MODEL-BASED DEEP LEARNING: KEY APPROACHES AND DESIGN GUIDELINES

Nir Shlezinger, Jay Whang, Yonina C. Eldar, and Alexandros G. Dimakis

ABSTRACT

Signal processing, communications, and control have traditionally relied on classical statistical modeling techniques. Such model-based methods tend to be sensitive to inaccuracies and may lead to poor performance when real systems display complex or dynamic behavior. On the other hand, purely data-driven approaches are becoming increasingly popular. Deep neural networks (DNNs) employ a highly flexible function class to learn mappings from data, and demonstrate excellent performance. However, DNNs typically require massive amounts of data and immense computational resources, limiting their applicability for some signal processing scenarios. We consider hybrid techniques that combine principled mathematical models with data-driven systems to benefit from the advantages of both approaches. Such *model-based deep learning* methods exploit both partial domain knowledge, via mathematical structures designed for specific problems, as well as learning from limited data. Here, we survey leading approaches for studying and designing model-based deep learning systems, along with concrete design guidelines and signal processing oriented examples.

***Index Terms*—** Model-based deep learning

1. INTRODUCTION

Traditional signal processing is dominated by algorithms that are based on mathematical models which are hand-designed from domain knowledge. Such knowledge typically comes from postulated statistical models. These domain-knowledge-based processing algorithms, referred to henceforth as *model-based methods*, infer based on knowledge of the underlying model relating the observations and the desired information. Model-based methods do not rely on data to learn their mapping, though data is often used to estimate a small number of parameters. Fundamental techniques like the Kalman filter and message passing algorithms are model-based methods. Classical models rely on simplifying assumptions that make them tractable, understandable and computationally efficient. However, simple models frequently fail to represent nuances of high-dimensional complex data and dynamic variations.

The incredible success of deep learning has initiated a general data-driven mindset. It is currently fashionable to replace simple principled models with purely data-driven pipelines, such as Deep neural networks (DNNs), trained with massive labeled datasets. The benefits of data-driven methods over model-based approaches are twofold: First, purely-data-driven methods do not rely on analytical approximations and thus can operate when the model is not known. Second, for complex systems, data-driven methods are able to extract semantic information from observed data [1]. This is difficult to achieve analytically, even when the model is known perfectly.

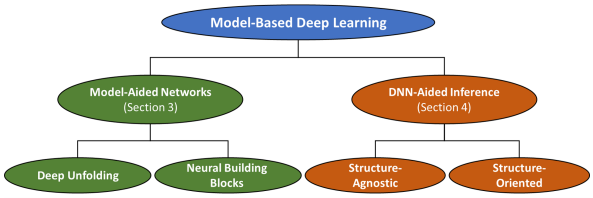

Fig. 1: Model-based deep learning strategies.

The fact that massive data sets are typically required to train DNNs to learn a desirable mapping, and the computational burden of training and utilizing these networks, may constitute a major drawback in various signal processing oriented applications. This severely limits their applicability on hardware-limited and portable devices [2]. Furthermore, DNNs are commonly utilized as black-boxes; understanding how their predictions are obtained tends to be challenging, and they do not yet offer the interpretability, flexibility, versatility, and reliability of model-based methods [3].

The limitations associated with model-based methods and black-box data-driven systems gave rise to a multitude of techniques aiming to benefit from both approaches. In this work we overview the leading strategies for combining domain knowledge and data via model-based deep learning. To that aim, we present a unified framework for studying and designing hybrid model-based/data-driven systems, while being geared towards signal processing oriented problems. The proposed framework builds upon the insight that such systems can be divided based on the component which eventually infers: The first category are DNNs whose architecture is specialized to the specific problem using model-based methods, referred to here as *model-aided networks*. The second one are techniques in which inference is carried out by a model-based algorithm whose operation is empowered by deep learning, which we refer to as *DNN-aided inference*. Based on this division, we provide concrete guidelines for studying model-based deep learning systems. An illustration of this division of model-based deep learning schemes is depicted in Fig. 1.

The rest of this article is organized as follows: Sec-

N. Shlezinger is with the School of ECE, Ben-Gurion University of the Negev (e-mail: nirshl@bgu.ac.il). J. Whang is with the Department of CS, University of Texas at Austin (e-mail: jaywhang@cs.utexas.edu). Y. C. Eldar is with the Faculty of Math and CS, Weizmann Institute of Science (e-mail: yonina@weizmann.ac.il). A. G. Dimakis is with the Department of ECE, University of Texas at Austin (e-mail: dimakis@austin.utexas.edu).

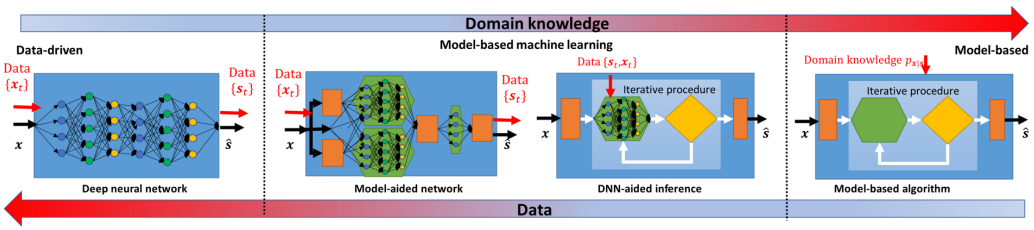

Fig. 2: Illustration of model-based versus data-driven inference. Red arrows represent computations performed before the inference stage.

tion 2 discusses the concepts of model-based and data-driven schemes. The hybrid strategies of model-aided networks and DNN-aided inference are detailed in Sections 3-4, respectively. Finally, we provide concluding remarks in Section 5.

2. MODEL-BASED VERSUS DATA-DRIVEN

We focus on systems that perform inference based on a set of observed variables. Here, an input variable $\boldsymbol{x} \in \mathcal{X}$ is mapped to a prediction of a label variable $\boldsymbol{s} \in \mathcal{S}$, denoted $\hat{\boldsymbol{s}}$, where $\mathcal{X}$ and $\mathcal{S}$ are referred to as the *input space* and the *label space*, respectively. An inference rule can be expressed as $f : \mathcal{X} \mapsto \mathcal{S}$, where the fidelity is measured using a loss function $\mathcal{L}(f)$. Both model-based methods and data-driven schemes aim to design the inference rule $f(\cdot)$ to minimize $\mathcal{L}(f)$.

2.1. Model-Based Methods

Model-based algorithms set their inference rule, i.e. tune f, to minimize $\mathcal{L}(\cdot)$ based on domain knowledge, namely, prior knowledge of the underlying statistics relating the input $\boldsymbol{x}$ and the label $\boldsymbol{s}$, such as the generative conditional distribution $p_{\boldsymbol{x}|\boldsymbol{s}}$. Model-based algorithms can often provably implement or approach the loss minimizing inference rule at controllable complexity, typically using iterative methods comprised of multiple stages, where each stage involves generic mathematical manipulations and model-specific computations.

Model-based methods do not rely on data, as illustrated in the right part of Fig. 2, though data is often used to estimate unknown model parameters. In practice, accurate knowledge of the underlying statistical model is typically unavailable, and it is commonly required to impose assumptions on the underlying statistics. Under 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.

2.2. Data-Driven Schemes

Data-driven systems learn their mapping from data. In a supervised setting, data is comprised of a training set consisting of n_t input-label pairs $\{\boldsymbol{x}_t, \boldsymbol{s}_t\}_{t=1}^{n_t}$. A leading data-driven strategy, used in deep learning, assumes a highly-expressive generic parametric model for $f(\cdot)$ which is dictated by a set of parameters denoted $\boldsymbol{\theta}$, and is written as $f_{\boldsymbol{\theta}}$. Deep learning typically implements $f_{\boldsymbol{\theta}}$ using a DNN, where $\boldsymbol{\theta}$ represents the network weights. By properly tuning $\boldsymbol{\theta}$ using the training set, one aims to obtain the desirable inference rule.

Purely-data-driven methods are model-agnostic, as illustrated in the left part of Fig. 2. The inference rule can be applied to a broad range of different problems, while the characteristics of the problem are encapsulated in the weights. Yet, one can incorporate some level of domain knowledge in the selection of the architecture. or by pre-processing of $\boldsymbol{x}$.

The generic nature of data-driven strategies induces some drawbacks, as learning a large number of parameters requires a massive data set to train on. Even when a sufficiently large data set is available, the training procedure is typically lengthy and involves high computational burden. Finally, their black-box nature implies that data-driven systems in general lack interpretability, making it difficult to provide performance guarantees and insights into its operation.

2.3. Model-Based Deep Learning

Model-based deep learning schemes incorporate domain knowledge in the form of an established model-based algorithm suitable for the problem at hand, combined with capabilities to learn from data via deep learning. Such systems tune their mapping of the input $\boldsymbol{x}$ based on both data, e.g., a labeled training set $\{\boldsymbol{s}_t, \boldsymbol{x}_t\}_{t=1}^{n_t}$, as well as some domain knowledge, such as partial knowledge of $p_{\boldsymbol{x}|\boldsymbol{s}}$. Such hybrid systems can typically learn their mappings from less data compared to purely model-agnostic DNNs, and commonly operate without full knowledge of the underlying model.

Hybrid model-based/data-driven design schemes can be divided into two main strategies, as illustrated in Fig. 2. The first of the two, which we refer to as *model-aided networks*, utilizes model-based methods as a form of domain knowledge in designing a DNN architecture, while using the resultant network for inference. The second strategy, which we call *DNN-aided inference systems*, uses conventional model-based methods for inference, while incorporating learning to make the resultant system more robust and model-agnostic.

3. MODEL-AIDED NETWORKS

Model-aided networks use custom DNN architectures that incorporate structures exhibited by model-based methods for the problem at hand, while using the resultant network for inference. Unlike conventional model-agnostic deep learning, model-aided networks utilize a unique architecture designed for a particular task by imitating the model-based algorithm with full model knowledge. We next describe two representative examples: deep unfolding and neural building blocks.

3.1. Deep Unfolding

Deep unfolding, also known as *deep unrolling* [3, 4], is a method for converting an iterative algorithm into a DNN by designing each layer to resemble a single iteration. Deep unfolding has been applied in image denoising [3, 5], sparse recovery [4,6], dictionary learning [7], communications [8–10], ultrasound [11, 12], and super resolution [13].

Design Outline:

1. Identify a suitable iterative optimization algorithm.
2. Fix a number of iterations in the optimization algorithm.
3. Design the layers to imitate the free parameters of each iteration in a trainable fashion.
4. Train the overall resulting network end-to-end.

Deep Unfolded Robust PCA (Example): Robust principal component analysis (PCA) refers to the problem of recovering a sparse matrix $\boldsymbol{S}$ where the measurements $\boldsymbol{X}$ are corrupted by low-rank clutter $\boldsymbol{L}$ as well as noise $\boldsymbol{W}$. In particular, the measurements are related to the sparse matrix via

$$\boldsymbol{X} = \boldsymbol{H}_1\boldsymbol{L} + \boldsymbol{H}_2\boldsymbol{S} + \boldsymbol{W}. \tag{1}$$

Here, $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ are measurement matrices of appropriate dimensions; $\boldsymbol{L}$ is a low-rank clutter matrix; and $\boldsymbol{W}$ is an additive noise signal. The domain knowledge that $\boldsymbol{S}$ is sparse and that $\boldsymbol{L}$ is low rank implies that their recovery can be obtained by solving the following relaxed optimization problem:

$$\min_{\boldsymbol{S},\boldsymbol{L}} \|\boldsymbol{X} - \boldsymbol{H}_1\boldsymbol{L} + \boldsymbol{H}_2\boldsymbol{S}\|_F^2 + \lambda_1\|\boldsymbol{L}\|_* + \lambda_2\|\boldsymbol{S}\|_{1,2}, \tag{2}$$

where $\|\cdot\|_F$, $\|\cdot\|_*$, and $\|\cdot\|_{1,2}$ are the Frobenious, nuclear, and mixed $\ell_{1,2}$ norms, while λ_1 and λ_2 are coefficients promoting low rankness of $\boldsymbol{L}$ and sparsity of $\boldsymbol{S}$, respectively. The robust PCA objective formulation in (2) can be solved by the generalization of iterative soft thresholding algorithm (ISTA) to the matrix domain, which we unfold into a DNN following [5].

To formulate the deep unfolded robust PCA system, we first fix a number of iterations Q, and design a DNN with Q layers, each imitating a single ISTA iteration. Letting $\hat{\boldsymbol{L}}_q$ and $\hat{\boldsymbol{S}}_q$ be the estimates of $\boldsymbol{L}$ and $\boldsymbol{S}$ produced at the output the qth layer, the operation of the unfolded iteration can be written as

$$\hat{\boldsymbol{L}}_{q+1} = \mathrm{SVT}_{\lambda_1^{(q)}}\left\{\boldsymbol{P}_5^{(q)}\hat{\boldsymbol{L}}_q + \boldsymbol{P}_3^{(q)}\hat{\boldsymbol{S}}_q + \boldsymbol{P}_1^{(q)}\boldsymbol{X}\right\}, \tag{3a}$$

$$\hat{\boldsymbol{S}}_{q+1} = \mathrm{T}_{\lambda_2^{(q)}}\left\{\boldsymbol{P}_6^{(q)}\hat{\boldsymbol{L}}_q + \boldsymbol{P}_4^{(q)}\hat{\boldsymbol{S}}_q + \boldsymbol{P}_2^{(q)}\boldsymbol{X}\right\}, \tag{3b}$$

where T_λ and SVT_λ are the soft thresholding operator and the singular value thresholding operator, respectively. The affine mappings are convolutional layers with kernels $\{\boldsymbol{P}_i^{(q)}\}$. This DNN is depicted in Fig. 3. To tune its trainable parameters, the network is trained in an end-to-end manner to minimize the Frobenius norm loss. As both $\boldsymbol{L}$ and $\boldsymbol{S}$ are estimated, the training accounts for the errors in both estimates.

Discussion: Deep unfolding uses domain knowledge to obtain a dedicated DNN design which resembles an iterative algorithm. Compared to conventional DNNs, unfolded networks are typically interpretable, tend to have fewer parameters, and can be trained more quickly [3, 8]. When the model

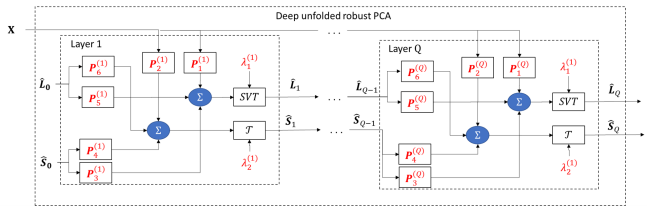

Fig. 3: Deep unfolded robust PCA illustration.

is accurately known, deep unfolding essentially incorporates it into the DNN. However, this may lead to degraded performance when the true model deviates from the one assumed in design, e.g., (1). Nonetheless, training an unfolded network designed with a mismatched model using data from the true setup can yield accurate inference as the unfolded network learns to compensate for this mismatch from data. Another advantage of deep unfolding over model-based optimization is in inference speed. For instance, unfolding the ISTA iterations into a DNN allows to infer with fewer layers compared to the number of iterations ISTA requires to converge.

3.2. Neural Building Blocks

Neural building blocks implement a DNN comprised of multiple sub-networks by representing a model-based algorithm suitable for the problem as an interconnection of building blocks. Each module carries out the computations of the different building blocks constituting the model-based algorithm [14], or to capture a known statistical relationship [15].

Design Outline:

1. Identify an algorithm or a flow-chart structure which is useful for the problem, and decompose it into building blocks.
2. Identify which of these building blocks should be learned from data, and what is their concrete task.
3. Design a dedicated neural network for each building block capable of learning to carry out its specific task.
4. Train the overall resulting network, either end-to-end or by training each building block network individually.

DeepSIC for MIMO Detection (Example): DeepSIC proposed in [14] is a hybrid model-based/data-driven implementation of the iterative soft interference cancellation (SIC) method [16]. The iterative SIC is a symbol detector for linear multiple-input multiple-output (MIMO) Gaussian channels:

$$\boldsymbol{x} = \boldsymbol{H}\boldsymbol{s} + \boldsymbol{w}. \tag{4}$$

Here, $\boldsymbol{x}$ is the $N\times 1$ observations, $\boldsymbol{H}$ is a known deterministic channel matrix, $\boldsymbol{w}$ is Gaussian noise, and $\boldsymbol{s}$ is comprised of K entries $\{s_k\}$ generated i.i.d. uniformly from $\mathcal{S} = \{\pm 1\}$. SIC operates in an iterative fashion where, in each iteration, an estimate of the conditional probability mass function (PMF) of s_k given $\boldsymbol{x}$ is generated for every k using the estimates of the interfering symbols $\{s_l\}_{l\neq k}$ obtained in the previous iteration. This iterative procedure is illustrated in Fig. 4(a).

DeepSIC learns to implement the iterative SIC from data as a set of neural building blocks, each implementing the two stages of interference cancellation and soft decoding. In particular, the kth building block of the qth iteration produces

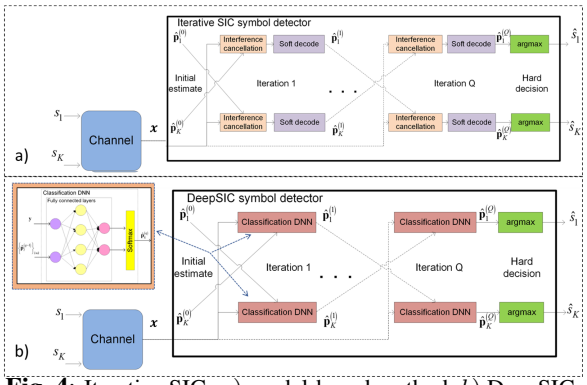

Fig. 4: Iterative SIC: a) model-based method; b) DeepSIC.

$\hat{\boldsymbol{p}}_k^{(q)}$, which is an estimate of the conditional PMF of s_k given $\boldsymbol{x}$ based on $\{\hat{\boldsymbol{p}}_l^{(q-1)}\}_{l\neq k}$. Such computations are naturally implemented by classification DNNs with a softmax output layer. Embedding these conditional PMF computations into the iterative SIC block diagram in Fig. 4(a) yields the overall receiver architecture depicted in Fig. 4(b). Using classification DNNs as the basic building blocks allows to compute conditional PMFs in complex non-linear setups. DeepSIC can thus implement SIC for arbitrary channel models. The DNNs can either be trained individually, where the qth iteration outputs are used as training inputs for the DNNs of the $q+1$th iteration. Alternatively, DeepSIC can be trained end-to-end.

Discussion: The main rationale in designing DNNs as interconnected neural building blocks is to facilitate learned inference by preserving the structured operation of a suitable model-based algorithm. Treating the algorithm as a set of building blocks with concrete tasks allows a DNN architecture designed to comply with this structure not only to learn to carry out the original model-based method from data, but also to make it more robust. In addition, the division into building blocks gives rise to the possibility to train each block separately. The main advantage in doing so is that training will likely be faster, though end-to-end training is likely to yield improved accuracy given sufficiently large training data.

4. DNN-AIDED INFERENCE

DNN-aided inference is a family of model-based deep learning algorithms in which DNNs are incorporated into model-based methods. Unlike model-aided networks, here inference is carried out using a traditional model-based method, while some of the intermediate computations are empowered by DNNs. We next describe both structure-agnostic and the structure-oriented approaches for DNN-aided inference.

4.1. Structure-Agnostic DNN-Aided Inference

The first family of DNN-aided inference utilizes deep learning to implicitly learn structures and statistical properties of the signal of interest, in a manner amenable to model-based optimization. Tackling such problems typically involves imposing a structure on the target signal. Deep learning allows to avoid such explicit constraints, thereby mitigating the

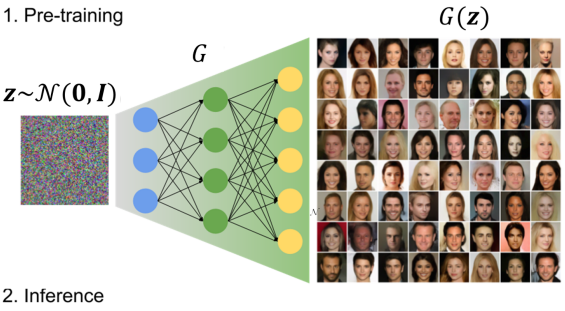

Fig. 5: High-level overview of CS with a DNN-based prior.

detrimental effects of crude approximation of the true structure. This can be achieved by using deep denoisers as learned proximal mappings, as in DNN-based plug-and-play methods [17–19]. DNN-based priors can also enable compressed sensing (CS) beyond the domain of sparse signals [20–23].

Design Outline:

1.Identify the optimization procedure for the problem at hand, given domain knowledge of the signal of interest.

2. The optimization parts which rely on complicated domain knowledge are replaced with a DNN.

3. The integrated data-driven module can either be trained separately from the inference system, possibly in an unsupervised manner as in [21], or end-to-end [24].

CS using Generative Models (Example): CS refers to the task of recovering some unknown signal from (possibly noisy) lower-dimensional observations. Here, we wish to reconstruct an unknown N-dimensional sparse signal $\boldsymbol{s}^*$ from:

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{s}^* + \boldsymbol{w}, \tag{5}$$

where $\boldsymbol{A}$ is an $M \times N$ random Gaussian matrix with entries $\boldsymbol{A}_{ij} \sim \mathcal{N}(0, 1/M)$, with $M < N$, and $\boldsymbol{w}$ is a noise vector. Our goal is to find the sparsest $\boldsymbol{s}$ that agrees with the noisy observations, which can be recovered by minimizing

$$\mathcal{L}_{\text{LASSO}}(\boldsymbol{s}) \triangleq \|\boldsymbol{A}\boldsymbol{s} - \boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{s}\|_1 \tag{6}$$

For $l = \|\boldsymbol{s}^*\|_0$ and $M = \Theta(l \log \frac{N}{l})$, the unique minimizer of the convex (6) equals $\boldsymbol{s}^*$ with high probability.

The method proposed in [21] uses a deep generative prior to replace the sparsity assumption on true signal $\boldsymbol{s}^*$, with a requirement that it lies in the range of a pre-trained generator network $G : \mathbb{R}^l \to \mathbb{R}^N$. To that aim, one first has to train a generative network G to map a latent vector $\boldsymbol{z}$ from an i.i.d. Gaussian distribution into a signal $\boldsymbol{s}$ in the domain of interest. Once a pre-trained G is available, it is used as an alternative prior for the inverse model in (5), as the range of G should only contain *plausible* signals. This is achieved by optimizing in the latent space to find $\boldsymbol{z}$ whose image $G(\boldsymbol{z})$ matches the observations, i.e., by minimizing the following loss function:

$$\mathcal{L}_{\text{CS}}(\boldsymbol{z}) = \|\boldsymbol{A}G(\boldsymbol{z}) - \boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{z}\|_2^2. \tag{7}$$

While (7) involves a highly non-convex function G, it is differentiable with respect to $\boldsymbol{z}$, so it can be tackled using

gradient-based optimization techniques. Once a suitable latent variable $\boldsymbol{z}$ is found, the actual signal we return is $G(\boldsymbol{z})$. An illustration of the system operation is depicted in Fig. 5.

Discussion: Using deep learning to empower regularized optimization builds upon the model-agnostic nature of DNNs to by pass the need for explicit regularization. The fact that structure-agnostic DNN-aided inference utilizes deep learning to capture the domain of interest facilitates using pre-trained networks. This property reduces the dependency of the system on massive amounts of labeled data, as e.g., deep generative priors are trained in an unsupervised manner, and thus relies mostly on unlabeled data, which are typically more accessible and easy to aggregate compared to labeled data. One can often utilize off-the-shelf pre-trained DNNs when such exist for domains related to that of interest, with possible adjustments to account for the subtleties of the problem.

4.2. Structure-Oriented DNN-Aided Inference

Structure-oriented DNN-aided inference algorithms utilize model-based methods designed to exploit an underlying statistical structure, while integrating DNNs to enable operation without additional explicit model characterization. These structures can be an a-priori known factorizable distribution, such as finite memory in communication channels [25–27]; as well as from established approximations, such as modelling of images as conditional random fields [28].

Design Outline:

1. A proper inference algorithm is chosen based on the available knowledge of the underlying statistical structure.
2. Once an algorithm is selected, we identify its model-specific parts, and replace them with compact DNNs.
3. The resulting DNNs are either trained individually, or the overall system can be trained in an end-to-end manner.

Learned Factor Graphs (Example): The sum-product (SP) algorithm efficiently computes marginals from factorizable distributions [29]. Consider the recovery of a series $\{s_i\}$ taking values in $\mathcal{S}$ from an observed sequence $\{x_i\}$ taking values in $\mathcal{X}$. The distribution of $\{s_i\}$ and $\{x_i\}$ obeys an lth-order Markovian stationary model, $l \geq 1$, such that the distribution of $\boldsymbol{x} = [x_1, \ldots, x_t]^T$ and $\boldsymbol{s} = [s_1, \ldots, s_t]^T$ satisfies

$$p(\boldsymbol{x}, \boldsymbol{s}) = \prod_{i=1}^{t} p\left(x_i|\boldsymbol{s}_{i-l}^{i}\right) p\left(s_i|\boldsymbol{s}_{i-l}^{i-1}\right), \tag{8}$$

where we write $\boldsymbol{s}_i^j \triangleq [s_i, s_{i+1}, \ldots, s_j]^T$ for $i < j$. When (8) is known the SP algorithm efficiently computes the maximum a-posteriori probability (MAP) detector [29]. This is achieved by defining the vector variable $\boldsymbol{s}_i \triangleq \boldsymbol{s}_{i-l+1}^{i} \in \mathcal{S}^l$, and the function $f\left(x_i, \boldsymbol{s}_i, \boldsymbol{s}_{i-1}\right) \triangleq p\left(x_i|\boldsymbol{s}_i, \boldsymbol{s}_{i-1}\right) p\left(\boldsymbol{s}_i|\boldsymbol{s}_{i-1}\right)$, The distribution (8) is then represented as a factor graph with t function nodes $\{f\left(x_i, \boldsymbol{s}_i, \boldsymbol{s}_{i-1}\right)\}$. Finally, the joint distribution of each $\boldsymbol{s}_i$ and $\boldsymbol{x}$ is computed by recursive message passing along its factor graph as illustrated in Fig. 6(a).

Learned factor graphs [26] utilize the known factorization to determine the factor graph structure, while using deep learning to compute the function nodes. In particular, the stationarity assumption implies that the complete factor graph is encapsulated in the single function $f(\cdot)$ for any t. A DNN is thus utilized to learn the mapping carried out at the function node separately from the inference task. The SP method is the applied over the resulting learned factor graph, as illustrated in Fig. 6(b). As learning a single function node is often simpler than recovering $\boldsymbol{s}$ from $\boldsymbol{x}$, one may use relatively compact DNNs, which can be learned with small data sets.

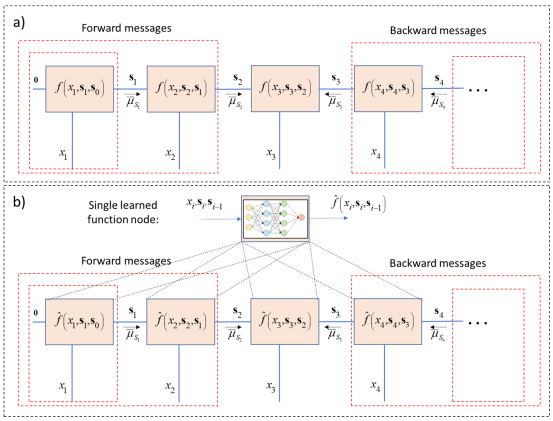

Fig. 6: Illustration of the SP method for Markovian sequences using a) the true factor graph; and b) a learned factor graph.

In order to learn a stationary factor graph from samples, one must only learn its function node, which here boils down to learning $p(x_i|\boldsymbol{s}_{i-l}^{i})$ and $p(s_i|\boldsymbol{s}_{i-l}^{i-1})$, where $p(s_i|\boldsymbol{s}_{i-l}^{i-1})$ can be learned via a histogram. For learning $p(x_i|\boldsymbol{s}_{i-l}^{i})$, a parametric estimate of $p\left(\boldsymbol{s}_i|x_i\right)$, denoted $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{s}_i|x_i)$, is obtained for each $\boldsymbol{s}_i \in \mathcal{S}^{l+1}$ by training a classification DNN. As SP is invariant to scaling $f(\boldsymbol{x}_i, \boldsymbol{s}_i, \boldsymbol{s}_{i-1})$, one can use $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{s}_i|x_i)$ as the function node without affecting the inference mapping.

Discussion: Structure-oriented DNN-aided inference is most suitable for setups in which structured domain knowledge naturally follows from the underlying physics or from established models of the problem. Such structural knowledge is often present in various problems in signal processing and communications, implies that this approach can facilitate inference in such scenarios in a manner which is ignorant of the unique and possibly intractable subtleties of the problem. Furthermore, such systems can be often trained using scarce data sets, facilitating adaptation to temporal model variations.

5. CONCLUSION

We presented a mapping of methods for combining domain knowledge and data-driven inference. We divided such systems into model-aided networks, which utilize model-based algorithms to design DNNs, and DNN-aided inference, where deep learning is integrated into model-based methods. In our review, we detailed design approaches and examples for each strategy, in order to facilitate extensions and applications.

6. REFERENCES

[1] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[2] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, 2019.

[3] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Process. Mag.*, 2020.

[4] S. Wu, A. Dimakis, S. Sanghavi, F. Yu, D. Holtmann-Rice, D. Storcheus, A. Rostamizadeh, and S. Kumar, "Learning a compressed sensing measurement matrix via gradient unrolling," in *International Conference on Machine Learning*, 2019, pp. 6828–6839.

[5] 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.*, 2019.

[6] C. Metzler, A. Mousavi, and R. Baraniuk, "Learned D-AMP: Principled neural network based compressive image recovery," in *Advances in Neural Information Processing Systems*, 2017, pp. 1772–1783.

[7] T. Chang, B. Tolooshams, and D. Ba, "RandNet: deep learning with compressed measurements of images," in *Proc. IEEE MLSP*, 2019.

[8] A. Balatsoukas-Stimming and C. Studer, "Deep unfolding for communications systems: A survey and some new directions," *arXiv preprint arXiv:1906.05774*, 2019.

[9] N. Samuel, T. Diskin, and A. Wiesel, "Learning to detect," *IEEE Trans. Signal Process.*, vol. 67, no. 10, pp. 2554–2564, 2019.

[10] H. He, C.-K. Wen, S. Jin, and G. Y. Li, "Model-driven deep learning for MIMO detection," *IEEE Trans. Signal Process.*, vol. 68, pp. 1702–1715, 2020.

[11] R. J. van Sloun, R. Cohen, and Y. C. Eldar, "Deep learning in ultrasound imaging," *Proc. IEEE*, vol. 108, no. 1, pp. 11–29, 2019.

[12] M. Mischi, M. A. L. Bell, R. J. van Sloun, and Y. C. Eldar, "Deep learning in medical ultrasound—from image formation to image analysis," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 67, no. 12, pp. 2477–2480, 2020.

[13] G. Dardikman-Yoffe and Y. C. Eldar, "Learned SPARCOM: Unfolded deep super-resolution microscopy," *Optics Express*, vol. 28, no. 19, pp. 4797–4812, 2020.

[14] N. Shlezinger, R. Fu, and Y. C. Eldar, "DeepSIC: Deep soft interference cancellation for multiuser MIMO detection," *IEEE Trans. Wireless Commun.*, 2020.

[15] M. Kocaoglu, C. Snyder, A. G. Dimakis, and S. Vishwanath, "CausalGAN: Learning causal implicit generative models with adversarial training," *arXiv preprint arXiv:1709.02023*, 2017.

[16] W.-J. Choi, K.-W. Cheong, and J. M. Cioffi, "Iterative soft interference cancellation for multiple antenna systems." in *Proc. WCNC*, 2000, pp. 304–309.

[17] H. K. Aggarwal, M. P. Mani, and M. Jacob, "MoDL: Model-based deep learning architecture for inverse problems," *IEEE Trans. Med. Imag.*, vol. 38, no. 2, pp. 394–405, 2018.

[18] 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.

[19] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep CNN denoiser prior for image restoration," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3929–3938.

[20] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett, "Deep learning techniques for inverse problems in imaging," *IEEE J. Sel. Areas Inform. Theory*, vol. 1, no. 1, pp. 39–56, 2020.

[21] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, "Compressed sensing using generative models," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 537–546.

[22] D. Van Veen, A. Jalal, M. Soltanolkotabi, E. Price, S. Vishwanath, and A. G. Dimakis, "Compressed sensing with deep image prior and learned regularization," *arXiv preprint arXiv:1806.06438*, 2018.

[23] J. Whang, Q. Lei, and A. G. Dimakis, "Compressed sensing with invertible generative models and dependent noise," *arXiv preprint arXiv:2003.08089*, 2020.

[24] D. Gilton, G. Ongie, and R. Willett, "Neumann networks for inverse problems in imaging," *arXiv preprint arXiv:1901.03707*, 2019.

[25] N. Shlezinger, N. Farsad, Y. C. Eldar, and A. J. Goldsmith, "ViterbiNet: A deep learning based Viterbi algorithm for symbol detection," *IEEE Trans. Wireless Commun.*, vol. 19, no. 5, pp. 3319–3331, 2020.

[26] ——, "Data-driven factor graphs for deep symbol detection," *arXiv preprint arXiv:2002.00758*, 2020.

[27] N. Farsad, N. Shlezinger, A. J. Goldsmith, and Y. C. Eldar, "Data-driven symbol detection via model-based machine learning," *arXiv preprint arXiv:2002.07806*, 2020.

[28] P. Knobelreiter, C. Sormann, A. Shekhovtsov, F. Fraundorfer, and T. Pock, "Belief propagation reloaded: Learning BP-layers for labeling problems," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7900–7909.

[29] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, 2001.