.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Model-Based Deep Learning: On the Intersection of Deep Learning and Optimization

NIR SHLEZINGER[1], (Member, IEEE), YONINA C. ELDAR[2], (Fellow, IEEE), and STEPHEN P. BOYD[3], (Fellow, IEEE)

[1]School of Electrical and Computer Engineering, Ben-Gurion University of the Negev, Beer-Sheva 8410501, Israel (e-mail: nirshl@bgu.ac.il)
[2]Faculty of Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 7610001, Israel (e-mail: yonina.eldar@weizmann.ac.il)
[3]Department of Electrical Engineering, Stanford University, Palo Alto, CA, USA (e-mail: boyd@stanford.edu)

Corresponding author: Nir Shlezinger (e-mail: nirshl@bgu.ac.il).

The work was supported in part by ACCESS (AI Chip Center for Emerging Smart Systems), Stanford SystemX, by the Igel Manya Center for Biomedical Engineering and Signal Processing, and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101000967).

ABSTRACT Decision making algorithms are used in a multitude of different applications. Conventional approaches for designing decision algorithms employ principled and simplified modelling, based on which one can determine decisions via tractable optimization. More recently, deep learning approaches that use highly parametric architectures tuned from data without relying on mathematical models, are becoming increasingly popular. Model-based optimization and data-centric deep learning are often considered to be distinct disciplines. Here, we characterize them as edges of a continuous spectrum varying in specificity and parameterization, and provide a tutorial-style presentation to the methodologies lying in the middle ground of this spectrum, referred to as model-based deep learning. We accompany our presentation with running examples in super-resolution and stochastic control, and show how they are expressed using the provided characterization and specialized in each of the detailed methodologies. The gains of combining model-based optimization and deep learning are demonstrated using experimental results in various applications, ranging from biomedical imaging to digital communications.

INDEX TERMS Optimization, deep learning, deep unfolding, learn-to-optimize

I. INTRODUCTION

OPTIMIZATION provides a framework for solving problems described in a tractable mathematical manner. Optimization-based methods have been successfully applied across a broad range of applications involving decision making, ranging from electrical engineering to control and finance. The conventional approach to carry out decision making involves the introduction of mathematical models for the problem and the solver based on domain knowledge. Such model-based methods form the basis for many classical and fundamental optimization techniques. Many of these classical approaches rely on simplified descriptions of the problem that make decision making tractable, computationally feasible, and interpretable. While model-methods often work well, their simplified approximations can limit performance in some applications.

The unprecedented success of machine learning (ML), and particularly of deep learning, in areas such as computer vision and natural language processing [1] gave rise to methodology geared towards data. It is becoming common practice to replace principled task-specific decision mappings with abstract purely data-driven pipelines, trained with massive data sets. Deep neural networks (DNNs) are trained end-to-end, often in a supervised manner, without relying on analytical approximations, and therefore, they can operate in scenarios where analytical models are unknown or highly complex [2]. However, the abstractness and extreme parameterization of DNNs results in them often being treated as black-boxes; understanding how their predictions are obtained and how reliable they are tends to be quite challenging, and thus deep learning lacks the interpretability, flexibility, versatility, and reliability of model-based techniques.

Due to the limitations of model-based methods and data-driven pipelines, recent years have witnessed growing inter-

est in decision mappings involving both principled mathematical optimization and data-centric deep learning [3]–[5]. These include frameworks such as deep unfolding [6] and learned optimization [7], as well as task-specific techniques augmenting optimizers with DNNs [8]–[11]. While hybrid model-based deep learning methods are often designed and studied for specific tasks, their underlying methodology is relevant to a broad range of applications, motivating the systematic characterization of the interplay between existing approaches.

In this article we introduce a general framework for model-based deep learning schemes. While classic optimization and deep learning are typically considered to be distinct disciplines, we view them as edges of a continuum varying in specificity and parameterization. We build upon this characterization to provide a tutorial-style presentation of the main methodologies which lie in the middle ground of this spectrum, and combine model-based optimization with ML as a form of model-based deep learning. Our presentation is exemplified with running examples from super-resolution imaging and stochastic control.

We begin by providing a unified characterization for decision making algorithms, focusing on the main pillars of their design, which we identify as the decision rule type, the decision rule objective, and the evaluation procedure. Then, we show how classical model-based optimization as well as data-centric deep learning are obtained as special instances of this unified characterization. We identify the components dictating the distinction between the methodologies in the formulated objectives, the corresponding decision rule types, and their associated parameters. We next present a spectrum of decision making approaches which vary in specificity and parameterization, with model-based optimization and deep learning constituting its edges, and provide a systematic categorization of model-based deep learning techniques into concrete strategies positioned along this continuous spectrum. The proposed categorization captures the interplay between the different techniques and their pros and cons in comparison with both model-based optimization and conventional deep learning. We present extensive experimental results applying model-based deep learning methodologies in various application areas, including ultrasound image processing, microscopy imaging, digital communications, and tracking of dynamic systems. The results demonstrate the gains in performance and run-time of combining model-based optimization with deep learning over favouring one discipline over the other.

II. DECISION MAKING

We consider a generic setup where the goal is to design a decision rule $f : \mathcal{X} \mapsto \mathcal{S}$. The decision rule maps the context $x \in \mathcal{X}$, i.e., the available observations, into a decision $s \in \mathcal{S}$.

Examples: This generic formulation encompasses a multitude of settings involving classification, prediction, control, and many more. It thus corresponds to a broad range of applications. The task dictates the context space $\mathcal{X}$ and the possible decisions $\mathcal{S}$. A partial list of such applications includes:

- Signal processing - The context x includes samples from an observed signal or an image, which is mapped by f into another signal (e.g., for denoising) or into some form of inference (e.g., anomaly detection).
- Communications - The decision rule represents the operation of a digital receiver, which decodes the channel output x into an estimate of the transmitted message s.
- Vehicular control - The decision rule f is the control algorithm. The context x can include the traditional state variables, i.e., the vehicle sensory data, and commands. The decision s is the control action.
- Finance - The decision rule is the trading algorithm. The context x includes quantities such as financial forecasts and current positions. The decision s is the trade list, i.e., the list of assets to buy and to sell.

FIGURE 1. Super-resolution recovery illustration.

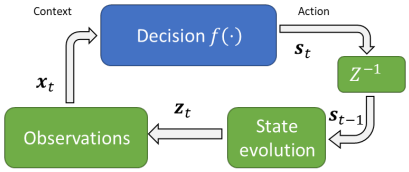


FIGURE 2. Stochastic control illustration.

To keep the presentation focused, we repeatedly use two concrete running examples:

Example 1 (Super-Resolution): Here, s is a high-resolution image, while x is a distorted low-resolution version of the image. Thus, $\mathcal{X}$ and $\mathcal{S}$ are the spaces of low-resolution and high-resolution images, respectively. Such decision rules, typically referred to as recovery methods, aim at reconstructing s^{true} from its distorted version x, as illustrated in Fig. 1.

Example 2 (Stochastic Control): In our second example, we consider a dynamic system, where the decision rule is a control policy. At each time period t, the goal is to map the noisy state observations x_t, where $\mathcal{X}$ is the space of possible sensory measurements, into an action s_t within an action space $\mathcal{S}$. The system is characterized by a latent state vector z_t that evolves in a random fashion which is related to the previous state z_{t-1} and action s_{t-1}, while being partially observable via the noisy x_t. This setup is illustrated in Fig. 2.

A. DECISION RULE TYPES

The above generic formulation allows the decision rule f to be any mapping from $\mathcal{X}$ into $\mathcal{S}$. In practice, decision rules are often carried out using a structured form. Some common types of decision rules are:

T1 An affine rule, i.e., $s = Wx + b$ for some (W, b).

T2 A decision tree chooses s from a finite set of possible decisions $\{s_k\}$ by examining a set of nested conditions $\{cond_k\}$, e.g., if $cond_1(x)$ then $s = s_1$; else inspect $cond_2(x)$, and so on.

T3 An optimization-based decision rule chooses s as a solution or approximate solution of an optimization problem parametrized by the context x, i.e., $\arg\min_{s\in\mathcal{S}}\mathcal{L}(s;x)$, where $\mathcal{L}$ is an objective function.

T4 An iterative algorithm finds its decision by executing a sequence of mappings $h_k : \mathcal{S}\times\mathcal{X} \mapsto \mathcal{S}$, repeating $s_{k+1} = h_k\big(s_k;x\big)$ from an initial guess s_0 until convergence, or a fixed number of steps K, i.e., $s = h_K\big(h_{K-1}\big(\cdots h_1(s_0;x);x\big);x\big)$.

T5 A neural network is a special case of an iterative algorithm, where $h_k(z) = \sigma\big(W_k z + b_k\big)$ with $\sigma(\cdot)$ being an activation function and (W_k, b_k) are parameters of the affine transformation. These mappings are referred to as layers. We have $s = h_K\big(h_{K-1}\big(\cdots h_1(x)\big)\big)$.

The boundaries between decision rule types are not always clear, and there is some overlap between the categories. For instance, an optimization-based decision rule with quadratic objective, where the context affects only the linear term in the objective, can be explicitly expressed as an affine decision rule. As another example, iterative decision rules often arise as iterations that solve an optimization problem. Moreover, an iterative algorithm with K iterations can often be viewed as a neural network, as we further elaborate on in the sequel.

Each of these decision rule types include parameters. For example in an affine decision rule, the parameters are W and b; in a decision tree, it is the values s_k and parameters that specify the conditions. In an iterative algorithm the parameters are those appearing in the functions h_k; and in a neural network, the parameters are W_k and b_k. In some cases the number of parameters is small, such as decision trees with a small number of conditions. In other cases, e.g., when f is a DNN, decision rules can involve a massive number of parameters. These parameters capture the different mappings one can represent as decision rules.

For a decision rule type, we let $\mathcal{F}$ denote the set of possible decision rules, over all choices of parameters. In general, the more parameters there are, the broader the family of mappings captured by $\mathcal{F}$, which in turn results in the decision rule capable of accommodating more diverse and generic functions. Decision rules with fewer parameters are typically more specific, capturing a limited family of mappings. Let Θ denote the parameter space for a decision rule family $\mathcal{F}$, such that for each $\theta \in \Theta$, $f(\cdot;\theta)$ is a mapping in $\mathcal{F}$. We refer to the choosing of the decision rule parameters θ as tuning. In principle, tuning can be carried out based on understanding and modeling of the task. In practice, tuning typically involves simulation with either synthetic or real data; this procedure can be done manually when there are a few parameters, or by an automated algorithm for decision rules involving many parameters. In the latter case, tuning is also referred to as training or learning.

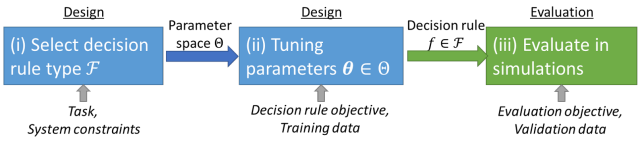


FIGURE 3. Decision rule selection procedure illustration.

B. EVALUATING A DECISION RULE

The evaluation of a decision rule is comprised of two ingredients: 1) simulations where the decision rule is to be applied; and 2) an objective function for measuring its performance during the conducted simulations.

Simulations represent the setting in which the decision rule is required to operate after its parameters are tuned. They can be as simple as applying f on held out data, or involve complex mechanisms for emulating an overall system where the decision rule is to be applied and the expected environment. Various terminologies are used to describe simulators in different domains, including validation (in ML), closed-loop simulation (in control), and back-testing (in finance).

Example 3: A simulation setup for super-resolution recovery (Example 1) can be comprised of a set of unseen images and a mapping that converts them into low-resolution data.

Example 4: A simulation setup for stochastic control (Example 2) can be software which emulates how an action s_{t-1} is translated into a state z_t and an observed context x_t.

Objective functions are measures used to evaluate a decision rule. In some cases, the objective is given by a cost or a loss function which one aims at minimizing, or it can be specified by an application utility or reward, which we wish to maximize. In many applications there are multiple objectives, which are scalarized into a single cost function, for example, by forming a weighted sum. The objective can be the average value of individual decisions, or a function of multiple decisions, e.g., a trajectory. In its most basic form, a loss function evaluates the decision rule for a given context as compared with some desired decision; such a loss function is formulated as a mapping [12]

$$l : \mathcal{F}\times\mathcal{X}\times\mathcal{S} \mapsto \mathcal{R}^+. \tag{1}$$

Broadly speaking, (1) dictates the success criteria of a decision mapping for a given context-decision pair. For instance, in inference tasks, candidate losses include the error rate (zero-one) loss $l_{\mathrm{Err}}(f,x,s^{true}) = \mathbf{1}_{f(x)\neq s^{true}}$ (for classification) and the ℓ_2 loss $l_{\mathrm{Est}}(f,x,s^{true}) = \|s^{true} - f(x)\|_2^2$ (for estimation).

In optimization-based decision rules, the objective used to formulate the optimization problem need not be the same as the evaluation objective function. The evaluation objective measures the performance of the decision rule in simulation; it may be complex and capture multiple utilities of the overall system. In some applications, e.g., medical imaging, it may involve inspecting the simulations outcome by human experts. The objective of the optimization-based decision rule, referred to as the decision rule objective, is used for tuning

the decision rule; it is often a surrogate of the evaluation objective, including e.g., simplifications, approximations, and regularizations, introduced for tractability and to facilitate tuning.

The selection of a decision rule depends on how the mapping is judged during tuning and evaluation. This is a three-step procedure, whose first two steps are its design, involving (i) selecting a type of decision rule $\mathcal{F}$ (e.g., linear model, decision tree, DNN, etc.); and (ii) tuning its parameters θ based on the decision rule objective. Then (iii), the tuned system is evaluated in simulations based on the evaluation objective. The evaluation is determined by the simulator, and is independent of the design steps. Fig. 3 illustrates the overall procedure. The traditional approaches to carry out the design procedure, referred to as model-based or classic methods, are based on modelling and knowledge; the data-centric approach uses ML, with deep learning being a leading family of ML techniques.

III. MODEL-BASED METHODS

A. DECISION RULE OBJECTIVE

The classic model-based approach sets decision rules based on domain knowledge. Namely, knowledge of an underlying model which mathematically describes the setup is used along with the loss measure $l(\cdot)$ to formulate an analytical surrogate decision rule objective $\mathcal{L} : \mathcal{F} \mapsto \mathcal{R}^+$. Both the model imposed and the objective are typically simplified approximations of the evaluation simulator and objective, respectively, introduced for analytical tractability. The decision rule objective also often includes sensitivity and regularization terms, resulting in an inductive bias on f. The decision rule objective is applied to select the decision rule from $\mathcal{F}$, which can be a pre-defined type or the entire space of mappings from $\mathcal{X}$ to $\mathcal{S}$. Once a simplified objective $\mathcal{L}$ is set, one can often find the optimal decision rule in $\mathcal{F}$ with respect to $\mathcal{L}$.

For instance, for inference tasks, given knowledge of a distribution $\mathcal{P}$ defined over $\mathcal{X} \times \mathcal{S}$, one can formulate the risk $\mathcal{L}(f) = \mathbb{E}_{(x,s^{true})\sim\mathcal{P}}\{l(f, x, s^{true})\}$, and set f to minimize the error rate risk among all mappings from $\mathcal{X}$ to $\mathcal{S}$ as the maximum a-posteriori probability (MAP) rule, given by:

$$f_{MAP}(x) = \arg\max_{s\in\mathcal{S}} \Pr(s^{true} = s|x). \tag{2}$$

The formulation of $\mathcal{L}(f)$ is dictated by the model imposed on the underlying relationship between x and the desired s^{true}. This objective typically contains parameters of the model, which we denote by $\theta^{\rm o}$, and henceforth write $\mathcal{L}_{\theta^{\rm o}}(f)$.

Example 5: A common approach to treat the super-resolution problem in Example 1 is to assume the compression obeys a linear Gaussian model, i.e.,

$$x = Hs^{true} + w, \qquad w \sim \mathcal{N}(0, \sigma^2 I). \tag{3}$$

The matrix H in (3) may represent the point-spread function of the system, a reduced measurement resolution, etc. The MAP rule in (2) becomes

$$f_{MAP}(x) = \arg\min_{s} \frac{1}{2}\|x - Hs\|_2^2 + \sigma^2\phi(s), \tag{4}$$

where $\phi(s) := -\log\Pr(s^{true} = s)$. The resulting decision rule objective requires imposing a prior on $\mathcal{S}$ encapsulated in $\phi(s)$. A popular selection is to impose sparsity in some known domain Ψ (e.g., wavelet), such that $s = \Psi r$, where r is sparse. This boils down to an objective defined on r, given by

$$\mathcal{L}_{\theta^{\rm o}}(r) = \frac{1}{2}\|x - H\Psi r\|_2^2 + \rho\|r\|_0, \tag{5}$$

where the parameter ρ encapsulates σ^2 and the expected sparsity level. The parameters of the objective in (5) are

$$\theta^{\rm o} = \{H, \Psi, \rho\}. \tag{6}$$

The above example shows how one can leverage domain knowledge to formulate an objective, which is dictated by the parameter vector $\theta^{\rm o}$. It also demonstrates two key properties of model-based approaches: (i) that surrogate models can be quite unfaithful to the true data, since, e.g., the Gaussianity of w implies that x in (3) can take negative values, which is not the case for image data; and (ii) that simplified models allow translating the task into a relatively simple closed-form objective, as in (5). Similar approaches can be used to tackle the stochastic control setting of Example 2.

Example 6: Traditional linear-quadratic-Gaussian (LQG) control considers dynamics that take the form of a linear Gaussian state-space model, where

$$z_{t+1} = Az_t + Bs_t + v_t, \tag{7a}$$
$$x_t = Cz_t + w_t. \tag{7b}$$

Here, the noise sequences v_t, w_t are zero-mean Gaussian signals, i.i.d. in time, with covariance matrices V, W, respectively. The objective at each time instance t is given by

$$\mathcal{L}_{\theta^{\rm o}}(f) = \mathbb{E}\{z_t^T Q z_t + s_t^T R s_t\}, \quad s_t = f(\{x_\tau\}_{\tau\le t}). \tag{8}$$

The parameters of the objective function (8) are thus

$$\theta^{\rm o} = (A, B, C, Q, R, V, W). \tag{9}$$

Example 7: Model predictive control replaces the expectation based objective in (8) with a deterministic optimization problem based on forecasting over some finite horizon H. Here, for the linear Gaussian state-space model of (7) with a quadratic loss, the objective at each time period t is given by

$$\mathcal{L}_{\theta^{\rm o}}(f) = \sum_{\tau=0}^{H-1} \left(\hat{z}_{t+\tau}^T Q_\tau \hat{z}_{t+\tau} + s_{t+\tau}^T R_\tau s_{t+\tau}\right), \tag{10}$$

where $s_t, \ldots, s_{t+H-1} = f(\{x_\tau\}_{\tau\le t})$ and $\{\hat{z}_{t+\tau}\}$ are computed via (7) with $\{v_{t+\tau}\}$ and $\{w_{t+\tau}\}$ replaced with some predicted values. The parameters $\theta^{\rm o}$ of the objective function (10) thus include these predictive mapping, as well as the matrices $A, B, C, \{Q_\tau, R_\tau\}$.

The formulation of the decision rule objectives in Examples 5-7 relies on full domain knowledge, e.g., one has to know the prior $\phi(\cdot)$ or the covariances V, W in order to express the objectives in (5) and (8), respectively.

B. DECISION RULE TYPE

Model-based methods determine the decision rule objective based on domain knowledge, obtained from measurements and from understanding of the underlying physics. Once the objective is fixed, evaluating the decision rule boils down to solving an optimization problem, typically resulting in highly-specific types of decision mappings whose structure follows from the optimization formulation. In particular, a decision rule is typically obtained as either an explicit solution of the problem, or in the form of an iterative solver.

Explicit solvers arise when the decision rule objective takes a relatively simplified form, such that one can characterize the optimal mapping. In such cases, the optimization-based decision rule of type T3 can specialize into an affine rule T1.

Example 8: The mapping which minimizes the LQG loss in Example 6 is known to be obtained by first predicting the latent state z_t using a Kalman filter, i.e.,

$$\hat{z}_t = A\hat{z}_{t-1} + L_t\left(x_t - C(A\hat{z}_{t-1} + Bs_{t-1}\right), \tag{11}$$

where L_t is the Kalman gain matrix. The action is taken to be

$$s_t = -K_t\hat{z}_t, \tag{12}$$

with K_t being the feedback gain matrix. Both L_t and K_t are determinstically determined by $\theta^{\rm o}$ in (9), and are updated based on internally tracked statistical moments that are recursively updated.

Example 8 demonstrates how the modelling of a complex task using a simplified linear Gaussian model, combined with the usage of a simple surrogate quadratic objective, results in an explicit solution, which here takes a linear form. While this surrogate model and the objective are likely to differ from the operation of the system, one can tune the objective parameters $\theta^{\rm o}$ (encapsulated in (11) and in K_t) via simulations, thus modifying the decision rule to match the expected operation.

Iterative solvers follow mathematical steps which gradually lead to the decision that achieves the decision rule objective, yielding a mapping as in type T4. A large body of optimization techniques are iterative, with common schemes based on first-order methods (i.e., gradient iterative steps) [13, Ch. 9]. Iterative optimizers typically give rise to additional parameters which affect the speed and convergence rate of the algorithm, but not the actual objective being minimized. We refer to these parameters of the solver as hyperparameters, and denote them by $\theta^{\rm h}$. As opposed to the objective parameters $\theta^{\rm o}$ (as in, e.g., (6)), they often have no effect on the solution when the algorithm is allowed to run to convergence, and so are of secondary importance. But when the iterative algorithm is stopped after a predefined number of iterations, they affect the decisions, and therefore also the decision rule objective. Due to the surrogate nature of the objective, such stopping does not necessarily degrade the evaluation performance.

Example 9: The super-resolution objective in (4) can be tackled using the alternating direction method of multipliers (ADMM) [14]. This method summarized as Algorithm 1, where we merge σ^2 in (4) into the prior $\phi(\cdot)$ for brevity, and the proximal mapping is defined as

$$prox_g(v) := \arg\min_z \left(g(z) + \frac{1}{2}\|z - v\|_2^2 \right). \tag{13}$$

Algorithm 1 ADMM

Fix step size μ, and $\lambda > 0$.
Initialize $k = 0$, u_0, v_0 randomly.
repeat
  Update $s_{k+1} = (H^T H + 2\lambda I)^{-1}(H^T x + 2\lambda(v_k - u_k))$.
  Update $v_{k+1} = prox_{\frac{1}{2\lambda}\phi}(s_{k+1} + u_k)$ (see (13)).
  Update $u_{k+1} = u_k + \mu\left(s_{k+1} - v_{k+1}\right)$.
  Set $k = k + 1$.
until convergence
Set estimate $s = s_k$.

ADMM converges to a solution of (4) when $\phi(\cdot)$ is convex, for any positive value of μ. When $\phi(\cdot)$ is not convex, there are no convergence guarantees, but it has been observed in practice that good results are obtained, when μ is chosen appropriately. Since convergence of iterative optimizers to an optimal decision can be generally guaranteed for convex objectives, one often has to relax and modify the objective.

Example 10: The non-convex super-resolution surrogate objective with sparse prior in (5) can be relaxed into

$$\mathcal{L}_{\theta^{\rm o}}(s) = \frac{1}{2}\|x - Hs\|_2^2 + \rho\|r\|_1, \tag{14}$$

where we set Ψ to the identity matrix for simplicity. This successive relaxation of an already surrogate objective yields a convex cost in (14). It can be solved, e.g., using proximal gradient descent with step size μ, which specializes here into the iterative soft thresholding algorithm (ISTA) [15, Ch. 7]. Letting $\mathcal{T}_\beta(\cdot) \triangleq sign(x)\max(0, |x| - \beta)$ be the element-wise soft-thresholding operation, the update equation is

$$s_{k+1} = \mathcal{T}_{\mu\rho}\left(s_k + \mu H^T(x - Hs_k)\right). \tag{15}$$

In Example 9, illustrated in Fig. 5(a), the iterative solver introduces two hyperparameters, i.e., $\theta^{\rm h} = [\lambda, \mu]$, which are used in the iterative minimization of (4). In Example 10, there is only one hyperparameter $\theta^{\rm h} = \mu$. The hyperparameters $\theta^{\rm h}$ are often set by manual hand tuning based on simulations.

C. SUMMARY

Model-based methods rely on decision rules of type T3, where an analytically tractable optimization problem is formulated based domain knowledge. The optimization problems solved are typically surrogates for the real application problem. The decision rule objectives and constraints are

often inspired by physical characteristics, understanding of the system operation, and existing models (of noise, disturbances, and other quantities). Yet, in practice, objectives are likely to differ from the system task, due to multiple reasons, including:

- Simplifying approximations, e.g., modelling super-resolution as a linear Gaussian setup in Example 5.
- Estimation inaccuracies, e.g., substituting estimated covariances W, V to compute the LQG objective in Example 6 or an estimated H in the MAP objective in Example 5.
- Introducing regularization terms in the objective, e.g., $\rho\|r\|_0$ in Example 5.
- Relaxations or approximations of the objective and constraints to render the optimization problem solvable.
- Scaling of some of the quantities involved.

Model-based techniques are particularly suitable for the resulting optimization problem. Once a solvable (e.g., convex) formulation is determined, these methods are guaranteed to obtain its solution. Furthermore, their operation is interpretable, and tends to be highly flexible, as one can substitute different values of the objective parameters θ^{o}.

In practice, accurate knowledge of the statistical model relating the context and the desired decision is often unavailable. Consequently, model-based techniques may require imposing assumptions on the underlying statistics, which in some cases reflect the actual behavior, but can also be a crude approximation. In the presence of inaccurate model knowledge, either as a result of estimation errors or due to enforcing a model which does not fully capture the environment, the performance of model-based techniques tends to degrade during evaluation. This limits the applicability of model-based schemes in scenarios where one cannot represent the task via a decision rule objective in a closed-form (and preferably simplified) expression, or alternatively, when the underlying model is costly to estimate accurately, or too complex to express analytically. Additional challenges stem from the fact that decision making can be slow, particularly using iterative solvers. Finally, setting the hyperparameters θ^{h} is often elusive, and may involve heuristics and cumbersome hand-crafted tunning.

IV. DEEP LEARNING

A. DECISION RULE OBJECTIVE

While in many applications coming up with accurate and tractable statistical modelling is difficult, we are often given access to data describing the setup. ML systems learn their mapping from data. In a supervised setting, data is comprised of a set of n_t pairs of inputs and labels, denoted $\mathcal{D} = \{x_i, s_i^{true}\}_{i=1}^{n_t}$. In reinforcement learning, data is obtained from a simulator, which on each decision produces a subsequent context. This data is referred to as the training set, and there is typically an additional data set used for evaluation and validation. Since no mathematical model relating the input and the desired decision is imposed, the decision rule objective is often the empirical risk. Focusing on a supervised setting, this objective is given by

$$\mathcal{L}_{\mathcal{D}}(f) \triangleq \frac{1}{n_t}\sum_{i=1}^{n_t} l(f, x_i, s_i^{true}). \tag{16}$$

Decision rule objectives that are based on data and do not rely on modeling are often a more faithful representation of the evaluation objective compared with model-based approaches. However, they are still surrogates. This follows not only from the difference between the training and validation data, but also from the frequent inclusions of regularizing terms and mechanisms such as dropout and batch normalization, whose operation differs between training and evaluation.

While we focus our description on supervised settings, ML systems can also learn in an unsupervised manner. In such cases, the data set $\mathcal{D}$ is comprised only of a set of examples $\{x_i\}_{i=1}^{n_t}$, and the loss measure l is defined over $\mathcal{F}\times\mathcal{X}$, instead of over $\mathcal{F}\times\mathcal{X}\times\mathcal{S}$ as in (1). Unsupervised ML is often used to discover patterns in the data, with tasks including clustering, anomaly detection, generative modeling, and compression.

B. DECISION RULE TYPE

In contrast to the model-based case, where decision rules can sometimes be derived by directly solving the optimization problem without initially imposing structure on the system, setting a decision rule based on data necessitates restricting the domain of feasible mappings. This stems from the fact that one can usually form a decision rule which minimizes the empirical loss of (16) by memorizing the data, i.e., overfit [12, Ch. 2]. A leading strategy in ML, upon which deep learning is based, is to assume a highly-expressive generic parametric model on the decision mapping, while incorporating optimization mechanisms and regularizing the empirical risk to avoid overfitting. In deep learning, f is a DNN, i.e., type T5, with the parameters θ being the network parameters, e.g., the weights and biases of each layer. By the universal approximation theorem, DNNs can approach any Borel measurable mapping [16, Ch. 6].

While model-based algorithms are specifically tailored to a given scenario, deep learning is model-agnostic. The unique characteristics of the scenario are encapsulated in the learned weights. The decision rule family $\mathcal{F}$, i.e., the possible DNN mappings, is generic and can be applied in a broad range of different problems. While standard DNN structures are model-agnostic and are commonly treated as black boxes, one can still incorporate some level of domain knowledge in the selection of the network architecture. For instance, when the input is known to exhibit temporal correlation, architectures based on recurrent neural networks (RNNs) [16, Ch. 10] or attention mechanisms [17] are often preferred. Alternatively, in the presence of spatially local patterns, one may utilize convolutional layers [18]. An additional method to incorporate domain knowledge into a black box DNN is by pre-processing of the input via, e.g., hand-crafted feature extraction.

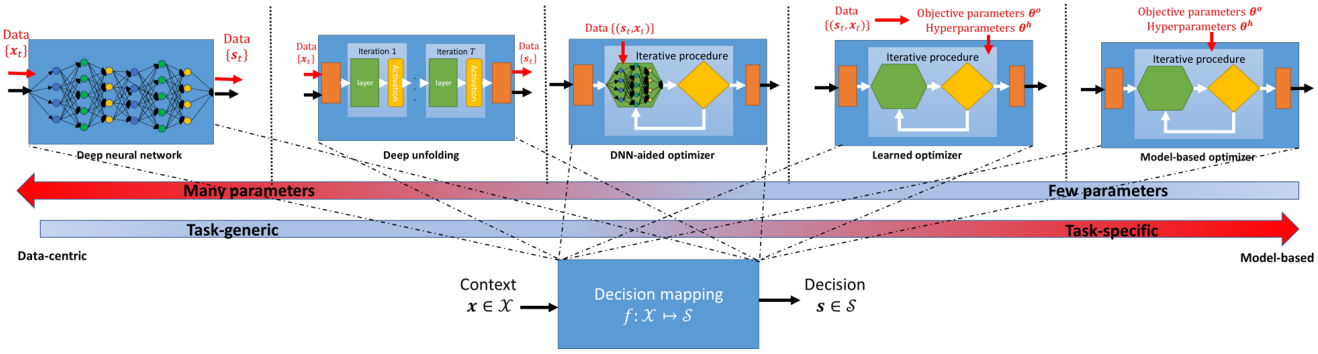


FIGURE 4. Continuous spectrum of specificity and parameterization with model-based methods and deep learning constituting the extreme edges of the spectrum.

Example 11: The super-resolution task (Example 1) can be carried out by training a deep convolutional autoencoder [19].
Example 12: Stochastic control (Example 2) can be carried by a DNN controller using deep reinforcement learning [20].

The fact that DNNs are comprised of a large number of parameters, and that massive data sets are often used for their training, makes it unlikely to recover θ that minimizes the empirical risk with affordable computational effort. Instead, the tuning of θ is typically carried out using first-order gradient-based algorithms, where gradients estimated from a small number of randomly chosen samples, e.g., by mini-batch stochastic gradient descent (SGD) iterations of the form

$$\theta_{j+1} = \theta_j - \eta_j \nabla_\theta \mathcal{L}_{\mathcal{D}_j}(f(\cdot;\theta_j)), \tag{17}$$

where $\mathcal{D}_j$ is a mini-batch sampled from $\mathcal{D}$, and η_j is the learning rate. The gradients in (17) are computed using back-propagation [21]. Mini-batch SGD is the basis for DNN training, with common variants using momentum and adaptive learning rates. Such training methods operate in an automated manner, enabling tuning of DNNs from massive data sets.

C. SUMMARY

DNNs operate in a model-agnostic manner, and can be tuned to implement an immense family of mappings, making them widely adopted in areas where principled mathematical models are scarce, such as computer vision and natural language processing. Despite their success, existing deep learning approaches are subject to several challenges, which limit their applicability in some application domains. The computational burden of training and utilizing highly parametrized DNNs, as well as the fact that massive data sets are often required for their training, constitute major drawbacks in various signal-processing, communications and in control applications. This limitation is particularly relevant when operating on hardware-constrained devices, e.g., mobile systems, unmanned aerial vehicles, and sensors. Such systems are typically limited in their ability to utilize highly parametrized DNNs, and they should be flexible to adapt to variations in the environment. Furthermore, the fact that the decision mapping is learned solely from data often gives rise to generalization issues on unseen data. Finally, due to the complex and generic structure of DNNs, it is often extremely challenging to understand how they obtain their predictions, track the rationale leading to their decisions, and characterize confidence intervals. Consequently, deep learning does not offer the interpretability, flexibility, versatility, and reliability of model-based methods. This is a major limitation for tasks involving critical and even life-saving decision making, such as the control of vehicular and aerospace systems.

V. HYBRID MODEL-BASED DEEP LEARNING OPTIMIZERS

Model-based methods and deep learning are often viewed as fundamentally different approaches for setting decision boxes. Nonetheless, both strategies typically use parametric mappings, i.e., the weights θ of DNNs and the parameters $(\theta^{\rm o}, \theta^{\rm h})$ of model-based optimizers, whose setting is determined based on data and on principled mathematical models. The core difference thus lies in the specificity and the parameterization of the decision rule type: Model-based methods are knowledge-centric, using decision rules that are task-specific, and usually involve a limited number of parameters that can often be set manually. Deep learning is data-centric, and thus uses highly-parametrized model-agnostic task-generic mappings.

The identification of model-based methods and deep learning as two ends of a spectrum of specificity and parameterization indicates the presence of a continuum, as illustrated in Fig. 4. In fact, many techniques lie in the middle ground, designing decision rules with different levels of specificity and parameterization by combining some balance of deep learning with model-based optimization [4]. In this section we review three systematic frameworks for designing decision mappings that are both knowledge- and data-centric as a form of hybrid model-based deep learning: The first strategy, coined learned optimizers [7], [22], uses deep learning automated tuning machinery to tune parameters of model-based optimization conventionally tuned by hand. The second family of techniques, referred to as deep unfolding [6], converts iterative optimizers into DNNs. The third type of model-based deep learning schemes, which we call DNN-aided optimizers [3], augments model-based optimization with dedicated DNNs.

A. LEARNED OPTIMIZATION

Learned optimizers use conventional model-based methods for decision making, while tuning the parameters and hyperparameters of classic solvers via automated deep learning training [7], [22]. This form of model-based deep learning leverages data to optimize the optimizer. While learned optimization bypasses the traditional daunting effort of manually fitting the decision rule parameters, it involves the introduction of new hyperparameters of the training procedure that must to be configured (typically by hand).

Learned optimization effectively converts an optimizer into an ML model. Since automated tuning of ML models typically uses gradient-based methods as in (17), a key requirement is for the optimizer to be differentiable, namely, that one can compute the gradient of its decision with respect to its parameters. Fortunately, convex optimization solvers are typically differentiable (under some regularity conditions) [23]. Alternatively, for non-convex optimization, one can differentiate numerically [24] or, in some cases, implicitly [25].

Examples: Learned optimization focuses on optimizing parameters conventionally tuned manually; these are parameters whose value does not follow from prior knowledge of the problem being solved, and thus their modification affects only the solver, and not the problem being solved. For model-based optimizers based on explicit solutions, the parameters available are only those of the objective θ^{o}. Nonetheless, some of these parameters stem from the fact the objective is inherently a surrogate for the actual problem being solved, and thus require tuning, as shown in the following example.

Example 13: Consider a dynamic system characterized by a state-space model as in (7). In such settings, the linear mappings A, B, C often arise from understanding the physics of the problem, while the objective parameters Q and R stem from the system requirements. Nonetheless, in practice, one typically does not have a concrete stochastic model for the noise signals, which are often introduced as a way to capture stochasticity, and thus V and W are often tuned by hand.

Given a data set of n_t trajectories of T observations with the corresponding states and actions $\mathcal{D} = \{\{x_{t,i}, s_{t,i}, z_{t,i}\}_{t=1}^{T}\}_{i=1}^{n_t}$, one set the trainable parameters to be $\theta = [V, W]$, and optimize them via (17) with $\mathcal{L}_{\mathcal{D}}$ being the empirical ℓ_2 distance between the Kalman filter prediction (11) and the true state [26]. The gradients of $\mathcal{L}_{\mathcal{D}}$ are computed using backpropagation through time (BPTT) [27], building upon the diffrentiability of the Kalman gain L_t with respect to both V and W [28].

When the optimizer being learned is an iterative solver, one can use data to tune the hyperparameters θ^{h}, whose value does not affect the optimization objective. This can be specialized for the ADMM optimizer of Example 9, as shown next.

Example 14: Consider the ADMM solver (Algorithm 1). Given a data set $\mathcal{D}$ of n_t labeled samples, the hyperparameter vector $\theta^{\mathrm{h}} = [\lambda, \mu]$ can be optimized by treating it as trainable parameters, as visualized in Fig. 5(b). Letting $f(\cdot; \theta^{\mathrm{h}})$ be the ADMM mapping with hyperparameters θ^{h}, this is given by

$$\theta^* = \underset{\theta^{\mathrm{h}}=[\lambda,\mu]\in\mathcal{R}^+}{\arg\min} \frac{1}{n_t}\sum_{i=1}^{n_t} \|f(x_i;\theta^{\mathrm{h}}) - s_i^{true}\|_2^2. \tag{18}$$

The problem in (18) is as DNN training, e.g., (17), where to compute each gradient of the objective with respect to θ^{h}, Algorithm 1 must first run until it reaches convergence, after which the gradients are computed via BPTT.

Summary. Learned optimizers are ML decision rules which completely preserve the operation of conventional model-based methods. As such, they share the core gains of principled optimization. These include its suitability for the problem at hand; the interpretability that follows from the ability to relate each feature involved to an operation meaning; and flexibility, as one can control the objective for which the decision is configured using its non-learned parameters in θ^{o}.

Compared with model-based optimization, learned optimizers facilitate the design procedure, avoiding the need to tune parameters by hand. Furthermore, the fact that tuning is carried out by observing the decision output and evaluating it based on data allows to improve performance when the surrogate objective differs from the (possibly analytically intractable) evaluation objective. Finally, when the decision box is an iterative solver, learned optimization can reduce the convergence speed compared with manually tuned θ^{h}.

B. DEEP UNFOLDING

A relatively common methodology for combining model-based methods and deep learning is that of deep unfolding, also referred to as deep unrolling [6]. Originally proposed by Greger and LeCun for sparse recovery [29], deep unfolding converts iterative optimizers into trainable DNNs. As the name suggests, the method unfolds an iterative algorithm into a sequential procedure with a fixed number of iterations. Then, each iteration is treated as a layer, with its trainable parameters θ being either only the hyperparameters θ^{h}, or also the decision rule objective parameters θ^{o}.

Unfolding an iterative optimizer into a DNN facilitates tuning different parameters for each iteration, being converted into trainable parameters of different layers. This is achieved by training end-to-end, i.e., by evaluating the system output based on data. Letting K be the number of unfolded iterations, deep unfolding can learn iteration-dependent hyperparameters $\{\theta_k^{\mathrm{h}}\}_{k=1}^{K}$ and even objective parameters $\{\theta_k^{\mathrm{o}}\}_{k=1}^{K}$. This increases the parameterization and abstractness compared with learned optimization of iterative solvers, which typically reuses the learned hyperparameters and runs until convergence (as in model-based optimizers). Nonetheless, for every setting of $\{\theta_k^{\mathrm{o}}\}_{k=1}^{K}$ and $\{\theta_k^{\mathrm{h}}\}_{k=1}^{K}$, a deep unfolded system effectively carries out its decision using K iterations of some principled iterative solver known to be suitable for the problem.

Examples: Deep unfolded networks can be designed to improve upon model-based optimization in convergence

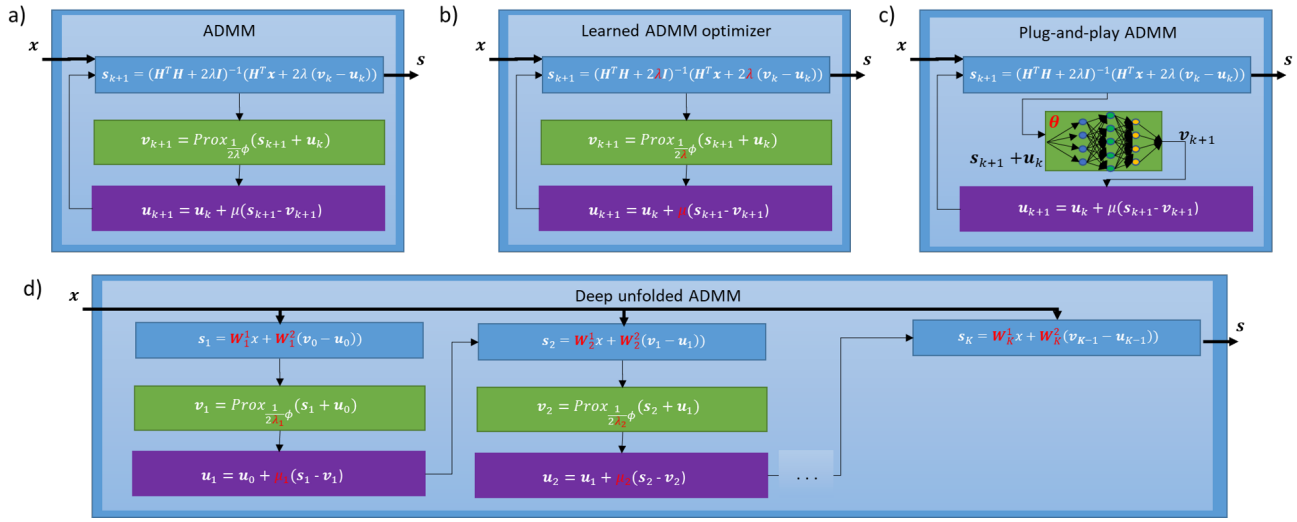


FIGURE 5. An illustration of the model-based deep learning strategies arising from the ADMM optimizer (Algorithm 1), where variables in red fonts represent trainable parameters: a) the model-based optimizer (Example 9); b) a learned ADMM optimizer (Example 14); c) plug-and-play ADMM (Example 20); and d) deep unfolded ADMM (Example 16).

speed and model abstractness. The former is achieved since the resulting system operates with a fixed number of iterations, which can be much smaller compared with that usually required to converge. This is combined with the natural ability of deep unfolding to learn iteration-dependent hyperparameters to enable accurate decisions to be achieved within this predefined number of iterations, as exemplified next:

Example 15: Let us consider again the ADMM optimizer of Algorithm 1. A deep unfolded ADMM is obtained by setting the decision to be $s = s_K$ for some fixed K, and allowing each iteration to use hyperparameters $[\lambda_k, \mu_k]$, that are stacked into the trainable parameters vector θ. Similarly to (18), these hyperparameters are learned from data via

$$\theta^* = \underset{\theta=\{\lambda_k,\mu_k\}_{k=1}^K}{\arg\min} \frac{1}{n_t}\sum_{i=1}^{n_t} \|f(x_i;\theta) - s_i^{true}\|_2^2. \quad (19)$$

Example 15 implements K ADMM iterations, as only the hyperparameters are learned. One can also transform iterative solvers into more abstract DNNs by also tuning the objective of each iteration. Continuing along the line of Examples 14-15, we show how this is can be achieved following [30]:

Example 16: An alternative approach to unfold the ADMM optimizer into a DNN is by repeating the procedure in Example 15, where algorithm is unfolded into K iterations with each assigned hypereparameters $[\lambda_k, \mu_k]$. Furthermore, the first update step of Algorithm 1 is now replaced with

$$s_{k+1} = W_k^1 x + W_k^2 (v_k - u_k). \quad (20)$$

For $W_k^1 = (H^T H + 2\lambda I)^{-1} H^T$ and $W_k^2 = 2\lambda(H^T H + 2\lambda I)^{-1}$, (20) coincides with the corresponding step in Algorithm 1. The trainable parameters $\theta = \{W_k^1, W_k^2, \lambda_k, \mu_k\}_{k=1}^K$ are learned from data by jointly minimizing

$$\theta^* = \underset{\theta=\{W_k^1,W_k^2,\lambda_k,\mu_k\}}{\arg\min} \frac{1}{n_t}\sum_{i=1}^{n_t} \|f(x_i;\theta) - s_i^{true}\|_2^2. \quad (21)$$

The resulting decision mapping of Example 16 is illustrated in Fig. 5(d). Unlike Example 15 which only learns the hyperparameters, the unfolded ADMM in Example 16 jointly learns the hyperparameters and the objective parameters $\theta^{\rm o}$ per each iteration. This can be viewed as if each iteration follows a different objective, such that the output after K iterations most accurately matches the desired value. While each layer in Example 16 has different parameters, one can enforce identical parameters across layers. The DNN can realize a larger family of mappings compared with the original model-based optimizer, which serves as a principled initialization for the system, rather than its fixed structure as in Example 15.

A popular application of deep unfolding, which follows the rationale of Example 16 with both $\theta^{\rm o}$ and $\theta^{\rm h}$ learned end-to-end, is the unfolding of ISTA into learned ISTA (LISTA) [29].

Example 17: The ISTA optimizer in Example 10 can be unfolded into the LISTA DNN architecture by fixing K iterations and replacing the update step in (15) with

$$s_{k+1} = \mathcal{T}_{\lambda_k}\left(W_k^1 x + \mu_k W_k^2 s_k\right). \quad (22)$$

For $W_k^1 = \mu H^T$, $W_k^2 = I - \mu H^T H$, $\mu_k = 1$ and $\lambda_k = \mu\rho$, (22) coincides with model-based ISTA. The trainable parameters $\theta = \left[\{W_k^1, W_k^2, \mu_k\}_{k=1}^K\right]$ are learned from data via end-to-end training as in (21).

Summary: Deep unfolding designs dedicated DNNs whose architecture follows iterative optimization algorithms. Compared with conventional DNNs applied to similar tasks, deep unfolded networks are more task-specific and less parameterized, as the setting of their trainable parameters and their interconnection is based on a iterative solver suitable for such problems. As a result, deep unfolded networks tend to require less data for training compared with standard DNNs, and often achieve improved performance and generalization

[6]. Furthermore, deep unfolded networks offer improved interpretability, as one can identify the meaning of some of its internal features, a task which is rarely achievable in conventional DNNs. In deep unfolded networks, the features exchanged between its layers represent the output of each iteration as in type T4, and can thus be associated with an estimate of the decision which is gradually refined as in iterative optimization.

Compared with model-based optimization, converting an iterative solver (T4) into a DNN with K layers (T5) typically results in faster inference. The fact that iteration-specific parameters are learned end-to-end allows deep unfolded networks to operate with much fewer layers compared with the number of iterations required by the model-based optimizer to achieve similar performance. Furthermore, the increased parameterization improves the abstractness of the decision rule, particularly when both the hyperparameters θ^{h} and the objective parameters θ^{o} are jointly learned as in Examples 16 and 17. Such unfolded networks depart from the iterative algorithm from which they originates, allowing them to overcome mismatches and approximation errors associated with the need to specify a mathematically tractable surrogate objective for decision making. In particular, training an unfolded network designed with a mismatched model using data corresponding to the true underlying scenario typically yields improved performance compared to the model-based iterative algorithm with the same model-mismatch, as the unfolded network can learn to compensate for this mismatch [31].

C. DNN-AIDED OPTIMIZATION

The third model-based deep learning strategy combines conventional DNN architectures with model-based optimization to enable the latter to operate reliably in complex domains. The rationale here is to preserve the objective and structure of a model-based decision mapping suitable for the problem at hand based on the available domain knowledge, while augmenting computations that rely on approximations and missing domain knowledge with model-agnostic DNNs. DNN-aided optimizers thus aim at benefiting from the best of both worlds by accounting in principled manner for the available domain knowledge while using deep learning to cope with the elusive aspects of the problem description.

Unlike the aforementioned strategies of learned optimizers and deep unfolding, which are relatively systematic and can be viewed as recipe-style methodologies, DNN-aided optimization accommodates a broad family of different techniques for augmenting model-based optimizers with DNNs. We next discuss some representative DNN-aided optimization approaches.

Examples: The straight-forward application of DNN-aided optimization replaces an internal computation of a model-based solver with a dedicated DNN, converting it into a trainable model-based deep learning system. An example of how this is done, based on [32], is detailed next.

Example 18: Consider again the setting of Example 13, where a Kalman filter is designed without knowing the distribution of the noise signals in (7). Since the dependency on the noise statistics in the Kalman filter is encapsulated in the Kalman gain L_t, its computation can be replaced with a trainable DNN, and thus (11) is replaced with

$$\hat{z}_t = A\hat{z}_{t-1} + h_\theta(x_t, s_{t-1})\left(x_t - C(A\hat{z}_{t-1} + Bs_{t-1})\right), \tag{23}$$

where h_θ is a DNN with parameters θ. Particularly, since L_t is updated recursively, its learned computation is carried out with an RNN. By letting $f(\cdot;\theta)$ be the latent state estimate computed using (23) with parameters θ, the overall system is trained end-to-end via

$$\theta^* = \arg\min_\theta \frac{1}{n_t T}\sum_{i=1}^{n_t}\sum_{t=1}^{T} \|f(x_{t,i}, s_{t-1,i};\theta) - z_{t,i}\|_2^2. \tag{24}$$

Example 18 was shown in [32] to overcome non-linearities and mismatches in the state-space model, outperforming the classical Kalman filter while retaining its data efficiency and interpretability. It is emphasized though that Example 18 represents one approach to combine Kalman filtering with DNN-aided optimization methodology. Additional techniques include the usage of an external DNN operating in parallel with the filter and providing correction terms, as proposed in [10], and the application of the Kalman filter to learned features extracted by a DNN as in [33], exemplified next.

Example 19: Consider a state-space model as in (7) where the observations x_t are complex and non-linear, i.e., (7b) does not hold. One can still apply a Kalman filter designed for a linear Gaussian setting by applying a DNN $h_\theta(\cdot)$ to transform x_t into features that follow the state-space model assumed by the model-based filter. Here, (11) becomes

$$\hat{z}_t = A\hat{z}_{t-1} + L_t\left(h_\theta(x_t) - C(A\hat{z}_{t-1} + Bs_{t-1})\right), \tag{25}$$

and the tuning is done via end-to-end training as in (24).

The latter approach, of applying a model-based optimizer to features extracted by a DNN as in Example 19, can also be used to enforce decisions made by a DNN to comply to some underlying physical requirements, see, e.g., [34].

The above examples build upon the differentiability of the model-based solver to train the DNN augmented into the method end-to-end. Nonetheless, DNN-aided optimization can also augment model-based methods with DNNs that are pre-trained, possibly even in an unsupervised manner thus alleviating the dependence on the availability of labeled data. One such family of DNN-aided optimization techniques, referred to as plug-and-play networks [8], is exemplified next.

Example 20: Consider the application of ADMM (Algorithm 1) to solving (4). Computing the proximal mapping in the second update step is often challenging, as the ability to evaluate the prior $\phi(\cdot)$ is required, which in practice may be unavailable or involve exhaustive computations. Nonetheless, the proximal mapping is invariant of the task, and can be viewed as a denoiser for samples in $\mathcal{S}$, e.g., high-resolution

images for the setting in Example 1. Denoisers are common DNN models, which can be trained in an unsupervised manner, and can reliably operate on signals with intractable priors (e.g., natural images). By letting $h_\theta(\cdot;\alpha)$ be a DNN trained to denoise data in $\mathcal{S}$ with noise level α, one can thus implement Algorithm 1 without specifying the prior $\phi(\cdot)$ by replacing the proximal mapping with [35]

$$v_{k+1} = h_\theta(s_{k+1} + u_k; \alpha_k). \tag{26}$$

The term plug-and-play is used to describe decision mappings as in Example 20 where pre-trained models are plugged into model-based optimizers without further tuning, as illustrated in Fig. 5(c). Nonetheless, this methodology can also incorporate deep learning into the optimization procedure by, e.g., unfolding the iterative optimization steps into a large DNN whose trainable parameters are those of the smaller networks augmenting each iteration, as in [36]. This approach allows to benefit from both the ability of deep learning to implicitly represent complex domains, as well as the inference speed reduction of deep unfolding along with its robustness to uncertainty and errors in the model parameters assumed to be known. Nonetheless, the fact that the iterative optimization must be learned from data in addition to the prior on $\mathcal{S}$ implies that larger amounts of labeled data are required to train the system, compared to using the model-based optimizer.

An alternative approach to augment model-based solvers with pre-trained DNNs is the usage of deep priors [9]. As opposed to plug-and-play networks, which augment the solver with a DNN in order to cope with complex modelling, deep priors use DNNs to directly compute the (possibly intractable) decision rule objective, as shown in the next example.

Example 21: Consider again the setting in Example 20, where one aims at solving (4) while the prior $\phi(\cdot)$ is unavailable and possibly intractable. However, now let us assume that we have access to some bijective mapping from some latent space $\mathcal{Z}$ to the signal space $\mathcal{S}$, denoted $g: \mathcal{Z} \mapsto \mathcal{S}$, such that the prior term $\phi(s)$ can be written in terms of z as $\phi(s) = \tilde{\phi}(z)|_{z=G^{-1}(s)}$. In this case, the MAP rule in (4) becomes

$$s = g(\hat{z}), \quad \hat{z} = \arg\min_z \frac{1}{2}\|x - HG(z)\|_2^2 + \sigma^2\tilde{\phi}(z). \tag{27}$$

Deep generative priors [9] use a pre-trained DNN-based prior $h_\theta(\cdot)$, typically a generative network trained to map Gaussian vectors to $\mathcal{S}$. The resulting objective becomes:

$$\hat{z} = \arg\min_z \frac{1}{2}\|x - Hh_\theta(z)\|_2^2 + \lambda\|z\|_2^2. \tag{28}$$

Even though the exact formulation of $h_\theta(\cdot)$ may be highly complex, one can tackle (28) via first-order optimization, building upon the fact that DNNs allow simple computation of gradients via backpropagation. These gradients are taken not with respect to the weights (as done in conventional DNN training), but with respect to the input of the network.

Summary: DNN-aided optimizers implement decision boxes via an interleaving of model-based principled mathematical procedures and trained DNNs. The approach is particularly suitable for enabling decision making in complex environments with partial domain knowledge, where the latter is used to determine the suitable model-based optimizer, whose complex computations are replaced with DNNs. Such augmentations facilitate the model-based optimizer in coping with mismatches in its objective model and its parameters, and makes it applicable in complex domains.

Compared with the direct application of deep learning for the decision mappings, DNN-aided optimizers are less generic and more task-specific due to the fact that they preserve the structure of a model-based optimizer. This property does not only facilitate their training procedure, which can sometimes be done unsupervised as in Examples 20-21, but also yields decision rules that are interpretable and suitable for their task. This interpretability can be exploited to extract additional measures of interest, e.g., uncertainty, as shown in [37] for the DNN-aided Kalman filter in Example 18; such measures, which are naturally obtained in model-based methods while being challenging to characterize for black-box DNNs, are often of importance in some applications.

VI. RESULTS

In this section we experimentally exemplify model-based deep learning methodology in a broad range of diverse application areas, including ultrasound imaging, optics, digital communications, and tracking of dynamic systems.

A. ULTRASOUND IMAGING

We first demonstrate the ability of deep unfolding, and particularly of LISTA-like architectures as detailed in Example 17, to facilitate the processing of ultrasound images. Our first example is taken from [38], which trained a deep unfolded decision box for clutter removal in contrast-enhanced ultrasound. Here the data was modeled as comprising a low-rank clutter background and a sparse blood flow image depicting the contrast agents. A generalization of ISTA was then applied to robust principled component analysis (RPCA) optimization leading to an unfolded network referred to as CORONA: Convolutional rObust pRincipal cOmpoNent Analysis. Here, both the context x and the decision s are maximum intensity projection ultrasound images; the decision rule type type is $K = 10$ iterations of a generalized ISTA, i.e., type T4, whose objective parameters $\theta^{\rm o}$ and hyperparameters $\theta^{\rm h}$ are tuned per-iteration from data via end-to-end training using the empirical risk (16) with the ℓ_2 loss $l_{\rm Est}(\cdot)$, computed over a set of $n_t = 4800$ images.

An experimental study of this application, showing that deep unfolding can infer both quickly and reliably, is presented in Fig. 6. Fig. 6(c) shows the recovered ultrasound (contrast agents) image from a cluttered image (Fig. 6(a)) achieved using deep unfolding of RPCA. Comparing the recovered image to the ground-truth in Fig. 6(b) demonstrates the accuracy in using a DNN to imitate the operations of

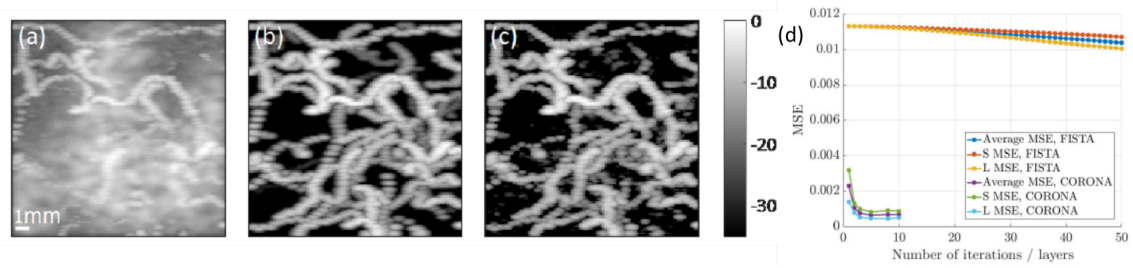


FIGURE 6. Experimental results (reproduced from [38]) for recovering ultrasound contrast agents from cluttered maximum intensity projection images: a) the observed image; b) the ground-truth sparse contrast agents; c) image recovered by deep unfolding; d) MSE versus iterations/layers of deep unfolded network (CORONA) compared to fast ISTA.

the generalized ISTA algorithm in a learned fashion. Furthermore, the fact that the unfolded network learns its parameters from data for each layer allows it to infer with a notably reduced number of layers compared to the corresponding number of iterations required by the model-based algorithm, which utilizes its full domain knowledge in applying the hard-coded iterative procedure. This is illustrated in Fig. 6(d) which demonstrates that the trained unfolded network can achieve with only a few layers a mean-squared error (MSE) accuracy which the model-based fast ISTA of [39] does not approach even in 50 iterations.

Deep unfolding can also be applied for super-resolution in ultrasound using micro bubbles. For instance, the work [40] applied LISTA for ultrasound-based breast lesion characterization. Here, the input x is a low-resolution ultrasound image, while the decision s is a high-resolution image. The decision rule is again an unfolded iterative algorithm, i.e., T4, with $\theta^{\rm o}$ and $\theta^{\rm h}$ jointly learned end-to-end as in Example 17.

The ability of LISTA to increase ultrasound resolution and facilitate diagnosis is demonstrated in Fig. 7. Here, a super-resolved recovery of a fibroadenoma (Fig. 7, top) shows an oval, well circumscribed mass with homogeneous high vascularization; a cyst (Fig. 7, middle) is visualized as a round structure with high concentration of blood vessels at the periphery of lesion; while an invasive ductal carcinoma (Fig. 7, bottom) shows an irregular mass with ill-defined margins, high concentration of blood vessels at the periphery of the mass, and a low concentrations of blood vessels at the center. These resolved features are not visually identifiable in the low-resolution.

B. MICROSCOPY IMAGING

Next, we demonstrate the application of model-based deep learning techniques in optics, considering again the usage of LISTA (applied in the correlation domain) for super-resolution. The context x is a low resolution microscopy image, and s^{true} is a high resolution image, with the decision rule being $K = 10$ iterations of ISTA (T4) where the parameters $\theta^{\rm o}$ and $\theta^{\rm h}$ are jointly learned from data to minimize the empirical risk with the ℓ_2 loss as its design objective.

Experimental results of applying the deep unfolded mapping trained for super-resolution in microscopy imaging are depicted in Fig. 8, which is reproduced from [41] based

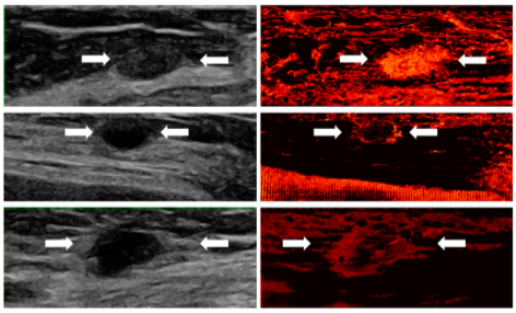


FIGURE 7. Experimental results (reproduced from [40]) for applying LISTA for super-resolution in human scans of three lesions in breasts of three patients. Left: B-mode images; Right: super-resolution recoveries; Top: fibroadenoma (benign); Middle: cyst (benign); Bottom: invasive ductal carcinoma (malignant).

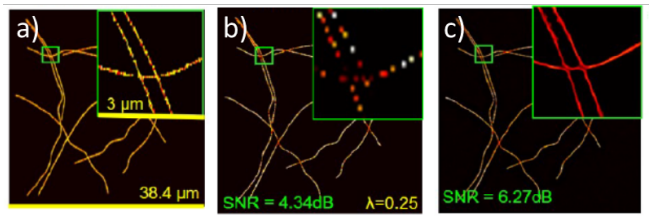


FIGURE 8. Sample results (reproduced from [41]) for applying deep unfolding for recovery of high resolution image. a) simulated ground truth tubulin structure; b) model-based recovery with hyperparameter $\theta^{\rm h} = 0.25$; c) deep unfolded resolved image.

on the method from [42]. Here, a super-resolved image is reconstructed from a simulated tubulins data set, composed of 350 high-density frames, where the deep unfolded network (Fig. 8(c)) is compared with 100 iterations of the model-based iterative sparse recovery algorithm from which it originates (Fig. 8(b)). These results demonstrate the ability of deep unfolding, where both the objective parameters $\theta^{\rm o}$ and the hyperparameters $\theta^{\rm h}$ are jointly learned end-to-end, to yield more abstract models that can overcome mismatches due to the surrogate objectives of model-based optimization with complex data, where mathematical descriptions are rarely accurate.

C. DIGITAL COMMUNICATIONS

The experimental evaluations so far focused on deep unfolding methodology and on tasks where the context x is an image. We proceed to a different family of tasks, arising in

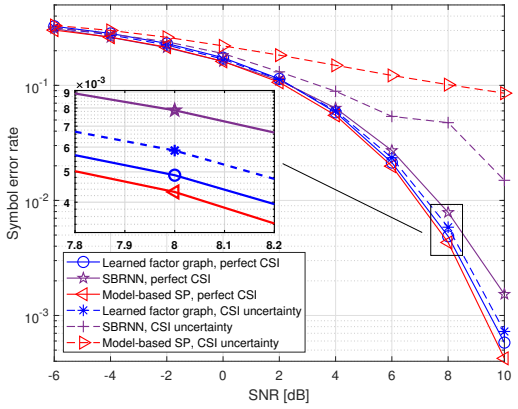


FIGURE 9. Experimental results from [43] of learned factor graphs compared to the model-based SP and the sliding bidirectional RNN (SBRNN) of [45]. *Perfect CSI* implies that the system is trained and tested using samples from the same channel, while in *CSI uncertainty* they are trained using samples from a set of different channels.

the operation of digital receivers, and present a a numerical example for DNN-aided optimization. We consider a scenario of symbol detection over causal stationary communication channels with finite memory, reproduced from [43]. Here, the input x is a real valued vector representing samples from an observed channel output, and s^{true} is a vector of the transmitted symbols, whose entries take value in a discrete binary phase shift keying constellation. The decision mapping which minimizes the error is the MAP rule which in such scenarios can be implemented with reduced complexity using the sum-product (SP) algorithm [44]. This mapping relies on accurate knowledge of the underlying channel which is captured using a factor graph. The parameters of the decision rule are the weights of an internal DNN used for evaluating the function nodes of the graph, and these parameters are tuned by minimizing the empirical cross entropy loss on a data set comprised of observations and their corresponding symbols.

Fig. 9 depicts the numerically evaluated symbol error rate achieved by applying a DNN-aided SP algorithm where deep learning is used to learn to compute the function nodes of the factor graph from $n_t = 5000$ labeled samples. The results are compared to the performance of the model-based SP, that requires complete knowledge of the underlying statistical model, as well as the sliding bidirectional RNN detector proposed in [45] for such setups, which utilizes a conventional DNN architecture. Fig. 9 demonstrates the ability of learned factor graphs to enable accurate message passing inference in a data-driven manner, as the performance achieved using learned factor graphs approaches that of the SP algorithm, which operates with full knowledge of the underlying statistical model. The numerical results also demonstrate that combining model-agnostic DNNs with model-aware optimization notably improves robustness to model uncertainty compared to applying the SP algorithm with the inaccurate model. Furthermore, it also observed that the principled incorporation of DNNs and SP inference allows to achieve improved performance compared to utilizing black-box DNN architectures such as the sliding bidirectional RNN detector, with limited training data.

	EKF	UKF	PF	KalmanNet	RNN
MSE [dB]	-6.432	-5.683	-5.337	**-11.284**	17.355
Run-time [sec]	5.440	6.072	62.946	**4.699**	2.291

TABLE 1. MSE performance and run-time of the DNN-aided KalmanNet, end-to-end RNN, and the model-based EKF, UKF, and PF.

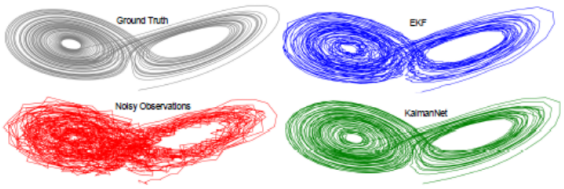


FIGURE 10. Tracking a single trajectory of the Lorentz attractor chaotic system using the DNN-aided KalmanNet compared with the model-based EKF (reproduced from [32]).

D. TRACKING OF DYNAMIC SYSTEMS

We conclude our experimental results with the application of DNN-aided optimization for tracking of dynamic systems. Here, we use the DNN-aided Kalman filter of Example 18 to track the Lorentz attractor non-linear chaotic system. Both the context and the decision are three-dimensional vectors, representing 3000 noisy observations and the trajectory of the Lorenz attractor, respectively. The decision rule is a combination of a DNN (T5) and an affine mapping (T1) trained end-to-end from supervised data, as detailed in Example 18. We compare this model-based deep learning mapping with several model-based tracking algorithms designed for such settings – the extended Kalman filter (EKF); unscented Kalman filter (UKF); and particle filter (PF) – as well as to a black-box RNN trained end-to-end.

The results, reproduced from [32] are summarized in Table 1, and a representative reconstruction is visualized in Fig. 10. It is observed in Table 1 that the gains of DNN-aided optimization here are two-fold: first, it achieves the best MSE results due to its incorporation of the state-space model as domain knowledge along with a DNN which learns to handle the complex dynamics and overcome the mismatches induced by the surrogate objective. Furthermore, the integration of deep learning allows DNN-aided optimization to operate more quickly than its model-based counterparts, as some of the internal exhaustive computations of the algorithms are replaced with a DNN inferring at fixed complexity.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[2] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
[3] N. Shlezinger, J. Whang, Y. C. Eldar, and A. G. Dimakis, "Model-based deep learning," arXiv preprint arXiv:2012.08405, 2020.

[4] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin, "Learning to optimize: A primer and a benchmark," arXiv preprint arXiv:2103.12828, 2021.

[5] A. Maier, H. Köstler, M. Heisig, P. Krauss, and S. H. Yang, "Known operator learning and hybrid machine learning in medical imaging—a review of the past, the present, and the future," *Progress in Biomedical Engineering*, 2022.

[6] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Process. Mag.*, vol. 38, no. 2, pp. 18–44, 2021.

[7] A. Agrawal, S. Barratt, and S. Boyd, "Learning convex optimization models," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 8, pp. 1355–1364, 2021.

[8] R. Ahmad, C. A. Bouman, G. T. Buzzard, S. Chan, S. Liu, E. T. Reehorst, and P. Schniter, "Plug-and-play methods for magnetic resonance imaging: Using denoisers for image recovery," *IEEE Signal Process. Mag.*, vol. 37, no. 1, pp. 105–116, 2020.

[9] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, "Compressed sensing using generative models," in *International Conference on Machine Learning*. JMLR, 2017, pp. 537–546.

[10] V. Garcia Satorras, Z. Akata, and M. Welling, "Combining generative and discriminative models for hybrid inference," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[11] N. Shlezinger, N. Farsad, Y. C. Eldar, and A. J. Goldsmith, "Model-based machine learning for communications," arXiv preprint arXiv:2101.04726, 2021.

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

[13] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[14] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[15] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in optimization*, vol. 1, no. 3, pp. 127–239, 2014.

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

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[18] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[19] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[22] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, "Learning convex optimization control policies," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 361–373.

[23] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[24] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*, 2015, pp. 2113–2122.

[25] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 1540–1552.

[26] S. T. Barratt and S. P. Boyd, "Fitting a Kalman smoother to data," in *2020 American Control Conference (ACC)*. IEEE, 2020, pp. 1526–1531.

[27] I. Sutskever, *Training recurrent neural networks*. University of Toronto, 2013.

[28] L. Xu and R. Niu, "EKFNet: Learning system noise statistics from measurement data," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2021, pp. 4560–4564.

[29] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *International Conference on International Conference on Machine Learning*, 2010, pp. 399–406.

[30] J. Johnston, Y. Li, M. Lops, and X. Wang, "ADMM-Net for communication interference removal in stepped-frequency radar," *IEEE Trans. Signal Process.*, vol. 69, pp. 2818–2832, 2021.

[31] S. Khobahi, N. Shlezinger, M. Soltanalian, and Y. C. Eldar, "LoRD-Net: Low resolution detection network for deep low-resolution receivers," *IEEE Trans. Signal Process.*, vol. 69, pp. 5651–5664, 2021.

[32] G. Revach, N. Shlezinger, X. Ni, A. L. Escoriza, R. J. van Sloun, and Y. C. Eldar, "KalmanNet: Neural network aided Kalman filtering for partially known dynamics," *IEEE Trans. Signal Process.*, vol. 70, pp. 1532–1547, 2022.

[33] H. Coskun, F. Achilles, R. DiPietro, N. Navab, and F. Tombari, "Long short-term memory Kalman filters: Recurrent neural estimators for pose regularization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5524–5532.

[34] T. Zhao, X. Pan, M. Chen, and S. H. Low, "Ensuring DNN solution feasibility for optimization problems with convex constraints and its application to DC optimal power flow problems," arXiv preprint arXiv:2112.08091, 2021.

[35] S. H. Chan, X. Wang, and O. A. Elgendy, "Plug-and-play ADMM for image restoration: Fixed-point convergence and applications," *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 84–98, 2016.

[36] D. Gilton, G. Ongie, and R. Willett, "Neumann networks for inverse problems in imaging," *IEEE Transactions on Computational Imaging*, vol. 6, pp. 328–343, 2019.

[37] I. Klein, G. Revach, N. Shlezinger, J. E. Mehr, R. J. van Sloun, and Y. Eldar, "Uncertainty in data-driven Kalman filtering for partially known state-space models," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2022.

[38] O. Solomon, R. Cohen, Y. Zhang, Y. Yang, Q. He, J. Luo, R. J. van Sloun, and Y. C. Eldar, "Deep unfolded robust PCA with application to clutter suppression in ultrasound," *IEEE Trans. Med. Imag.*, vol. 39, no. 4, pp. 1051–1063, 2019.

[39] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[40] O. Bar-Shira, A. Grubstein, Y. Rapson, D. Suhami, E. Atar, K. Peri-Hanania, R. Rosen, and Y. C. Eldar, "Learned super resolution ultrasound for improved breast lesion characterization," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2021, pp. 109–118.

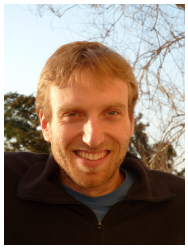
[41] Y. B. Sahel, J. P. Bryan, B. Cleary, S. L. Farhi, and Y. C. Eldar, "Deep unrolled recovery in sparse biological imaging," *IEEE Signal Process. Mag.*, vol. 39, no. 2, pp. 45–57, 2022.

[42] G. Dardikman-Yoffe and Y. C. Eldar, "Learned SPARCOM: unfolded deep super-resolution microscopy," *Optics express*, vol. 28, no. 19, pp. 27 736–27 763, 2020.

[43] N. Shlezinger, N. Farsad, Y. C. Eldar, and A. J. Goldsmith, "Learned factor graphs for inference from stationary time sequences," *IEEE Trans. Signal Process.*, vol. 70, pp. 366–380, 2021.

[44] H.-A. Loeliger, "An introduction to factor graphs," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 28–41, 2004.

[45] N. Farsad and A. Goldsmith, "Neural network detection of data sequences in communication systems," *IEEE Trans. Signal Process.*, vol. 66, no. 21, pp. 5663–5678, 2018.



NIR SHLEZINGER (M'17) is an assistant professor in the School of Electrical and Computer Engineering in Ben-Gurion University, Israel. He received his B.Sc., M.Sc., and Ph.D. degrees in 2011, 2013, and 2017, respectively, from Ben-Gurion University, Israel, all in electrical and computer engineering. From 2017 to 2019 he was a postdoctoral researcher in the Technion, and from 2019 to 2020 he was a postdoctoral researcher in Weizmann Institute of Science, where he was awarded the FGS prize for outstanding research achievements. His research interests include communications, information theory, signal processing, and machine learning.

YONINA C. ELDAR (S'98–M'02–SM'07–F'12) received the B.Sc. degree in Physics in 1995 and the B.Sc. degree in Electrical Engineering in 1996 both from Tel-Aviv University (TAU), Tel-Aviv, Israel, and the Ph.D. degree in Electrical Engineering and Computer Science in 2002 from the Massachusetts Institute of Technology (MIT), Cambridge. She is currently a Professor in the Department of Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel. She was previously a Professor in the Department of Electrical Engineering at the Technion, where she held the Edwards Chair in Engineering. She is also a Visiting Professor at MIT, a Visiting Scientist at the Broad Institute, and an Adjunct Professor at Duke University and was a Visiting Professor at Stanford. She is a member of the Israel Academy of Sciences and Humanities (elected 2017), an IEEE Fellow and a EURASIP Fellow. Her research interests are in the broad areas of statistical signal processing, sampling theory and compressed sensing, learning and optimization methods, and their applications to biology, medical imaging and optics.

Dr. Eldar has received many awards for excellence in research and teaching, including the IEEE Signal Processing Society Technical Achievement Award (2013), the IEEE/AESS Fred Nathanson Memorial Radar Award (2014), and the IEEE Kiyo Tomiyasu Award (2016). She was a Horev Fellow of the Leaders in Science and Technology program at the Technion and an Alon Fellow. She received the Michael Bruno Memorial Award from the Rothschild Foundation, the Weizmann Prize for Exact Sciences, the Wolf Foundation Krill Prize for Excellence in Scientific Research, the Henry Taub Prize for Excellence in Research (twice), the Hershel Rich Innovation Award (three times), the Award for Women with Distinguished Contributions, the Andre and Bella Meyer Lectureship, the Career Development Chair at the Technion, the Muriel & David Jacknow Award for Excellence in Teaching, and the Technion's Award for Excellence in Teaching (two times). She received several best paper awards and best demo awards together with her research students and colleagues including the SIAM outstanding Paper Prize, the UFFC Outstanding Paper Award, the Signal Processing Society Best Paper Award and the IET Circuits, Devices and Systems Premium Award, was selected as one of the 50 most influential women in Israel and in Asia, and is a highly cited researcher. She was a member of the Young Israel Academy of Science and Humanities and the Israel Committee for Higher Education. She is the Editor in Chief of Foundations and Trends in Signal Processing, a member of the IEEE Sensor Array and Multichannel Technical Committee and serves on several other IEEE committees. In the past, she was a Signal Processing Society Distinguished Lecturer, member of the IEEE Signal Processing Theory and Methods and Bio Imaging Signal Processing technical committees, and served as an associate editor for the IEEE Transactions On Signal Processing, the EURASIP Journal of Signal Processing, the SIAM Journal on Matrix Analysis and Applications, and the SIAM Journal on Imaging Sciences. She was Co-Chair and Technical Co-Chair of several international conferences and workshops. She is author of the book "Sampling Theory: Beyond Bandlimited Systems" and co-author of five other books published by Cambridge University Press.

STEPHEN P. BOYD (F' 99) is the Samsung Professor of engineering, and Professor of electrical engineering in the Information Systems Laboratory at Stanford University, with courtesy appointments in computer science and management science and engineering. He received the A.B. degree in mathematics from Harvard University, USA, in 1980, and the Ph.D. in electrical engineering and computer science from the University of California, USA, in 1985, and then joined the faculty at Stanford. His current research interests include convex optimization applications in control, signal processing, machine learning, finance, and circuit design.