where z is independently drawn from $\mathcal{N}(0, \mathbb{1}_n)$ at each discretization step. The procedure described above relies on knowledge of the score function $s_t(x \mid y)$, which in turn requires solving (6.2.1). However, the initial condition of this forward diffusion involves the very target distribution from which we ultimately wish to sample, making direct access to $s_t(x \mid y)$ infeasible. Conditional diffusion models address this circular dependency by replacing the true score $s(x, t, y) = s_t(x \mid y)$ with a *learned* surrogate $s_\theta(x, t, y)$. This substitution yields the surrogate drift term:

$$\mu_\theta(x,t;y) = \frac{b(t)}{2}x + \frac{(1+\alpha)g(t)}{2}s_\theta(x,t,y) \tag{6.2.7}$$

which can then be employed within the Euler-Maruyama discretization in (6.2.6) to generate approximate samples from $p_{\text{tgt}}(x \mid y)$. In order to train the *score network* $s_\theta(t,x,y)$, a common choice of loss function is:

$$\mathcal{L}(\theta) = \int_{\mathbb{R}^m} \left[\int_0^T \int_{\mathbb{R}^n} \|s_\theta(x,t,y) - s(x,t,y)\|_2^2 p_t(x \mid y)\,\mathrm{d}x\,\mathrm{d}t \right] p(y)\,\mathrm{d}y \tag{6.2.8}$$

which can be approximated by a Monte Carlo sum obtained by sampling from the *joint* distribution $p(x,y)$. Specifically, given a set $\{(t^i,x^i,y^i,z^i)\}_{i=1}^N$, where t^i is sampled from $\mathcal{U}(0,T)$, (x^i,y^i) is sampled from $p(x,y)$, and z^i is sampled from $\mathcal{N}(0,\mathbb{1}_n)$, the loss in (6.2.8) can be approximated [30] by the conditional denoising score matching objective:

$$\mathcal{L}(\theta) = \sum_{i=1}^N \|\sigma(t^i)s_\theta(\tilde{x}^i,t^i,y^i) + z^i\|_2^2 \tag{6.2.9}$$

where $\tilde{x}^i = m(t^i)x^i + \sigma(t^i)z^i$.
We conclude this section by illustrating how conditional diffusion models can be embedded within a particle filtering framework. Consider the setting of Section 5.4.2 and assume that, at time $k-1$, an ensemble of N particles $\{x^i(k-1)\}_{i=1}^N$ is available, approximating the filtering distribution $p(x(k-1) \mid k-1)$. Given a new observation $y(k)$, the objective is to generate N particles from the updated posterior $p(x(k) \mid k)$. The procedure can be summarized as follows:

1. As in the classical particle filter, the particles are first propagated through the forward operator and observation model using (5.4.16) and (5.4.17), yielding predicted states $x^i(k \mid k-1)$ and corresponding predicted observations $y^i(k \mid k-1)$;

2. Leveraging the framework introduced above, a score network $s_\theta(x,t,y)$ is then trained using the loss in (6.2.9), treating the pairs $(x^i(k \mid k-1), y^i(k \mid k-1))$ as samples from the joint distribution $p(x(k),y(k))$. The optimized network parameters are denoted by $\hat{\theta}$;

3. Once training is complete, the predicted particles are discarded and a new set of N particles $\{x^i\}_{i=1}^N$ is drawn from the Gaussian distribution $\mathcal{N}(0,\sigma^2(T)\mathbb{1}_n)$. Each particle is then evolved according to the Euler-Maruyama discretization (6.2.6), using the surrogate drift (6.2.7) conditioned on the observation $y(k)$:

$$\begin{cases} x^i_{\tau+\Delta\tau} &= x^i_\tau + \mu_{\hat{\theta}}(x^i_\tau,t;y(k))\Delta\tau + \sqrt{2D(t)\Delta\tau}z \qquad \forall\tau = 0,\Delta\tau,\ldots,T-\Delta\tau \\ x^i_0 &= x^i \end{cases} \tag{6.2.10}$$

   where $z \sim \mathcal{N}(0,\mathbb{1}_n)$ is resampled independently at each step;

4. The updated particles approximating the posterior $p(x(k) \mid k)$ are finally obtained as $\{x^i(k)\}_{i=1}^N$, with $x^i(k) = x_T^i$.

As can be expected, training a score network at every filtering step is computationally demanding and may preclude real-time deployment. A possible strategy to mitigate this issue, together with a detailed discussion of the score network architecture, is presented in Section 6.5.

6.3 KalmanNet

KalmanNet [110] is an RNN-based architecture that extends Kalman filtering to settings where the standard assumptions, namely linear system dynamics and known Gaussian noise statistics, are violated. KalmanNet considers a state-space system in the form:

$$\begin{aligned} x(k) &= f(x(k-1)) + w(k) \\ y(k) &= h(x(k)) + v(k) \end{aligned} \tag{6.3.1}$$

where f and h denote the (possibly nonlinear) forward and observation operators, respectively. The additive noise terms $w(k)$ and $v(k)$ are assumed to be uncorrelated, zero-mean Gaussian, i.e., $w(k) \sim \mathcal{N}(0, Q(k))$ and $v(k) \sim \mathcal{N}(0, R(k))$, with unknown covariance matrices $Q(k)$ and $R(k)$. This state-space system is a particular case of the formulation used for the Ensemble Kalman filter (5.4.7).
For an initial state estimate $\hat{x}(0)$, KalmanNet employs a 2-step prediction-correction procedure akin to the traditional Kalman filter. In particular, for the system in (6.3.1), the prediction step is in the form:

$$\hat{x}(k \mid k-1) = f(\hat{x}(k-1)) \tag{6.3.2}$$

while the correction step is given by:

$$\hat{x}(k) = \hat{x}(k \mid k-1) + \mathcal{K}_G^{(k)} \left(y(k) - h(\hat{x}(k \mid k-1)) \right) \tag{6.3.3}$$

Since classical Kalman theory is not applicable, the gain matrix $\mathcal{K}_G^{(k)}$ is instead computed recursively at each filtering step by an RNN. A high-level overview of the architecture is shown in Figure 6.1.
With respect to its RNN component, KalmanNet supports two distinct parameterizations of the output gain matrix:

1. *One-component parameterization.* The gain matrix $\mathcal{K}_G^{(k)}$ is produced by a single Gated Recurrent Unit (GRU) module, formed by a standard GRU embedded between two fully connected layers. The dimension of the GRU hidden state is $C(n^2+m^2)$, for some integer $C > 0$;

2. *Three-component parameterization.* The gain matrix $\mathcal{K}_G^{(k)}$ is obtained from three interconnected GRU modules, each of which updates a distinct surrogate component of the Kalman gain expressed in (5.3.12). More specifically, the three GRU modules are tasked with computing surrogate representations of $Q(k)$, $P(k \mid k-1)$, and the quantity that reduces to $C(k)P(k \mid k-1)C(k) + R(k)$ in the linear setting. Consequently, their hidden states have dimensions n^2, n^2, and m^2, respectively.

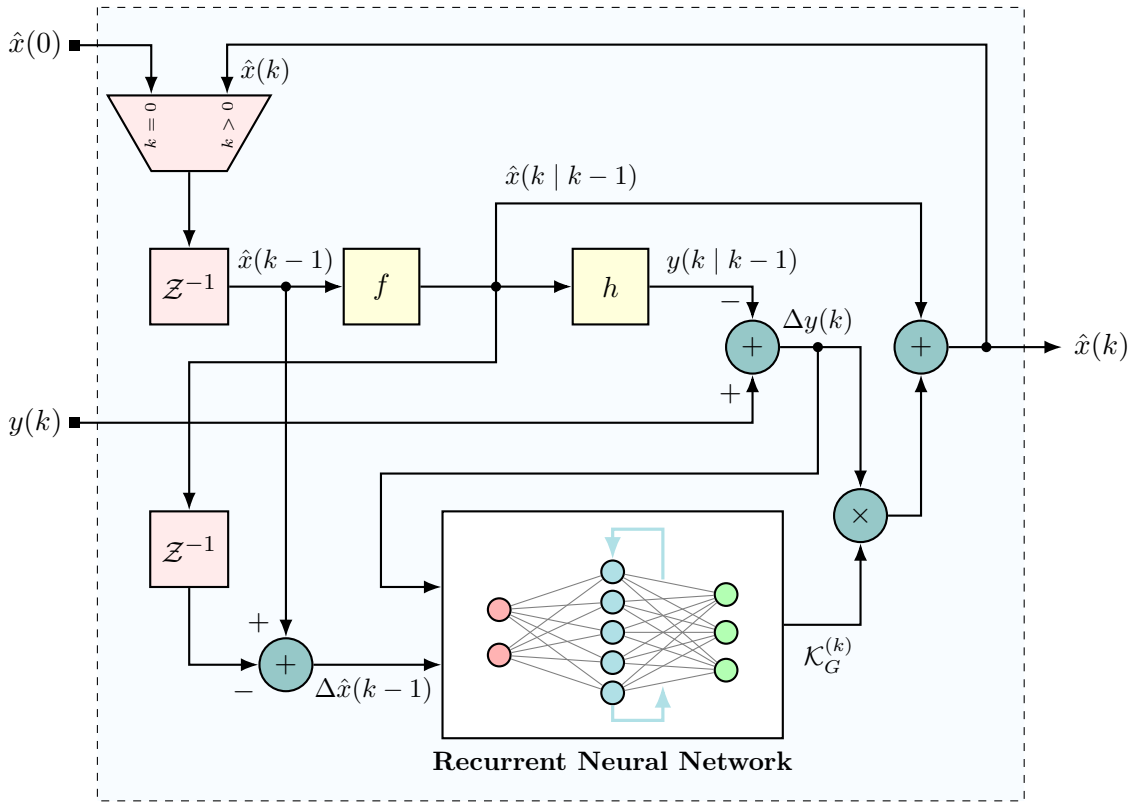


Figure 6.1: KalmanNet architecture. The gain matrix $\mathcal{K}_G^{(k)}$ is computed recursively by an RNN, whose inputs encode the stochastic components of the system (6.3.1). In the displayed architecture, the chosen inputs are the output prediction error $\Delta y(k) = e_y(k)$ at time k, and the correction $\Delta x(k-1)$ at time $k-1$.

The choice of inputs provided to the RNN may also vary. Since the network is tasked with producing the gain matrix, according to either of the two parameterizations introduced above, its inputs must convey information about the stochastic components of the underlying system. KalmanNet supports four such inputs: the observation increment $\Delta\tilde{y}(k) = y(k) - y(k-1)$, the output prediction error $\Delta y(k) = y(k) - y(k \mid k-1)$, the state increment $\Delta\tilde{x}(k) = \hat{x}(k) - \hat{x}(k-1)$, and the state correction $\Delta x(k) = \hat{x}(k) - \hat{x}(k \mid k-1)$. Lastly, we mention that KalmanNet is trained using backpropagation through time, as described in Section 1.2.2. In particular, for a given state trajectory $\{x(k)\}_{k=0}^{K}$, the loss is in the form:

$$\mathcal{L} = \frac{1}{2K} \sum_{k=1}^{K} \|\hat{x}(k) - x(k)\|_2^2 \tag{6.3.4}$$

which is equivalent to (1.2.9) for the particular choice of $k = K = T$. To mitigate numerical instabilities, the trajectory $\{x(k)\}_{k=0}^{K}$ is, in practice, partitioned into multiple sub-trajectories of fixed length T, and the loss in (1.2.11) is adopted.

6.4 Deep Kalman Filter

This section reviews and slightly extends the Deep Kalman Filter architecture originally proposed in [21, 23]. Let us begin by considering a state-space system in the form:

$$\begin{aligned} x(k) &= f(x(k-1), p^{(k)}, u(k)) + w(k) \\ y(k) &= C^{(k)} x(k) + v(k) \end{aligned} \tag{6.4.1}$$

where f is the system's nonlinear forward operator, depending on a possibly time-varying parameter vector $p \in \mathbb{R}^r$. The additive noise terms $w(k)$ and $v(k)$ are assumed only to be uncorrelated. This state-space system lies between the formulations used for the traditional Kalman filter (5.3.1) and the Ensemble Kalman filter (5.4.7). It adopts a linear observation model as in (5.3.1), while extending the forward operator to the nonlinear case considered in (5.4.7). Unlike the EnKF formulation, however, the present model assumes strictly additive process and observation noise.
Given an initial state estimate $\hat{x}(0)$, we consider the iterations of a Kalman-like filter that operates through prediction (5.3.7) and correction (5.3.10) steps. In particular, for the system in (6.4.1), the prediction step is given by:

$$\hat{x}(k \mid k-1) = f(\hat{x}(k-1), p^{(k)}, u(k)) \tag{6.4.2}$$

Similarly, the correction step:

$$\hat{x}(k) = \hat{x}(k \mid k-1) + \mathcal{K}_G^{(k)} \left(y(k) - C^{(k)} \hat{x}(k \mid k-1) \right) \tag{6.4.3}$$

assumes a known gain matrix $\mathcal{K}_G^{(k)}$, analogous to (5.3.11). As with KalmanNet, classical Kalman theory for computing the gains is not applicable. Indeed, the forward operator f is retained in its fully nonlinear form, in contrast to the Extended Kalman filter, which relies on a linearized forward operator. Moreover, the prediction step (6.4.2) is entirely deterministic, unlike in the Ensemble Kalman filter or particle filter. In contrast, the Deep Kalman Filter determines the gain matrices $\mathcal{K}_G^{(k)}$ required in the correction step (6.4.3) via backpropagation-based learning on measured data. More precisely, we consider the network architecture obtained by unrolling the correction updates (6.4.3) for a fixed number K of iterations. The network is shown in Figure 6.2.

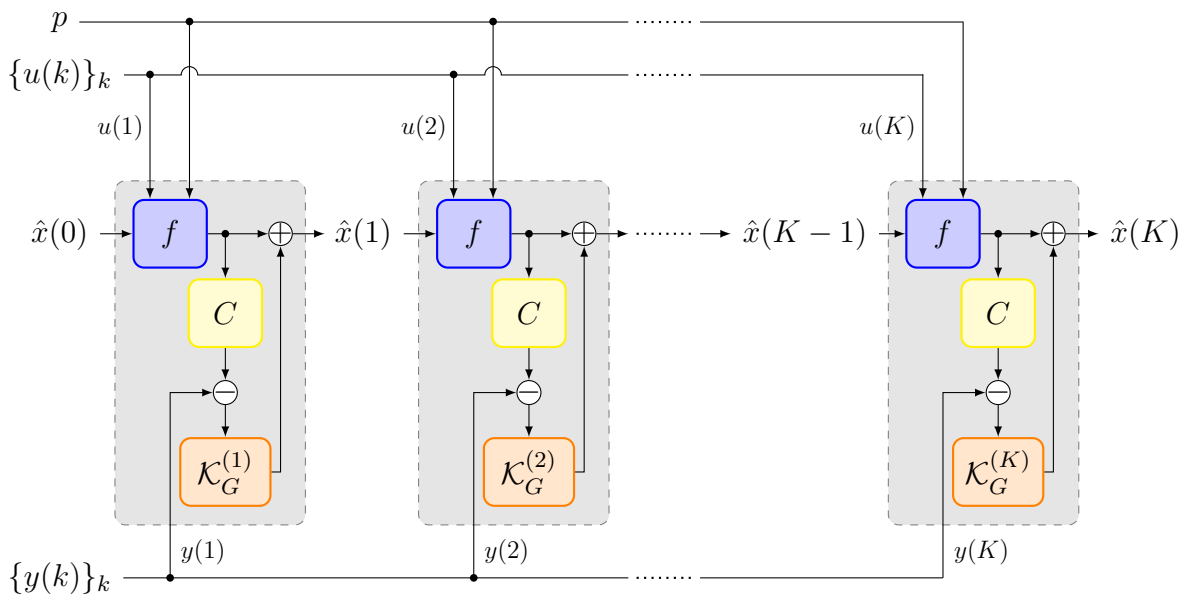


Figure 6.2: Structure of the Deep Kalman Filter network. Each layer is contained in a gray box. The k-th layer receives a state $\hat{x}(k-1)$, observation $y(k)$, input $u(k)$, parameter p, and outputs the new estimated state $\hat{x}(k)$. The predictor component f (blue box) receives $u(k)$ and p to form the predicted state $\hat{x}(k \mid k-1)$ (6.4.2). The corrector component, composed of the observation matrix C (yellow box) and gain matrix $\mathcal{K}_G^{(k)}$ (orange box), receives $y(k)$ and assembles the correction. Predicted state and correction are then summed to form the new state (6.4.3).

In addition to the *untied* gain matrices $\mathcal{K}_G^{(k)}$, this machine-learning reinterpretation of the Kalman filter allows other parameters to be treated as learnable weights. In its most general form, the Deep Kalman Filter treats both $p^{(k)} \equiv p$ and $C^{(k)} \equiv C$ as *shared* weights across layers. Overall, the weights learnable via backpropagation in this architecture include:

- **Gain matrices:** $\{\mathcal{K}_G^{(k)}\}_{k=1}^K$;
- **Forward operator parameters:** $p^{(k)} \equiv p$;
- **Observation model parameters:** $C^{(k)} \equiv C$.

In Section 6.4.1, we will show how the proposed architecture can further optimize the forward operator f in cases where the weights $p^{(k)}$ are insufficient. Moreover, although $p^{(k)}$ and $C^{(k)}$ are shared across layers, the architecture can be readily generalized to accommodate untied parameters. In addition, certain weights can be fixed at predetermined optimal values instead of being learned, as will be the case in most of the numerical experiments of Sections 6.4.4 and 6.4.5. From a high-level perspective, the three sets of weights address uncertainties associated with different components of the reference system (6.4.1). Specifically, the gain matrices $\{\mathcal{K}_G^{(k)}\}_{k=1}^K$ capture stochastic contributions, whereas $p^{(k)}$ and $C^{(k)}$ model deterministic discrepancies. This clear separation provides a level of control over the network architecture that is crucial for maintaining interpretability. In contrast, the KalmanNet architecture [110], briefly reviewed in Section 6.3, does

not differentiate between stochastic and deterministic error sources, instead relying on a single RNN to perform state estimation. This aspect will be discussed in more detail in later sections. Furthermore, as we will see in the following, the Deep Kalman Filter does not require access to the complete state trajectory during training, a requirement that can be restrictive in many applications and potentially infeasible for PDE-based models. Another key feature of the Deep Kalman Filter is that, while the gain matrices are untied and seamlessly integrated within the algorithm unrolling framework described in Section 1.2.1, the shared parameters are updated using the conventional backpropagation through time procedure outlined in Section 1.2.2, thereby combining these two computational paradigms.

6.4.1 Architecture details

Building on the general overview of the previous section, we now examine the Deep Kalman Filter architecture in more detail. In particular, we discuss alternative parameterizations, explain the rationale behind the training loss function by connecting it to the traditional Kalman filter equations, and show how the Deep Kalman Filter can further refine its internal model using established data-driven techniques, such as SINDy.

Different parameterizations for $\mathcal{K}_G^{(k)}$

The most direct approach involves learning the elements of the gain matrices $\mathcal{K}_G^{(k)}$. An alternative network architecture, which imposes additional structure and aligns more closely with the original Kalman filter algorithm, can be obtained by decomposing the gain matrices as follows:

$$\mathcal{K}_G^{(k)} = C_{xy}^{(k)} \left(C_{yy}^{(k)} \right)^{-1} \qquad \forall k = 1, \dots, K \tag{6.4.4}$$

where $C_{xy}^{(k)}$ corresponds to the covariance matrix between the state and the observation, while $C_{yy}^{(k)}$ corresponds to the covariance matrix of the observations, as in (5.3.14). Since the positive definiteness of these matrices should be preserved, this alternative architecture replaces the weights $\{\mathcal{K}_G^{(k)}\}_{k=1}^K$ with $\{L_{xy}^{(k)}\}_{k=1}^K$ and $\{L_{yy}^{(k)}\}_{k=1}^K$, where $L_{xy}^{(k)}$, $L_{yy}^{(k)}$ are lower triangular matrices satisfying:

$$C_{xy}^{(k)} = L_{xy}^{(k)} L_{xy}^{(k)\top} \qquad \left(C_{yy}^{(k)} \right)^{-1} = L_{yy}^{(k)} L_{yy}^{(k)\top} \tag{6.4.5}$$

Since $\mathcal{K}_G^{(k)} \in \mathcal{M}_{n\times m}(\mathbb{R})$, $p \in \mathbb{R}^r$ and $C \in \mathcal{M}_{m\times n}(\mathbb{R})$, the number of parameters in this second architecture increases from $mn(K+1)+r$ to $\frac{1}{2}\left[\, n(n+1) + m(m+1) \,\right] K + mn + r$. For this reason, we adopt the first formulation.

Loss function and backpropagation

We now turn to the loss function used to train the Deep Kalman Filter architecture. Given a prescribed terminal state $x(K)$, we consider the following loss function:

$$\mathcal{L} = \frac{\lambda_K}{2}\left\|\hat{x}(K) - x(K)\right\|_2^2 \; + \; \frac{1}{2}\sum_{k=1}^{K}\left\|y(k) - C^{(k)}\hat{x}(k)\right\|_{R_k^{-1}}^2 \; +$$
$$+ \; \frac{1}{2}\sum_{k=1}^{K}\left\|\mathcal{K}_G^{(k)}\left(y(k) - C^{(k)}\hat{x}(k \mid k-1)\right)\right\|_{P_k^{-1}}^2 \; + \; \frac{\lambda_D}{2mn}\sum_{i,j=1}^{m,n}\left\|Dv^{ij}\right\|_2^2 \; = \tag{6.4.6}$$
$$= \mathcal{E}_{\lambda_K} + \sum_{k=1}^{K}\mathcal{F}^k + \sum_{k=1}^{K}\mathcal{G}^k + \mathcal{H}_{\lambda_D}$$

where R_k, P_k are symmetric weight matrices, D is the second order finite differences matrix, and $v^{ij} = \left[(\mathcal{K}_G^{(1)})_{ij} \; \ldots \; (\mathcal{K}_G^{(K)})_{ij}\right]^\top$.
The loss function defined in (6.4.6) consists of four distinct components:

- $\mathcal{E}_{\lambda_K}$ penalizes deviations of the state at the final network layer from a prescribed (measured) terminal state $x(K)$. This term can be used to mitigate the effects of model uncertainty and errors in the initial state, particularly in chaotic systems, where small perturbations may significantly affect the final state. Since full knowledge of the terminal state is rarely available, especially in PDE-based models, $x(K)$ may be replaced by a coarse approximation, with the weighting parameter λ_K tuned accordingly.

- $\mathcal{F}^k$ penalizes the observation error at the k-th layer and is weighted by the matrix R_k^{-1}. This term enforces consistency between the intermediate estimated states and the available observation $y(k)$, where R_k can be selected to reflect the known stochastic properties of the observation noise.

- $\mathcal{G}^k$ penalizes the correction step at the k-th layer and is weighted by the matrix P_k^{-1}. Indeed, rearranging (6.4.3) yields $\mathcal{K}_G^{(k)}\left(y(k) - C^{(k)}\hat{x}(k \mid k-1)\right) = \hat{x}(k) - \hat{x}(k \mid k-1)$. This term encourages the updated state estimates to remain close to the corresponding predictions $\hat{x}(k \mid k-1)$, with P_k chosen to reflect the associated stochastic properties.

- $\mathcal{H}_{\lambda_D}$ acts as a regularization term on the gain matrices, promoting smooth variations of their entries across network layers.

The last three terms in the loss function are essential for ensuring consistency between the Deep Kalman Filter and its model-based counterpart. Recall that, in the classical Kalman filter, the corrected state $\hat{x}(k)$ is obtained as the solution of the MAP problem (5.4.6), whose objective function comprises two components analogous to $\mathcal{F}^k$ and $\mathcal{G}^k$, for the specific choice $R_k = R(k)$ and $P_k = P(k \mid k-1)$. In (5.4.6), the optimization variable is the state vector $\hat{x}(k)$, whereas in the proposed architecture (6.4.6), the optimization variable is the gain matrix $\mathcal{K}_G^{(k)}$. We return to this distinction at the end of this section. Finally, the state covariance matrices $P(k)$ generated by the Kalman filter evolve according to a well-known Riccati-type equation [6, 7], which induces a smooth temporal evolution of the Kalman gain entries. The role of the regularization term $\mathcal{H}_{\lambda_D}$ is therefore to enforce this intrinsic property on the learned gain matrices, which would otherwise be completely independent across layers.

As for the backpropagation procedure, we refer the reader to Appendix 6.A, where explicit expressions for the partial derivatives required to compute the overall gradients of the proposed architecture are provided:

$$\begin{cases} \nabla_{\mathcal{K}_G^{(k)}}\mathcal{L} &= \nabla_{\mathcal{K}_G^{(k)}}\mathcal{E}_{\lambda_N} + \sum_{l=1}^{K}\nabla_{\mathcal{K}_G^{(k)}}\mathcal{F}^l + \sum_{l=1}^{K}\nabla_{\mathcal{K}_G^{(k)}}\mathcal{G}^l + \nabla_{\mathcal{K}_G^{(k)}}\mathcal{H}_{\lambda_D} \qquad \forall k = 1,\dots,K \\ \nabla_p\mathcal{L} &= \sum_{k=1}^{K}\left(\nabla_{p^{(k)}}\mathcal{E}_{\lambda_K} + \sum_{l=1}^{K}\nabla_{p^{(k)}}\mathcal{F}^l + \sum_{l=1}^{K}\nabla_{p^{(k)}}\mathcal{G}^l\right) \\ \nabla_C\mathcal{L} &= \sum_{k=1}^{K}\left(\nabla_{C^{(k)}}\mathcal{E}_{\lambda_K} + \sum_{l=1}^{K}\nabla_{C^{(k)}}\mathcal{F}^l + \sum_{l=1}^{K}\nabla_{C^{(k)}}\mathcal{G}^l\right) \end{cases} \tag{6.4.7}$$

The weight update rules, for given learning rates $\mu_{\mathcal{K}}$, μ_p, and μ_C, are therefore given by:

$$\begin{cases} \mathcal{K}_G^{(k)} &\Leftarrow \mathcal{K}_G^{(k)} - \mu_{\mathcal{K}}\nabla_{\mathcal{K}_G^{(k)}}\mathcal{L} \qquad \forall k = 1,\dots,K \\ p &\Leftarrow p - \mu_p\nabla_p\mathcal{L} \\ C &\Leftarrow C - \mu_C\nabla_C\mathcal{L} \end{cases} \tag{6.4.8}$$

In practice, an optimizer is used in place of the above stochastic gradient descent. For our implementation we adopted Adam with moment parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We will expand on the initialization of the weights in sec. 6.4.3.

We now examine in greater detail the consistency between the Deep Kalman Filter and the classical Kalman filter. With regard to the Kalman gains, the nature of the *complete* backpropagation procedure (6.A.3)-(6.A.4) implies that the gain matrices $\mathcal{K}_G^{(k)}$ are updated during training using observations $y(k)$ through $y(K)$. In other words, future measurements influence the present state estimation. While access to the full observation sequence can enhance the learning process and potentially yield better-performing gain matrices, it complicates a direct comparison with the classical Kalman filter, which operates using only past and present observations. Indeed, complete backpropagation is closely related to Kalman smoothing [116], as reviewed in Section 5.3. Consequently, to facilitate a more meaningful comparison with the standard Kalman filter, it is useful to introduce a *truncated* version of the backpropagation algorithm for the gain matrices $\mathcal{K}_G^{(k)}$. In particular, recalling that the state $\hat{x}(k)$ estimated by the Kalman filter is the solution of the MAP problem (5.4.6), we assume $\hat{x}(k)$ to be of the form:

$$\begin{aligned} \hat{x}(k) &= \hat{x}(k \mid k-1) + \mathcal{K}_G^{(k)}\left(y(k) - C^{(k)}\hat{x}(k \mid k-1)\right) \\ &= \hat{x}(k \mid k-1) + \mathcal{K}_G^{(k)}e_y(k) \end{aligned} \tag{6.4.9}$$

For some unknown, unconstrained $\mathcal{K}_G^{(k)}$, we can therefore rewrite (5.4.6) as:

$$\begin{aligned} \mathcal{K}_G^{(k)} \in \underset{\mathcal{K}}{\operatorname{argmin}} &\left\| y(k) - C^{(k)}\left(\hat{x}(k \mid k-1) + \mathcal{K}e_y(k)\right)\right\|^2_{R(k)^{-1}} + \\ &+ \|\mathcal{K}e_y(k)\|^2_{P(k|k-1)^{-1}} \end{aligned} \tag{6.4.10}$$

This observation shows that, given the predictor $\hat{x}(k \mid k-1)$, its associated covariance matrix $P(k \mid k-1)$ and observation covariance matrix $R(k)$, the Kalman gain $\mathcal{K}_G^{(k)}$ depends solely on the observation at time k, namely $y(k)$. The proposed *truncated* backpropagation scheme therefore ensures that each term in the loss function (6.4.6) influences only the gain matrix at the corresponding network layer. This can be implemented by replacing (6.A.4) with (6.A.5). We note that the gradients with respect to the parameters p and C in the proposed architecture remain unchanged.

It is also worth noting that the optimization problem in (6.4.10) admits infinitely many minimizers. Indeed, owing to the unconstrained nature of the optimization variable $\mathcal{K}$, any matrix of the form $\mathcal{K}_G^{(k)} + M(k)$, with $e_y(k) \in \ker M(k)$, constitutes an alternative optimal solution. This observation further motivates the inclusion of the regularization term $\mathcal{H}_{\lambda_D}$. We emphasize, however, that although this smoothness-promoting term, whose effectiveness is demonstrated in Section 6.4.4, encourages a gradual temporal evolution of the gain matrices, it does not guarantee exact convergence of the truncated backpropagation algorithm to the Kalman gains of the model-based filter. Indeed, while $\mathcal{H}_{\lambda_D}$ enforces smooth temporal behavior, the model-based Kalman gains $\widetilde{\mathcal{K}}_G^{(k)}$ evolve according to more intricate dynamics. Under suitable non-singularity assumptions [7], one can show that the latter satisfy:

$$\widetilde{\mathcal{K}}_G^{(k+1)} = \left(\mathcal{C} + \mathcal{D}\widetilde{\mathcal{K}}_G^{(k)}\right)\left(\mathcal{A} + \mathcal{B}\widetilde{\mathcal{K}}_G^{(k)}\right)^{-1} = F(\,\widetilde{\mathcal{K}}_G^{(k)}\,) \tag{6.4.11}$$

where $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ depend on the state-space matrices (5.3.1). As a consequence, one possible strategy to achieve exact convergence of the gain matrices, and thus full consistency between the proposed architecture and the classical Kalman filter, is to replace the regularization term $\mathcal{H}_{\lambda_D}$ with a more specialized smoothing term of the form:

$$\widetilde{\mathcal{H}}_{\lambda_F} = \frac{\lambda_F}{2} \sum_{k=1}^{K-1} \left\| \mathcal{K}_G^{(k+1)} - F(\,\mathcal{K}_G^{(k)}\,) \right\|_2^2 \tag{6.4.12}$$

Nevertheless, since (6.4.12) relies heavily on the underlying model satisfying the assumptions of the Kalman filter, we choose to partially relax consistency with the classical formulation by adopting $\mathcal{H}_{\lambda_D}$ as our smoothing term. This choice, in turn, allows the proposed architecture to be applied to a significantly broader class of models.

To conclude this section, we present experimental evidence showing that, in the absence of the smoothing term $\mathcal{H}_{\lambda_D}$, the gain matrices learned by the Deep Kalman Filter differ from the model-based Kalman gains by matrices whose kernels contain the innovation vectors. Specifically, we consider the toy model defined in (5.3.1) and set:

$$A(k) \equiv \begin{bmatrix} 1.05 & 0.05 \\ -0.005 & 1 \end{bmatrix} \qquad B(k) \equiv \mathbb{0} \qquad C(k) \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The observations $y(k)$ are generated using:

$$Q(k) \equiv \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \qquad R(k) \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad P(0) = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$$

with initial condition $\mu_0 = \binom{0}{2}$. We then train the Deep Kalman Filter to convergence on a single trajectory (with $K = 50$), using the truncated backpropagation scheme (6.A.5),

setting $\lambda_K = \lambda_D = 0$, and choosing the weights $R_k = R(k)$ and $P_k = P(k \mid k-1)$. From the resulting gain matrices $\mathcal{K}_G^{(k)}$, we construct the residuals:

$$M(k) = \widetilde{\mathcal{K}}_G^{(k)} - \mathcal{K}_G^{(k)}$$

and compute their singular value decompositions (SVDs), denoting by $V(k)$ the matrix of right singular vectors.

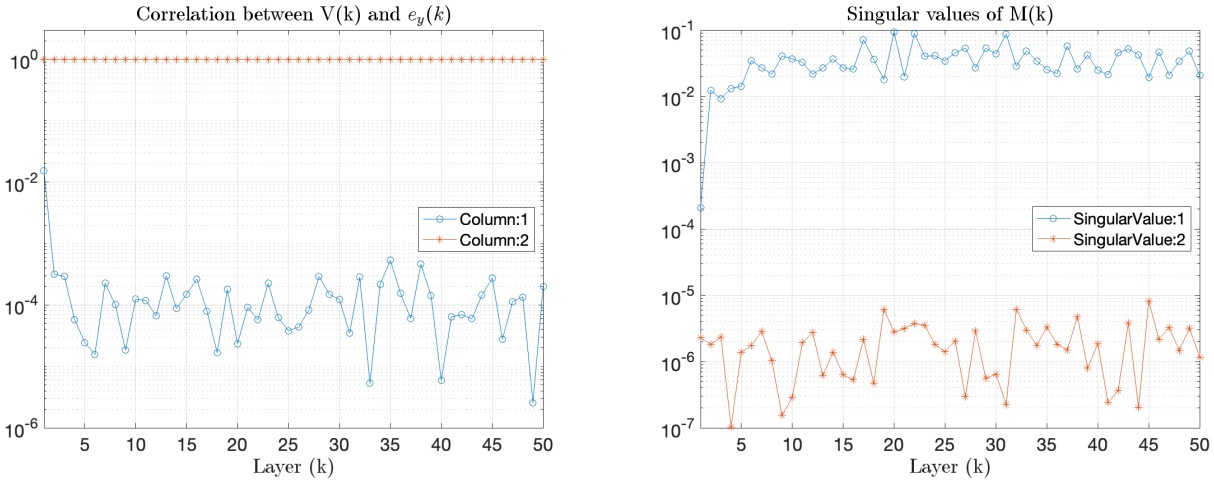


Figure 6.3: Left: Absolute correlations between the columns of $V(k)$ and the innovation $e_y(k)$ at each layer. Right: Singular values of $M(k)$ across the layers.

Figure 6.3 illustrates the correlation between the columns of $V(k)$ and the corresponding innovation vectors $e_y(k)$, as well as the singular values of $M(k)$. As expected, the innovations are closely aligned with the right singular vectors associated with the smallest singular values.

Forward operator extensions

In this section, we discuss several important extensions to the forward operator (or predictor) f. A common approach to define f in state-space systems of the form (6.4.1) is to employ a parametric model governed by differential equations and discretize it. For PDE systems, however, the full state vector can be prohibitively large. In such cases, a surrogate model obtained via operator inference methods can be used within the predictor. Examples include extensions of the Variationally Mimetic Operator Network (VarMiON) [105], Latent Dynamics Networks [108], or classical Operator Inference [49]. The Deep Kalman Filter allows the propagation of even a subset of state variables predicted by the operator network, an option that is not feasible in the traditional model-based Kalman filter, which requires propagation of the complete state vector.
Moreover, since the Kalman filter implements essentially a proportional control of the state-estimation error, certain problems may yield poor performance. In such cases, adding a feed-forward term to the predictor f can substantially improve results; for example, [92] proposes an adaptive feed-forward with gains chosen according to the maximum principle for the heat equation. Importantly, such feed-forward terms can be incorporated directly into f in (6.4.1) without modifying the Deep Kalman Filter formulation.

The map f can also be extended with a general parametric term to enable data-driven model discovery during training, compensating for potential inaccuracies or unmodeled dynamics beyond mere parameter mismatches. One such approach is Sparse Identification of Nonlinear Dynamics (SINDy) [17], which preserves interpretability by representing the learned correction explicitly as a function of the state variables. For instance, suppose the known part of the model is a linear system of differential equations:

$$M\dot{x} = Kx + d \tag{6.4.13}$$

where M, K, and d are known. A state matrix $X \in \mathcal{M}_{n\times K}(\mathbb{R})$ can be constructed by stacking the state estimates $\hat{x}(k)$ from a forward run of the Deep Kalman Filter. An overcomplete dictionary $\Phi(X) \in \mathcal{M}_{N\times n_s}(\mathbb{R})$ can then be generated from a predetermined set of functions of the state vector [37]. Numerical derivatives of the states, computed using noise-robust methods available in the Python `derivative` package and utilized by PySINDy [73], are stored in $\dot{X} \in \mathcal{M}_{n\times K}(\mathbb{R})$. Sparse recovery can then be performed on the underdetermined system:

$$\Phi(X)s = \left(M\dot{X} - KX - D\right)^\top, \tag{6.4.14}$$

where $s \in \mathcal{M}_{n_s\times n}(\mathbb{R})$ is the sparse coefficient matrix and $D \in \mathcal{M}_{n\times K}(\mathbb{R})$ is constructed column-wise from d. Finally, the predictor at each layer can be defined using the discretization of the extended, data-driven model:

$$M\dot{x} = Kx + d + s^\top \Phi(x)^\top, \tag{6.4.15}$$

so that f now depends on s as well: $f = f(x, p, s, u)$. This approach naturally generalizes to nonlinear models; the linear system (6.4.13) is used here solely for notational convenience. It is important to note that the SINDy extension may overlap with the learnable model parameters p, potentially leading to identifiability issues. Consequently, the construction of the dictionary $\Phi(X)$ must be carried out with care, ensuring that no terms are included that could undermine the identifiability of the overall parametric model. Further discussion on this matter is provided in Section 6.4.5. This framework naturally aligns with the hybrid approach proposed by Glasner [52], where the Deep Kalman Filter minimizes the prediction error while the SINDy residual enforces sparsity in the discovered dynamics. In summary, when an accurate model of the underlying dynamics is unavailable or incomplete, the Deep Kalman Filter provides a flexible framework that can incorporate three core categories of unknown dynamics:

- Parametric dynamical models with unknown parameters;
- Complex or partially unknown dynamics captured by interpretable surrogate models, such as neural networks;
- Inaccurate or partially known dynamical models that can be augmented through data-driven extensions.

Finally, even data-driven extensions that introduce "fictitious" terms can remain physically interpretable if they correspond to difficult-to-formulate physical properties. For example, estimating inner cavities in a nonlinear geometric inverse problem can be reformulated as estimating fictitious heat sources in a linear inverse problem [51].

6.4.2 Regularization strategies

The Kalman filter updates the state trajectory by computing the innovations (5.3.8), interpreting them as stochastic model and/or observation errors, and correcting the trajectory using the Kalman gains as a proportional feedback on these prediction errors. The dynamics of the gains, and consequently of the resulting corrections, are computed using only a priori knowledge of the reference model (5.3.1) and the second-order moments (covariances) of the model errors. In contrast, the Deep Kalman Filter learns the gains by minimizing the model-based loss function (6.4.6) directly from the output and final state prediction errors. These data reflect not only stochastic model and observation errors, but also other phenomena such as unmodeled deterministic dynamics, incorrect model parameters, and similar effects. Therefore, it is crucial to guide the learning process so that the algorithm can discriminate between error components arising from stochastic versus deterministic sources, updating each set of parameters accordingly. The underlying assumption is that there exists a unique set of optimal parameters, which can be obtained when all error components are minimized through precise tuning of the corresponding subset of parameters. Crucially, the individual error contributions are not directly observable; only their combined effect can be measured. In this section, we show how these errors can be separated implicitly through regularization, guided by two key principles: the *whiteness of the prediction error* as an indicator of a well-tuned reference model [48, 63], and the classical *parsimony principle* (Occam's razor). It is important to note that the different error components cannot be distinguished instantaneously. Each source of error evolves according to its own dynamics, yet multiple sources may produce similar contributions at any given time step. In the final part of this section, we describe the dynamic regularization procedure employed during the training of the Deep Kalman Filter to address this challenge.

Whiteness of the predictor error

The underlying motivation for the following procedure is that the Kalman Filter is considered perfectly tuned when the output prediction errors $e_y(k)$ behave as white noise [116]. We now detail how this property can be leveraged to guide the learning process of the Deep Kalman Filter and prevent mis- or overfitting during training.

Let us first establish some notation. For a vector $v \in \mathbb{R}^n$, we denote by $\mathcal{N}(v) \in \mathbb{R}^q$ its *normalized cumulative periodogram* (NCP)[1] and by $\mathcal{D} \in \mathbb{R}^q$ a vector of q equispaced points on $[0, 1]$, namely $\mathcal{D}_i = \frac{i-1}{q-1}$ for $i = 1, \ldots, q$. According to Bartlett's whiteness test, if v is white noise, the quantity:

$$\mathcal{W}_{\text{dev}}(v) = \|\mathcal{D} - \mathcal{N}(v)\|_2 \tag{6.4.16}$$

will be small, since the NCP of white noise lies within a confidence region centered at $\mathcal{D}$ [62, 48]. In the Deep Kalman Filter, we use (6.4.16) as a whiteness deviation index for the predictor estimates during training. This index allows us to determine when to stop updating specific rows of the gain matrices $\mathcal{K}_G^{(k)}$ via backpropagation. Specifically, if the prediction error for a given observed state begins to lose whiteness during training (i.e., its whiteness deviation index $\mathcal{W}_{\text{dev}}$ increases), we freeze the corresponding row of the

[1]The dimension q of the NCP depends on the length n of the input vector.

gain matrix. The rationale is that further updates could overfit the state estimate. Once the prediction errors for all observed states achieve whiteness, one can either update the predictor f and restart the whitening procedure or terminate training altogether.

We now describe the whiteness-control procedure in detail. Assume we are at the j-th backpropagation update of the Deep Kalman Filter for a single training trajectory. After performing a forward pass, we assemble the predictor error vectors for each observed state:

$$v_i^j = \{e_y(k)_i\}_{k=1}^K \tag{6.4.17}$$

and compute their corresponding whiteness deviation indices $\mathcal{W}_{\text{dev}}(v_i^j)$. We then identify the set of observed states satisfying a prescribed *stop condition*, for instance:

$$\mathcal{W}_{\text{dev}}(v_i^j) \geq \mathcal{W}_{\text{dev}}(v_i^{j-1}) \tag{6.4.18}$$

which encodes that *"after the $(j-1)$-th update, the whiteness of the i-th observed state prediction error has not increased"*. Let $I_j \subseteq \{1, \ldots, m\}$ denote the indices satisfying (6.4.18). From iteration j onward, we stop updating the rows of $\mathcal{K}_G^{(k)}$ corresponding to $\phi(I_j)$, where $\phi : \{1, \ldots, m\} \to \{1, \ldots, n\}$ maps each observed state index to its corresponding full-state index. If the observation matrix C consists of identity rows, ϕ is single-valued; otherwise, $\phi(i)$ identifies the support of the i-th row of C. From an implementation perspective, we introduce a *gain mask* $M^{\mathcal{K}} \in \mathcal{M}_{n\times m}(\mathbb{R})$, initialized as $M^{\mathcal{K}} = \mathbb{1}_{n\times m}$, and set the rows corresponding to $\phi(I_j)$ to zero as training progresses. The backpropagation update for the gain matrices in (6.4.8) is then replaced with:

$$\mathcal{K}_G^{(k)} \Leftarrow \mathcal{K}_G^{(k)} - \mu_{\mathcal{K}} \left(M^{\mathcal{K}} \circ \nabla_{\mathcal{K}_G^{(k)}} \mathcal{L} \right) \tag{6.4.19}$$

where $\circ$ denotes element-wise multiplication. It is important to note that this procedure does not entirely prevent updates to observed states that have reached the stop condition. If the predictor f is nonlinear or couples multiple states, updates to rows of $\mathcal{K}_G^{(k)}$ outside of $\phi(\{1, \ldots, m\})$ may still influence the observed states. The primary goal of this strategy is to significantly slow down learning for states that have reached a locally optimal configuration with respect to the Kalman filter tuning condition and to reduce the risk of overfitting to the observations.

Model discovery parsimony

Sparse recovery over a redundant dictionary using SINDy is a data-driven modeling operation that requires parsimony [90]. In the Deep Kalman Filter framework [23], various sparse solvers can be employed, such as Orthogonal Matching Pursuit (OMP) and Lawson-Hanson (LH) with the positivity trick [33]. In OMP, the sparsity level must be specified a priori, while in LH it is estimated at runtime. However, in both cases, there is no guarantee that the recovered solution satisfies broader parsimony requirements. For example, in OMP, if the sparsity level is set too low, the recovered solution may fail to capture essential components of the target right-hand side; if it is too high, unnecessary dictionary columns may be included, contributing little to the reconstruction. This challenge is exacerbated in the presence of noisy data, since choosing the correct sparsity level is crucial to avoid overfitting. To address this, we employ a leave-one-out support pruning procedure based on the relative residues of the sparse solution provided by the

solver. In the literature, successive pruning or thresholding of the solution's support has been extensively studied [1, 69, 81, 97]. What distinguishes the approach presented here, however, is its simplicity and low computational cost, enabling implementation without altering the underlying sparse solver. Consider the following sparse regression problem:

$$\begin{aligned} \min \quad & \|x\|_0 \\ \text{s.t.} \quad & Ax = b \end{aligned}$$

Let $\hat{x}$ denote its sparse solution, with initial support $S = S^{(0)} = \{i \mid \hat{x}_i \neq 0\}$, and let $\tau > 0$ be a user-defined parsimony parameter. For any set of indices I, the relative residue is defined as:

$$r(I) = \frac{\|b - A_I \hat{x}_I\|_2}{\|b\|_2} \tag{6.4.20}$$

The pruning procedure iteratively removes from the current support the index whose omission produces the smallest increase in relative residue relative to the original solution. At iteration $j \geq 0$, define:

$$\bar{i} = \underset{i \in S^{(j)}}{\operatorname{argmin}}\ r(S_i^{(j)}) \tag{6.4.21}$$

The support is then updated according to:

$$S^{(j+1)} = \begin{cases} S_{\bar{i}}^{(j)} & \text{if } \ r(S_{\bar{i}}^{(j)}) < (1+\tau) r(S) \\ S^{(j)} & \text{if } \ r(S_{\bar{i}}^{(j)}) \geq (1+\tau) r(S) \end{cases} \tag{6.4.22}$$

Pruning terminates when $S^{(j+1)} = S^{(j)}$, and all components of $\hat{x}$ outside of $S^{(j+1)}$ are set to zero. This procedure allows OMP to be run with an initially high sparsity level while using pruning to remove unnecessary indices, thereby enforcing parsimony and reducing the risk of overfitting.

Regularized learning process

For interpretability, the Deep Kalman Filter separates the learning of its components by starting from a single quantity: the prediction error. This error arises from several sources, including parameter mismatch in the model, unmodelled dynamics, and stochastic errors in both the model and observations. Because these contributions cannot be separated instantaneously, the Deep Kalman Filter implements a dynamic strategy to disentangle them. Here, we outline the rationale behind this design.

The three main sources of error can be addressed in the Deep Kalman Filter via different components: the model parameters p and C handle parameter mismatch, the sparse matrix s (learned via SINDy) captures unmodelled dynamics, and the gain matrices $\mathcal{K}_G^{(k)}$ account for stochastic error terms. The learning procedure aims to balance these components, respecting their individual roles and interactions. First, it is important to note the role of the gain matrices $\mathcal{K}_G^{(k)}$: in classical Kalman filtering, these gains are designed to correct for stochastic errors assuming an accurate deterministic model. If the model is inaccurate, the gains may inadvertently compensate for deterministic "model gaps", which

should instead be corrected by p, C, or s. Therefore, these parameters are given priority. Similarly, the recovery of unmodelled dynamics via SINDy depends on a sufficiently accurate current model, so sparse recovery should be activated only after p and C have largely converged. Ignoring this ordering may result in poor or unstable support estimation. Based on these considerations, we adopt a three-step training procedure (Algorithm 6.3.3), each targeting a specific source of prediction error:

1. *Model parameter mismatch.* After initializing p, C, and $\mathcal{K}_G^{(k)}$, training begins to correct any model parameter discrepancies. To prevent the gain matrices from learning deterministic gaps, the learning rates μ_p, μ_C, and $\mu_\mathcal{K}$ are set to prioritize p and C:

$$\frac{\mu_p}{\mu_\mathcal{K}}, \frac{\mu_C}{\mu_\mathcal{K}} \gg 1 \tag{6.4.23}$$

   In practice, scale factors of 10^2–10^3 are used. Prediction error whiteness is then monitored to determine convergence: when all predictor error vectors $v_i^j \ \forall i = 1, \ldots, m$ satisfy the stop condition (6.4.18), the p and C parameters are considered locally converged.

2. *Unmodelled dynamics.* The next step estimates unmodelled dynamics via SINDy using the state estimates from the Deep Kalman Filter. Here, the gain matrices are temporarily used to provide accurate state estimates for SINDy, even though their intended role is to handle stochastic errors. This step involves cyclic updates:

   2.1. Update s with SINDy using current state estimates;

   2.2. Re-initialize the gain matrices $\mathcal{K}_G^{(k)}$ and reset the Gain Mask $M^\mathcal{K} = \mathbb{1}_{n\times m}$;

   2.3. Resume training until all predictor error vectors v_i^j satisfy the whiteness stop condition.

   This cycle can be repeated until s stabilizes or after a preset number of iterations. Each cycle incrementally improves the predictor map f by incorporating the learned deterministic dynamics.

3. *Stochastic terms.* Finally, the gain matrices $\mathcal{K}_G^{(k)}$ are trained to account for stochastic errors. With p, C, and s considered converged, the learning rate $\mu_\mathcal{K}$ can be increased to accelerate training. The procedure stops once the predictor error vectors reach the whiteness stop condition or according to other standard stopping criteria.

Figure 6.4 illustrates the effect of this training procedure on the loss function of a numerical example (detailed in Section 6.4.5). Each step is highlighted in different shades of gray, and the SINDy cycles are separated by dash-dotted vertical lines.
Depending on the application, one or more steps may be skipped. For instance, if f does not depend on p, or if unmodelled dynamics are negligible, the first and second steps can be omitted.

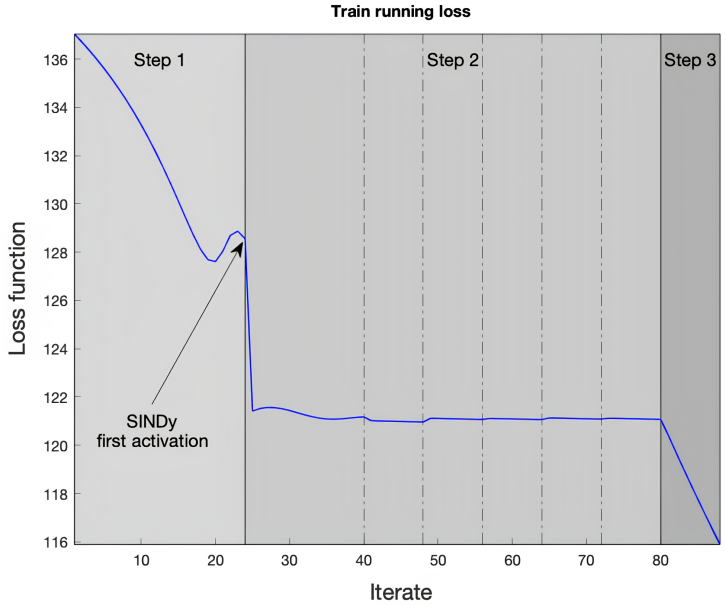


Figure 6.4: Effect of the training process on the running loss function. Three steps of the training procedure are shown in different shades of gray. Each SINDy cycle is separated by dash-dotted vertical lines.

6.4.3 Parameter initialization

Let us now discuss how parameter initialization is handled in the Deep Kalman Filter. Unlike standard neural networks, where parameters are typically initialized only once at the *start* of training, the Deep Kalman Filter may require re-initialization of certain parameters *during* training, as noted in Section 6.4.2. This aspect is crucial, as it can significantly affect overall performance. We distinguish between the three families of parameters updated via backpropagation in our architecture:

- *Forward model parameters* p are initialized at the beginning of training and never reset. Their purpose is to correct discrepancies between the initial predictor map f and the true model underlying the observed data. It is therefore natural to set $p = 0$, so that $f_0 = f(\,\cdot\,, 0, \cdot\,)$ represents our current best estimate of the governing model. Initializing p to any other value, say p_0, is equivalent to setting $p = 0$ and redefining the predictor as $\tilde{f}(\,\cdot\,, p, \cdot\,) = f(\,\cdot\,, p_0 + p, \cdot\,)$, which sets $f_{p_0} = f(\,\cdot\,, p_0, \cdot\,)$ as the initial model estimate. As long as f embeds all available prior knowledge, starting with $p = 0$ is straightforward and natural.

- *Observation model parameters* C are also initialized at the start and never reset. Since their role is to adjust the observation model to match the true system, it is reasonable to initialize $C = C_0$, where C_0 represents our current best estimate of the observation matrix.

- *Gain matrices* $\mathcal{K}_G^{(k)}$ are initialized at the start of training and additionally re-initialized after each SINDy cycle. During SINDy updates, the gains temporarily encode deterministic model gaps to provide accurate state estimates for sparse recovery (see Section 6.4.2). To ensure that the state estimates $\hat{x}(k)$ follow the

observations $y(k)$ rather than potentially inaccurate predictions $\hat{x}(k \mid k-1)$, in traditional Kalman filtering, one can choose the covariance matrices $Q(k)$, $R(k)$, and $P(0)$ such that $Q(k) \equiv Q = \sigma_Q^2 \mathbb{1}_n$, $R(k) \equiv R = \sigma_R^2 \mathbb{1}_m$, and $P(0) = \sigma_P^2 \mathbb{1}_n$, with $\sigma_Q \approx \sigma_P \gg \sigma_R$. This choice biases the initial Kalman gains towards the observations, with the first gain $\widetilde{\mathcal{K}}_G^{(1)} = P(1)C^\top R^{-1}$ particularly influenced by $P(0)$. Accordingly, in the Deep Kalman Filter, all gains are (re-)initialized as:

$$\mathcal{K}_G^{(k)} = P(1)C^\top R^{-1} \qquad \forall k = 1, \ldots, K \tag{6.4.24}$$

for an appropriate choice of Q, R, and $P(0)$. This initialization ensures that state estimates are primarily guided by observations during the early backpropagation steps following each initialization, while preserving interpretability and consistency with classical Kalman filtering.

For completeness, we note that standard random initialization of learnable parameters generally led to worse performance compared to the strategy described above. Consequently, all experiments reported in subsequent sections use this proposed initialization approach. Algorithm 6.3.3 provides a schematic overview of the complete Deep Kalman Filter training procedure.

6.4.4 Numerical experiments: linear state-space models

For the first set of experiments, we consider several linear state-space configurations of the form (5.3.1). We start from the linear ODE (6.4.13), treating $d(t)$ as an input and augmenting it with the output equation

$$\begin{aligned} M\dot{x}(t) &= Kx(t) + d(t) \\ y(t) &= Cx(t) \end{aligned} \tag{6.4.25}$$

Discretizing (6.4.25) with an implicit Euler scheme and time step Δt, we obtain at iteration k:

$$M\left(\frac{x(k+1) - x(k)}{\Delta t}\right) = Kx(k+1) + d(k+1)$$

$$\Rightarrow \quad (\mathbb{1}_n - \Delta t\ M^{-1}K)x(k+1) = x(k) + \Delta t\ M^{-1}d(k+1)$$

This yields a discrete-time system in the form (5.3.1) with:

$$A(k) \equiv A = (\mathbb{1}_n - \Delta t\ M^{-1}K)^{-1} \qquad B(k) \equiv B = (\mathbb{1}_n - \Delta t\ M^{-1}K)^{-1}\Delta t\ M^{-1}$$

$$u(k) = d(k+1)$$

Finally, we include process noise $w(k) \sim \mathcal{N}(0, Q)$ and observation noise $v(k) \sim \mathcal{N}(0, R)$, with prescribed covariance matrices Q and R. By varying the parameters in (6.4.25) and the noise covariance matrices, we can generate a range of distinct state-space models for the following sections.

Algorithm 6.3.3: Deep Kalman Filter training

Input: A set of gain matrices $\{\mathcal{K}_G^{(k)}\}_{k=1}^K$ and model parameters p, C. A set of training trajectories comprising of an initial state $\hat{x}(0)$, observations $\{y(k)\}_{k=1}^K$, inputs $\{u(k)\}_{k=1}^K$ and target final state $x(K)$. An initial forward map f.

Output: A set of trained gain matrices $\{\mathcal{K}_G^{(k)}\}_{k=1}^K$, estimated model parameters p, C and sparse dynamics matrix s describing unmodeled dynamics.

Initialize $M^{\mathcal{K}} = \mathbb{1}_{n\times m}$
for Epoch $= 1 :$ EpochsNum **do:**
  for TrajectoryIdx $= 1 :$ TrajectoryNum **do:**
    select TrainTrajectory(TrajectoryIdx)
    for $k = 1 : K$ **do:**
      $\hat{x}(k) = \hat{x}(k \,|\, k-1) + \mathcal{K}_G^{(k)}\left(y(k) - C\hat{x}(k \,|\, k-1)\right)$ (6.4.2)-(6.4.3)
    end for
    update whiteness deviation indices for current trajectory's output states (6.4.16)
    if whiteness deviation indices have increased for a given output state across all trajectories (6.4.18)
      set corresponding row of $M^{\mathcal{K}}$ to zero
    end if
    if $(CM^{\mathcal{K}} == \mathbb{0}_{m\times m})$
      Update s with SINDy
      Re-initialize $\{\mathcal{K}_G^{(k)}\}_{k=1}^K$ and reset ADAM moments
      Re-initialize $M^{\mathcal{K}} = \mathbb{1}_{n\times m}$
    else
      for $k = 1 : K$ **do:**
        Compute $\nabla_{\mathcal{K}_G^{(k)}}\mathcal{L}$ (6.A.3)-(6.A.4)-(6.A.5)-(6.4.7)
      end for
      Compute $\nabla_p\mathcal{L}$ (6.A.3)-(6.A.4)-(6.A.5)-(6.4.7)
      Compute $\nabla_C\mathcal{L}$ (6.A.3)-(6.A.4)-(6.A.5)-(6.4.7)
      for $k = 1 : K$ **do:**
        Update $\mathcal{K}_G^{(k)} \Leftarrow \mathcal{K}_G^{(k)} - \mu_{\mathcal{K}}\left(M^{\mathcal{K}} \circ \nabla_{\mathcal{K}_G^{(k)}}\mathcal{L}\right)$ (6.4.19)
      end for
      Update $p \Leftarrow p - \mu_p\nabla_p\mathcal{L}$
      Update $C \Leftarrow C - \mu_C\nabla_C\mathcal{L}$
    end if
  end for
end for

Strongly observable and known model

Consider the state-space system defined by:

$$A = \begin{bmatrix} 1.01 & -0.002 \\ 0 & 1.005 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad u(k) \equiv \mathbb{0}$$

with initial state $\hat{x}(0) = \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$ and diagonal covariance matrices:

$$Q = \sigma_Q^2 \mathbb{1}_2 \qquad R = \sigma_R^2 \mathbb{1}_2 \qquad P(0) = \sigma_P^2 \mathbb{1}_2$$

where $\sigma_Q = 2 \times 10^{-2}$, $\sigma_R = 3.5 \times 10^{-2}$, and $\sigma_P = 10^{-1}$. For this DLTI system, the observability Gramian (5.2.9) is:

$$G_O = \begin{bmatrix} 2.0201 & -0.0020 \\ -0.0020 & 2.0100 \end{bmatrix}$$

indicating that both states are comparably and strongly observable. We now assess the *interpretable consistency* of the Deep Kalman Filter. In the loss function (6.4.6), the matrices R_k and P_k correspond to $R(k)$ and $P(k \mid k-1)$ in the MAP formulation (5.4.6), which determine the Kalman gain update (5.3.11). Although the Deep Kalman Filter gains are more general than their model-based counterparts, the first column of Figure 6.5 shows that, for this class of systems, using truncated backpropagation (6.A.5), setting $R_k = R$, $P_k = P(k \mid k-1)$ for all k, and choosing $\lambda_K = 0$, $\lambda_D > 0$, the learned gains $\mathcal{K}_G^{(k)}$ converge rapidly in k to the classical Kalman gains (5.3.11). The role of the smoothing regularization is highlighted by the second column of Figure 6.5, where $\lambda_D = 0$. In this case, the gains exhibit pronounced oscillations, whereas $\lambda_D > 0$ enforces the smooth temporal evolution characteristic of the Riccati-based update (6.4.11). As with any regularization term, this introduces a small bias in the learned gains. The importance of a k-dependent weighting matrix P_k is illustrated in the third and fourth columns of Figure 6.5, where P_k is set constant to $P(0)$ and Q, respectively. In both cases, the learned gains deviate noticeably from those in the first column. These results capture the essence of *interpretable consistency* for the Deep Kalman Filter: when applied to problems where the model-based Kalman filter is optimal, backpropagation does not introduce significant bias. At the same time, the Deep Kalman Filter naturally extends the classical framework to more general settings, including nonlinear dynamics, partially known models, and non-Gaussian noise, as demonstrated in the following subsections.

Model with incorrect covariance matrices

The Deep Kalman Filter is more robust than its model-based counterpart to uncertainty in the model specification when additional information is available, such as knowledge of the final state $x(K)$. This information enters the loss function (6.4.6) through the term weighted by λ_K, inducing a trade-off between deviations in the state trajectory, caused by inaccurate a priori estimates of Q, R, and $P(0)$, and enforcement of the prescribed final state. Here we assume, for simplicity, that the final state is fully and accurately observed, although the approach readily extends to partial observations. Numerical results show that, for sufficiently large λ_K (and λ_D), this information is effectively backpropagated through the network, yielding performance superior to that of the model-based

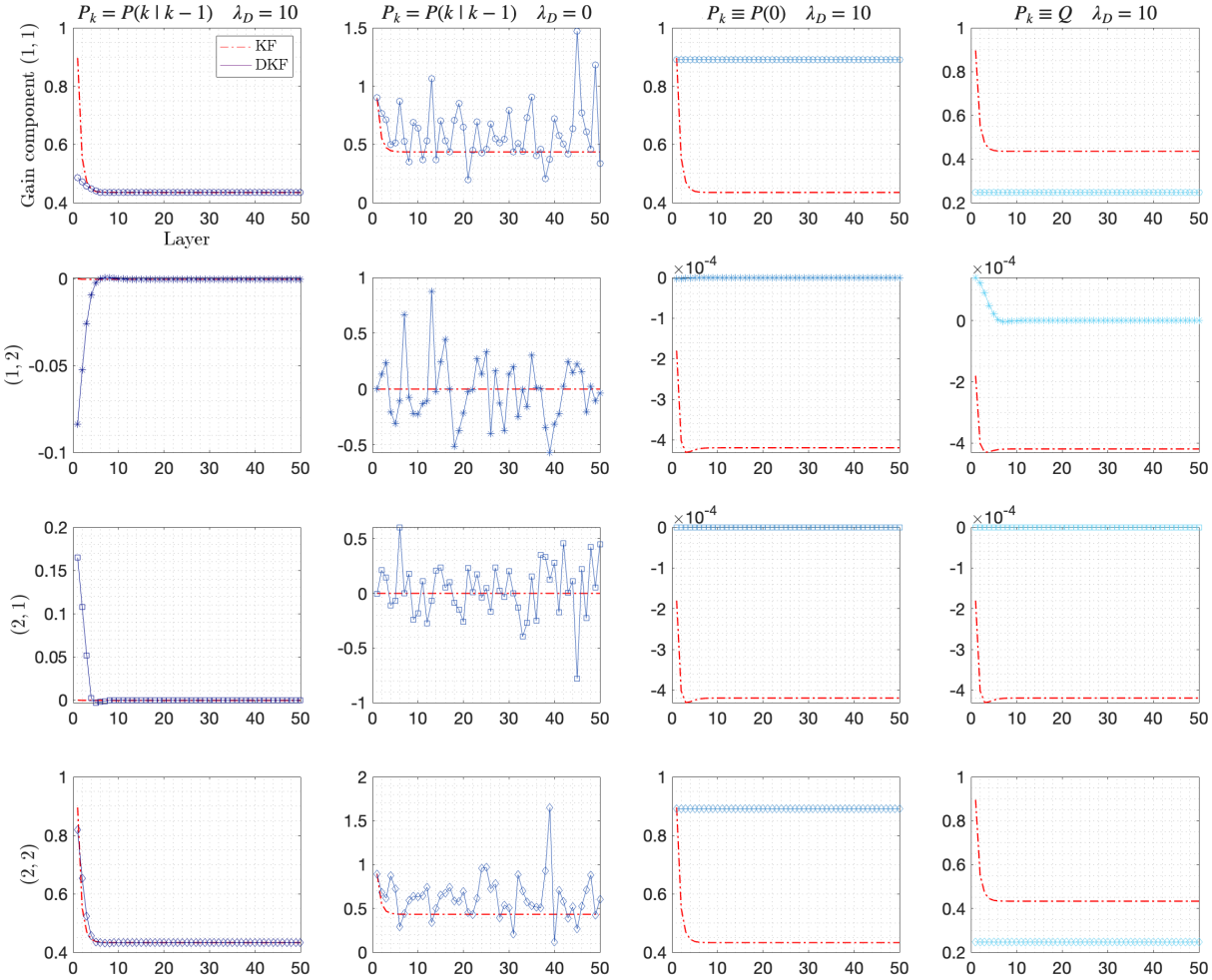


Figure 6.5: Gain matrices comparison for different values of P_k and λ_D. First column: DKF gains obtained for $P_k = P(k \mid k-1)$ and $\lambda_D = 10$; Second column: DKF gains obtained for $P_k = P(k \mid k-1)$ and $\lambda_D = 0$; Third column: DKF gains obtained for $P_k \equiv P(0)$ and $\lambda_D = 10$; Fourth column: DKF gains obtained for $P_k \equiv Q$ and $\lambda_D = 10$. The components of the gains vary across the rows. In each plot, the component of the DKF gains is denoted by a blue continued curve, while the corresponding component of the KF gains is denoted by a red dash-dotted curve.

Kalman filter. We now examine the effect of the loss term $\mathcal{E}_{\lambda_K}$ on the gain matrices $\mathcal{K}_G^{(k)}$ learned by the DKF under both truncated and complete backpropagation. In the truncated backpropagation case (6.A.5), we use the same state-space model as in the previous experiments, but deliberately provide an inaccurate estimate of the process noise covariance, $\widetilde{Q} = 0.1Q$. Figure 6.6 shows that, for sufficiently large values of λ_K and λ_D, the DKF outperforms the standard Kalman filter up to an optimal regime. Beyond this regime, the state estimates become unstable (exhibiting oscillations, not shown), leading to degraded performance.

In the complete backpropagation case (6.A.4), we again use the same state-space model, but now provide inaccurate estimates for all stochastic components: $\widetilde{Q} = 0.1Q$, $\widetilde{R} = 5R$, and $\widetilde{P}(0) = 0.1P(0)$. We generate 100 trajectories with different noise realizations and

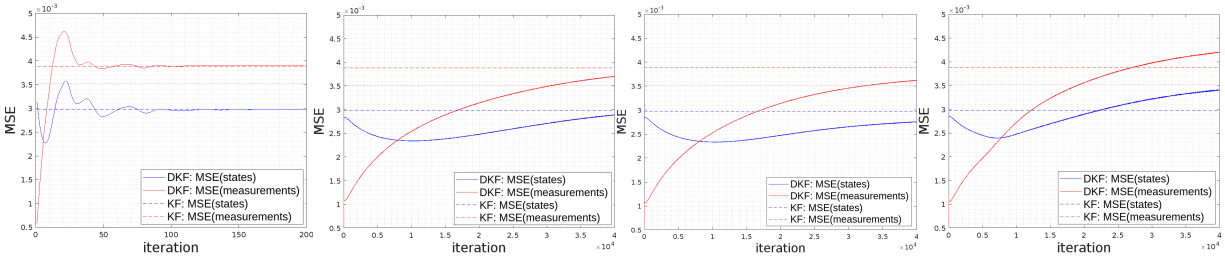


Figure 6.6: Case of wrong covariance matrix $\widetilde{Q} = 0.1Q$: running DKF MSEs computed with respect to the measurements (red) and the true trajectory without measurement noise (blue). The horizontal dashed lines are the corresponding KF MSEs. From left to right: $\lambda_K = 0$ and $\lambda_D = 1$; $\lambda_K = 10^3$ and $\lambda_D = 2.5 \times 10^5$; $\lambda_K = 7 \times 10^4$ and $\lambda_D = 2.5 \times 10^5$; $\lambda_K = 5 \times 10^5$ and $\lambda_D = 2.5 \times 10^5$.

train the DKF for various values of λ_K and λ_D.

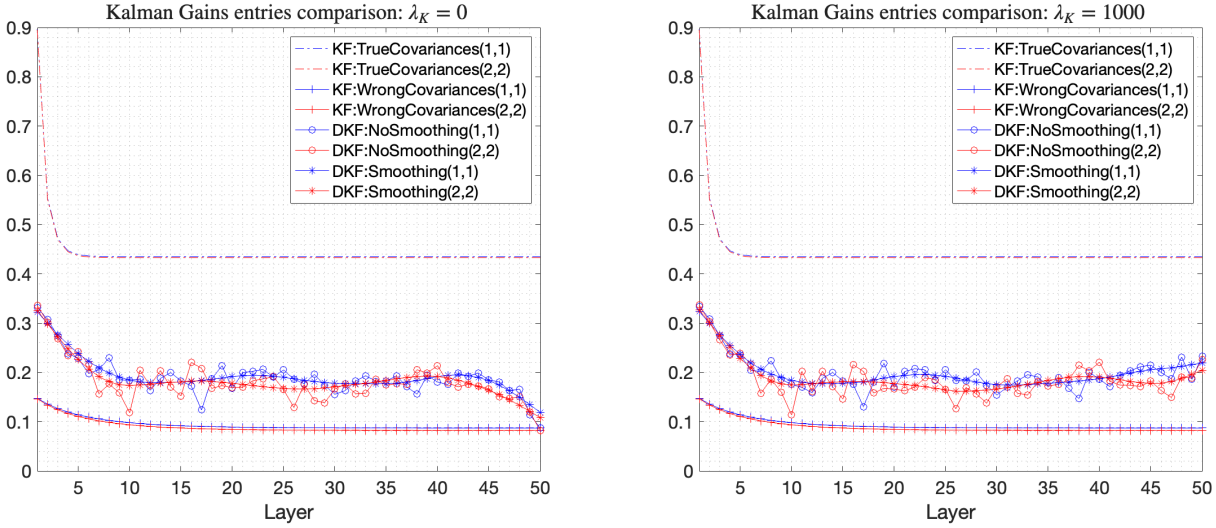


Figure 6.7: Case of wrong covariance matrices $\widetilde{Q} = 0.1Q$, $\widetilde{R} = 5R$ and $\widetilde{P}(0) = 0.1P(0)$: effect of the loss-function term $\mathcal{E}_{\lambda_K}$ on the DKF gain matrices. Diagonal entries of the gains as a function of the layer for $\lambda_K = 0$ (left) and $\lambda_K = 10^3$ (right). In both figures "NoSmoothing" refers to the choice $\lambda_D = 0$ (circles) and "Smoothing" to $\lambda_D = 10^3$ (asterisks). We also include the diagonal entries of the model-based Kalman gains obtained from the true covariances Q, R, $P(0)$ (dash-dots) and wrong covariances $\widetilde{Q}$, $\widetilde{R}$, $\widetilde{P}(0)$ (crosses).

Figure 6.7 illustrates the effect of $\mathcal{E}_{\lambda_K}$ on the learned gain matrices. In particular, this term influences the gains at the final layers, preventing their diagonal entries from drifting toward the stationary Kalman gains computed using the inaccurate covariances $\widetilde{Q}$, $\widetilde{R}$, and $\widetilde{P}(0)$. Finally, for each set of learned gain matrices, we simulate the 100 estimated trajectories and compute their mean-squared error with respect to the corresponding true trajectories generated without observation noise but with identical noisy initial conditions. The averaged results, reported in Table 2, show that when the stochastic components of the model are misspecified, the Deep Kalman Filter substantially improves performance

over the standard Kalman filter and achieves accuracy comparable to that of a model-based Kalman filter constructed using the true covariance matrices.

Table 2: Average MSE between true and simulated trajectories obtained from different sets of gain matrices.

MSE	KF		DKF			
	True covariances	Wrong covariances	$\lambda_D = 0$		$\lambda_D = 10^3$	
			$\lambda_K = 0$	$\lambda_K = 10^3$	$\lambda_K = 0$	$\lambda_K = 10^3$
All states	3.30×10^{-4}	2.10×10^{-3}	7.72×10^{-4}	7.59×10^{-4}	7.78×10^{-4}	7.64×10^{-4}
Last 10 states	3.69×10^{-4}	2.22×10^{-3}	8.44×10^{-4}	7.47×10^{-4}	8.50×10^{-4}	7.47×10^{-4}

Weakly observable and known model

Consider the state-space system defined by:

$$A = \begin{bmatrix} 1.0030 & 0.0502 & 0.0001 \\ 0.0502 & 1.0030 & 0.0001 \\ 0.0000 & 0.0000 & 1.0005 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad u(k) \equiv \mathbb{0}$$

with initial state $\hat{x}(0) = \begin{pmatrix} 2.0 \\ 1.0 \\ 1.5 \end{pmatrix}$ and diagonal covariance matrices:

$$Q = \sigma_Q^2 \mathbb{1}_3 \qquad R = \sigma_R^2 \mathbb{1}_2 \qquad P(0) = \sigma_P^2 \mathbb{1}_3$$

where $\sigma_Q = 10^{-4}$, $\sigma_R = 10^{-4}$ and $\sigma_P = 5$. For this DLTI system, the observability Gramiam (5.2.9) is:

$$\mathcal{G}_O = \begin{bmatrix} 3.0358 & 0.3037 & 0.0002 \\ 0.3037 & 3.0358 & 0.0002 \\ 0.0002 & 0.0002 & 3 \times 10^{-8} \end{bmatrix}$$

indicating that the third state is weakly observable. In particular, the model-based Kalman filter applies a proportional feedback correction based on the output prediction error and typically performs poorly for weakly observable state variables. The Deep Kalman Filter shows improved performance in this setting, but at the cost of an improper use of the gains: the weak observability of the third state is compensated through the gains rather than through the predictor. We recall that, in this set of experiments, the forward-model parameters p are not employed. For this reason, we introduce a feed-forward (FF) term $\delta\hat{x}_{FF}(k)$ in the state estimate, following [92]. With this addition, the prediction step for the considered DLTI system becomes:

$$\hat{x}(k \mid k-1) = A\,(\,\hat{x}(k-1) + G_{FF}\delta\hat{x}_{FF}(k-1)\,)$$

where $\delta\hat{x}_{FF}(k-1)$ represents an estimate of the state-estimation increment anticipated by the feed-forward action. In the illustrative example considered here, the feed-forward

gain G_{FF} is trained during the learning phase using the known difference $x(K) - x(0)$, while during operation it is computed iteratively. In practical applications, this approach is most effective when the feed-forward term can exploit analytic insights, such as the maximum principle for the heat equation employed in [92].
Figure 6.8 shows that complete backpropagation (second panel from the left) offers no improvement over truncated backpropagation (leftmost panel) and, overall, the DKF performs worse than the model-based Kalman filter (dashed lines). As training proceeds (third panel from the left), the loss continues to decrease (not shown), yet state-estimation accuracy degrades: overfitting the observed outputs comes at the expense of poorer state estimates. By contrast, the rightmost panel highlights the effect of the feed-forward term, which preserves accurate state estimates even at convergence despite overfitting. Table 3 further shows that, with feed-forward, the gains associated with the observable states are closer to the Kalman filter gains and remain bounded for the weakly observable state, whereas the Kalman filter gains diverge for that variable.

Table 3: Comparison between the gain matrices $\mathcal{K}_G^{(k)}$ obtained with KF and DKF at the first ($k = 1$) and last ($k = 50$) layer.

$\mathcal{K}_G^{(k)}$	KF		DKF no feed-forward		DKF with feed-forward	
$k = 1$	1.0000	0.0000	1.0602	0.0602	1.0001	−0.0002
	0.0000	1.0000	0.0602	1.0602	0.0001	0.9998
	0.0000	0.0000	−0.4651	−0.4650	−0.0923	−0.0922
$k = 50$	0.6227	0.0146	0.9518	−0.0483	0.6885	−0.3017
	0.0146	0.6227	−0.0592	0.9390	−0.2767	0.7254
	123.9904	123.9904	0.3487	0.3607	0.1695	0.1993

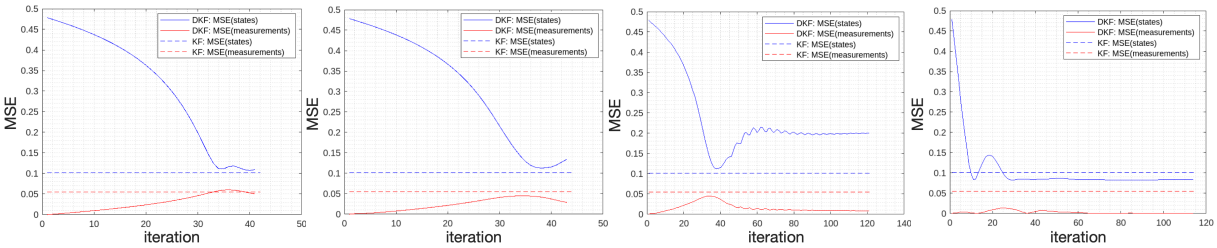


Figure 6.8: Case of a weakly-observable, known model, first example: running DKF MSEs computed with respect to the observations (red) and the true trajectory without observation noise (blue). The horizontal dashed lines are the corresponding KF MSEs. From left to right: running MSEs obtained with truncated backpropagation over 40 iterates; complete backpropagation over 40 iterates; complete backpropagation over 120 iterates; complete backpropagation with feed-forward over 120 iterates.

In the previous example the Kalman filter performed satisfactorily and was therefore used as a reference. Figure 6.9 instead illustrates a case in which it performs poorly: here the Deep Kalman Filter yields a clear improvement over the Kalman filter (left panel). The benefit of the feed-forward term is also evident (right panel): without it,

convergence is significantly slower (note the different x-axis scales), the final MSEs are slightly higher, and the gain components associated with the weakly observable state exhibit larger magnitudes and noticeable oscillations, as reported in Table 4.

Table 4: Comparison between the gain matrices $\mathcal{K}_G^{(k)}$ obtained with the KF and DKF at the first ($k = 1$) and last ($k = 50$) layer.

$\mathcal{K}_G^{(k)}$	KF		DKF no feed-forward		DKF with feed-forward	
$k = 1$	1.0000	0.0000	1.0640	0.0097	1.0013	−0.0043
	0.0000	1.0000	0.1203	1.0269	0.0013	0.9957
	0.0000	0.0000	−1.0322	−0.9437	−0.0908	−0.0859
$k = 50$	0.6226	0.0144	1.0135	−0.1437	1.0194	−0.0025
	0.0144	0.6226	−0.0599	0.8049	0.0019	0.9937
	119.7517	119.7517	0.3895	0.4775	0.0409	0.1119

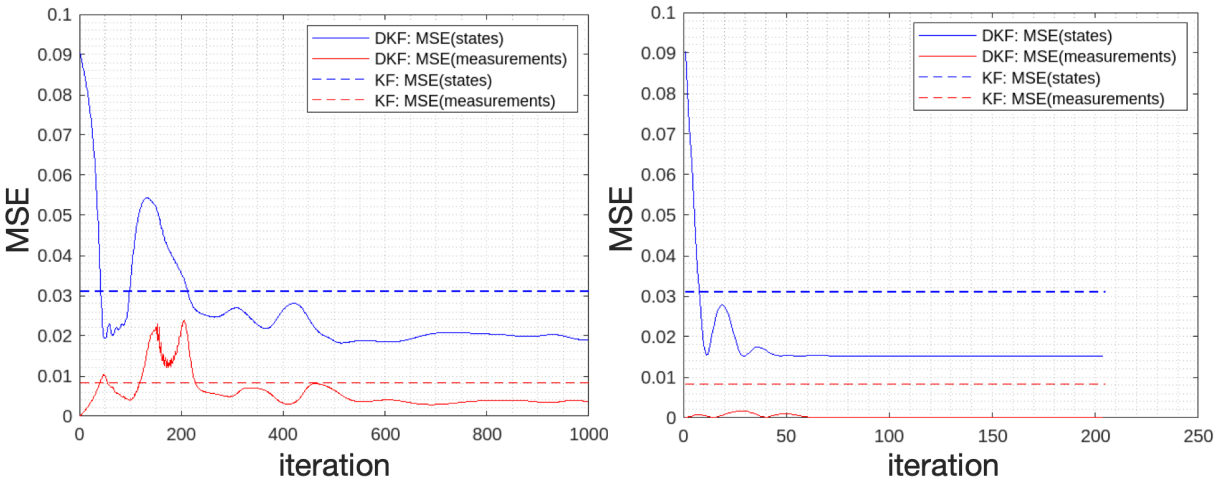


Figure 6.9: Case of a weakly-observable, known model, second example: running DKF MSEs computed with respect to the observations (red) and the true trajectory without observation noise (blue). The horizontal dashed lines are the corresponding KF MSEs. From left to right: running MSEs obtained with truncated backpropagation over 1000 iterates; truncated backpropagation with feed-forward over 200 iterates.

Model with non-diagonal covariance matrices

Consider again the state-space system from the strongly observable experiment, but now assume that the covariance matrices Q and R are not diagonally constant. We test the Deep Kalman Filter both with the true covariances Q and R and with diagonally constant approximations $\widetilde{Q}$ and $\widetilde{R}$, where the constant diagonal entries are set to the mean of the corresponding true diagonals. Figure 6.10 shows a representative example of the improvement obtained by using matrix-valued weightings instead of scalar penalties, and again highlights convergence toward the Kalman filter.

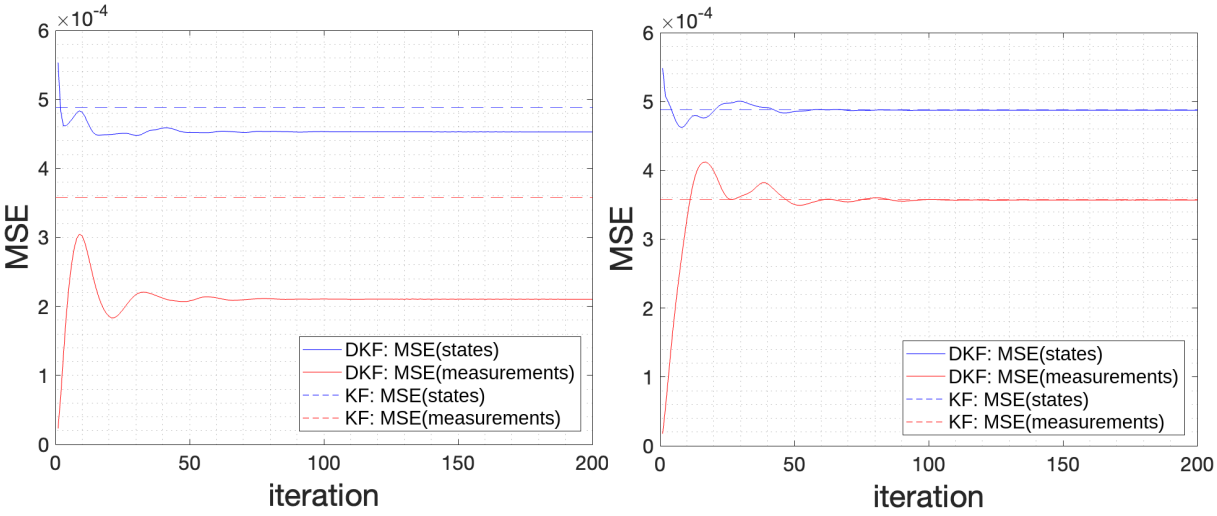


Figure 6.10: Case of non-diagonally constant covariance matrices Q and R: running DKF MSEs computed with respect to the observations (red) and the true trajectory without observation noise (blue). The horizontal dashed lines are the corresponding KF MSEs. Left: running DKF MSEs obtained with truncated backpropagation and wrong, diagonally constant covariances $\widetilde{Q}$, $\widetilde{R}$. Right: running DKF MSEs obtained with truncated backpropagation and true, non-diagonally constant covariances Q, R.

6.4.5 Numerical experiments: nonlinear state-space models

This section examines the ability of the Deep Kalman Filter to generalize the model-based Kalman filter to nonlinear state-space systems. Unlike the previous experiments, we fully exploit the regularized training procedure from Section 6.4.2, using the forward model parameters p to correct small mismatches and the sparse SINDy dictionary s to capture unmodeled dynamics.

FitzHugh-Nagumo

For the first set of experiments on non-linear state-space models, we consider the perturbed FitzHugh-Nagumo ODE system:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}0.01 & 2\\ -3 & -0.08\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix} + \begin{bmatrix}\alpha x^3\\ 0\end{bmatrix} + \begin{bmatrix}\beta\\ \gamma x^2\end{bmatrix} \tag{6.4.26}$$

where $\alpha = -0.5$, $\beta = 1$, $\gamma = 0.1$ and provide the Deep Kalman Filter with only partial knowledge of the true dynamics, namely the linear ODE:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}0.01 & 2\\ -3 & -0.08\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix} \tag{6.4.27}$$

We then integrate numerically (using MATLAB's `ode45`) equation (6.4.26) with starting condition $x(0) = 0.1$, $y(0) = 0.2$ for $t \in [0, 20]$ evaluated at a regular time step of $\Delta t = 8 \times 10^{-2}$. We train the Deep Kalman Filter with $C = \mathbb{1}_3$ (fixed, not optimized by backpropagation) on single trajectories obtained by adding different levels of process and observation noise. The goal is to estimate the unmodeled dynamics and the corresponding

parameters $\alpha^* = -0.5$, $\beta^* = 1$, $\gamma^* = 0.1$. In particular, we consider process and observation noises with scalar covariance matrices and standard deviations σ_Q, σ_R respectively, sampled linearly in the intervals $\Sigma_Q = [0, 0.4]$, $\Sigma_R = [0, 0.4]$ and for each we train the network on 20 different realizations. We also add initial state uncertainty with covariance $P(0) = R$. Lastly, the chosen loss-function weight matrices are $R_k = R$, $P_k = P(k \mid k-1)$ $\forall k = 1, \ldots, K$ and we let $\lambda_K = \lambda_D = 1$. The SINDy dictionary is constructed with polynomial terms up to degree three. For every pair (σ_Q, σ_R) the euclidean norms of the relative residuals $\left(\frac{|\hat{\alpha}-\alpha^*|}{|\alpha^*|}, \frac{|\hat{\beta}-\beta^*|}{|\beta^*|}, \frac{|\hat{\gamma}-\gamma^*|}{|\gamma^*|}\right)$ are averaged over all realizations and displayed in Figure 6.11. Moreover, Figure 6.12 shows the training results obtained with a realization of process and observation noise for $\sigma_Q = 0.4$, $\sigma_R = 0.4$.

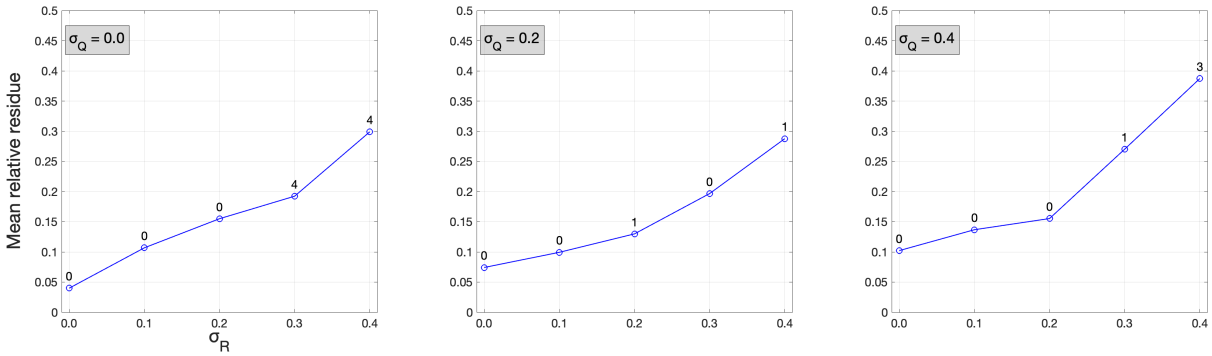


Figure 6.11: Mean relative residues of parameter estimates for the perturbed FitzHugh-Nagumo system (6.4.26). The residues were averaged over 20 realizations of synthetic data for each combination of σ_Q and σ_R. Above each mean residue is shown the number of noise realizations for which the support of the sparse SINDy matrix was estimated incorrectly.

We remark that, unlike other similar works [52], here we do not explicitly tell the network which unmodeled dynamics we expect to be present in the noisy trajectory. As a matter of fact, all entries in the SINDy sparse matrix are essentially parameters that the Deep Kalman Filter can estimate, thus greatly increasing the algorithm complexity. In Figure 6.11 we also include, for each pair (σ_Q, σ_R), the number of noise realizations that led the algorithm to recover an *incorrect* support for the matrix. As expected, increasing levels of process and observation noise affects SINDy's ability to correctly estimate the unmodeled dynamics. As one can observe from the left and right plots of Figure 6.11, the higher values for the mean relative residues are directly correlated to an incorrect support estimation.

Lorenz '63

For the last set of non-linear state-space experiments, we begin by considering the perturbed Lorenz '63 ODE system:

$$\begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -7 & 7 & 0 \\ 28 & -1 & 0 \\ 0 & 0 & -\frac{8}{3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ -xz \\ xy \end{bmatrix} + \begin{bmatrix} \varepsilon x^3 \\ 0 \\ 0 \end{bmatrix} \tag{6.4.28}$$

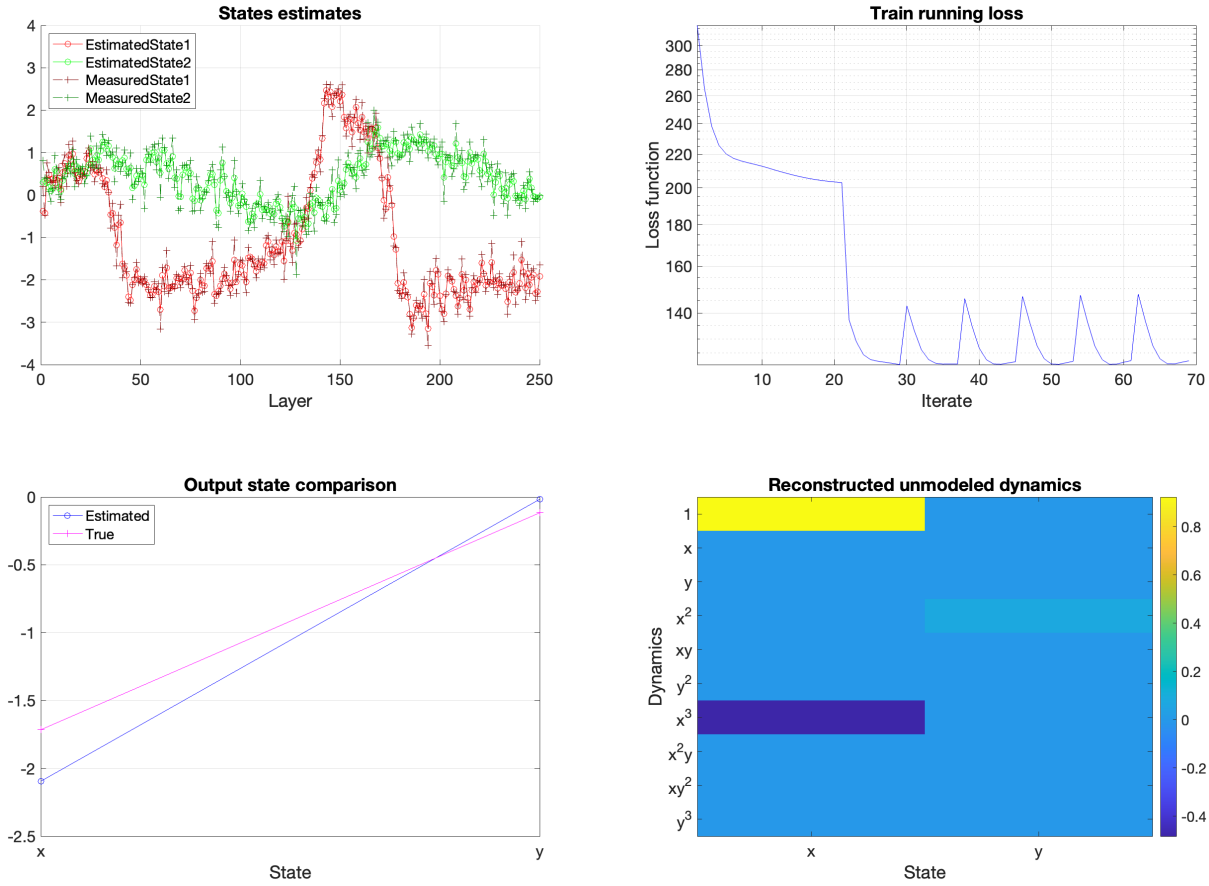


Figure 6.12: Top left: states estimates (circles) and given noisy observations (crosses) for the perturbed FitzHugh-Nagumo system (6.4.26). Top right: train running loss decrease. Bottom left: comparison between estimated (circles) and true (crosses) final state. Bottom right: reconstructed sparse SINDy matrix.

where $\varepsilon = 0.01$ and provide the Deep Kalman Filter with only partial knowledge of the true dynamics, namely the true Lorenz ODE:

$$\begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -7 & 7 & 0 \\ 28 & -1 & 0 \\ 0 & 0 & -\frac{8}{3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ -xz \\ xy \end{bmatrix} \tag{6.4.29}$$

We then integrate numerically (using MATLAB's `ode45`) equation (6.4.28) with starting condition $x(0) = -30$, $y(0) = 25$, $z(0) = 50$ for $t \in [0, 0.2]$ evaluated at a regular time step of $\Delta t = 10^{-3}$. We choose to observe the first two states, thus $C = \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{smallmatrix}\right)$ (fixed, not optimized by backpropagation) and add both process and observation noise with scalar covariance matrices and standard deviations $\sigma_Q = 4$, $\sigma_R = 2$. We also add initial state uncertainty with covariance $P(0) = R$. Lastly, the chosen loss-function weight matrices are $R_k = R$, $P_k = P(k \mid k-1)$ $\forall k = 1, \ldots, K$ and we let $\lambda_K = \lambda_D = 1$. The SINDy dictionary is constructed with polynomial terms up to degree three. Figure 6.13 shows the results after training on a single trajectory.

To conclude this section, we again consider equation (6.4.28) as the true dynamics model

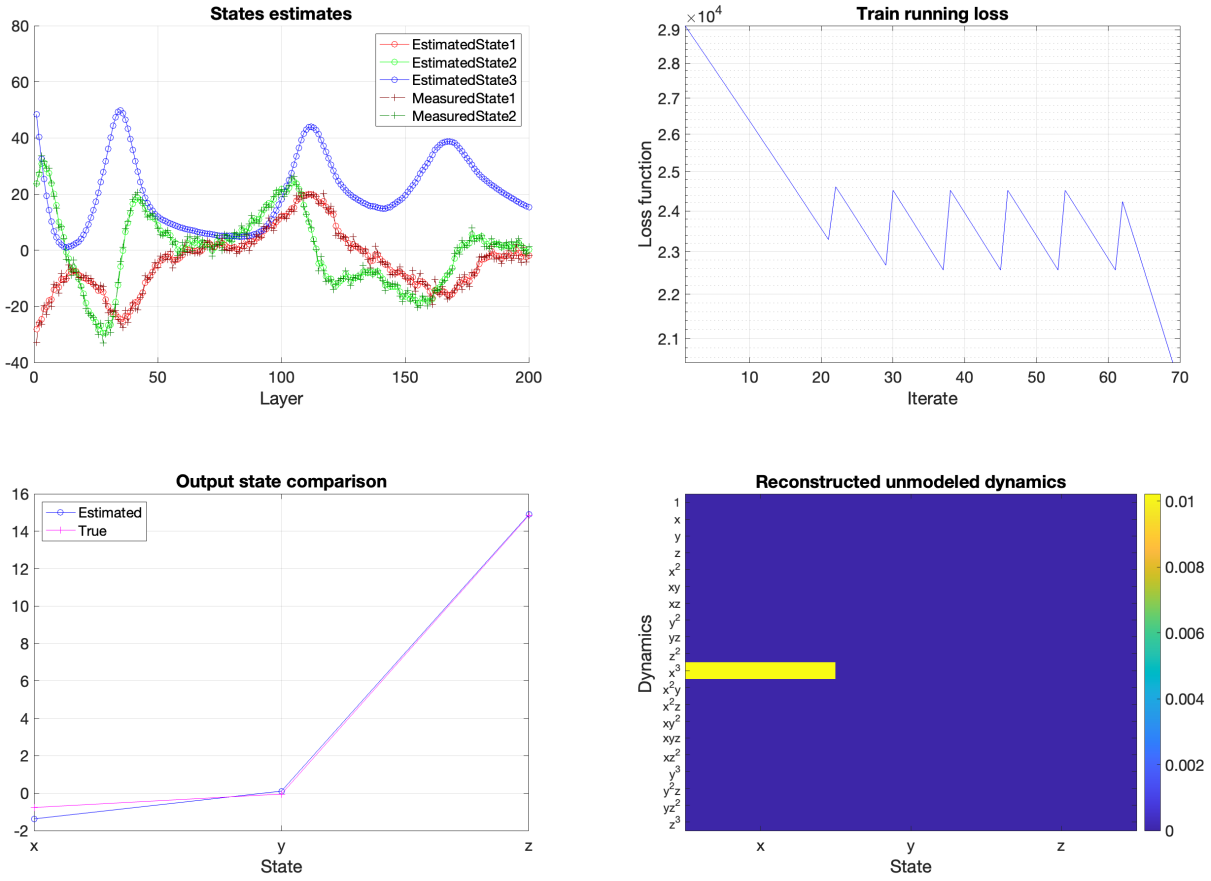


Figure 6.13: Top left: states estimates (circles) and given noisy observations (crosses) for the perturbed Lorenz system (6.4.28). Top right: train running loss decrease. Bottom left: comparison between estimated (circles) and true (crosses) final state. Bottom right: reconstructed sparse SINDy matrix.

but provide the Deep Kalman Filter with a rescaled version of equation (6.4.29), namely:

$$\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -7 & 7 & 0 \\ 28 & -1 & 0 \\ 0 & 0 & -\frac{8}{3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ -xz \\ xy \end{bmatrix} \tag{6.4.30}$$

Instead of relying completely on SINDy, which would need to re-discover the given (correct) x-state dynamic and rescale the linear terms accordingly, we employ the p parameters in our architecture to account for such mass-like discrepancy. In particular, the complete forward model encoded in the network is obtained by discretizing:

$$\begin{bmatrix} 0.5+p & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -7 & 7 & 0 \\ 28 & -1 & 0 \\ 0 & 0 & -\frac{8}{3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ -xz \\ xy \end{bmatrix} \tag{6.4.31}$$

This formulation highlights the inherent risk of identifiability issues arising when simultaneously employing SINDy and the learned parameters p. In this particular case, exact identifiability could be achieved by excluding the linear terms associated with the x-state from the SINDy dictionary. However, such a modification is unnecessary, as the adopted

regularized learning procedure is explicitly designed to prioritize the accurate estimation of the known deterministic model discrepancies encoded by p. We train the Deep Kalman Filter with $C = \mathbb{1}_3$ (fixed, not optimized by backpropagation) on single trajectories obtained by adding different levels of process and observation noise. The goal is to estimate the missing dynamic, its associated parameter $\varepsilon^* = 0.01$, and the mass discrepancy $p^* = -0.4$. We consider process and observation noises with standard deviations σ_Q and σ_R respectively, sampled linearly in the intervals $\Sigma_Q = [0, 4]$, $\Sigma_R = [0, 2]$ and for each we train the network on 20 different realizations. We also add initial state uncertainty with covariance $P(0) = R$. The loss-function is constructed as in the previous example. For every pair (σ_Q, σ_R) the euclidean norms of the relative residuals $\left(\frac{|\hat{\varepsilon}-\varepsilon^*|}{|\varepsilon^*|}, \frac{|\hat{p}-p^*|}{|p^*|}\right)$ are averaged over all realizations and displayed in Figure 6.14. Moreover, Figure 6.15 shows the training results obtained with a realization of process and observation noise for $\sigma_Q = 4$, $\sigma_R = 2$.

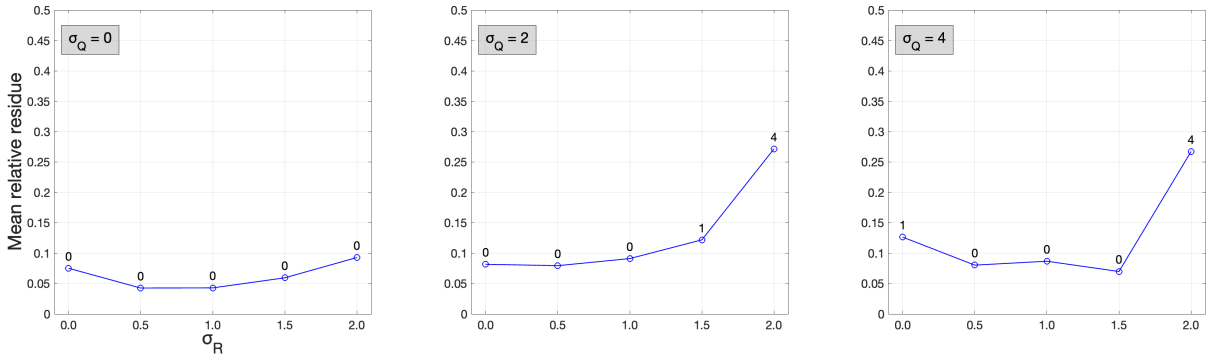


Figure 6.14: Mean relative residues of parameter estimates for the perturbed Lorenz system (6.4.28). The residues were averaged over 20 realizations of synthetic data for each combination of σ_Q and σ_R. Above each mean residue is shown the number of noise realizations for which the support of the sparse SINDy matrix was estimated incorrectly.

6.5 Comparisons and experiments

The remainder of this chapter is devoted to comparing the Deep Kalman Filter with alternative data assimilation approaches introduced earlier. We first provide a qualitative comparison with KalmanNet, highlighting both conceptual similarities and key methodological differences. We then introduce a numerical experimental framework designed to assess the impact of predictor accuracy on filtering performance, thereby illustrating how the Deep Kalman Filter leverages its ability to optimize and extend the forward operator in partially unknown models.

6.5.1 Deep Kalman Filter and KalmanNet

In this section, we provide a qualitative comparison between the Deep Kalman Filter and KalmanNet, structured around four key aspects.

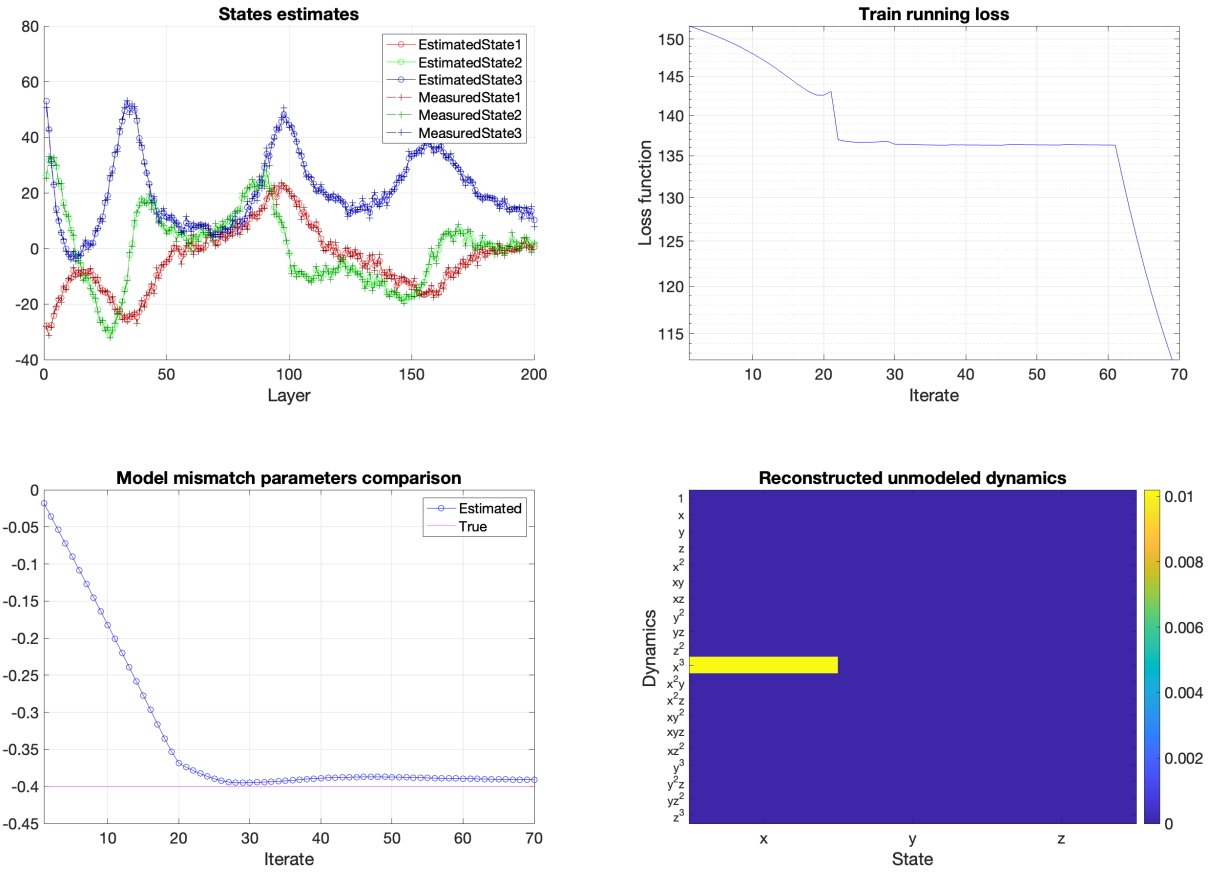


Figure 6.15: Top left: states estimates (circles) and given noisy observations (crosses) for the Lorenz system (6.4.28). Top right: train running loss decrease. Bottom left: train running comparison between estimated (circles) and true (line) p parameter. Bottom right: reconstructed sparse SINDy matrix.

- **Interpretability.** From a methodological standpoint, the Deep Kalman Filter and KalmanNet share several important similarities. Both provide learning-enhanced frameworks for data assimilation within the Kalman filtering paradigm, alleviating some of the restrictive assumptions of the classical formulation while retaining a clear predictor-corrector structure. In both approaches, learning primarily targets the construction of the gain matrices $\mathcal{K}_G^{(k)}$, which govern the correction step and compensate for unknown or misspecified noise statistics. The Deep Kalman Filter, however, extends this philosophy further by also allowing the forward operator and the observation model to be refined through the learning of shared parameters p, C, and the sparse SINDy coefficients s. This additional modeling flexibility, combined with the regularized training strategy described in Section 6.4.2, enables the Deep Kalman Filter to explicitly distinguish between deterministic modeling errors and stochastic uncertainty, and to adapt the corresponding components accordingly. In contrast, KalmanNet absorbs all sources of discrepancy, both deterministic and stochastic, into the learned gain matrices. As a result, the Deep Kalman Filter offers a higher degree of interpretability and modularity, particularly when model correction and uncertainty quantification must be disentangled.

- **Consistency.** Both algorithms exhibit a degree of consistency with the classical Kalman

filter in regimes where the model-based solution is optimal. In KalmanNet, this consistency is primarily demonstrated empirically through convergence of the mean squared error in numerical experiments. The Deep Kalman Filter, by contrast, is explicitly designed to mirror the analytical structure of the Kalman filter. This design choice is reflected in its training objective, which closely parallels the optimization problems underlying the model-based correction step, as discussed in Section 6.4.1. Moreover, the truncated backpropagation strategy ensures that information flow during training remains aligned with the causal structure of the filtering recursion. In comparison, KalmanNet relies on a purely data-driven loss optimized via standard backpropagation through time, without explicitly enforcing consistency with the analytical Kalman filtering objectives.

- **Architecture and training.** At a high level, KalmanNet combines an auto-regressive filtering structure with recurrent neural components. The auto-regressive aspect mirrors the Kalman predictor-corrector recursion, while the recurrent GRU modules are responsible for producing the gain matrices. Training KalmanNet therefore constitutes a canonical application of backpropagation through time. The Deep Kalman Filter, on the other hand, fully embraces the algorithm unrolling paradigm by focusing on the auto-regressive structure of Kalman filtering and untying the gain matrices across layers. At the same time, the shared parameters of the forward and observation models are optimized through backpropagation through time, making the Deep Kalman Filter an instance that combines both facets of deep unfolding, as presented in Chapter 1. From a data perspective, KalmanNet typically requires access to full state trajectories during training, whereas the Deep Kalman Filter can be trained using only observation data $y(k)$, which often reside in a significantly lower-dimensional space.

- **Computational cost.** Owing to its close correspondence with the model-based Kalman filter, the Deep Kalman Filter has essentially the same online computational cost as its classical counterpart, provided that the gain matrices can be computed offline. This assumption is valid in specific settings, such as problems with a known time horizon and fixed model parameters, but may not hold in more adaptive or nonstationary scenarios. KalmanNet offers greater flexibility in this regard, as it constructs the gain matrices online through its recurrent architecture and does not require a predefined horizon. The trade-off is an increased online computational burden associated with the GRU-based gain computation, which, while avoiding explicit matrix inversions, introduces additional neural network evaluations at each filtering step.

6.5.2 Numerical experiments: predictor impact on the performance of Data Assimilation

One of the defining features of the Deep Kalman Filter is its ability to correct and extend the forward operator f, which we will hereafter refer to simply as the predictor. To illustrate the importance of predictor accuracy in the data assimilation process, we evaluate the performance of three methods, the Ensemble Kalman Filter (EnKF), the Deep Kalman Filter (DKF), and conditional diffusion models (CDM), under two distinct predictor configurations: one accurate and one inaccurate. In particular, we again consider a perturbed Lorenz '63 ODE system:

$$\begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.05 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -7 & 7 & 0 \\ 28 & -1 & 0 \\ 0 & 0 & -\frac{8}{3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ -xz \\ xy \end{bmatrix} + \begin{bmatrix} \varepsilon x^3 \\ 0 \\ 0 \end{bmatrix} \tag{6.5.1}$$

as the true model that generates the observations to be assimilated. As before, we set $\varepsilon = 0.01$. Likewise, the associated partially unknown model is:

$$\begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.05 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -7 & 7 & 0 \\ 28 & -1 & 0 \\ 0 & 0 & -\frac{8}{3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ -xz \\ xy \end{bmatrix} \tag{6.5.2}$$

The *accurate* predictor $f = f_\varepsilon$ is obtained by applying a single step of a fourth-order Runge-Kutta (RK4) discretization to the true system in (6.5.1). The *inaccurate* predictor $\tilde{f} = f_0$ is constructed in an analogous manner by applying the same discretization scheme to the approximate system in (6.5.2). These two predictors are used in conjunction with each of the three data assimilation methods considered in this study. To generate the trajectories used for data assimilation, we integrate the true system (6.5.1) using the Python numerical solver `DOPRI5`, under varying levels of process and observation noise, namely $\sigma_Q \in \{0, 0.15, 0.3\}$ and $\sigma_R \in \{0, 0.33, 0.66, 1\}$. Initial state uncertainty is introduced by sampling from a Gaussian distribution with standard deviation $\sigma_P = 0.05$. The initial condition for (6.5.1) is set to $x(0) = -5.686$, $y(0) = -8.492$, and $z(0) = 17.845$. The system is integrated with time step $\Delta t = 10^{-3}$ for $K = 100$ steps. All three state components are observed, corresponding to $C(= h) = \mathbb{1}_3$. Figure 6.16 shows some of the trajectories for different noise levels.

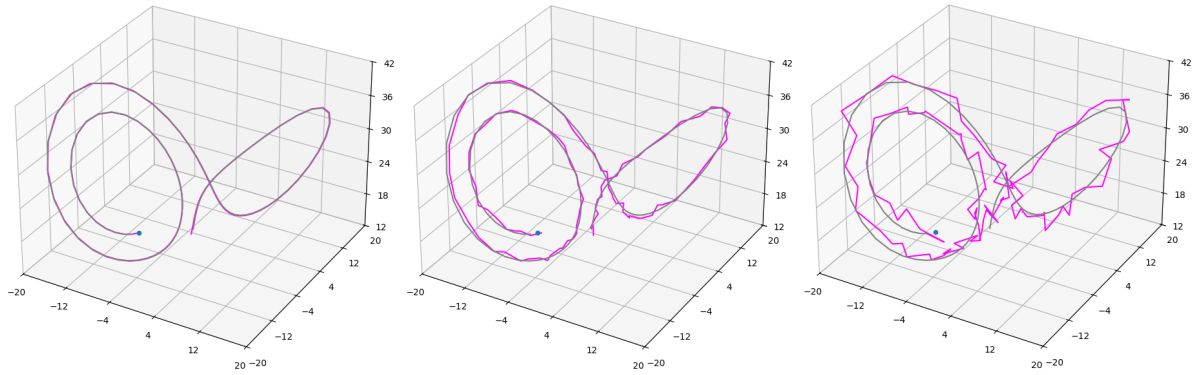


Figure 6.16: Generated trajectories across different noise levels. The reference noiseless trajectory is shown in gray while the noisy observations are shown in magenta. The initial state is marked in green. Left: low noise ($\sigma_Q = 0.15, \sigma_R = 0$). Center: medium noise ($\sigma_Q = 0.15, \sigma_R = 0.33$). Right: high noise ($\sigma_Q = 0.3, \sigma_R = 1$).

We now describe the specific configurations adopted for each method:

- **EnKF.** The Ensemble Kalman Filter is used as a benchmark, as it provides a favorable compromise between accuracy and computational cost in nonlinear systems. We use an ensemble of $N = 1000$ particles, and conduct experiments with both the accurate predictor f and the inaccurate predictor $\tilde{f}$.

- **DKF.** The Deep Kalman Filter is trained using complete backpropagation on the gain matrices $\mathcal{K}_G^{(k)}$, with learning rate $\mu_{\mathcal{K}} = 0.2$ and batch size 30. The shared parameters p and C are not optimized. We set $\lambda_K = 1$ and $\lambda_D = 10^3$, and employ early stopping based on the whiteness of the prediction error. Training is performed either using the accurate predictor f with SINDy disabled, or using the inaccurate predictor $\tilde{f}$ with SINDy enabled. Comparing these two settings enables a direct evaluation of the model discovery capabilities of the Deep Kalman Filter.

- **CDM.** The score network $s_\theta(x, t, y)$ is implemented as a five-layer fully connected neural network with ReLU activations. In our previous notation, the input to the network is the concatenated vector $[\tilde{x}, t_{\text{emb}}, y]$, where the time embedding is defined as:

$$t_{\text{emb}} = [t - 0.5, \cos(2t), \sin(2t), -\cos(4t)]$$

  As a result, in this experiment the input dimension is 10. The hidden layers have dimension 50, and the output dimension matches the state dimension, namely 3.
  To reduce training time at each filtering step, the network is warm-started at the beginning of each experiment using a learning rate of 10^{-3} for 500 epochs with batch size 256. The resulting weights are then used as initialization during each filtering step, allowing training to converge using only 200 epochs and a reduced learning rate of 10^{-4}.
  The reverse-time SDE is integrated using the Euler–Maruyama scheme with time increments $\Delta\tau = 10^{-2}$. We adopt the variance-preserving formulation with parameters $T = 1$, $\alpha = 1$, $\mu = 2$, and $\beta(t) = 0.01 + 19.99t$. With these settings, a single filtering step requires less than one second. As with the EnKF, experiments are conducted on an ensemble of $N = 1000$ particles using both the accurate predictor f and the inaccurate predictor $\tilde{f}$.

For each experiment and noise configuration, the accuracy of the estimated trajectories is assessed by computing the Root Mean Squared Error (RMSE) with respect to a reference noiseless trajectory. The reported RMSE values are then obtained by averaging over 15 independent filtering runs, each corresponding to a different realization of process and observation noise. The results are shown in Figure 6.17. For both the EnKF and CDM,

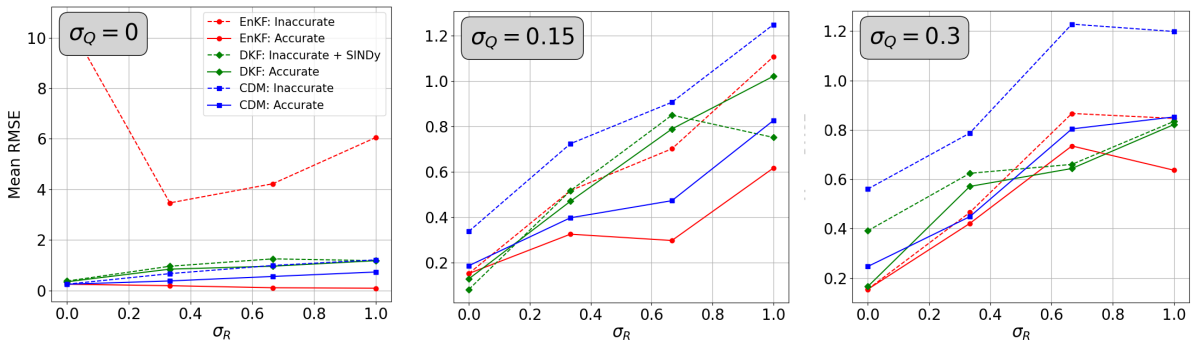


Figure 6.17: Mean RMSE across different noise levels. The reported values are averaged over 15 synthetic data realizations for each combination of σ_Q and σ_R. Solid lines correspond to the use of the accurate predictor f, while dashed lines indicate the inaccurate predictor $\tilde{f}$. Results for EnKF, DKF, and CDM are shown in red, green, and blue, respectively.

the use of the inaccurate predictor $\tilde{f}$ leads to a noticeable degradation in performance, reflected by a clear increase in RMSE. This effect is especially pronounced for the EnKF in the absence of process noise, i.e. $\sigma_Q = 0$, underscoring its strong reliance on model accuracy. As the process noise level increases, however, the EnKF with an inaccurate predictor becomes progressively more competitive, indicating that deterministic model errors are effectively absorbed and compensated for as stochastic uncertainty. In contrast, the performance of CDM does not exhibit a comparable improvement as the process noise grows. Notably, the DKF shows a markedly different behavior: its performance remains largely insensitive to the initial use of the inaccurate predictor $\tilde{f}$, which is rapidly corrected through the SINDy-based model adaptation during training. Lastly, Figure 6.18 reports the corresponding results for the mean generalized variance [133, 2] computed for the two ensemble-based methods. As a scalar summary of the ensemble covariance, the generalized variance provides a global measure of the uncertainty associated with the state estimates over the entire filtering horizon. In particular, we observe that the accuracy of the predictor has a limited impact on the estimated state uncertainty. By contrast, increasing levels of process noise lead to a marked growth in uncertainty for the EnKF, whereas the uncertainty estimates produced by CDM remain largely insensitive to such variations.

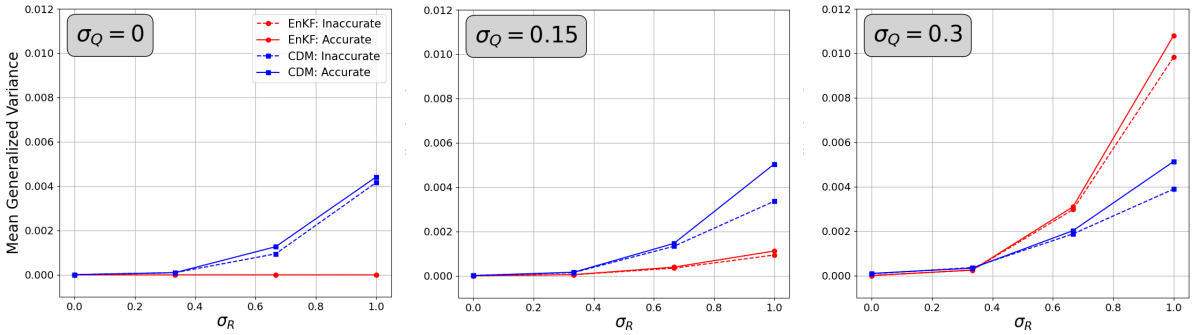


Figure 6.18: Mean generalized variance across different noise levels. The reported values are averaged over 15 synthetic data realizations for each combination of σ_Q and σ_R. Solid lines correspond to the use of the accurate predictor f, while dashed lines indicate the inaccurate predictor $\tilde{f}$. Results for EnKF and CDM are shown in red and blue, respectively.

Appendix

6.A DKF backpropagation details

This section derives the backpropagation relations for the Deep Kalman Filter introduced in Section 6.4. In particular, we present a detailed analytical treatment of both the *complete* and *truncated* backpropagation algorithms. For conciseness, not all relations are derived explicitly; instead, we focus on a small selection of key quantities.
The state correction relation (6.4.3) written elementwise is in the form:

$$\hat{x}(k)_i = \hat{x}(k \mid k-1)_i + \sum_j (\mathcal{K}_G^{(k)})_{i,j} \left[y(k)_j - \sum_q C_{j,q}^{(k)} \hat{x}(k \mid k-1)_q \right] \tag{6.A.1}$$

In order to assemble the backpropagation algorithm for the Deep Kalman Filter architecture and loss (6.4.6), we need four key components:

$$\frac{\partial \hat{x}(k)_i}{\partial \hat{x}(k-1)_{\bar{i}}} \qquad \frac{\partial \hat{x}(k)_i}{\partial (\mathcal{K}_G^{(k)})_{\bar{i},\bar{j}}} \qquad \frac{\partial \hat{x}(k)_i}{\partial p^{(k)}} \qquad \frac{\partial \hat{x}(k)_i}{\partial C_{\bar{i},\bar{j}}^{(k)}}$$

The first quantity is the derivative of the state at time k with respect to the state at time $k-1$. This term is central to the backpropagation procedure, as it enables the propagation of gradients from one layer to the previous one. The remaining three quantities are associated with the network's learned parameters and are required to assemble the gradients of the loss function. From (6.A.1), these expressions follow directly:

$$\begin{aligned}
\tfrac{\partial \hat{x}(k)_i}{\partial \hat{x}(k-1)_{\bar{i}}} &= \partial_{x_{\bar{i}}} \hat{x}(k \mid k-1)_i - \sum_j (\mathcal{K}_G^{(k)})_{i,j} \left[\sum_q C_{j,q}^{(k)} \partial_{x_{\bar{i}}} \hat{x}(k \mid k-1)_q \right] \\
&= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k)} C^{(k)} \right) \nabla_x \hat{x}(k \mid k-1) \right]_{i,\bar{i}} \\
\tfrac{\partial \hat{x}(k)_i}{\partial (\mathcal{K}_G^{(k)})_{\bar{i},\bar{j}}} &= \left[y(k)_{\bar{j}} - \sum_q C_{\bar{j},q}^{(k)} \hat{x}(k \mid k-1)_q \right] \delta_{i,\bar{i}} = \left[y(k) - C^{(k)} \hat{x}(k \mid k-1) \right]_{\bar{j}} \delta_{i,\bar{i}} \\
\tfrac{\partial \hat{x}(k)_i}{\partial p^{(k)}} &= \partial_p \hat{x}(k \mid k-1)_i - \sum_j (\mathcal{K}_G^{(k)})_{i,j} \left[\sum_q C_{j,q}^{(k)} \partial_p \hat{x}(k \mid k-1)_q \right] \\
&= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k)} C^{(k)} \right) \nabla_p \hat{x}(k \mid k-1) \right]_i \\
\tfrac{\partial \hat{x}(k)_i}{\partial C_{\bar{i},\bar{j}}^{(k)}} &= (\mathcal{K}_G^{(k)})_{i,\bar{i}} \hat{x}(k \mid k-1)_{\bar{j}}
\end{aligned} \tag{6.A.2}$$

In particular, using (6.A.2), one obtains the recursive relations:

$$\begin{cases}
\nabla_{\hat{x}(K)}\mathcal{E}_{\lambda_K} &= \lambda_K\left(\hat{x}(K) - x(K)\right) \\
\nabla_{\hat{x}(l)}\mathcal{F}^l &= -C^{(l)^\top}\left[R_l^{-1}\left(y(l) - C^{(l)}\hat{x}(l)\right)\right] \qquad \forall l = 1,\dots,K \\
\nabla_{\hat{x}(l-1)}\mathcal{G}^l &= \left(-\mathcal{K}_G^{(l)}C^{(l)}\nabla_x\hat{x}(l \mid l-1)\right)^\top\left[P_l^{-1}\mathcal{K}_G^{(l)}\left(y(l) - C^{(l)}\hat{x}(l \mid l-1)\right)\right] \qquad \forall l = 2,\dots,K \\
\nabla_{\hat{x}(k)}\mathcal{E}_{\lambda_K} &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k+1)}C^{(k+1)}\right)\nabla_x\hat{x}(k+1 \mid k)\right]^\top\nabla_{\hat{x}(k+1)}\mathcal{E}_{\lambda_K} \qquad \forall k = 1,\dots,K-1 \\
\nabla_{\hat{x}(k)}\mathcal{F}^l &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k+1)}C^{(k+1)}\right)\nabla_x\hat{x}(k+1 \mid k)\right]^\top\nabla_{\hat{x}(k+1)}\mathcal{F}^l \quad \forall k = 1,\dots,l-1, \quad \forall l = 1,\dots,K \\
\nabla_{\hat{x}(k)}\mathcal{G}^l &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k+1)}C^{(k+1)}\right)\nabla_x\hat{x}(k+1 \mid k)\right]^\top\nabla_{\hat{x}(k+1)}\mathcal{G}^l \quad \forall k = 1,\dots,l-2, \quad \forall l = 2,\dots,K
\end{cases} \tag{6.A.3}$$

as well as the partial gradients:

$$\begin{cases}
\nabla_{\mathcal{K}_G^{(k)}}\mathcal{E}_{\lambda_K} &= \nabla_{\hat{x}(k)}\mathcal{E}_{\lambda_K}\left(y(k) - C^{(k)}\hat{x}(k \mid k-1)\right)^\top \qquad \forall k = 1,\dots,K \\
\nabla_{\mathcal{K}_G^{(k)}}\mathcal{F}^l &= \nabla_{\hat{x}(k)}\mathcal{F}^l\left(y(k) - C^{(k)}\hat{x}(k \mid k-1)\right)^\top \qquad \forall k = 1,\dots,l, \quad \forall l = 1,\dots,K \\
\nabla_{\mathcal{K}_G^{(l)}}\mathcal{G}^l &= \left[P_l^{-1}\mathcal{K}_G^{(l)}\left(y(l) - C^{(l)}\hat{x}(l \mid l-1)\right)\right]\left(y(l) - C^{(l)}\hat{x}(l \mid l-1)\right)^\top \qquad \forall l = 1,\dots,K \\
\nabla_{\mathcal{K}_G^{(k)}}\mathcal{G}^l &= \nabla_{\hat{x}(k)}\mathcal{G}^l\left(y(k) - C^{(k)}\hat{x}(k \mid k-1)\right)^\top \qquad \forall k = 1,\dots,l-1, \quad \forall l = 1,\dots,K \\
\nabla_{\mathcal{K}_G^{(k)}}\mathcal{H}_{\lambda_D} &= \lambda_D T^{(k)} \qquad \forall k = 1,\dots,K \\
\nabla_{p^{(k)}}\mathcal{E}_{\lambda_K} &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k)}C^{(k)}\right)\nabla_p\hat{x}(k \mid k-1)\right]^\top\nabla_{\hat{x}(k)}\mathcal{E}_{\lambda_K} \qquad k = 1,\dots,K \\
\nabla_{p^{(k)}}\mathcal{F}^l &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k)}C^{(k)}\right)\nabla_p\hat{x}(k \mid k-1)\right]^\top\nabla_{\hat{x}(k)}\mathcal{F}^l \qquad \forall k = 1,\dots,l-1, \quad \forall l = 1,\dots,K \\
\nabla_{p^{(l)}}\mathcal{G}^l &= \left(-\mathcal{K}_G^{(l)}C^{(l)}\nabla_p\hat{x}(l \mid l-1)\right)^\top\left[P_l^{-1}\mathcal{K}_G^{(l)}\left(y(l) - C^{(l)}\hat{x}(l \mid l-1)\right)\right] \qquad \forall l = 1,\dots,K \\
\nabla_{p^{(k)}}\mathcal{G}^l &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k)}C^{(k)}\right)\nabla_p\hat{x}(k \mid k-1)\right]^\top\nabla_{\hat{x}(k)}\mathcal{G}^l \qquad \forall k = 1,\dots,l-1, \quad \forall l = 1,\dots,K \\
\nabla_{C^{(k)}}\mathcal{E}_{\lambda_K} &= -\left(\mathcal{K}_G^{(k)^\top}\nabla_{\hat{x}(k)}\mathcal{E}_{\lambda_K}\right)\hat{x}(k \mid k-1)^\top \qquad \forall k = 1,\dots,K \\
\nabla_{C^{(l)}}\mathcal{F}^l &= -R_l^{-1}\left(y(l) - C^{(l)}\hat{x}(l \mid l-1)\right)\hat{x}(l \mid l-1)^\top - \left(\mathcal{K}_G^{(l)^\top}\nabla_{\hat{x}(l)}\mathcal{F}^l\right)\hat{x}(l \mid l-1)^\top \quad \forall l = 1,\dots,K \\
\nabla_{C^{(k)}}\mathcal{F}^l &= -\left(\mathcal{K}_G^{(k)^\top}\nabla_{\hat{x}(k)}\mathcal{F}^l\right)\hat{x}(k \mid k-1)^\top \qquad \forall k = 1,\dots,l-1, \quad \forall l = 1,\dots,K \\
\nabla_{C^{(l)}}\mathcal{G}^l &= -\left[\mathcal{K}_G^{(l)^\top}P_l^{-1}\mathcal{K}_G^{(l)}\left(y(l) - C^{(l)}\hat{x}(l \mid l-1)\right)\right]\hat{x}(l \mid l-1)^\top \qquad \forall l = 1,\dots,K \\
\nabla_{C^{(k)}}\mathcal{G}^l &= -\left(\mathcal{K}_G^{(k)^\top}\nabla_{\hat{x}(k)}\mathcal{G}^l\right)\hat{x}(k \mid k-1)^\top \qquad \forall k = 1,\dots,l-1, \quad \forall l = 1,\dots,K
\end{cases} \tag{6.A.4}$$

where $T^{(k)} \in \mathcal{M}_{n\times m}(\mathbb{R})$ is given by $T_{ij}^{(k)} = \left(D^\top D v^{ij}\right)_k$ and v^{ij} is defined as in (6.4.6). In particular, (6.A.4) describes the *complete* backpropagation algorithm. In contrast, the *truncated* backpropagation algorithm uses the following partial gradients, as motivated in Section 6.4.1:

$$
\begin{cases}
\nabla_{\mathcal{K}_G^{(K)}} \mathcal{E}_{\lambda_K} &= \nabla_{\hat{x}(K)} \mathcal{E}_{\lambda_K} \left(y(K) - C^{(K)} \hat{x}(K \mid K-1) \right)^\top \\
\nabla_{\mathcal{K}_G^{(k)}} \mathcal{E}_{\lambda_K} &= \mathbb{0}_{n \times m} \qquad \forall k = 1, \ldots, K-1 \\
\nabla_{\mathcal{K}_G^{(l)}} \mathcal{F}^l &= \nabla_{\hat{x}(l)} \mathcal{F}^l \left(y(l) - C^{(l)} \hat{x}(l \mid l-1) \right)^\top \qquad \forall l = 1, \ldots, K \\
\nabla_{\mathcal{K}_G^{(k)}} \mathcal{F}^l &= \mathbb{0}_{n \times m} \qquad \forall k = 1, \ldots, l-1, \quad \forall l = 1, \ldots, K \\
\nabla_{\mathcal{K}_G^{(l)}} \mathcal{G}^l &= \left[P_l^{-1} \mathcal{K}_G^{(l)} \left(y(l) - C^{(l)} \hat{x}(l \mid l-1) \right) \right] \left(y(l) - C^{(l)} \hat{x}(l \mid l-1) \right)^\top \qquad \forall l = 1, \ldots, K \\
\nabla_{\mathcal{K}_G^{(k)}} \mathcal{G}^l &= \mathbb{0}_{n \times m} \qquad \forall k = 1, \ldots, l-1, \quad \forall l = 1, \ldots, K \\
\nabla_{\mathcal{K}_G^{(k)}} \mathcal{H}_{\lambda_D} &= \lambda_D T^{(k)} \qquad \forall k = 1, \ldots, K \\
\nabla_{p^{(k)}} \mathcal{E}_{\lambda_K} &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k)} C^{(k)} \right) \nabla_p \hat{x}(k \mid k-1) \right]^\top \nabla_{\hat{x}(k)} \mathcal{E}_{\lambda_K} \qquad \forall k = 1, \ldots, K \\
\nabla_{p^{(k)}} \mathcal{F}^l &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k)} C^{(k)} \right) \nabla_p \hat{x}(k \mid k-1) \right]^\top \nabla_{\hat{x}(k)} \mathcal{F}^l \qquad \forall k = 1, \ldots, l-1, \quad \forall l = 1, \ldots, K \\
\nabla_{p^{(l)}} \mathcal{G}^l &= \left(-\mathcal{K}_G^{(l)} C^{(l)} \nabla_p \hat{x}(l \mid l-1) \right)^\top \left[P_l^{-1} \mathcal{K}_G^{(l)} \left(y(l) - C^{(l)} \hat{x}(l \mid l-1) \right) \right] \qquad l = 1, \ldots, K \\
\nabla_{p^{(k)}} \mathcal{G}^l &= \left[\left(\mathbb{1}_n - \mathcal{K}_G^{(k)} C^{(k)} \right) \nabla_p \hat{x}(k \mid k-1) \right]^\top \nabla_{\hat{x}(k)} \mathcal{G}^l \qquad \forall k = 1, \ldots, l-1, \quad \forall l = 1, \ldots, K \\
\nabla_{C^{(k)}} \mathcal{E}_{\lambda_K} &= - \left(\mathcal{K}_G^{(k)^\top} \nabla_{\hat{x}(k)} \mathcal{E}_{\lambda_K} \right) \hat{x}(k \mid k-1)^\top \qquad \forall k = 1, \ldots, K \\
\nabla_{C^{(l)}} \mathcal{F}^l &= -R_l^{-1} \left(y(l) - C^{(l)} \hat{x}(l \mid l-1) \right) \hat{x}(l \mid l-1)^\top - \left(\mathcal{K}_G^{(l)^\top} \nabla_{\hat{x}(l)} \mathcal{F}^l \right) \hat{x}(l \mid l-1)^\top \quad \forall l = 1, \ldots, K \\
\nabla_{C^{(k)}} \mathcal{F}^l &= - \left(\mathcal{K}_G^{(k)^\top} \nabla_{\hat{x}(k)} \mathcal{F}^l \right) \hat{x}(k \mid k-1)^\top \qquad \forall k = 1, \ldots, l-1, \quad \forall l = 1, \ldots, K \\
\nabla_{C^{(l)}} \mathcal{G}^l &= - \left[\mathcal{K}_G^{(l)^\top} P_l^{-1} \mathcal{K}_G^{(l)} \left(y(l) - C^{(l)} \hat{x}(l \mid l-1) \right) \right] \hat{x}(l \mid l-1)^\top \qquad \forall l = 1, \ldots, K \\
\nabla_{C^{(k)}} \mathcal{G}^l &= - \left(\mathcal{K}_G^{(k)^\top} \nabla_{\hat{x}(k)} \mathcal{G}^l \right) \hat{x}(k \mid k-1)^\top \qquad \forall k = 1, \ldots, l-1, \quad \forall l = 1, \ldots, K
\end{cases}
\tag{6.A.5}
$$

Conclusions

In this dissertation, we investigated a broad class of modern machine learning architectures that seek to bridge the gap between traditional model-based methods and data-driven approaches. Motivated by the increasing need for methods that are both expressive and interpretable, we focused on the paradigm of deep unfolding as a unifying framework capable of reconciling these two perspectives. We showed that deep unfolding naturally encompasses two complementary manifestations, algorithm unrolling as an architectural design principle and backpropagation through time as an optimization mechanism, both of which recur across many recent developments in learning-enhanced signal processing and state estimation.

A central discussion point of this work is the identification of bilevel optimization as a rigorous mathematical foundation underlying deep unfolding. By framing learning problems in a bilevel setting, iterative algorithms can be systematically transformed into structured neural architectures whose parameters are optimized end-to-end while preserving the semantics of the original algorithm. This perspective not only clarifies the relationship between classical optimization and modern deep learning, but also provides a principled path toward highly interpretable models whose behavior can be directly linked to well-understood mathematical procedures.

The first application domain explored in this dissertation was audio processing, with a focus on source separation and detection. Within this context, we revisited nonnegative matrix factorization (NMF) and its variants as a powerful and flexible model-based framework. We extended automatic relevance determination (ARD) to the nonnegative matrix factorization deconvolution (NMFD) setting, demonstrating its effectiveness on standard benchmark datasets and highlighting its ability to promote parsimonious representations in time-frequency analysis. Building upon this foundation, we applied algorithm unrolling to NMF-based methods, resulting in competitive deep architectures such as PAD-NMF and Deep-NMFD. These models overcome key limitations of traditional NMF, notably slow convergence and limited adaptability, while retaining interpretability. Importantly, we analyzed how time correlations are embedded differently in these architectures: PAD-NMF relies on Hankel-augmented representations that encode temporal structure as extended features, whereas Deep-NMFD exploits the intrinsic convolutional structure of NMFD to model temporal dependencies directly. Through extensive experimentation, we showed that these approaches are complementary, each excelling under different practical constraints and application scenarios.

The second major theme of this dissertation concerned state estimation in dynamical systems, a domain historically dominated by Kalman filtering and its extensions. After reviewing the Kalman filter and ensemble-based variants, we emphasized the challenges

that arise in real-world applications, including model mismatch, nonlinearity, unknown noise statistics, and high dimensionality. Particle filters were introduced as a flexible alternative capable of addressing some of these issues, and were employed to design a novel audio tracking algorithm for continuous time-frequency pattern analysis. While powerful, particle filters also revealed intrinsic limitations, particularly their reliance on explicit likelihood models and their computational burden. By reframing state estimation as a data assimilation problem and allowing for learned components, we then explored a new generation of learning-enhanced data assimilation methods. Conditional diffusion models were presented as a data-driven alternative to particle filters, capable of sampling from complex conditional posteriors without requiring explicit likelihood evaluations. In parallel, KalmanNet and the Deep Kalman Filter were introduced as learned extensions of classical Kalman filtering that remain closely tied to its predictor-corrector structure. These methods exemplify deep unfolding in both of its forms: backpropagation through time enables the optimization of temporally structured models, while algorithm unrolling preserves the interpretability and consistency of the filtering process. Both architectures focus on learning gain matrices to overcome classical assumptions such as linearity and known Gaussian noise, while maintaining a clear connection to the original Kalman framework. A distinctive contribution of this dissertation is the review and slight extension of the Deep Kalman Filter architecture, which further generalizes Kalman filtering by enabling the data-driven enhancement of the dynamical model itself. By jointly optimizing gain matrices and components of the governing dynamical system within a regularized learning framework, the Deep Kalman Filter is able to disentangle deterministic modeling errors from stochastic uncertainty. This capability proved crucial in scenarios involving inaccurate predictors, where the method demonstrated robustness and adaptability beyond that of ensemble-based and purely data-driven alternatives. Through carefully designed numerical experiments, we highlighted the importance of model fidelity in data assimilation and showed how learning-enhanced approaches can compensate for, and even correct, structural deficiencies in the underlying model.

Overall, this dissertation demonstrates that deep unfolding provides a powerful conceptual and practical framework for merging model-based reasoning with data-driven learning. By grounding neural architectures in established algorithms and optimization principles, it is possible to achieve a balance between expressiveness, interpretability, and computational efficiency. The results presented here suggest that learning-enhanced formulations of classical methods, whether in audio processing or state estimation, are not merely heuristic extensions, but principled generalizations that preserve the strengths of both paradigms. As the boundaries between model-based and data-driven approaches continue to blur, deep unfolding stands out as a compelling direction for the design of next-generation algorithms in signal processing, dynamical systems, and beyond.

Lastly, several avenues for future research emerge from the results presented in this dissertation. From a theoretical perspective, an important open question concerns the convergence properties and stability guarantees of deep unfolded architectures, particularly in the widely used untied setting where parameters are allowed to vary across layers. While algorithm unrolling inherits intuition from the underlying iterative procedures, the introduction of learned and untied parameters breaks many of the classical assumptions that ensure convergence. Developing a rigorous analysis for such networks, possibly within the broader framework of bilevel optimization and dynamical systems, would therefore

represent a significant step toward establishing stronger theoretical foundations for deep unfolding. In the context of NMF-based models for audio processing, further investigation of the role of the β parameter in the multiplicative update rules is also warranted. The architectures explored in this work, namely Deep-NMF, PAD-NMF and Deep-NMFD, have focused exclusively on the case $\beta = 1$, corresponding to the Kullback-Leibler divergence that is commonly adopted in audio applications. Extending these models to other values of β within the β-divergence family may provide additional flexibility and robustness, potentially improving performance in scenarios involving different noise statistics or signal characteristics. Finally, in the context of learning-enhanced state estimation, the Deep Kalman Filter framework could benefit from more expressive representations of the underlying dynamical model. In particular, integrating learned operator networks within the prediction step may offer a promising direction for capturing complex and possibly infinite-dimensional dynamics, thereby extending the applicability of the approach to a broader class of systems. Taken together, these directions highlight how the deep unfolding paradigm continues to open both theoretical and practical challenges, suggesting a rich landscape for future developments at the intersection of optimization, machine learning, and dynamical systems.

Bibliography

[1] Sebastian Ament and Carla Gomes. On the optimality of backward regression: Sparse recovery and subset selection. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5599–5603. IEEE, 2021.

[2] Theodore W. Anderson. *An Introduction to Multivariate Statistical Analysis*. Wiley, 3 edition, 2003.

[3] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

[4] Athanasios C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. SIAM, 2005.

[5] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[6] Nicholas Assimakis and Maria Adam. Kalman filter riccati equation for the prediction, estimation, and smoothing error covariance matrices. *ISRN Computational Mathematics*, 2013:1–7, 2013.

[7] Nicholas Assimakis and Maria Adam. Iterative and algebraic algorithms for the computation of the steady state Kalman filter gain. *ISRN Applied Mathematics*, 2014:1–10, 2014.

[8] Wafa Barkhoda, Amjad Seyedi, Nicolas Gillis, and Fardin Akhlaghian Tab. Instance-wise distributionally robust nonnegative matrix factorization. *Pattern Recognition*, 169:111732, 2026.

[9] Julian M. Becker and Christian Rohlfing. Custom sized non-negative matrix factor deconvolution for sound source separation. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2124–2128, 2014.

[10] Murat Belge, Misha E Kilmer, and Eric L Miller. Efficient determination of multiple regularization parameters in a generalizedl-curve framework. *Inverse problems*, 18(4):1161, 2002.

[11] Dimitri Bertsekas. *Nonlinear Programming: Second Edition*. Athena Scientific, 1999.

[12] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484, 2023.

[13] Marc Bocquet, Julien Brajard, Alberto Carrassi, and Laurent Bertino. Data assimilation as a learning problem. *Nonlinear Processes in Geophysics*, 27(4):743–768, 2020.

[14] Silvia Bonettini, Giorgia Franchini, Danilo Pezzi, and Marco Prato. Linesearch-enhanced forward–backward methods for inexact nonconvex scenarios. *SIAM Journal on Imaging Sciences*, 18(2):1314–1343, 2025.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[16] Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the national academy of sciences*, 101(12):4164–4169, 2004.

[17] Steven L. Brunton, Marko Budisic, Eurika Kaiser, and J. Nathan Kutz. Modern koopman theory for dynamical systems. *SIAM Review*, 64(2):229–340, 2022.

[18] Dan Butnariu and Elena Resmerita. Bregman distances, totally convex functions, and a method for solving operator equations in banach spaces. In *Abstract & Applied Analysis*, 2006.

[19] Daniela Calvetti and Erkki Somersalo. *Bayesian scientific computing*, volume 215. Springer, 2023.

[20] Sibo Cheng, César Quilodrán-Casas, Said Ouala, Alban Farchi, Che Liu, Pierre Tandeo, Ronan Fablet, Didier Lucor, Bertrand Iooss, Julien Brajard, et al. Machine learning with data assimilation and uncertainty quantification for dynamical systems: a review. *IEEE/CAA Journal of Automatica Sinica*, 10(6):1361–1387, 2023.

[21] Erik Chinellato and Fabio Marcuzzi. State estimation of partially unknown dynamical systems with a deep Kalman filter. In *Computational Science – ICCS 2024*, pages 307–321. Springer Nature Switzerland, 2024.

[22] Erik Chinellato and Fabio Marcuzzi. Hit detection in audio mixtures by means of a physics-aware deep-nmf algorithm. *Mechanical Systems and Signal Processing*, 224:112162, 2025.

[23] Erik Chinellato and Fabio Marcuzzi. State, parameters and hidden dynamics estimation with the deep Kalman filter: regularization strategies. *Journal of Computational Science*, 87:102569, 2025.

[24] Erik Chinellato and Fabio Marcuzzi. Deep unfolding for scientific computing on embedded systems. In *Scientific Machine Learning: Emerging Topics*. Springer, 2026 (in press).

[25] Erik Chinellato, Fabio Marcuzzi, and Paolo Martin. Real-time generation of a targeted clean audio sequence from source separation of noisy environmental mixtures using a deep nonnegative matrix factorization on iot devices. *Smart Innovation, Systems and Technologies*, 404 SIST:265–275, 2024.

[26] Erik Chinellato, Fabio Marcuzzi, and Simone Pierobon. Physics-aware soft sensors for embedded digital twins. In *Proceedings of 9th International Congress on Information and Communication Technology (ICICT 2024)*, pages 417–427. Springer LNNS, 2024.

[27] Erik Chinellato, Paolo Martin, Laura Rinaldi, and Fabio Marcuzzi. Exploiting scientific machine learning on embedded digital twins. In *Emerging Technologies in Computational Sciences for Industry, Sustainability and Innovation*, pages 191–205. Springer Nature Switzerland, 2025.

[28] Andrzej Cichocki, Rafal Zdunek, and Shun-ichi Amari. Csiszár's divergences for non-negative matrix factorization: Family of new algorithms. In Justinian Rosca, Deniz Erdogmus, José C. Príncipe, and Simon Haykin, editors, *Independent Component Analysis and Blind Signal Separation*, pages 32–39. Springer Berlin Heidelberg, 2006.

[29] Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. Hierarchical als algorithms for nonnegative matrix and 3d tensor factorization. *Independent Component Analysis and Signal Separation*, pages 169–176, 2007.

[30] Agnimitra Dasgupta, Alexsander Marciano da Cunha, Ali Fardisi, Mehrnegar Aminy, Brianna Binder, Bryan Shaddy, and Assad A Oberai. Unifying and extending diffusion models through pdes for solving inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 448:118431, 2026.

[31] Pierre De Handschutter and Nicolas Gillis. A consistent and flexible framework for deep matrix factorizations. *Pattern Recognition*, 134:109102, 2023.

[32] Nicoletta Del Buono, Flavia Esposito, Laura Selicato, and Rafał Zdunek. Penalty hyperparameter optimization with diversity measure for nonnegative low-rank approximation. *Applied Numerical Mathematics*, 208:189–204, 2025.

[33] Monica Dessole, Marco Dell'Orto, and Fabio Marcuzzi. The lawson-hanson algorithm with deviation maximization: Finite convergence and sparse recovery. *Numerical Linear Algebra with Applications*, 2023.

[34] Chris Ding, Xiaofeng He, Horst D. Simon, and Rong Jin. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM International Conference on Data Mining (SDM)*, pages 606–610, 2005.

[35] David Donoho and Victoria Stodden. When does non-negative matrix factorization give a correct decomposition into parts? In *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003.

[36] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[37] Megan R. Ebers, Katherine M. Steele, and J. Nathan Kutz. Discrepancy modeling framework: Learning missing physics, modeling systematic residuals, and disambiguating between deterministic and random effects. *SIAM Journal on Applied Dynamical Systems*, 23(1):440–469, 2024.

[38] Carl Eckart and G. Marion Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[39] Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99(C5):10143–10162, 1994.

[40] Geir Evensen. *Data Assimilation: The Ensemble Kalman Filter*. Springer, 2009.

[41] Geir Evensen, Femke C Vossepoel, and Peter Jan Van Leeuwen. *Data assimilation fundamentals: A unified formulation of the state and parameter estimation problem*. Springer Nature, 2022.

[42] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[43] Massimo Fornasier, Valeriya Naumova, and Sergei V Pereverzyev. Parameter choice strategies for multipenalty regularization. *SIAM Journal on Numerical Analysis*, 52(4):1770–1794, 2014.

[44] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International conference on machine learning*, pages 1165–1173. PMLR, 2017.

[45] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, 2018.

[46] Cédric Févotte, Nancy Bertin, and Jean-Louis Durrieu. Nonnegative matrix factorization with the itakura-saito divergence: With application to music analysis. *Neural Computation*, 21(3):793–830, 2009.

[47] Cédric Févotte and Jérôme Idier. Algorithms for nonnegative matrix factorization with the beta-divergence. *Neural Computation*, 23(9):2421–2456, 2011.

[48] Marta Gatto and Fabio Marcuzzi. An algorithm for model-based denoising of input-output data. *Dolomites Research Notes on Approximation*, 12:73–85, 2019.

[49] Rudy Geelen, Stephen Wright, and Karen Willcox. Operator inference for non-intrusive model reduction with quadratic manifolds. *Computer Methods in Applied Mechanics and Engineering*, 403, 2023.

[50] Nicolas Gillis. *Nonnegative Matrix Factorization*. SIAM, 2020.

[51] Giulio G. Giusteri, Fabio Marcuzzi, and Laura Rinaldi. Replacing voids and localized parameter changes with fictitious forcing terms in boundary-value problems. *Results in Applied Mathematics*, 20, 2023.

[52] Karl Glasner. Data-driven learning of differential equations: combining data and model uncertainty. *Computational and Applied Mathematics*, 42(1), 2023.

[53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[54] Riccardo Grazzi, Luca Franceschi, Massimiliano Pontil, and Saverio Salzo. On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, pages 3748–3758. PMLR, 2020.

[55] Riccardo Grazzi, Massimiliano Pontil, and Saverio Salzo. Convergence properties of stochastic hypergradients. In *Proceedings of the 24th International Conference on Artifical Intelligence and Statistics (AISTATS)*, volume 130, 2021.

[56] Riccardo Grazzi, Massimiliano Pontil, and Saverio Salzo. Bilevel optimization with a lower-level contraction: Optimal sample complexity without warm-start. *Journal of Machine Learning Research*, 24(167):1–37, 2023.

[57] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning*, pages 399–406, 2010.

[58] Mohinder S. Grewal and Angus P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. John Wiley & Sons, 2014.

[59] Luigi Grippo and Marco Sciandrone. On the convergence of the block nonlinear Gauss–Seidel method under convex constraints. *Operations research letters*, 26(3):127–136, 2000.

[60] Shuting Guo, Liyan Luo, Mei Wang, Zou Zhou, Xiaofang Deng, Lu Bai, and Dancheng Zhao. Denoising algorithm of environmental sound fused nmf and omlsa in non-stationary noise environment. In *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 364–368, 2021.

[61] Mark H. Hansen and Bin Yu. Model selection and the principle of minimum description length. *Journal of the american statistical association*, 96(454):746–774, 2001.

[62] Per Christian Hansen. *Discrete Inverse Problems: Insight and Algorithms*. Society for Industrial and Applied Mathematics, 2010.

[63] Per Christian Hansen, Misha Elena Kilmer, and Rikke Høj Kjeldsen. Exploiting residual information in the parameter choice for discrete ill-posed problems. *BIT Numerical Mathematics*, 46(1):41–59, 2006.

[64] John Hershey, Jonathan Le Roux, and Felix Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *ArXiv*, ArXiv:1409.2574, 2014.

[65] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[66] Patrik O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

[67] Jeffrey Humpherys, Preston Redd, and Jeremy West. A fresh look at the Kalman filter. *SIAM Review*, 54(4):801–823, 2012.

[68] Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5–28, 1998.

[69] Prateek Jain, Ambuj Tewari, and Inderjit Dhillon. Orthogonal matching pursuit with replacement. *Advances in neural information processing systems*, 24, 2011.

[70] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[71] Thomas Kailath. *Linear Systems*. Prentice Hall, 1980.

[72] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[73] Alan A. Kaptanoglu, Brian M. de Silva, Urban Fasel, Kadierdan Kaheman, Andy J. Goldschmidt, Jared L. Callaham, Charles B. Delahunt, Zachary G. Nicolaou, Kathleen Champion, Jean-Christophe Loiseau, Nathan J. Kutz, and Steven L. Brunton. Pysindy: A comprehensive python package for robust sparse system identification. *ArXiv*, ArXiv:2111.08481, 2021.

[74] Seon Man Kim, Ji Hun Park, Hong Kook Kim, Sung Joo Lee, and Yun Keun Lee. Non-negative matrix factorization based noise reduction for noise robust automatic speech recognition. In *Latent Variable Analysis and Signal Separation*, pages 338–346, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[75] Serap Kirbiz, A Taylan Cemgil, and Bilge Günsel. Bayesian inference for nonnegative matrix factor deconvolution models. In *2010 20th International Conference on Pattern Recognition*, pages 2812–2815. IEEE, 2010.

[76] Deguang Kong, Chris Ding, and Heng Huang. Robust nonnegative matrix factorization using l21-norm. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, page 673–682. Association for Computing Machinery, 2011.

[77] Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

[78] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[79] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by nonnegative matrix factorization. *Nature*, 401:788–791, 1999.

[80] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 13, 2001.

[81] Namyoon Lee. Map support detection for greedy sparse signal recovery algorithms in compressive sensing. *IEEE Transactions on Signal Processing*, 64(19):4987–4999, 2016.

[82] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

[83] Liang Lin, Ding Xingyun, Wen Haobin, and Liu Fei. Impulsive components separation using minimum-determinant kl-divergence nmf of bi-variable map for bearing diagnosis. *Mechanical Systems and Signal Processing*, 2022.

[84] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ArXiv*, ArXiv:1806.09055, 2018.

[85] Lennart Ljung et al. Theory for the user. *System identification*, 1987.

[86] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1540–1552. PMLR, 2020.

[87] Shuai Lu, Sergei V Pereverzev, Yuanyuan Shao, and Ulrich Tautenhahn. Discrepancy curves for multi-parameter regularization. *Journal of Inverse & Ill-Posed Problems*, 18(6), 2010.

[88] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015.

[89] Julien Mairal, Francis Bach, and Jean Ponce. Task-driven dictionary learning. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):791–804, 2011.

[90] Niall M. Mangan, Nathan J. Kutz, Steven L. Brunton, and Joshua L. Proctor. Model selection for dynamical systems via sparse regression and information criteria. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2204), 2017.

[91] Md Abdul Mannan, Md R Rahman, Halima Akter, Nazmun Nahar, and Samiran Mondal. A study of banach fixed point theorem and it's applications. *American Journal of Computational Mathematics*, 11(2):157–174, 2021.

[92] Fabio Marcuzzi. A numerical feed-forward scheme for the augmented kalman filter. In *International Conference on Computational Science*, pages 131–145. Springer, 2024.

[93] Arthur Marmin, José Henrique de Morais Goulart, and Cédric Févotte. Majorization-minimization for sparse nonnegative matrix factorization with the beta-divergence. *IEEE transactions on signal processing*, 71:1435–1447, 2023.

[94] Arthur Marmin, José Henrique de Morais Goulart, and Cédric Févotte. Joint majorization-Minimization for nonnegative matrix factorization with the beta-divergence. *Signal Processing*, 209:109048, 2023.

[95] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, 1979.

[96] Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

[97] Deanna Needell and Joel A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and computational harmonic analysis*, 26(3):301–321, 2009.

[98] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[99] Peter Ochs, René Ranftl, Thomas Brox, and Thomas Pock. Bilevel optimization with nonsmooth lower level problems. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 654–665. Springer, 2015.

[100] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Press, 3rd edition, 2009.

[101] Art B. Owen and Patrick O. Perry. Bi-cross-validation of the svd and the nonnegative matrix factorization. *ArXiv*, ArXiv:0908.2062, 2009.

[102] Alexey Ozerov and Cédric Fevotte. Multichannel nonnegative matrix factorization in convolutive mixtures for audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):550–563, 2010.

[103] Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[104] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.

[105] Dhruv Patel, Deep Ray, Michael R.A. Abdelmalik, Thomas J.R. Hughes, and Assad A. Oberai. Variationally mimetic operator networks. *Computer Methods in Applied Mechanics and Engineering*, 419, 2024.

[106] Fernando Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *Neural information processing systems*, 1987.

[107] Meisam Razaviyayn, Mingyi Hong, and Zhi-Quan Luo. A unified convergence analysis of block successive minimization methods for nonsmooth optimization. *SIAM Journal on Optimization*, 23(2):1126–1153, 2013.

[108] Francesco Regazzoni, Stefano Pagani, Matteo Salvador, Luca Dede, and Alfio Quarteroni. Latent dynamics networks (ldnets): learning the intrinsic dynamics of spatio-temporal processes. *ArXiv*, ArXiv:2305.00094, 2023.

[109] Sebastian Reich and Colin Cotter. *Probabilistic Forecasting and Bayesian Data Assimilation*. Cambridge University Press, 2015.

[110] Guy Revach, Nir Shlezinger, Xiaoyong Ni, Adria Escoriza, Ruud van Sloun, and Yonina Eldar. KalmanNet: Neural network aided Kalman filtering for partially known dynamics. *IEEE Transactions on Signal Processing*, 70:1–1, 2022.

[111] Hannes Risken. Fokker-planck equation. In *The Fokker-Planck equation: methods of solution and applications*, pages 63–95. Springer, 1989.

[112] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[113] Simo Särkkä and Lennart Svensson. *Bayesian filtering and smoothing*, volume 17. Cambridge university press, 2023.

[114] Mikkel N. Schmidt and Morten Mørup. Nonnegative matrix factor 2-d deconvolution for blind single channel source separation. In *Independent Component Analysis and Blind Signal Separation*, pages 700–707. Springer Berlin Heidelberg, 2006.

[115] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *International Conference on Artificial Intelligence and Statistics*, 2018.

[116] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons, 2006.

[117] Paris Smaragdis. Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs. In *Independent Component Analysis and Blind Signal Separation*, pages 494–499, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[118] Paris Smaragdis and Judith C Brown. Non-negative matrix factorization for polyphonic music transcription. In *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No. 03TH8684)*, pages 177–180. IEEE, 2003.

[119] Nafiseh Soleymani, Mohammad Hossein Moattar, and Reza Sheibani. Dealing with high dimensional multi-view data: A comprehensive review of non-negative matrix factorization approaches in data mining and machine learning. *Computer Science Review*, 58:100788, 2025.

[120] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020.

[121] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *ArXiv*, ArXiv:2011.13456, 2020.

[122] Pablo Sprechmann, Roee Litman, Tal Ben Yakar, Alexander M. Bronstein, and Guillermo Sapiro. Supervised sparse analysis and synthesis operators. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[123] Ensio Suonperä and Tuomo Valkonen. Linearly convergent bilevel optimization with single-step inner methods. *Computational Optimization and Applications*, 87(2):571–610, 2024.

[124] Vincent Y.F. Tan and Cédric Févotte. Automatic relevance determination in nonnegative matrix factorization with the beta-divergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(7):1592–1605, 2013.

[125] George Trigeorgis, Konstantinos Bousmalis, Stefanos Zafeiriou, and Björn W Schuller. A deep matrix factorization method for learning attribute representations. *IEEE transactions on pattern analysis and machine intelligence*, 39(3):417–429, 2016.

[126] Mohammad Valipour, Mohammad Ebrahim Banihabib, and Seyyed Mahmood Reza Behbahani. Comparison of the arma, arima, and the autoregressive artificial neural network models in forecasting the monthly inflow of dez dam reservoir. *Journal of hydrology*, 476:433–441, 2013.

[127] Nicolaas G. Van Kampen. Stochastic differential equations. *Physics reports*, 24(3):171–228, 1976.

[128] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[129] Pedro J. Villasana T. and Stanislaw Gorlow. Exact multiplicative factor updates for convolutional beta-nmf in 2d. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, 2019.

[130] Tuomas Virtanen. Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparsity criteria. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(3):1066–1074, 2007.

[131] Yong Wang, Xinbin Luo, Lu Ding, Shan Fu, and Shiqiang Hu. Multi-task nonnegative matrix factorization for visual object tracking. *Pattern Analysis and Applications*, 23(1):493–507, 2020.

[132] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 2002.

[133] Samuel S. Wilks. Certain generalizations in the analysis of variance. *Biometrika*, 24(3–4):471–494, 1932.

[134] Kevin W. Wilson, Bhiksha Raj, Paris Smaragdis, and Ajay Divakaran. Speech denoising using nonnegative matrix factorization with priors. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4029–4032, 2008.

[135] Yi Wu, Bin Shen, and Haibin Ling. Visual tracking via online nonnegative matrix factorization. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(3):374–383, 2014.

[136] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273, 2003.

[137] Ruixue Yuan, Chengcai Leng, Bing Li, and Anup Basu. beta-divergence nmf with biorthogonal regularization for data representation. *Engineering Applications of Artificial Intelligence*, 121:106014, 2023.