Bilevel Methods for Image Reconstruction

Caroline Crockett
University of Michigan
cecro@umich.com

Jeffrey A. Fessler
University of Michigan
fessler@umich.edu

September 21, 2021

Abstract

This review discusses methods for learning parameters for image reconstruction problems using bilevel formulations. Image reconstruction typically involves optimizing a cost function to recover a vector of unknown variables that agrees with collected measurements and prior assumptions. State-of-the-art image reconstruction methods learn these prior assumptions from training data using various machine learning techniques, such as bilevel methods.

One can view the bilevel problem as formalizing hyperparameter optimization, as bridging machine learning and cost function based optimization methods, or as a method to learn variables best suited to a specific task. More formally, bilevel problems attempt to minimize an upper-level loss function, where variables in the upper-level loss function are themselves minimizers of a lower-level cost function.

This review contains a running example problem of learning tuning parameters and the coefficients for sparsifying filters used in a regularizer. Such filters generalize the popular total variation regularization method, and learned filters are closely related to convolutional neural networks approaches that are rapidly gaining in popularity. Here, the lower-level problem is to reconstruct an image using a regularizer with learned sparsifying filters; the corresponding upper-level optimization problem involves a measure of reconstructed image quality based on training data.

This review discusses multiple perspectives to motivate the use of bilevel methods and to make them more easily accessible to different audiences. We then turn to ways to optimize the bilevel problem, providing pros and cons of the variety of proposed approaches. Finally, we overview bilevel applications in image reconstruction.
# Contents

1 Introduction .......................................................... 4  
1.1 Notation ................................................................. 5  
1.2 Bilevel Method: Set-up and Running Example ......................... 7  
1.3 Conclusion ............................................................... 11  

2 Background: Image Reconstruction .................................... 12  
2.1 Image Reconstruction ................................................... 12  
2.2 Sparsity-Based Regularizers ............................................ 14  
2.3 Brief History of Analysis Regularizer Learning ......................... 17  
2.4 Summary ................................................................. 20  

3 Background: Hyperparameter Optimization .......................... 21  
3.1 Image Quality Metrics .................................................. 22  
3.2 Parameter Search Strategies ............................................ 26  
3.3 Summary ................................................................. 30  

4 Gradient Based Bilevel Methodology: The Groundwork ............ 31  
4.1 Set-up ................................................................. 32  
4.2 Minimizer Approach .................................................... 32  
4.3 Translation to a Single Level ........................................... 37  
4.4 Unrolled Approaches ................................................... 39  
4.5 Summary ................................................................. 43  

5 Gradient-Based Bilevel Optimization Methods ....................... 46  
5.1 Double-Loop Algorithms ............................................... 47  
5.2 Single-Loop Algorithms ............................................... 56  
5.3 Summary of Methods .................................................. 60  

6 Survey of Applications .................................................. 62  
6.1 Lower-level Cost Function Design .................................... 62  
6.2 Upper-Level Loss Function Design ................................... 65  
6.3 Conclusion ............................................................... 68  

7 Connections and Future Directions ..................................... 69  
7.1 Connection: Learnable Optimization Algorithms ..................... 70  
7.2 Connection: Plug-and-play Priors .................................... 72  
7.3 Connection: Single-Level Parameter Learning ........................ 73
| Section                                                                 | Page |
|------------------------------------------------------------------------|------|
| 7.4 Future Directions                                                  | 76   |
| 7.5 Summary of Advantages and Disadvantages                           | 77   |
| Acknowledgements                                                       | 80   |
| A Additional Running Example Results                                  | 80   |
| A.1 Derivatives for Convolutional Filters                              | 80   |
| A.2 Evaluating Conditions for the Running Example                     | 82   |
| References                                                             | 87   |
1 Introduction

Methods for image recovery aim to estimate a good-quality image from noisy, incomplete, or indirect measurements. Such methods are also known as computational imaging. For example, image denoising and image deconvolution attempt to recover a clean image from a noisy and/or blurry input image, and image inpainting tries to complete missing measurements from an image. Medical image reconstruction aims to recover images that humans can interpret from the indirect measurements recorded by a system like an MRI or CT scanner. Such image reconstruction applications are a type of inverse problem [1].

New methods for image reconstruction attempt to lower complexity, decrease data requirements, or improve image quality for a given input data quality. For example, in CT one goal is to provide doctors with information to help their patients while reducing radiation exposure [2]. To achieve these lower radiation doses, the CT system must collect data with lower beam intensity or fewer views. Similarly, in MRI collecting fewer k-space samples can reduce scan times. Such “undersampling” leads to an under-determined problem, with fewer knowns (measurements from a scanner) than unknowns (pixels in the reconstructed image), requiring advanced image reconstruction methods.

Existing reconstruction methods make different assumptions about the characteristics of the images being recovered. Historically, the assumptions are based on easily observed (or assumed) characteristics of the desired output image, such as a tendency to have smooth regions with few edges or to have some form of sparsity [3]. More recent machine learning approaches use training data to discover image characteristics. These learning-based methods often outperform traditional methods, and are gaining popularity in part because of increased availability of training data and computational resources [4], [5].

There are many design decisions in learning-based reconstruction methods. How many parameters should be learned? What makes a set of parameters “good?” How can one learn these good parameters? Using a bilevel methodology is one systematic way to address these questions.

Bilevel methods are so named because they involve two “levels” of optimization: an upper-level loss function that defines a goal or measure of goodness (equivalently, badness) for the learnable parameters and a lower-level cost function that uses the learnable parameters, typically as part of a regularizer. The main benefits of bilevel methods are learning task-based hyperparameters in a principled approach and connecting machine learning techniques with image reconstruction methods that are defined in terms of optimizing a cost function, often called model-based image reconstruction methods. Conversely, the main challenge with bilevel methods is the computational complexity. However, like with neural networks, that complexity is highest during the training process, whereas deployment has lower complexity because it requires only the lower-level problem.

The methods in this review are broadly applicable to bilevel problems, but we focus
on formulations and applications where the lower-level problem is an image reconstruction cost function that uses regularization based on analysis sparsity. The application of bilevel methods to image reconstruction problems is relatively new [5], but there are a growing number of promising research efforts in this direction. We hope this review serves as a primer and unifying treatment for readers who may already be familiar with image reconstruction problems and traditional regularization approaches but who have not yet delved into bilevel methods.

This review lies at the intersection of a specific machine learning method, bilevel, and a specific application, filter learning for image reconstruction. For a broad overview of machine learning in image reconstruction, see [5]. For an overview of image reconstruction methods, including classical, variational, and learning-based methods, see [6]. Finally, for a historical overview of bilevel optimization and perspective on its use in a wide variety of fields, see [7]. Within the image recovery field, bilevel methods have also been used, e.g., in learning synthesis dictionaries [8], [9].

The structure of this review is as follows. The remainder of the introduction defines our notation and presents a running example bilevel problem. Section 2 provides background information on the lower-level image reconstruction cost function and analysis regularizers. Section 3 provides background information on the upper-level loss function, specifically loss function design and hyperparameter optimization strategies. These background sections are not meant to be exhaustive overviews of these broad topics. Section 4 presents building blocks for optimizing a bilevel problem. Section 5 uses these building blocks to discuss optimization methods for the upper-level loss function. Section 6 discusses previous applications of the bilevel method in image recovery problems, including signal denoising, image inpainting, and medical image reconstruction. It also overviews bilevel formulations for blind learning and learning space-varying tuning parameters. Finally, Section 7 offers summarizing commentary on the benefits and drawbacks of bilevel methods for computational imaging and our thoughts for future directions for the field.

1.1 Notation

This review considers continuous-valued, discrete space signals. Some papers, e.g., [10], [11], analyze signals in function space, arguing that the goal of high resolution imagery is to approximate a continuous space reality and that analysis in continuous domain can yield insights and optimization algorithms that are resolution independent. However, the majority of bilevel methods are motivated and described in discrete space. The review does not include discrete-valued settings, such as image segmentation [12]; those problems often require different techniques to optimize the lower-level cost function.

The literature is inconsistent in how it refers to variables in machine learning problems.
For consistency within this document, we define the following terms:

- **Hyperparameters**: Any adjustable parameters that are part of a model. Tuning parameters and model parameters are both sub-types of hyperparameters. This document uses $\gamma$ to denote a vector of hyperparameters.

- **Tuning parameters**: Scalar parameters that weight terms in a cost function to determine the relative importance of each term. This review uses $\beta$ to denote individual tuning parameters.

- **Model parameters**: Parameters, generally in vector or matrix form, that are used in the structure of an cost or loss function, typically as part of the regularization term. In the running example in the next section, the model parameters are typically filter coefficients, denoted $c$.

We write vectors as column vectors and use bold to denote matrices (uppercase letters) and vectors (lowercase letters). Subscripts index vector elements, so $x_i$ is the $i$th element in $x$. For functions that are applied element-wise to vectors, we use notation following the Julia programming language [13], where $f(x)$ denotes the function $f$ applied element wise to its argument:

$$x \in \mathbb{F}^N \implies f(x) = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix} \in \mathbb{F}^N.$$  

The field $\mathbb{F}$ can be either $\mathbb{R}$ or $\mathbb{C}$, depending on the application.

Convolution between a vector, $x$, and a filter, $c$, is denoted as $c \odot x$. This review assumes all convolutions use circular boundary conditions. Thus, the same convolution is equivalent to multiplication with a square, circulant matrix:

$$c \odot x = Cx.$$

The conjugate mirror reversal of $c$ is denoted as $\tilde{c}$ and its application is equivalent to multiplying with the adjoint of $C$:

$$\tilde{c} \odot x = C'x.$$

Finally, for partial derivatives, we use the notation that

$$\nabla_x f(x, y) = \frac{\partial f(x, y)}{\partial x} \in \mathbb{F}^N,$$

$$\nabla_{xy} f(x, y) = \begin{bmatrix} \frac{\partial^2 f(x, y)}{\partial x_i \partial y_j} \\ \vdots \\ \frac{\partial^2 f(x, y)}{\partial x_N \partial y_M} \end{bmatrix} \in \mathbb{F}^{N \times M}, \quad \text{(1.1)}$$

$$\nabla_{xy} f(\hat{x}, y) = \nabla_{xy} f(x, y) \bigg|_{x=\hat{x}}.$$

Tables 1.1 and 1.2 summarize our frequently used notation for variables and functions.
1.2 Bilevel Method: Set-up and Running Example

This section introduces a generic bilevel problem as well as a specific bilevel problem that serves as a running example throughout the review. Later sections discuss many of the ideas presented here more thoroughly. Our hope is that an early introduction to the formal problem motivates readers and that this section acts as a quick-reference guide to our notation.

This review considers the image reconstruction problem where the goal is to form an estimate $\hat{x} \in \mathbb{F}^N$ of a (vectorized) latent image, given a set of measurements $y \in \mathbb{F}^M$. For denoising and inpainting problems, $N = M$, but the two dimensions may differ significantly.

| Variable | Dim | Description |
|----------|-----|-------------|
| $x_j^{\text{true}}$ | $N$ | One of $J$ clean, noiseless training signals. Often used in a supervised training set-up. |
| $A$ | $M \times N$ | Forward operator for the system of interest. |
| $y_j$ | $M$ | During the bilevel learning process, this refers to simulated measurements, where $y_j = Ax_j^{\text{true}} + n_j$. Once $\gamma$ is learned, $y$ refers to collected measurements. |
| $n$ | $N$ | A noise realization. |
| $\hat{x}$ | $N$ | A reconstructed image. |
| $\gamma$ | $R$ | The vector of parameters to learn using bilevel methods. This often includes $c_k$ and/or $\beta_k$. |
| $c_k$ | $S$ | One of $K$ convolutional filters. A 2D filter might be $\sqrt{S} \times \sqrt{S}$. |
| $\tilde{c}_k$ | $S$ | Conjugate mirror reversal of filter $c_k$. |
| $C_k$ | $N \times N$ | The convolution matrix such that $C_k x = c_k \ast x$ and $C_k' x = \tilde{c}_k \ast x$. |
| $\beta_k$ | $\mathbb{R}$ | The tuning parameter associated with $c_k$. |
| $\beta_0$ | $\mathbb{R}$ | An overall regularization (tuning) parameter, appearing as $e^{\beta_0}$ the running example. |
| $z$ | Varies | A sparse vector, often from $C_k x$. |
| $\epsilon$ | $\mathbb{R}_+$ | Parameter used to define $\phi$. Typically determines the amount of corner-rounding. |
| $t$ | $0, \ldots, T$ | Iteration counter for lower-level optimization iterates, e.g., $x^{(t)}$ is the estimate of the lower-level optimization variable $x$ at the $t$th iteration. |
| $u$ | $0, 1, \ldots$ | Iteration counter for the upper-level optimization iterates, e.g., $\gamma^{(u)}$. |

Table 1.1: Overview of frequently used symbols in the review.
in more general image reconstruction problems. The forward operator, $A \in \mathbb{R}^{M \times N}$ models the physics of the system such that one would expect $y = Ax$ in an ideal (noiseless) system. We focus on linear imaging systems here, but the concepts generalize readily to nonlinear forward models. When known (in a supervised training setting), we denote the true, underlying signal as $x^{\text{true}} \in \mathbb{R}^N$.

We focus on model-based image reconstruction methods where the goal is to estimate $x$ from $y$ by solving an optimization problem of the form

$$\hat{x} = \hat{x}(\gamma) = \arg\min_{x \in \mathbb{R}^N} \Phi(x; \gamma, y).$$

(1.2)

To simplify notation, we drop $y$ from the list of $\Phi$ arguments except where needed for clarity. The quality of the estimate $\hat{x}$ can depend greatly on the choice of the hyperparameters $\gamma$. Historically there have been numerous approaches pursued for choosing $\gamma$, such as cross validation [14], generalized cross validation [15], the discrepancy principle [16] and Bayesian methods [17], among others.

Bilevel methods provide a framework for choosing hyperparameters. Without further ado, a bilevel problem for learning hyperparameters $\gamma$ has the following “double minimization” form:

$$\gamma = \gamma^* = \arg\min_{\gamma \in \mathbb{R}^F} \ell(\gamma; \hat{x}(\gamma))$$ where $\hat{x}(\gamma) = \arg\min_{x \in \mathbb{R}^N} \ell(x; \gamma)$.

(UL)

$$\hat{x}(\gamma) = \arg\min_{x \in \mathbb{R}^N} \Phi(x; \gamma).$$

(LL)

Fig. 1.1 depicts a generic bilevel problem for image reconstruction. The upper-level (UL), or outer, loss function, $\ell : \mathbb{R}^R \times \mathbb{R}^N \rightarrow \mathbb{R}$, quantifies how (not) good is a vector $\gamma$ of learnable parameters; it depends on the solution to the lower-level (LL) cost function, $\Phi$, which

| Function | Description |
|----------|-------------|
| $\ell(\gamma) \mapsto \mathbb{R}$ or $\ell(\gamma, x) \mapsto \mathbb{R}$ | Upper-level loss function used as a fitness measure of $\gamma$. Although $\ell$ is a function of $\gamma$, it is often helpful to write it with two inputs, where typically $x = \hat{x}$. |
| $\Phi(x; \gamma) \mapsto \mathbb{R}$ | Lower-level cost function used to reconstruct an image. |
| $R(x) \mapsto \mathbb{R}$ | Regularization function. Incorporates prior information about likely image characteristics. |
| $d(x, y) \mapsto \mathbb{R}$ | Data-fit term. |
| $\phi(z) \mapsto \mathbb{R}$ | Sparsity promoting function, e.g., 0-norm, 1-norm, or corner-rounded 1-norm. Typically used in $R$. |

Table 1.2: Overview of frequently used functions in the review.
depends on $\gamma$. We will also write the upper-level loss function with a single parameter as $\ell(\gamma)$. We write the lower-level cost as an optimization problem with “argmin” and thus implicitly assume that $\Phi$ has unique minimizer with respect to $x$. (See Section 4 for more discussion of this point).

Figure 1.1: Depiction of a typical bilevel problem for image reconstruction, illustrated using XCAT phantom from [18]. The upper box represents the training process, with the upper-level loss and lower-level cost function. During training, one minimizes the upper-level loss with respect to a vector of parameters, $\gamma$, that are used in the image reconstruction task. Once learned, $\hat{\gamma}$ is typically deployed in the same image reconstruction task, shown in the lower box.

Bilevel methods typically use training data. Specifically, one often assumes that a given a set of $J$ good quality images $x_{1}^{\text{true}}, \ldots, x_{J}^{\text{true}} \in \mathbb{F}^{N}$ are representative of the images of interest in a given application. (For simplicity of notation we assume the training images have the same size, but they can have different sizes in practice.) We generate corresponding simulated measurements for each training image using the imaging system model:

$$y_{j} = A x_{j}^{\text{true}} + n_{j}, \quad j = 1, \ldots, J,$$

where $n_{j} \in \mathbb{F}^{M}$ denotes an appropriate random noise realization. In (1.3), we add one noise realization to each of the $J$ images; in practice one could add multiple noise realizations to each $x_{j}^{\text{true}}$ to augment the training data. We then use the training pairs $(x_{j}^{\text{true}}, y_{j})$ to learn a good value of $\gamma$. After those parameters are learned, subsequent test images are reconstructed using (1.2) with the learned hyperparameters, $\hat{\gamma}$.

\footnote{A more general system model allows the noise to depend on the data and system model, \textit{i.e.}, $n_{j}(A, x_{j})$. This generality is needed for applications with certain noise distributions such as Poisson noise.}
An alternative to (UL) is the following stochastic formulation of bilevel learning:

\[
\hat{\gamma} = \arg\min_{\gamma \in \mathbb{F}} \mathbb{E}[\ell(\gamma)] \quad \text{where} \quad \mathbb{E}[\ell(\gamma)] \approx \frac{1}{J} \sum_{j=1}^{J} \ell(\gamma; \hat{x}_j(\gamma)).
\]

The expectation, taken with respect to the training data and noise distributions, is typically approximated as a sum over the training data.

To offer a concrete example, this review will frequently refer to the following running example (Ex), a filter learning bilevel problem:

\[
\hat{\gamma} = \arg\min_{\gamma \in \mathbb{F}} \frac{1}{2} \|\hat{x}(\gamma) - x^{true}\|_2^2,
\]

where

\[
\hat{x}(\gamma) = \arg\min_{x \in \mathbb{F}} \frac{1}{2} \|Ax - y\|_2^2 + e^{\beta_0} \sum_{k=1}^{K} e^{\beta_k} 1^{\phi}(c_k \otimes x; \epsilon),
\]

(Ex)

where \(\gamma \in \mathbb{F}^R\) contains all variables that we wish to learn: the filter coefficients \(c_k \in \mathbb{F}^S\) and tuning parameters \(\beta_k \in \mathbb{R}\) for all \(k \in [1, K]\). We include an auxiliary tuning parameter, \(\beta_0 \in \mathbb{R}\), for easier comparison to other models. Fig. 1.2 depicts the running example. In (Ex), we drop the sum over \(J\) training images for simplicity; the methods easily extend to multiple training signals. In general, the filters may be of different lengths with minimal impact on the methods presented in this review. However, for ease of notation, we will consider \(c_k\) to be of length \(S\) for all \(k\), e.g., a 2D filter might be \(\sqrt{S} \times \sqrt{S}\).

The function \(\phi\) in (Ex) is a sparsity-promoting function. If we were to choose \(\phi(z) = |z|\), then the regularizer would involve 1-norm terms of the type common in compressed sensing formulations:

\[
1^{\phi}(c_k \otimes x) = \|c_k \otimes x\|_1.
\]

However, to satisfy differentiability assumptions (see Section 4), this review will often consider \(\phi\) to denote the following “corner rounded” 1-norm having the shape of a hyperbola with the corresponding first and second derivative:

\[
\phi(z) = \begin{cases} \sqrt{z^2 + \epsilon^2} & \text{for the first} \\ \frac{z}{\sqrt{z^2 + \epsilon^2}} & \in [0, 1] \\ \frac{\epsilon^2}{(z^2 + \epsilon^2)^{3/2}} & \in (0, \frac{1}{\epsilon}], \end{cases}
\]

(CR1N)

where \(\epsilon\) is a small, relative to the expected range of \(z\), parameter that controls the amount of corner rounding. (Here, we use a dot over the function rather than \(\nabla\) to indicate a derivative because \(\phi\) is a scalar function.)
Figure 1.2: Example bilevel problem in (Ex). The vector of learnable hyperparameters, $\gamma$, includes the tuning parameters, $\beta_k$, and the filter coefficients, $c_k$, shown as example filters. Although this review will generally consider learning filters of a single size, the figure depicts how the framework easily extends to 2d filters of different sizes.

1.3 Conclusion

Bilevel methods for selecting hyperparameters offer many benefits. Previous papers motivate them as a principled way to approach hyperparameter optimization [7], [19], as a task-based approach to learning [11], [20], [21], and/or as a way to combine the data-driven improvements from learning methods with the theoretical guarantees and explainability provided by cost function-based approaches [10], [22], [23]. A corresponding drawback of bilevel methods are their computational cost; see Sections 4 and 5 for further discussion.

The task-based nature of bilevel methods is a particularly important advantage; Section 7.3 exemplifies why by comparing the bilevel problem to single-level, non-task-based approaches for learning sparsifying filters. Task-based refers to the hyperparameters being learned based on how well they work in the lower-level cost function—the image reconstruction task in our running example. The learned hyperparameters also adapt to the training dataset and noise characteristics. The task-based nature yields other benefits, such as making constraints or regularizers on the hyperparameters generally unnecessary; Section 6.2 presents some exceptions and [7] discusses further bilevel methods for applications with constraints.

There are three main elements to a bilevel approach. First, the lower-level cost function in a bilevel problem defines a goal, such as image reconstruction, including what hyperparameters can be learned, such as filters for a sparsifying regularizer. Section 2 provides background on this element specifically for image reconstruction tasks, such as the one in (Ex). Section 6.1 reviews example tasks used in bilevel methods.
Second, the upper-level loss function determines how the hyperparameters should be learned. While the mean square error (MSE) loss function in the running example is a common choice, Section 3 discusses other loss functions based on supervised and unsupervised image quality metrics. Section 6.2 then reviews example loss functions used in bilevel methods.

While not apparent in the written optimization problem, the third main element for a bilevel problem is the optimization approach. Section 3.2 briefly discusses various hyperparameter optimization strategies, then Sections 4 and 5 present multiple gradient-based bilevel optimization strategies. Throughout the review, we refer to the running example to show how the bilevel optimization strategies apply.

2 Background: Image Reconstruction

This review focuses on bilevel problems having image reconstruction as the lower-level problem. Image reconstruction involves undoing any transformations inherent in an imaging system, e.g., a camera or CT scanner, and removing measurement noise, e.g., thermal and shot noise, to realize an image that captures an underlying object of interest, e.g., a patient’s anatomy. Fig. 2.1 shows an example image reconstruction pipeline for CT data. The following sections formally define image reconstruction, discuss why regularization is important, and overview common approaches to regularization.

2.1 Image Reconstruction

Although the true object is in continuous space, image reconstruction is almost always performed on sampled, discretized signals [24]. Without going into detail of the discretization process, we define $x^{\text{true}} \in \mathbb{F}^N$ as the “true,” discrete signal. The goal of image reconstruction

![Figure 2.1: Example image reconstruction pipe-line, illustrated using XCAT phantom from [18]. Here $A$ denotes the actual physical mapping of the imaging system and $A$ denotes the numerical system matrix used for reconstruction.](image-url)

12
is to recover an estimate $\hat{x} \approx x^{\text{true}}$ given corrupted measurements $y \in \mathbb{F}^N$. Although we define the signal as a one-dimensional vector for notational convenience, the mathematics generalize to arbitrary dimensions.

To find $\hat{x}$, image reconstruction involves minimizing a cost function, $\Phi(x; \gamma)$, with two terms:

$$\hat{x} = \text{argmin}_{x \in \mathbb{F}^N} \underbrace{d(x; y)}_{\text{Data-fit}} + \beta \underbrace{R(x; \gamma)}_{\text{Regularizer}}$$

(2.1)

The first term, $d(x; y)$, is a data-fit term that captures the physics of the ideal (noiseless) system using the matrix $A \in \mathbb{F}^{M \times N}$; that matrix models the physical system such that we expect an observation, $y$, to be $y \approx Ax$. The most common data-fit term penalizes the square Euclidean norm of the “measurement error,” $d(x; y) = \|A \hat{x} - y\|_2^2$. This intuitive data-fit term can be derived from a maximum likelihood perspective, assuming a white Gaussian noise distribution [25].

Many image reconstruction problems have linear system models. In image denoising problems, one takes $A = I$. For image inpainting, $A$ is a diagonal matrix of 1’s and 0’s, where the 0’s correspond to sample indices of missing data [26]. In MRI, the system matrix is often approximated as a diagonal matrix times a discrete Fourier transform matrix, though more accurate models are often needed [27]. In some settings one can learn $A$ [28], or at least parts of $A$ [29], as part of the estimation process. Although the bilevel method generalizes to learning $A$, the majority of papers in the field assume $A$ is known; Section 6 discusses a few exceptions.

Using the system model (1.3), if $n$ were known and $A$ were invertible, we could simply compute $\hat{x} = x^{\text{true}} = A^{-1}(y - n)$. However, $n$ is random and, while we may be able to model its characteristics, we never know it exactly. Further, the system matrix, $A$, is often not invertible because the reconstruction problem is frequently under-determined, with fewer knowns than unknowns ($M < N$). Therefore, we must include prior assumptions about $x^{\text{true}}$ to make the problem feasible. These assumptions about $x^{\text{true}}$ are captured in the second, regularization term in (2.1), which depends on $\gamma$. The following section further discusses regularizers.

In sum, image reconstruction involves finding $\hat{x}$ that matches the collected data and satisfies a set of prior assumptions. The data-fit term encourages $\hat{x}$ to be a good match for the data; without this term, there would be no need to collect data. The regularization term encourages $\hat{x}$ to match the prior assumptions. Finally, the tuning parameter, $\beta$, controls the relative importance of the two terms. The cost function can be minimized using different optimization techniques depending on the form of each term.

This section is a very short overview of image reconstruction methods. See [6] for a more thorough review of biomedical image reconstruction.
2.2 Sparsity-Based Regularizers

The regularization, or prior assumption, term in (2.1) often involves assumptions about sparsity \[3\]. The basic idea behind sparsity-based regularization is that the true signal is sparse in some representation, while the noise or corruption is not. Thus, one can use the representation to separate the noise and signal, and then keep only the sparse signal component.

The design problem therefore becomes determining what representation best sparsifies the signal. There are two main types of sparsity-based regularizers corresponding to two representational assumptions: synthesis and analysis \[25\], \[30\]; Fig. 2.2 depicts both. While both are popular, this review concentrates on analysis regularizers, which are more widely represented in the bilevel image reconstruction literature. This section briefly compares the analysis and synthesis formulations. Here we simplify the formulas by considering \[A = I\]; the discussion generalizes to reconstruction by including \[A\]. For more thorough discussions of analysis and synthesis regularizers, see \[25\], \[30\], \[31\].

\[
\text{Synthesis: } \mathbf{x} = D \mathbf{z}
\]

\[
\text{Analysis: } \Omega \mathbf{x} = \mathbf{z}
\]

**Figure 2.2**: Depiction of synthesis and analysis sparsity. Under the synthesis model of sparsity (left), \(\mathbf{x}\) is a linear combination of a few dictionary atoms. The dictionary, \(D\), is typically wide, with more atoms (columns) than elements in \(\mathbf{x}\). Under the analysis model of sparsity (right), \(\mathbf{x}\) is orthogonal to many filters. The filter matrix, \(\Omega\), is typically tall, with more filters (rows) than elements in \(\mathbf{x}\).

2.2.1 Synthesis Regularizers

Synthesis regularizers model a signal being composed of building blocks, or “atoms.” Small subsets of the atoms span a low dimensional subspace and the sparsity assumption is that the signal requires using only a few of the atoms. More formally, the synthesis model is \(\mathbf{y} = \mathbf{x} + \mathbf{n}\), where the signal \(\mathbf{x} = D\mathbf{z}\) and \(\mathbf{z}\) is a sparse vector. The columns of \(D \in \mathbb{R}^{N \times K}\) contain the \(K\) dictionary atoms and form a low dimensional subspace for the signal. If \(D\) is a wide matrix \((N < K)\), the dictionary is over-complete and it is easier to represent a wide range of signals with a given number of dictionary atoms. The dictionary is complete when \(D\) is square (and full rank) and under-complete if \(D\) is tall (an uncommon choice).
Assuming one knows or has already learned $D$, one can use the sparsity synthesis assumption to denoise a noisy signal $y$ by optimizing

$$\hat{x} = D \cdot \left( \arg\min_z \frac{1}{2} \| y - Dz \|^2 + \phi(z) \right).$$

(2.2)

The estimation procedure involves finding the sparse codes, $\hat{z}$, from which the image is synthesized via $\hat{x} = D\hat{z}$. Common sparsity-inducing functions, $\phi$, are the 0-norm and 1-norm. The 2-norm is occasionally used as $\phi$, but it does not yield true sparse codes and it over-penalizes large values [32].

As written in (2.2), the synthesis formulation constrains the signal, $x$, to be in the range of $D$. This “strict synthesis” model can be undesirable in some applications, e.g., when one is not confident in the quality of the dictionary. An alternative formulation is

$$\hat{x} = \arg\min_x \frac{1}{2} \| y - x \|^2 + \beta R(x),$$

$$R(x) = \min_z \frac{1}{2} \| x - Dz \|^2 + \phi(z),$$

(2.3)

which no longer constrains $x$ to be exactly in the range of $D$. One can also learn $D$ while solving (2.3) [33].

Both synthesis denoising forms have equivalent sparsity constrained versions; one can replace $\phi$ a characteristic function that is 0 within some desired set and infinite outside it, e.g.,

$$\phi(z) = \begin{cases} 0 & \text{if } \|z\|_0 \leq \alpha \\ \infty & \text{else}, \end{cases}$$

(2.4)

for some sparsity constraint given by the hyperparameter $\alpha \in \mathbb{N}$.

See [32], [34] for discussions of when the synthesis model can guarantee accurate recovery of signals. The minimization problem in (2.3) is called sparse coding and is closely related to the LASSO problem [35]. One can think of the entire dictionary $D$ as a hyperparameter that can be learned with a bilevel method [36].

2.2.2 Analysis Regularizers

Analysis regularizers model a signal as being sparsified when mapped into another vector space by a linear transformation, often represented by a set of filters. More formally, an analysis model assumes the signal satisfies $\Omega x = z$ for a sparse coefficient vector $z$. Often the rows of the matrix $\Omega \in \mathbb{F}^{K \times N}$ are thought of as filters and the rows of $\Omega$ where $[\Omega x]_k = 0$ span a subspace to which $x$ is orthogonal. The analysis operator is called over-complete if $\Omega$ is tall ($N < K$), complete if $\Omega$ is square (and full rank), and under-complete if $\Omega$ is wide.
A particularly common analysis regularizer is based on a discretized version of total variation (TV) [37], and uses finite difference filters (or, more generally, filters that approximate higher-order derivatives). The finite difference filters sparsify any piece-wise constant (flat) regions in the signal, leaving the edges that are often approximately sparse in natural images. Other common analysis regularizers include the discrete Fourier transform (DFT), curvelets, and wavelet transforms [38].

The literature is less consistent in analysis regularizer vocabulary, and \( \Omega \) has been called an analysis dictionary, an analysis operator, a filter matrix, and a cosparse operator. The term “cosparse” comes from the sparsity holding in the codomain of the transformation \( T\{x\} = \Omega x \). The cosparseity of \( x \) with respect to \( \Omega \) is the number of zeros in \( \Omega x \) or \( K - \|\Omega x\|_0 \) [31]. Correspondingly, “cosupport” describes the indices of the rows where \( \Omega x = 0 \). We find the phrase “analysis operator” intuitive for general \( \Omega \)’s and “filter matrix” more descriptive when referring to the specific (common) case when the rows of \( \Omega \) are dictated by a set of convolutional filters.

Assuming one knows, or has already learned, \( \Omega \), one can use the analysis sparsity assumption to denoise a noisy signal, \( y \), by optimizing

\[
\hat{x} = \arg\min_x \frac{1}{2} \|y - x\|^2 + \beta \phi(\Omega x).
\]

(2.5)

An alternative version is

\[
\hat{x} = \arg\min_x \frac{1}{2} \|y - x\|^2 + \beta R(x)
\]

(2.6)

\[
R(x) = \min_z \frac{1}{2} \|\Omega x - z\|^2 + \phi(z).
\]

As in the synthesis case, both analysis formulations have equivalent sparsity constrained forms using a \( \phi \) as in (2.4).

See [38] for an error bound on the estimated signal \( \hat{x} \) when using a 1-norm as the sparsity function.

2.2.3 Comparing Analysis and Synthesis Approaches

The analysis and synthesis models are equivalent when the dictionary and analysis operator are invertible, with \( D = \Omega^{-1} \) [25]. Furthermore, in the denoising scenario where the system matrix \( A \) is identity, the two are almost equivalent in the under-complete case [25], with the lack of full equivalence stemming from the analysis form not constraining \( x \) to be in the range space \( D \). However, the two models diverge in the more common over-complete case.

Whether analysis-based or synthesis-based regularizers are preferable is an open question, and the answer likely depends on the application and the relative importance of reconstruction accuracy and speed [25]. Synthesis regularization is perhaps easier to interpret because of its constructive nature, and the synthesis approach used to be “widely considered
to provide superior results” [25, p. 950]. However, [25] goes on to show that an analysis regularizer produced more accurate reconstructed images in experiments on real images. Later analysis-based results also show competitive, if not superior, quality results when compared to similar synthesis models [39], [40]. See [41] for a survey of optimization methods for MRI reconstruction and a comparison of the computational challenges for cost functions with synthesis and analysis-based regularizers.

The analysis and synthesis regularizers in (2.2) and (2.6) quickly yield infeasibly large operators as the signal size increases. In practice, both approaches are usually implemented with patch-based formulations. For the synthesis approach, the patches typically overlap and there is an averaging effect. Analysis regularizers that have rows corresponding to filters, called the convolutional analysis model, extend very naturally to a global image regularizer. For example, in the lower-level cost function of our running filter learning example (Ex), we can define an analysis regularizer matrix as follows:

\[
\Omega = \begin{bmatrix} C_1 \\ \vdots \\ C_K \end{bmatrix} \in \mathbb{F}^{K \times N}. 
\]

See [42] and [43] for discussion of the connections between global models and patch-based models for analysis regularizers. This survey focuses on bilevel learning of analysis regularizers.

### 2.3 Brief History of Analysis Regularizer Learning

In 2003, Haber and Tenorio [21] proposed using bilevel methods to learn part of the regularizer in inverse problems. The authors motivate the use of bilevel methods through the task-based nature, noting that “the choice of good regularization operators strongly depends on the forward problem.” They consider learning tuning parameters, space-varying weights, and regularization operators (comparable to defining \( \phi \)), all for regularizers based on penalizing the energy in the derivatives of the reconstructed image. Their framework is general enough to handle learning filters. Ref. [21] was published a few years earlier than the other bilevel methods we consider in this review and was not cited in most other early works; [44] calls it a “groundbreaking, but often overlooked publication.”

In 2005, Roth and Black [45] proposed the Field of Experts (FoE) model to learn filters. Many papers on bilevel methods for filter learning cite FoE as a starting or comparison point. The FoE model is a translation-invariant analysis operator model, built on convolutional filters. It is motivated by the local operators and presented as a Markov random field model, with the order of the field determined by the filter size.
Under the FoE model, the negative log\(^1\) of the probability of a full image, \(x\), is proportional to
\[
\sum_k \beta_k \phi(c_k \odot x) \text{ where } \phi(z) = \log \left( 1 + \frac{1}{2} z^2 \right).
\] (2.7)

This (non-convex) choice of sparsity function \(\phi\) stems from the Student-t distribution. Ref. [45] learns the filters and filter-dependent tuning parameters such that the model distribution is as close as possible (defined using Kullback-Leibler divergence) to the training data distribution.

In 2007, Tappen, Liu, Adelson, et al. [46] proposed a different model based on convolutional filters: the Gaussian Conditional Random Field (GCRF) model. Rather than using a sparsity promoting regularizer, the GCRF uses a quadratic function for \(\phi\). The authors introduce space-varying weights, \(W\), so that the quadratic model does not overly penalize sharp features in the image. The general idea behind \(W\) is to use the given (noisy) image to guess where edges occur, and correspondingly penalize those areas less to avoid blurring edges. The likelihood for GCRF model is thus (to within a proportionality constant and monotonic function transformations):
\[
\sum_k \|c_k \odot x - e_k \{x\}\|_W^2,
\]
where the term \(e_k \{x\}\) captures the estimated value of the filtered image. For example, [46] used one averaging filter and multiple differencing filters for the \(c_k\)’s. The corresponding estimated values are \(x\) for the averaging filter and zero for the differencing filters.

The filters, \(c_k\), are pre-determined in the GCRF model; the learned element is how to form the weights as a function of image features. Specifically, each \(W_k\) is formed as a linear combination of the (absolute) responses to a set of edge-detecting filters, with the linear combination coefficients learned from training data. Rather than maximizing the likelihood of training data as in [45], [46] learns these coefficients to minimize the (corner-rounded) \(l_1\) norm of the error of the predicted image, which is a form of bilevel learning even though not described with that terminology.

Apparently one of the first papers to explicitly propose using bilevel methods to learn filters appeared in 2009, where Samuel and Tappen [47] considered a bilevel formulation where the upper-level loss was mean square error on training data and the lower-level cost was a denoising task based on filter sparsity equivalent to (Ex). The method builds on the FoE model, using the same \(\phi\) as in [45], but now learns the filters using a bilevel formulation rather than by maximizing a likelihood.

In 2011, Peyré and Fadili [20] proposed a similar bilevel method to learn analysis regularizers. The authors generalized the denoising task to use an analysis operator matrix

---

\(^1\)By taking the log of the probability model in [45], the connection between the FoE and the regularization term in the lower-level of the running filter learning example (Ex) is more evident.
and a wider class of sparsifying functions. Their results concentrate on the convolutional filter case with a corner-rounded 1-norm for $\phi$.

Both [47] and [20] focus on introducing the bilevel method for analysis regularizer learning, with denoising or inpainting as illustrations. Section 4 further discusses the methodology of both papers. Many of the bilevel based papers in this review build on one or both of their efforts. The rest of the review will summarize other bilevel based papers; here, we continue to highlight some of papers in the non-bilevel thread of the literature for context and comparison.

Ophir, Elad, Bertin, et al. [48] proposed another approach to learning an analysis operator. The method learns the operator one row at a time by searching for vectors orthogonal to the training signals. Algorithm parameters were chosen empirically without an upper-level loss function as a guide.

Between 2011 [49] and 2013 [50], Yaghoobi, Nam, Gribonval, and Davies were among the first to formally present analysis operator learning as an optimization problem. Their conference paper [49] considered noiseless training data and proposed learning an analysis operator as

$$\arg\min_{\Omega} \| \Omega X^\text{true} \|_1 \text{ s.t. } \Omega \in \mathcal{S}$$

for some constrained set $\mathcal{S}$. Each column of $X^\text{true}$ contains a training sample. The authors discussed varying options for $\mathcal{S}$, including a row norm, full rank, and tight frame constrained set.

Without any constraint on $\Omega$, the trivial solution to (2.8) would be to learn the zero matrix, which is not informative for any problem such as image denoising. Section 7.3 discusses in more detail the need for constraints and the various options proposed for filter learning.

Ref. [50] extends (2.8) to the noisy case where one does not have access to $X^\text{true}$. The proposed cost function is

$$\arg\min_{\Omega, X} \| \Omega X \|_1 + \frac{\beta}{2} \| Y - X \|_2^2 \text{ s.t. } \Omega \in \mathcal{S},$$

where each column of $Y$ contains a noisy data vector. Ref. [50] minimized (2.9) by alternating updating $X$, using alternating direction method of multipliers (ADMM), and $\Omega$, using a projected subgradient method for various constraint sets $\mathcal{S}$, especially Parseval tight frames.

Also in 2013, Ravishankar and Bresler [40] made a distinction between the analysis model, where one models $y = x + n$ with $z = \Omega x$ being sparse, and the transform model, where $\Omega y = z + n$ where $z$ is sparse. The analysis version models the measurement as being a cosparse signal plus noise; the transform version models the measurement as being approximately cosparse. Another perspective on the distinction is that, if there is no noise, the analysis model constrains $y$ to be in the range space of $\Omega$, while there is no such
constraint on the transform model. The corresponding transform learning problem is

\[
\arg\min_{\Omega} \min_Z \|\Omega Y - Z\|_2^2 + R(\Omega) \quad \text{s.t.} \quad \|Z_i\|_0 \leq \alpha \quad \forall i, \quad (2.10)
\]

where \(i\) indexes the columns of \(Z\). Ref. [40] considers only square matrices \(\Omega\). The regularizer, \(R\), promotes diversity in the rows of \(\Omega\) to avoid trivial solutions, similar to the set constraint in (2.9).

A more recent development is directly modeling the convolutional structure during the learning process. In 2020, [51] proposed Convolutional Analysis Operator Learning (CAOL) to learn convolutional filters without patches. The CAOL cost function is

\[
\arg\min_{c_1, \ldots, c_K} \sum_{k=1}^{K} \min_z \frac{1}{2} \|c_k \odot x - z\|_2^2 + \beta \|z\|_0 \quad \text{s.t.} \quad [c_1 \ldots c_K] \in S. \quad (2.11)
\]

Unlike the previous cost functions, which typically require patches, CAOL can easily handle full-sized training images \(x\) due to the nature of the convolutional operator.

While model-based methods were being developed in the signal processing literature, convolutional neural network (CNN) models were being advanced and trained in the machine learning and computer vision literature [52] [53] [54]. The filters used in CNN models like U-Nets [55] can be thought of as having analysis roles in the earlier layers, and synthesis roles in the final layers [56]. See also [57] for further connections between analysis and transform models within CNN models. CNN training is usually supervised, and the supervised approach of bilevel learning of filters strengthens the relationships between the two approaches. A key distinction is that CNN models are generally feed-forward computations, whereas bilevel methods of the form (LL) have a cost function formulation. See Section 7 for further discussion of the parallels between CNNs and bilevel methods.

### 2.4 Summary

This background section focused on the lower-level problem: image reconstruction with a sparsity-based regularizer. After defining the problem and the need for regularization, Section 2.3 reviewed the history of analysis regularizer learning and included many examples of methods to learn hyperparameters.

Bilevel methods are just one, task-based way to learn such hyperparameters. Section 7.3 further expands on this point, but we can already see benefits of the task-based nature of bilevel methods. Without the bilevel approach, filters are learned so that they best sparsify training data. These sparsifying filters can then be used in a prior for image reconstruction tasks. However, they are learned to sparsify, not necessarily to best reconstruct. In contrast, the bilevel approach aims to learn filters that best reconstruct images (or whatever other task is desired), even if those filters are not the ones that best sparsify. Although this may
seem like a subtle distinction, [9] shows that different (synthesis) filters work better for image denoising versus image inpainting.

Having provided some background on the lower-level cost function and motivated bilevel methods, this review now turns to defining the upper-level loss function and surveying methods of hyperparameter optimization.

3 Background: Hyperparameter Optimization

Most inverse problems involve at least one hyperparameter. For example, the general reconstruction cost function (2.1) requires choosing the tuning parameter $\beta$ that trades-off the influence of the data-fit and regularization terms. The field of hyperparameter optimization is quite large and can include categorical hyperparameters, such as which optimizer to use; conditional hyperparameters, where certain hyperparameters are relevant only if others take on certain values; and integer or real-valued hyperparameters [58]. Here, we focus on learning real-valued, continuous hyperparameters.

A hyperparameter’s value can greatly influence the properties of the minimizer and a tuned hyperparameter typically improves over a default setting [58]. Fig. 3.1 illustrates how changing a tuning parameter can dramatically impact the visual quality of the reconstructed
If $\beta$ is too low, not enough weight is on the regularization term, and the minimizer is likely to be corrupted by noise in the measurements. If $\beta$ is too high, the regularization term dominates, and the minimizer will not align with the measurements.

Generalizing to an arbitrary learning problem that could have multiple hyperparameters, the goal of hyperparameter optimization is to find the “best” set of hyperparameters, $\gamma$, to meet a goal, described by a loss function $\ell$. Specifically, we wish to solve

$$\hat{\gamma} = \arg\min_{\gamma \in \Gamma} \mathbb{E}[\ell(\gamma)],$$

where $\Gamma$ is the set of all possible hyperparameters and the expectation is taken with respect to the distribution of the input data. If evaluating $\ell$ uses the output of another optimization problem, e.g., $\hat{x}$, then (3.1) is a bilevel problem as defined in (UL).

There are two key tasks in hyperparameter optimization. The first is to quantify how good a hyperparameter is; this step is equivalent to defining $\ell$ in (3.1). The second step is finding a good hyperparameter, which is equivalent to designing an optimization algorithm to minimize (3.1). The next two sections address each of these tasks in turn.

### 3.1 Image Quality Metrics

This section concentrates on the part of the upper-level loss function that compares the reconstructed image, $\hat{x}(\gamma)$, to the true image, $x_{\text{true}}$. As mentioned in Section 1, bilevel methods rarely require additional regularization for $\gamma$, but it is simple to add a regularization term to any of the loss functions if useful for a specific application. To discuss only the portion of the loss function that measures image quality, we use the notation $\ell(\gamma; \hat{x}(\gamma)) = l(\hat{x}, x_{\text{true}})$.

Picking a loss function is part of the engineering design process. No single loss function is likely to work in all scenarios; users must decide on the loss function that best fits their system, data, and goals. Consequently, there are a wide variety of loss functions proposed in the literature and some approaches combined multiple loss functions [5], [61].

One important decision criteria when selecting a loss function is the end purpose of the image. Much of the image quality assessment (IQA) literature focuses on metrics for images of natural scenes and is often motivated by applications where human enjoyment is the end-goal [62], [63]. In contrast, in the medical image reconstruction field, image quality is not the end-goal, but rather a means to achieving a correct diagnosis. Thus, the perceptual quality is less important than the information content.

There are two major classes of image quality metrics in the IQA literature, called full reference and no reference IQA\(^1\). The principles are somewhat analogous to supervised and unsupervised approaches in the machine learning literature. This section discusses some of

\(^1\)There are also reduced-reference image quality metrics, but we will not consider those here.
Figure 3.2: Example distortions that yield images with identical normalized MSE values: $\|x^{\text{true}} - x\| / \|x^{\text{true}}\| = 0.17$. (a) The original image, $x^{\text{true}}$, is a SheppLogan phantom [59]. The remaining images are displayed with the same colormap and have the following distortions: (b) blurred with an averaging filter, (c) additive, white Gaussian noise, (d) salt and pepper noise, and (e) a constant value added to every pixel.

The most common full reference and no reference loss functions; see [64] for a comparison of 11 full-reference IQA metrics and [65] for additional no-reference IQA metrics.

Perhaps surprisingly, the bilevel filter learning literature contains few examples of loss functions other than MSE or slight variants (see Section 6.2). While this is likely at least partially due to the computational requirements of bilevel methods (see Section 4 and 5), exploring additional loss functions is an interesting future direction for bilevel research.

3.1.1 Full Reference IQA

Full reference IQA metrics assume that you have a noiseless image, $x^{\text{true}}$, for comparison. Some of the simplest (and most common) full reference loss functions are:

- Mean square error (or $\ell_2$ error): $l_{\text{MSE}}(\hat{x}, x^{\text{true}}) = \frac{1}{N} \|\hat{x} - x^{\text{true}}\|_2^2$
- Mean absolute error (or $\ell_1$ error): $l_{\text{MAE}}(\hat{x}, x^{\text{true}}) = \frac{1}{N} \|\hat{x} - x^{\text{true}}\|_1$
- Signal to Noise Ratio (SNR, commonly expressed in dB):
  $$l_{\text{SNR}}(\hat{x}, x^{\text{true}}) = 10 \log \left( \frac{\|x^{\text{true}}\|_2^2}{\|\hat{x} - x^{\text{true}}\|_2^2} \right)$$
- Peak SNR (PSNR, in dB): $l_{\text{PSNR}}(\hat{x}, x^{\text{true}}) = 10 \log \left( \frac{N \|x^{\text{true}}\|_{\infty}}{\|\hat{x} - x^{\text{true}}\|_2} \right)$

The Euclidean norm is also frequently used as the data-fit term for reconstruction.

MSE (and the related metrics SNR and PSNR) are common in the signal processing field; they are intuitive and easy to use because they are differentiable and operate point-wise. However, these measures do not align well with human perceptions of image quality [64], [66]. For example, scaling an image by 2 leads to the same visual quality but causes 100% MSE. Fig. 3.2 shows a clean image and five images with different degradations. All five degraded images have almost equivalent MSEs, but humans judge their qualities as very different.

Tuning parameters using MSE tends to lead to images that are overly-smoothed,
sacrificing high frequency information [67], [68]. High frequency details are particularly important for perceptual quality as they correspond to edges in images. Therefore, some authors use the MSE on edge-enhanced versions of images to discourage solutions that blur edges. For example, [69] used a “high frequency error norm” metric consisting of the MSE of the difference of $\hat{x}$ and $x^{\text{true}}$ after applying a Laplacian of Gaussian (LoG) filter.

Another common full-reference IQA is Structural SIMilarity (SSIM) [70] that attempts to address the issues with MSE discussed above. SSIM is defined in terms of the local luminance, contrast, and structure in images. A multiscale extension of SSIM, called MS-SSIM, considers information at different resolutions [71]. The method computes the contrast and structure measures of SSIM for downsampled versions of the input images and then defines MS-SSIM as the product of the luminance at the original scale and the contrast and structure measures at each scale. However, SSIM and MS-SSIM may not correlate well with human observer performance on radiological tasks [72].

Recent works, e.g., [65], [73], consider using (deep) CNN models for IQA. CNN methods are increasingly popular and their use as a model for the human visual system [74] makes them an attractive tool for assessing images. For example, [73] proposed a CNN with convolutional and pooling layers for feature extraction and fully connected layers for regression. They used VGG [75], a frequently-cited CNN design with $3 \times 3$ convolutional kernels, as the basis of the feature extraction portion of their network. Ref. [73] showed that deeper networks with more learnable parameters were able to better predict image quality. However, datasets of images with quality labels remain relatively scarce, making it difficult to train deep networks.

### 3.1.2 No Reference IQA

No reference, or unsupervised, IQA metrics attempt to quantify an image’s quality without access to a noiseless version of the image. These metrics rely on modeling statistical characteristics of images or noise.

Many no reference IQA metrics assume the noise distribution is known. The discrepancy principle is a classic example that uses an assumed noise distribution to characterize the expected relation between the reconstructed image and the noisy data. For additive zero-mean white Gaussian noise with known variance $\sigma^2$, the discrepancy principle uses the fact that the expected MSE in the data space is the noise variance [16]:

$$\mathbb{E}\left[\frac{1}{M} \| A\hat{x}(\gamma) - y \|^2_2 \right] = \sigma^2.$$  

The discrepancy principle can be used as a stopping criteria in machine learning methods or as a loss function, e.g.,

$$\ell(\gamma; \hat{x}(\gamma)) = \left( \frac{1}{M} \| A\hat{x}(\gamma) - y \|^2_2 - \sigma^2 \right)^2.$$
However, images of varying quality can yield the same noise estimate, as seen in Fig. 3.2. Related methods have been developed for Poisson noise as well [76].

Paralleling MSE’s popularity among supervised loss metrics, Stein’s Unbiased Risk Estimator (SURE) [77] is an unbiased estimate of MSE that does not require noiseless images. Let \( \mathbf{y} = \mathbf{x}^{\text{true}} + \mathbf{n} \) denote a signal plus noise measurement where \( \mathbf{n} \) is, as above, Gaussian noise with known variance \( \sigma^2 \). The SURE estimate of the MSE of a denoised signal, \( \hat{\mathbf{x}} \), is

\[
\frac{1}{N} \| \mathbf{y} - \hat{\mathbf{x}}(\mathbf{y}) \|_2^2 - \sigma^2 + \frac{2\sigma^2}{N} \nabla_{\mathbf{y}} \hat{\mathbf{x}}(\mathbf{y}),
\]

(3.2)

where we write \( \hat{\mathbf{x}} \) as a function of \( \mathbf{y} \) to emphasize the dependence. For large signal dimensions, \( N \), such as is common in image reconstruction problems, the law of large numbers suggests SURE is a fairly accurate approximation of the true MSE.

However, it is often impractical to evaluate the divergence term in (3.2), due to computational limitations or not knowing the form of \( \hat{\mathbf{x}}(\mathbf{y}) \). A Monte-Carlo approach to estimating the divergence [78] uses the following key equation:

\[
\nabla_{\mathbf{y}} \hat{\mathbf{x}}(\mathbf{y}) = \lim_{\epsilon \to 0} \mathbb{E} \left[ b' \cdot \frac{\hat{\mathbf{x}}(\mathbf{y} + \epsilon \mathbf{b}) - \hat{\mathbf{x}}(\mathbf{y})}{\epsilon} \right],
\]

(3.3)

where \( \mathbf{b} \) is a independent and identically distributed (i.i.d.) random vector with zero mean, unit variance, and bounded higher order moments. Theoretical and empirical arguments show that a single noise vector can well-approximate the divergence [78], so only two calls to the lower-level solver \( \hat{\mathbf{x}}(\mathbf{y}) \) are required. This method treats the lower-level problem like a blackbox, thus allowing one to estimate the divergence of complicated functions, including those that may not be differentiable.

See [79]–[81] for examples of applying the Monte-Carlo estimation of SURE to train deep neural networks, and [82], [83] for two examples of learning a tuning parameter using a bilevel approach with SURE as the upper-level loss function. For extensions to inverse problems and noise from exponential families, see [84].

While SURE and the discrepancy principle are popular no-reference metrics in the signal processing literature, there are many additional no-reference metrics in the image quality assessment literature. These metrics typically depend on modeling one (or more) of three things [63]:

- image source characteristics,
- image distortion characteristics, e.g., blocking artifact from JPEG compression, and/or
- human visual system perceptual characteristics.

As an example of a strategy that can capture both image source and human visual system characteristics, natural scene\(^2\) statistics characterize the distribution of various features

\(^2\)Natural scenes are those captured by optical cameras (not created by computer graphics or other artificial processes) and are not limited to outdoor scenes.
in natural scenes, typically using some filters [63], [85]. If a feature reliably follows a specific statistical pattern in natural images but has a noticeably different distribution in distorted images, one can use that feature to assign quality scores to images. Some IQA metrics attempt to first identify the type of distortion and measure features specific to that distortion, while others use the same features for all images.

In addition to their use in full-reference IQA, CNN models have been trained to perform no-reference IQA [73], [86]. For example, Kang, Ye, Li, et al. [86] propose a CNN model that extracts small (32 × 32) patches from images, estimates quality for each one, and averages the scores over all patches to get the quality score for the entire image. Briefly, their method involves local contrast normalization for each patch, applying (learned) convolutional filters to extract features, maximum and minimum pooling, and fully connected layers with rectified linear units (ReLUs). As with most no reference IQAs, [86] trained their CNN on a dataset of (human encoded) image quality scores (see [87] for a commonly used collection of publicly available test images with quality scores). Unlike most other IQA approaches, [86] used backpropagation to learn all the CNN weights rather than learning a transformation from (handcrafted) features to quality scores.

Interestingly, some of the no-reference IQA metrics [63], [85], [86] approach the performance of the full-reference IQAs in terms of their ability to match human judgements of image quality. This observation suggests that there is room to improve full-reference IQA metrics and that assessing image quality is a very challenging problem!

### 3.2 Parameter Search Strategies

After selecting a metric to measure how good a hyperparameter is, the next task is devising a strategy to find the best hyperparameter according to that metric. Search strategies fall into three main categories: (i) model-free, ℓ-only; (ii) model-based, ℓ-only; and (iii) gradient-based, using both ℓ and ∇ℓ. Model-free strategies do not assume any information about the hyperparameter landscape, whereas model-based strategies use historical ℓ evaluations to predict the loss-function at untested hyperparameter values.

The following sections describe common model-free and model-based hyperparameter search strategies that only use ℓ. See [7, Ch. 13 and Ch. 20.6] for discussion of additional gradient-free methods for bilevel problems, e.g., population-based evolutionary algorithms, and [88] for a general discussion of derivative-free optimization methods.

The third class of hyperparameter optimization schemes are approaches based on gradient descent of a bilevel problem. The high-level strategy in bilevel approaches is to calculate the gradient of the upper-level loss function, ℓ, with respect to and then use any gradient descent method to minimize . Although this approach can be computationally challenging, it generalizes well to a large number of hyperparameters. Section 4 and Section 5 discuss
this point further and go into depth on different methods for computing this gradient.

3.2.1 Model-free

The most common search strategy is probably an empirical search, where a researcher tries different hyperparameter combinations manually. A punny, but often accurate, term for this manual search is GSD: grad[uate] student descent [89]. Bergstra and Bengio [90] hypothesizes that manual search is common because it provides some insight as the user must evaluate each option, it requires no overhead for implementation, and it can perform reliably in very low dimensional hyperparameter spaces.

Grid search is a more systematic alternative to manual search. When there are only one or two continuous hyperparameters, or the possible set of hyperparameters, $\Gamma$, is small, a grid search (or exhaustive search) strategy may suffice to find the optimal value, $\hat{\gamma}$, to within the grid spacing. However, the complexity of grid search grows exponentially with the number of hyperparameters. Regularizers frequently have many hyperparameters, so one must use a more sophisticated search strategy.

One popular approach is random search, which Bergstra and Bengio [90] shows is superior to a grid search, especially when some hyperparameters are more important than others. There are also variations on random search, such as using Poisson disk sampling theory to explore the hyperparameter space [91]. The simplicity of random search makes it popular, and, even if one uses a more complicated search strategy, random search can provide a useful baseline or an initialization strategy. However, random search, like grid search, suffers from the curse of dimensionality, and is less effective as the hyperparameter space grows.

Another group of model-free blackbox strategies are population-based methods such as evolutionary algorithms. A popular population-based method is the covariance matrix adaption evolutionary strategy (CMA-ES) [92]. In short, every iteration, CMA-ES involves sampling a multivariate normal distribution to create a number of “offspring” samples. Mimicking natural selection, these offspring are judged according to some fitness function, a parallel to the upper-level loss function. The fittest offspring determine the update to the normal distribution and thus “pass on” their good characteristics to the next generation.

3.2.2 Model-based

Model-based search strategies assume a model (or prior) for the hyperparameter space and use only loss function evaluations (no gradients). This section discusses two common model-based strategies: Bayesian approaches and trust region methods.
Bayesian Approaches

Bayesian approaches fit previous hyperparameter trials’ results to a model to select the hyperparameters that appear most promising to evaluate next [93]. For example, a common model for the hyperparameters is the Gaussian Process prior. Given a few hyperparameter and cost function points, a Bayesian method involves the following steps.

1. Find the mean and covariance functions for the Gaussian Process. The mean function will generally interpolate the sampled points. The covariance function is generally expressed as a kernel function, often using squared exponential functions [94].

2. Create an acquisition function. The acquisition function captures how desirable it is to sample (“acquire”) a hyperparameter setting. Thus, it should be large (desirable) for hyperparameter values that are predicted to yield small loss function values or that have high enough uncertainty that they may yield low losses. The design of the acquisition function thus trades-off between exploring new areas of the hyperparameter landscape with high uncertainty and a more locally focused exploitation of the current best hyperparameter settings. See [94] for a discussion of specific acquisition function designs.

3. Maximize the acquisition function (typically designed to be easy to optimize) to determine which hyperparameter point to sample next.

4. Evaluate the loss function at the new hyperparameter candidate.

These steps repeat for a given amount of time or until convergence.

Trust Region Method

Another derivative-free optimization method that uses only loss function evaluations is a trust-region method. This section describes the specific trust-region method as presented in [95] (see references therein for previous, similar methods).

The derivative-free, trust-region method (TRM) [96] is similar to Bayesian optimization in that it involves fitting an easier to optimize function to the loss function of interest, $\ell$, and then minimizing the easier, surrogate function (the “model”). Thus, TRM requires only function evaluations, not gradients, to construct and then minimize the model. However, unlike most Bayesian optimization-based approaches, TRM uses a local (often quadratic) model for $\ell$ around the current iterate, rather than a surrogate that fits all previous points. In taking a step based on this local information, TRM is more similar to gradient-based approaches.

The trust-region captures how well the model matches the observed $\ell$ values and determines the maximum step at every iteration. The “goodness” of the model is typically
quantified as the ratio of the actual decrease in $\ell$ (based on observed function evaluations) to the predicted decrease (based on the model). If this ratio is relatively large (close to one), then the model is a good approximation of $\ell$ and the trust-region grows for the next iteration. If this ratio is close to zero, then the observed decrease is much less than predicted and the trust-region shrinks.

Recall that evaluating $\ell$ is typically expensive in bilevel problems as each upper-level function evaluation involves optimizing the lower-level cost. Thus, even constructing the model for a TRM can be expensive. To mitigate this computational complexity, [95] incorporated a dynamic accuracy component, with the accuracy for the lower-level cost initially set relatively loose (leading to rough estimates of $\ell$) but increasing with the upper-level iterations (leading to refined estimates of $\ell$ as the algorithm nears a stationary point). One can use any optimization method for the lower-level cost; [95] used a gradient method for the lower-level optimization method with well-known convergence results to facilitate establishing convergence and computational complexity results.

The upper-level loss function considered in [95] is additively separable and quadratic:

$$\ell(\gamma) = \frac{1}{J} \sum_{j=1}^{J} \ell(\gamma; \hat{x}_j(\gamma)) = \frac{1}{J} \sum_{j=1}^{J} (r(\gamma; \hat{x}_j(\gamma)))^2,$$

where $r$ is typically $\hat{x}_j(\gamma) - x_j^{\text{true}}$. (Although we define the sum to be over the number of training samples, this expression easily generalizes to include a regularization term on $\gamma$ by defining an additional $r_{J+1}$ term.)

Given a current value for the hyperparameters, $\gamma^*$, the TRM models the local upper-level loss function by creating a linear model such that

$$r_j(\gamma^* + \delta) \approx m_j(\delta) := r_j(\gamma^*) + g'_j \delta,$$

where $g_j \in \mathbb{R}^R$ approximates $\nabla r_j(\gamma^*)$. Then, the overall model for the upper-level problem is quadratic:

$$\ell(\gamma^* + \delta) \approx \sum_j m_j^2(\delta) = \ell(\gamma^*) + \frac{1}{J} \sum_j 2r_j(\gamma^*)g'_j \delta + (g'_j \delta)^2. \quad (3.4)$$

One can estimate the gradients, $g_j$, by interpolating a set of $R$ samples (recall $\gamma \in \mathbb{R}^R$) of the upper-level loss function. This process involves choosing a set of interpolating points, $\{\gamma^* + \delta^{(1)}, \ldots, \gamma^* + \delta^{(R)}\}$, (approximately) evaluating $r$ at each one, then solving

$$\begin{bmatrix} r_j(\gamma^* + \delta^{(1)}) - r_j(\gamma^*) \\ \vdots \\ r_j(\gamma^* + \delta^{(R)}) - r_j(\gamma^*) \end{bmatrix}_{R \times 1} = \begin{bmatrix} (\delta^{(1)})' \\ \vdots \\ (\delta^{(R)})' \end{bmatrix}_{R \times R} g_j$$
for \( j \in [1 \ldots J] \).

After forming the quadratic model for the upper-level loss, the TRM minimizes the model (3.4) within some trust region, which is a simple convex-constrained quadratic problem. After computing \( \hat{\delta} \), the TRM accepts the step and updates the hyperparameters \( (\gamma^{(i+1)} = \gamma^{(i)} + \hat{\delta}) \) if the actual reduction (based on the estimated loss function values) to predicted reduction (based on the quadratic model of the loss function) ratio is large enough. Otherwise, if the ratio is low, the update step is rejected and the trust region shrinks.

While the TRM appears to involve \( R \) upper-level function evaluations every iteration to construct the gradient estimates, after an initialization, one can generally reuse samples, gradually replacing old samples with the samples at new hyperparameter iterates. Ref. [95] discusses requirements on the interpolation set to guarantee a good geometry and conditions for re-setting the interpolation sample if the model is not sufficiently accurate.

A main result from [95] is a bound on the number of iterations to reach an \( \epsilon \)-optimal point (defined as \( \min_u \| \nabla \ell(\gamma^{(u)}) \| < \epsilon \), where \( u \) indexes the upper-level iterates). The bound derivation assumes (i) \( \Phi \) is differentiable in \( x \), (ii) \( \Phi \) is \( \mu \)-strongly convex, i.e., \( \Phi(x) - \frac{\mu}{2} \| x \|^2 \) is convex, (iii) the derivative of \( \Phi \) is Lipschitz continuous, and (iv) the first and second derivative of the lower-level cost with respect to \( x \) exist and are continuous. These requirements are satisfied by the example filter learning problem (Ex), when \( A \) has full column rank, and more generally when there are certain constraints on the hyperparameters. The iteration bound is a function of the following:

- the tolerance \( \epsilon \),
- the trust region parameters (parameters that control the increase and decrease in trust region size based on the actual to predicted reduction, the starting trust region size, and the minimum possible trust region size),
- the initialization for \( \gamma \), and
- the maximum possible error between the gradient of the upper-level loss function and the gradient of the model for the upper-level loss within a trust region (when the gradient of \( \ell \) is Lipschitz continuous, this bound is the corresponding Lipschitz constant).

The number of iterations required to reach such an \( \epsilon \)-optimal point is \( O\left( \frac{1}{\epsilon^2} \right) \) [95] and the number of required upper-level loss function evaluations depends more than linearly on \( N \) [97]. That growth with \( N \) impedes its use in problems with many hyperparameters, though new techniques such as [98] may be able to decrease or remove the dependency.

### 3.3 Summary

Turning from the discussion of the lower-level problem in Section 2, this section concentrated on the other two aspects of bilevel problems: the upper-level loss function and the
optimization strategy.

The loss function defines what a “good” hyperparameter is, typically using a metric of image quality to compare $\hat{x}(y)$ to a clean, training image, $x^{\text{true}}$. MSE is the most common upper-level loss function. It is well known from statistical estimation that the estimator that minimizes MSE is the conditional mean, $\hat{x}(y) = \mathbb{E}[x|y]$. Thus, if MSE is the true metric of interest, then lower-level problems should be designed to try to approximate $\mathbb{E}[x|y]$ closely. Yet lower-level formulations in most bilevel papers are not described as conditional mean estimators or approximations thereof. Section 3.1 discussed many other full reference and no reference options, including ones motivated by human judgements of perceptual quality, from the image quality assessment literature; Section 6.2 gives examples of bilevel methods that use some of these other loss functions.

The second half of this section concentrated on model-free and model-based hyperparameter search strategies. The grid search, CMA-ES, and trust region methods described above all scale at least linearly with the number of hyperparameters. Similarly, Bayesian optimization is best-suited for small hyperparameter dimensions; [94] suggests it is typically used for problems with 20 or fewer hyperparameters.

The remainder of this review considers gradient-based strategies for hyperparameter optimization. The main benefit of gradient-based methods is that they can scale to the large number of hyperparameters that are commonly used in machine learning applications. Correspondingly, the main drawback of a gradient-based method over the methods discussed in this section is the implementation complexity and the differentiability requirement. Sections 4 and 5 discuss multiple options for gradient-based methods.

4 Gradient Based Bilevel Methodology: The Groundwork

When the lower-level optimization problem (LL) has a closed-form solution, $\hat{x}$, one can substitute that solution into the upper-level loss function (UL). In this case, the bilevel problem is equivalent to a single level problem and one can use classic single-level optimization methods to minimize the upper-level loss. (See [99] for analysis and discussion of some simple bilevel problems with closed-form solutions for $\hat{x}$.) This review focuses on the more typical bilevel problems that lack a closed-form solution for $\hat{x}$.

Although there are a wide variety of optimization methods for this challenging category of bilevel problems, many methods are built on gradient descent of the upper-level loss. The primary challenge with gradient-based methods is that the gradient of the upper-level function depends on a variable that is itself the solution to an optimization problem involving the hyperparameters of interest. This section describes two main approaches for overcoming this challenge. The first approach uses the fact that the gradient of the lower-level cost
function is zero at the minimizer to compute an exact gradient at the exact minimizer. The second approach uses knowledge of the update scheme for the lower-level cost function to calculate the exact gradient for an approximation to the minimizer after a specific number of lower-level optimization steps.

With this (approximation of the) gradient of the lower-level optimization variable with respect to the hyperparameters, one can compute the gradient of the upper-level loss function with respect to the hyperparameters, \( \gamma \). Section 5 uses the building blocks from this section to explain various bilevel methods based on this gradient.

4.1 Set-up

Recall from Section 1.2 that a generic bilevel problem is

\[
\text{argmin}_{\gamma} \ell(\gamma ; \hat{x}(\gamma)) \quad \text{where} \quad \hat{x}(\gamma) = \text{argmin}_x \Phi(x ; \gamma).
\]  

(4.1)

The way (4.1) expresses the lower-level problem implies a unique minimizer, \( \hat{x} \). This is satisfied when \( \Phi \) is a strictly convex function of \( x \), which is typically the case when we use a strictly convex regularizer. However, the following derivations also hold in the case of non-unique or local minimizers. For more discussion, [19] defines optimistic and pessimistic versions of the bilevel problem for the case of multiple lower-level solutions.

For simplicity, hereafter we focus on the case \( \mathbb{F} = \mathbb{R} \). Using the chain rule, the gradient of the upper-level loss function with respect to the hyperparameters is

\[
\nabla \ell(\gamma) = \nabla_\gamma \ell(\gamma ; \hat{x}(\gamma)) + (\nabla_\gamma \hat{x}(\gamma))' \nabla_x \ell(\gamma ; \hat{x}(\gamma)),
\]  

(4.2)

where on the right hand side \( \nabla_\gamma \) and \( \nabla_x \) denote partial derivatives w.r.t. the first and second arguments of \( \ell(\gamma ; x) \), respectively. We typically select the loss function such that it is easy to compute these partials. For example, if \( \ell \) is the MSE training loss, i.e.,

\[
\ell(\gamma ; \hat{x}(\gamma)) = \frac{1}{2} \| \hat{x}(\gamma) - x^{\text{true}} \|_2^2,
\]

then

\[
\nabla_\gamma \ell(\gamma ; \hat{x}(\gamma)) = 0 \quad \text{and} \quad \nabla_x \ell(\gamma ; \hat{x}(\gamma)) = \hat{x}(\gamma) - x^{\text{true}}.
\]

The following sections survey methods to find \( \nabla_\gamma \hat{x}(\gamma) \), which is the more challenging piece in (4.2).

4.2 Minimizer Approach

The first approach finds \( \nabla_\gamma \hat{x}(\gamma) \) by assuming the gradient of \( \Phi \) at the minimizer is zero. There are two ways to arrive at the final expression: the implicit function theorem (IFT)
perspective (as in [47], [100]) and the Lagrangian/KKT transformation perspective (as
in [19], [42]). This section presents both perspectives in sequence. The end of the section
summarizes the required assumptions and discusses computational complexity and memory
requirements.

The first step in both perspectives is to assume we have computed \( \hat{x}(\gamma) \) and that
the lower-level problem 4.1 is unconstrained (e.g., no non-negativity or box constraints).
Therefore, the gradient of \( \Phi \) with respect to \( x \) and evaluated at \( \hat{x} \) must be zero:

\[
\nabla_x \Phi(x; \gamma) \bigg|_{x = \hat{x}(\gamma)} = \nabla_x \Phi(\hat{x}; \gamma) = 0.
\] (4.3)

After this point, the two perspectives diverge.

4.2.1 Implicit Function Theorem Perspective

In the IFT perspective, we apply the IFT (cf. [101]) to define a function \( h \) such that
\( \hat{x}(\gamma) = h(y, \gamma) \). If we could write \( h \) explicitly, then the bilevel problem could be converted
to an equivalent single-level. However, per the IFT, we do not need to define \( h \), we only
state that such an \( h \) exists. Combining this definition with (4.3) yields

\[
0 = \nabla_x \Phi(h(y, \gamma); \gamma).
\] (4.4)

Using the chain rule, we differentiate both sides of (4.4) with respect to \( \gamma \). The \( I \) in the
equation below follows from the chain rule because \( \nabla_\gamma \gamma = I \). We then rearrange terms to
solve for the desired quantity, noting that \( \nabla_\gamma \hat{x}(\gamma) = \nabla_\gamma h(y, \gamma) \). Thus, evaluating all terms
at \( \hat{x} \) leads to a gradient expression for the lower-level problem:

\[
0 = \nabla_{xx} \Phi(h(y, \gamma); \gamma) \nabla_\gamma h(y, \gamma) + I \cdot \nabla_{xy} \Phi(h(y, \gamma); \gamma)
\]

\[
\nabla_\gamma h(y, \gamma) = -[\nabla_{xx} \Phi(h(y, \gamma); \gamma)]^{-1} \cdot \nabla_{xy} \Phi(h(y, \gamma); \gamma)
\]

\[
\nabla_\gamma \hat{x}(\gamma) = -[\nabla_{xx} \Phi(\hat{x}; \gamma)]^{-1} \cdot \nabla_{xy} \Phi(\hat{x}; \gamma).
\] (4.5)

When \( \Phi \) is strictly convex, the Hessian of \( \Phi \) is positive definite and \( \nabla_{xx} \Phi(\hat{x}; \gamma) \) is invertible.

Substituting (4.5) into (4.2) yields the following expression for the gradient of the
upper-level loss function with respect to \( \gamma \):

\[
\nabla \ell(\gamma) = \nabla_\gamma \ell(\gamma; \hat{x}(\gamma)) - (\nabla_{xy} \Phi(\hat{x}; \gamma))' (\nabla_{xx} \Phi(\hat{x}; \gamma))^{-1} \nabla_x \ell(\gamma; \hat{x}).
\]

Ref. [100] provides example simple bilevel problems that verify that the IFT gradient
agrees with the analytic gradient obtained by substituting the closed-form solution of the
lower-level problem into the upper-level problem and differentiating directly.
4.2.2 KKT Conditions

In the Lagrangian perspective, \((4.3)\) is treated as a constraint on the upper-level problem, creating a single-level problem with \(N\) equality constraints:

\[
\argmin_\gamma \ell(\gamma; x) \text{ subject to } \nabla_x \Phi(x; \gamma) = 0_N. \tag{4.6}
\]

The corresponding Lagrangian is

\[
L(x, \gamma, \nu) = \ell(\gamma; x) + \nu^T \nabla_x \Phi(x; \gamma)
\]

where \(\nu \in \mathbb{R}^N\) is a vector of Lagrange multipliers associated with the \(N\) equality constraints in \((4.6)\). The Lagrange reformulation is generally well-posed because many bilevel problems, such as \((\text{Ex})\), satisfy the linear independence constraint qualification (LICQ) \([102], [103]\). LICQ requires that the matrix of derivatives of the constraint have full row rank \([103]\), i.e.

\[
\text{rank} \left( \begin{bmatrix} \nabla_{xY} \Phi(x; \gamma) & \nabla_{xx} \Phi(x; \gamma) \end{bmatrix} \right) = N.
\]

Strict convexity of \(\Phi(x; \gamma)\) is therefore a sufficient condition for LICQ to hold. (Note the similarity to the IFT perspective, where strict convexity is sufficient for the Hessian to be invertible.)

The first KKT condition states that, at the optimal point, the gradient of the Lagrangian with respect to \(x\) must be \(0\). We can use this fact to solve for the optimal Lagrangian multiplier, \(\hat{\nu}\):

\[
\nabla_x L(\hat{x}; \gamma, \hat{\nu}) = \nabla_x \ell(\gamma; \hat{x}) + \nabla_{xx} \Phi(\hat{x}; \gamma) \hat{\nu} = 0
\]

\[
\hat{\nu} = - (\nabla_{xx} \Phi(\hat{x}; \gamma))^T \nabla_x \ell(\gamma; \hat{x}).
\]

Substituting the expression for \(\hat{\nu}\) into the gradient of the Lagrangian with respect to \(\gamma\) yields

\[
\nabla_\gamma L(\hat{x}, \gamma, \hat{\nu}) = \nabla_\gamma \ell(\gamma; \hat{x}) + \left( \nabla_{xy} \Phi(\hat{x}; \gamma) \right)^T \hat{\nu}
\]

\[
= \nabla_\gamma \ell(\gamma; \hat{x}) - \left( \nabla_{xy} \Phi(\hat{x}; \gamma) \right)^T \left( \nabla_{xx} \Phi(\hat{x}; \gamma) \right)^{-1} \nabla_x \ell(\gamma; \hat{x}),
\]

which is equivalent to \((4.8)\).

Ref. \([19]\) generalized the Lagrangian approach to the case where the forward model is defined only implicitly, \(e.g.,\) as the solution to a differential equation. The authors write the lower-level problem as

\[
\hat{x} = \argmin_x \min_{\tilde{y}} \|y - \tilde{y}\|^2_2 + R(x) \text{ s.t. } e(\tilde{y}, x) = 0, \tag{4.7}
\]

where the constraint function, \(e\), incorporates the implicit system model. For example, when the forward model is linear \((Ax)\), taking \(e(\tilde{y}, x) = \|Ax - \tilde{y}\|^2_2\) shows the equivalence of the approach here to the one in \([19]\).

\[\text{The KKT transformation (4.6) relates bilevel optimization to mathematical programs with equilibrium constraints (MPEC). Some authors use the terms interchangeably; see [7, Ch. 12].}\]
4.2.3 Summary of Minimizer Approach

In summary, the upper-level gradient expression for the minimizer approach (i.e., when one “exactly” minimizes the lower-level cost function) is

\[
\nabla \ell(\gamma) = \nabla_x \ell(\gamma; \hat{x}) - (\nabla_{xy} \Phi(\hat{x}; \gamma))' (\nabla_{xx} \Phi(\hat{x}; \gamma))^{-1} \nabla_x \ell(\gamma; \hat{x}).
\]

Thus, for a given loss function and cost function, calculating the gradient of the upper-level loss function (with respect to \( \gamma \)) requires the following components all evaluated at \( x = \hat{x} \): \( \nabla_x \ell(\gamma; x) \in \mathbb{R}^r \), \( \nabla_{xy} \Phi(\gamma; x) \in \mathbb{R}^{rx} \), \( \nabla_{xx} \Phi(\gamma; x) \in \mathbb{R}^{rx} \), and \( \nabla_x \ell(\gamma; x) \in \mathbb{R}^r \).

Continuing the specific example of learning filter coefficients and tuning parameters \((\text{Ex})\), the components are:

\[
\begin{align*}
\nabla_x \Phi(\hat{x}; \gamma) &= \Lambda'(Ax - y) + e^{\beta_0} \sum_{k=1}^K e^{\beta_k} \hat{c}_k \odot \hat{\phi}.(c_k \odot x; \epsilon) \\
\nabla_{x\beta_k} \Phi(\hat{x}; \gamma) &= e^{\beta_0 + \beta_k} \hat{c}_k \odot \hat{\phi}.(c_k \odot \hat{x}) \\
\nabla_{x\phi_{k,s}} \Phi(\hat{x}; \gamma) &= e^{\beta_0 + \beta_k} \left( \hat{\phi}.((c_k \odot \hat{x})^{(s)}) + \hat{c}_k \odot (\hat{\phi}.(c_k \odot \hat{x}) \odot \hat{x}^{(s)}) \right) \\
\nabla_{xx} \Phi(\hat{x}; \gamma) &= \Lambda' A + e^{\beta_0} \sum_k e^{\beta_k} C_k \delta \text{diag}(\hat{\phi}.(c_k \odot \hat{x})) C_k \\
\n\nabla_y \ell(\gamma; x) &= 0 \\
\n\nabla_x \ell(\gamma; \hat{x}) &= \hat{x}(\gamma) - x^{\text{true}}.
\end{align*}
\]

Here, the notation \( x^{(i)} \) means circularly shifting the vector \( x \) by \( i \) elements, and \( c_{k,s} \) denotes the \( s \)th element of the \( k \)th filter \( c_k \), where \( s \) is a tuple that indexes each dimension of \( c_k \). Appendix A.1 gives examples of using the \( x^{(i)} \) notation and derives \( \nabla_{x\phi_{k,s}} \Phi(\hat{x}; \gamma) \), which is the key step to expressing \( \nabla_{x\phi_{k,s}} \Phi(\hat{x}; \gamma) \). The other gradients easily follow from \( \nabla_x \Phi(\hat{x}; \gamma) \) using standard gradient tools for matrix expressions [104].

The minimizer approach to finding \( \nabla \ell(\gamma) \) uses the following assumptions:

1. Both the upper and lower optimization problems have no inequality constraints.
2. \( \hat{x} \) is the minimizer to the lower-level cost function, not an approximation of the minimizer. This constraint ensures that (4.3) holds.
3. The cost function \( \Phi \) is twice-differentiable in \( x \) and differentiable with respect to \( x \) and \( \gamma \). This condition disqualifies the 0-norm and 1-norm as choices for the function \( \phi \).
4. The Hessian of the lower-level cost function, \( \nabla_{xx} \Phi(\gamma; x) \), is invertible; this is guaranteed when \( \Phi \) is strictly convex.

The first condition technically excludes applications like CT imaging, where the image is typically constrained to be non-negative. However, non-negativity constraints are rarely
required when good regularizers are used, so the resulting non-constrained image can still be useful in practice [101].

The second constraint is often the most challenging since the lower-level problem typically uses an iterative algorithm that runs for a certain number of iterations or until a given convergence criteria is met. As previously noted, if there were a closed-form solution for $\hat{x}$, then we would not have needed to use the IFT or Lagrangian to find the partial derivative of $\hat{x}$ with respect to $\gamma$. Since one usually does not reach the exact minimizer, the calculated gradient will have some error in it, depending on how close the final iterate is to the true minimizer $\hat{x}$. Thus, the practical application of this method is more accurately called Approximate Implicit Differentiation (AID) [105], [106]. Section 4.5 further discusses gradient accuracy.

4.2.4 Computational Costs

The largest cost in computing the gradient of the upper-level loss using (4.8) is often finding (an approximation of) $\hat{x}$. However, this cost is difficult to quantify, as the IFT approach is agnostic to the lower-level optimization methodology. To compare the bilevel gradient methods, we will later assume the cost is comparable to the gradient descent calculations used in the unrolled approach (described in the following section). However, this is an over-estimation of the cost, as the IFT approach is not constrained to smooth lower-level updates, and one can use optimization methods with, e.g., warm starts and restarts to reduce this cost.

When the lower-level problem satisfies the assumptions above, and assuming one has already found $\hat{x}$, a straight-forward approach to computing the gradient (4.8) would be dominated by the $O(N^3)$ operations required to compute the Hessian’s inverse. For many problems, $N$ is large, and that matrix inversion is infeasible due to computation or memory requirements. Instead, as described in [107], one can use a conjugate gradient (CG) method to compute the matrix-vector product

$$
(\nabla_{xx}\Phi(\hat{x} ; \gamma))^{-1}\nabla_x\ell(\gamma ; \hat{x})
$$

(4.10)

because the Hessian is symmetric and positive definite (see assumption #4 in the previous section). For a generic $A$, each CG iteration requires multiplying the Hessian by a vector, which is $O(N^2)$.

CG takes $N$ iterations to converge fully (ignoring finite numerical precision), so the final complexity is still $O(N^3)$ in general. However, the Hessian often has a special structure that simplifies computing the matrix-vector product. Consider the running example of learning.
filters per (Ex). The Hessian, as given in (4.9), multiplied with any vector \( v \in \mathbb{F}^N \) is

\[
\nabla_{xx} \Phi(\hat{x}; \gamma, y) \cdot v = A'(A'v) + 2N^2 \varepsilon^0 \sum_k c_k \beta_k \underbrace{C_k' \cdot \text{diag}(\hat{\phi}(c_k \circ \hat{x}))}^{N} \cdot \underbrace{(C_k v)}_{NS}.
\]

(4.11)

The annotations show the multiplications required for each component, where we used the simplifying assumption that the number of measurements matches the number of unknowns \((M = N)\).

As written, (4.11) does not made any assumptions on \( A \), so the first term is still computationally expensive. If \( A \) is the identity matrix (as in denoising), the \( N^2 \) term could instead be zero cost. If \( A'A \) is circulant, \( e.g. \), if \( A \) is a MRI sampling matrix that can be written in terms of a discrete Fourier transform, then the cost is \( N \log(N) \). More generally, the computational cost for one (of \( N \)) iterations of CG is \( O(c_A N) \) where \( c_A \in [0, N] \) is some constant dependent on the structure of \( A \).

For the second addend in (4.11), we assume that \( S \ll N \), so direct convolution is most efficient and the matrix-vector product requires \( O(NS) \) multiplies. When the filters are relatively large, one can use Fourier transforms for the filtering, and the cost is \( O(N \log(N)) \). The final cost of the Hessian-vector product for (Ex) is \( O(c_A N + RN) \). This cost includes a multiplication by \( K \) to account for the sum over all filters, which simplifies since \( SK \) is\(^2\) \( O(R) \).

### 4.3 Translation to a Single Level

Before discussing the other widely used approach to calculating the gradient of the upper-level loss, we summarize a specialized approach for 1-norm regularizers. Like the minimizer approach described above, this approach assumes we have computed an (almost) exact minimizer of the lower-level cost function. It writes the minimizer as an (almost everywhere) differentiable function in terms of that \( \hat{x} \), then substitutes this expression into the upper-level loss to create a single-level optimization problem that is suitable for one hyperparameter update step.

Ref. [108] proposed the translation to a single-level approach to solve a bilevel problem with both synthesis and analysis operators. McCann and Ravishankar [109] recently presented a version specific to analysis operators. Here, we summarize the approach in [109] that avoids an auxiliary variable from splitting the one-norm sparsity penalty into two terms in [108].

\(^2\)The full parameter dimension includes the filters and tuning parameters, so \( R = S(K + 1) + 1 \).
The specific bilevel problem considered in [109] is:

\[
\begin{align*}
\argmin_{\gamma} \sum_{j} \frac{1}{2} \| \hat{x}_j(\gamma) - x_{j}^{\text{true}} \|_2^2 \\
\hat{x}_j(\gamma) = \argmin_x \frac{1}{2} \| Ax - y_j \|_2^2 + \beta \| \Omega x \|_1,
\end{align*}
\]

where \( \Omega_\gamma \) is a matrix constructed based on \( \gamma \), e.g., as a convolutional matrix as in our running example. (In the results, [109] parameterizes \( \Omega_\gamma \) to have \( 3 \times 3 \) convolutional filters as the learnable parameters.) We write \( \Omega \) in the following discussion to simplify notation.

Eqn. (4.12) slightly generalizes [109] by considering an inverse problem, rather than just denoising where \( A = I \).

As in the minimizer approach, the first step is to compute \( \hat{x}(\gamma) \) for the current guess of \( \gamma \), e.g., using a fast proximal gradient method [110] or ADMM (as used in [109]). One can then re-write the 1-norm in (4.12) based on the sign pattern of \( \Omega \hat{x}(\gamma) \) since

\[
\| \Omega \hat{x}(\gamma) \|_1 = \sum_{i \in \mathcal{I}_+(\gamma)} [\Omega \hat{x}(\gamma)]_i - \sum_{i \in \mathcal{I}_-(\gamma)} [\Omega \hat{x}(\gamma)]_i,
\]

where \( \mathcal{I}_+(\gamma) \) and \( \mathcal{I}_-(\gamma) \) denote the set of indices where \( \Omega \hat{x}(\gamma) \) is positive and negative, respectively. Ref. [109] defines a diagonal sign matrix, \( S(\gamma) = \text{diag} (\text{sign} (\Omega \hat{x}(\gamma))) \), having positive and negative diagonal elements at the appropriate indices. The lower-level problem (4.12) is thus equivalent to

\[
\hat{x}_j(\gamma) = \argmin_x \frac{1}{2} \| Ax - y_j \|_2^2 + \beta' \nu \| \Omega x \|_1, \quad \text{s.t.} \quad [\Omega x]_{\mathcal{I}_0(\gamma)} = 0,
\]

where \( \mathcal{I}_0(\gamma) \) denotes the set of indices where \( [\Omega \hat{x}(\gamma)]_i = 0 \).

The solution to the rewritten optimization problem (4.13) is the same as the original version (4.12) with the 1-norm. (The rewriting process requires \( \hat{x}(\gamma) \), so one cannot use the equivalence to solve the lower-level problem.) However, the rewritten problem has a simple structure: it is a quadratic cost function with a linear equality constraint. Therefore, using KKT conditions, if \( A' A \) is positive definite, then the minimizer of (4.13) is the solution to a linear system of equations

\[
\begin{bmatrix}
A' A & \Omega_0'(\gamma) \\
\Omega_0(\gamma) & 0
\end{bmatrix}
\begin{bmatrix}
\hat{x}(\gamma) \\
\nu
\end{bmatrix}
= \begin{bmatrix}
A' y - \beta \Omega' S(\gamma) 1 \\
0
\end{bmatrix},
\]

where \( \Omega_0 \) denotes the rows of \( \Omega \) corresponding to the indices \( \mathcal{I}_0(\gamma) \) and \( \nu \) is a vector of Lagrange multipliers. See [111, Sect. 10.1.1] for discussion of existence and uniqueness of solutions to this linear system.

Because of the discontinuity of the sign function in (4.13), \( \hat{x}(\gamma) \) is not everywhere differentiable in \( \gamma \). However, [109] states that \( \hat{x}(\gamma) \) is differentiable everywhere except a
set of measure zero when $A = I$ and when $\Omega_0$ has full row rank. Then [109] uses matrix calculus to derive a gradient expression for $\hat{x}(\gamma)$ that is suitable for standard automatic differentiation software tools.

In summary, the translation to a single level approach involves computing $\hat{x}$, creating a closed-form expression for $\hat{x}$, and then differentiating the closed-form expression to compute the desired gradient, $\nabla_{\gamma} \hat{x}(\gamma)$. In terms of computation, the approach requires optimizing the lower-level cost and then two calls to a CG method as part of calculating the gradient, making the cost similar to the minimizer approach.

### 4.4 Unrolled Approaches

Another popular approach to finding $\nabla_{\gamma} \hat{x}(\gamma)$ is to assume that the lower-level cost function is approximately minimized by applying $T$ iterations of some (sub)differentiable optimization algorithm, where we write the update step at iteration $t \in [1 \ldots T]$ as

$$x^{(t)} = \Psi(x^{(t-1)}; \gamma),$$

for some mapping $\Psi : \mathbb{F}^N \to \mathbb{F}^N$ that should have the fixed-point property $\Psi(\hat{x}(\gamma); \gamma) = \hat{x}(\gamma)$. For example, GD has $\Psi(x; \gamma) = x - \alpha \nabla \Phi(x; \gamma)$ for some step size $\alpha$. (Algorithms with momentum depend on earlier iterates like $x^{(t-2)}$ as well, but we omit that dependence to simplify notation.) In contrast to the two approaches described above, this “unrolled” approach no longer assumes the solution to the lower-level problem is an exact minimizer. Instead, the unrolled approach reformulates the bilevel problem (LL) as

$$\arg\min_{\gamma} \ell(\gamma; x^{(T)}(\gamma)) \quad \text{s.t.} \quad x^{(t)}(\gamma) = \Psi(x^{(t-1)}; \gamma), \quad \forall t \in [1 \ldots T],$$

where $x^{(0)}$ is an initialization, e.g., the pseudo-inverse of $A$ times $y$. One can then take the (sub)gradient of a finite number $T$ of iterations of $\Psi$, hoping that $x^T$ approximately minimizes the lower-level function, $\Phi$.

The chain rule for derivatives is the foundation of the unrolled method. The gradient of interest, $\nabla \ell(\gamma)$, depends on the gradient of the optimization algorithm step with respect to $x$ and $\gamma$. For readability, define the following gradient matrices for the $t$th unrolled iteration

$$H_t := \nabla_x \Psi(x^{(t-1)}; \gamma) \in \mathbb{F}^{N \times N} \quad \text{and} \quad J_t := \nabla_{\gamma} \Psi(x^{(t-1)}; \gamma) \in \mathbb{F}^{N \times R},$$

for $t \in [1, T]$. We use these letters because, when using gradient descent as the optimization algorithm, $\nabla_x \Psi(x; \gamma)$ is closely related to the Hessian of $\Phi$ and $\nabla_{\gamma} \Psi(x; \gamma)$ is the Jacobian of the gradient. Thus, when $\Psi$ corresponds to GD, an unrolled approach involves computing the same quantities as required by the IFT approach (4.8).
By the chain rule, the gradient of (4.14) is

$$\nabla \ell(\gamma) = \nabla \gamma \ell(\gamma; x^{(T)}) + \left( \sum_{t=1}^{T} (H_T \cdots H_{t+1}) J_t \right)' \nabla_x \ell(\gamma; x^{(T)}) \in \mathbb{F}^R. \tag{4.15}$$

One can derive this gradient expression using a reverse or forward perspective, with parallels to back-propagation through time and real-time recurrent learning respectively [112]. We present the update here in terms of $x$ for simplicity, but the idea easily extends to updates in terms of a state vector that allows one to include momentum terms, weights, and other accessory variables in $\gamma$ [112].

Unlike the minimizer approach, where the goal is to run the lower-level optimization until (close to) convergence, most unrolling methods set $T$ in advance. This set number of lower-level iterations mimics the set depth of CNNs and allows a precise estimate of how much computational effort each lower-level optimization takes. However, if $T$ is relatively small, the lower-level optimization may not near convergence, and the unrolled method is only loosely tied to the original bilevel optimization problem. To maintain a strong connection to the bilevel problem while avoiding setting $T$ larger than necessary for convergence, [113] used a convergence criterion to determine the number of $\Psi$ iterations rather than pre-specifying a number of iterations. Section 7.1 further discusses the relation between bilevel problems and unrolling methods in the broader literature [114].

### 4.4.1 Computational and Memory Complexity

To compare the forward and reverse approaches to gradient computation for unrolled methods, we introduce notation for an ordered product of matrices. We indicate the arrangement of the multiplications by the set endpoints, $s \in [s_1 \leftrightarrow s_2]$ with the left endpoint, $s_1$, corresponding to the index for the left-most matrix in the product and the right endpoint, $s_2$, corresponding to the right-most matrix. Thus, for any sequence of square matrices $\{A\}$:

$$\prod_{s \in [t \leftrightarrow T]} A_s := A_t A_{t+1} \cdots A_T = (A'_T A'_{T-1} \cdots A'_t)' = \left( \prod_{s \in [T \leftrightarrow t]} A'_s \right)'.$$  

The above double arrow notation does not indicate order of operations. In the following notation the arrow direction does not affect the product result (ignoring finite precision effects), but rather signifies the direction (order) of calculation:

$$\prod_{s \in [T-t]} A_s := A_T (A_{T-1} \cdots (A_{t+1}(A_t)))$$

$$\prod_{s \in [T-t]} A_s := ((A_T A_{T-1}) \cdots) A_{t+1}) A_t.$$
We use a similar arrow notation to denote the order that terms are computed for sums; as above, the order is only important for computational considerations and does not affect the final result.

Figure 4.1: Reverse mode computation of the unrolled gradient from (4.1). The first gradient computation requires \( x(T) \), so all computations occur after the lower-level optimization algorithm is complete. The final gradient is \( \nabla \ell(\gamma) = \nabla \gamma \ell(\gamma; x(T)) + r \).

Using this notation, the reverse gradient calculation of (4.15) is

\[
\nabla \gamma \ell(\gamma; x(T)) + \sum_{t \in [T-1]} J_t' \left( \prod_{s \in [(t+1) \leftarrow T]} H_s' \right) \nabla x \ell(\gamma; x(T)). \tag{4.16}
\]

This expression requires \( \prod_{s \in [(T+1) \leftarrow T]} H_s' = I \), because \( H_{T+1} \) is not defined. For example, for \( T = 3 \), we have

\[
\nabla \gamma \ell(\gamma; x^{(3)}) + \underbrace{J_3'(I)g}_t + \underbrace{J_2'(H_3')g}_t + \underbrace{J_1'(H_2' H_3')g}_t,
\]

where \( g \) is shorthand for \( \nabla x \ell(\gamma; x(T)) \) here. This version is called reverse as all computations (arrows) begin at the end, \( T \).

The primary benefit of the reverse mode comes from the ability to group \( \nabla x \ell(\gamma; x(T)) \) with the right-most \( H_T \), such that all products are matrix-vector products, as seen in Fig. 4.1 Further, one can save the matrix-vector products for use during the next iteration.
and avoid duplicating the computation. Continuing the example for $T = 3$, we have

$$\nabla_\gamma \ell(\gamma; x^{(3)}) + \sum_{t=1}^{T-1} \left( \prod_{s \in [T \leftarrow (t+1)]} H_s \right) J_t' \nabla_x \ell(\gamma; x^{(T)}).$$

where one only needs to compute $\Delta$ once. This ability to rearrange the parenthesis to compute matrix-vector products greatly decreases the computational requirement compared to matrix-matrix products. Excluding the costs of the optimization algorithm steps and forming the $H_s$ and $J_t$ matrices (these costs will be the same in the forward mode computation), reverse mode requires $O(T)$ Hessian-vector multiplies and $O(TNR)$ additional multiplies. The trade-off is that reverse mode requires storing all $T$ iterates, $x^{(t)}$, so that one can compute the corresponding Hessians and Jacobians from them as needed, and thus has a memory complexity $O(TN)$.

Figure 4.2: Forward mode computation of the unrolled gradient from (4.17). The intermediate computation matrix, $Z$, is initialized to zero ($Z_0 = 0$) then updated every iteration. The final gradient is $\nabla \ell(\gamma) = \nabla_\gamma \ell(\gamma; x^{(T)}) + Z_T' \nabla_x \ell(\gamma; x^{(T)}).$

The forward mode calculation of (4.15), depicted in Fig. 4.2, has all computations (arrows) starting at the earlier iterate:

$$\nabla_\gamma \ell(\gamma; x^{(T)}) + \sum_{t \in [1 \rightarrow T]} \left( \prod_{s \in [T \leftarrow (t+1)]} H_s \right) J_t' \nabla_x \ell(\gamma; x^{(T)}).$$

As before, $H_{T+1}$ is not defined, so we take $\prod_{s \in [T \leftarrow (T+1)]} H_s = I$. For example, for $T = 3$ we have

$$\nabla_\gamma \ell(\gamma; x^{(T)}) + \left( (H_3 H_2) J_1 \right)' + \left( (H_3) J_2 \right)' + \left( (I) J_3 \right)' \cdot g.$$ 

How the forward mode avoids storing $x$ iterates is evident after rearranging the parenthesis to avoid duplicate calculations, as illustrated in Fig. 4.2. Continuing the example for $T = 3,$
we have
\[ \nabla_{\gamma} \ell(\gamma; x^{(T)}) + \left[ H_3 \left( \begin{array}{c} Z_2 \\ H_2 (H_1 \cdot 0 + J_1) + J_2 \\ Z_1 \end{array} \right) + J_3 \right] g, \]
where \( Z_s = H_s Z_{s-1} + J_s \in \mathbb{R}^{N \times R} \) stores the intermediate calculations. The above formula also illustrates why \( H_1 \) is not needed in (4.14); \( \nabla_{\gamma} x^{(0)} = 0 \) is the last element from applying the chain rule.

There is no way to rearrange the terms in the forward mode formula to achieve matrix-vector products (while preserving the computation order). Therefore, the computation requirement is much higher at \( O(TR) \) Hessian-vector multiplications. The corresponding benefit of the forward mode method is that it does not require storing iterates, thus decreasing (in the common case when \( T > R \)) the memory requirement to \( O(NR) \) for storing the intermediate matrix \( Z_s \) during calculation.

As with the IFT/Lagrangian approach, the computational complexity of the unrolled approach is lower than the generic bound when we consider the specific example of learning convolutional filters according to (Ex). Nevertheless, the general comparison that reverse mode takes more memory but less computation holds true. See Tab. 4.1 for a full comparison of the computational and memory complexities.

Neural network training methods commonly use the reverse method, backpropagation, to compute (4.15) because it requires less computation than forward mode. However, in some problems with large dimensions, storing all the iterates is infeasible. A strategy to trade-off the memory and computation requirements is checkpointing, which stores \( x \) every few iterations. Checkpointing is an active research area; see [115] for an overview.

### 4.5 Summary

This section focused on computing \( \nabla \ell(\gamma) \), the gradient of the upper-level loss function with respect to the learnable parameters. Section 5 builds on this foundation to consider optimization methods for bilevel problems. Many of those optimization methods are agnostic of if one uses the minimizer, translation to a single level, or unrolled approaches to compute \( \nabla \ell(\gamma) \). Thus, how one selects an approach may depend on the structure of the specific bilevel problem, how closely tied one wishes to be to the original bilevel problem, computational cost, and/or gradient accuracy.

The translation to a single level approach is tailored to a specific bilevel problem. A benefit of the translation approach is the ability to use the 1-norm (without any corner rounding) in the lower-level cost function. However, the corresponding drawback is the
lack of generality in the minimizer approach; the closed-form expression derived in [108], [109] is specific to using the 1-norm as φ and applications where A has full column rank, like denoising problems, rather than inverse problems, like compressed sensing, where A is wide. Expanding this approach to regularizers other than the 1-norm is a possible avenue for future work.

One difference among the methods is whether they depend on the lower-level optimization algorithm; while the unrolled approach depends on the specific optimization algorithm, the minimizer approach and the translation to a single level approach do not. A resulting downside of unrolling is that one cannot use techniques such as warm starts and non-differentiable restarts, so \( x^{(T)} \) may be farther from the minimizer than the approximation from a similar number of iterations of a more sophisticated, non-differentiable update method. However, the unrolled method’s dependence on \( \Psi \) is also a benefit, as an unrolled method can be applied to non-smooth cost functions, as long as the resulting update mapping \( \Psi \) is smooth.

Another advantage of unrolling is that one can run a set number of iterations of the optimization algorithm, without having to reach convergence, and still calculate a valid gradient, thus avoiding the issue of the second constraint for the minimizer approach. Particularly in image reconstruction problems, where finding \( \hat{x} \) exactly can be time intensive, the benefit of a more flexible run-time could outweigh the disadvantages. However, the corresponding downside of unrolling is that the learned hyperparameters are less clearly tied to the original cost function than when one uses the minimizer approach. Section 7.1 further discusses this point in connection to how unrolling for bilevel methods can differ from (deep) learnable optimization algorithms.

Gradient accuracy and computational cost are, unsurprisingly, trade-offs. Tab. 4.1 summarizes the cost of the minimizer and unrolled approaches, derived in Section 4.2.4 and Section 4.4.1 respectively, but the total computation will depend on the required gradient accuracy. By accuracy, we mean error from the true bilevel gradient

\[
\| \tilde{\nabla}_T \ell(\gamma) - \nabla \ell(\gamma) \|,
\]

where \( T \) is the number of lower-level optimization steps.

The unrolled gradient is always accurate for the unrolled mapping, but not for the original bilevel problem. Therefore, unrolling may be more computationally feasible when one cannot run a sufficient number of lower-level optimization steps to reach close enough to a minimizer to assume the gradient is approximately zero [23]. When using the unrolling approach, one must decide between the forward-mode and reverse-mode gradient calculation. Most approaches use the reverse-mode due to the smaller computational burden, but unrolling with reverse mode differentiation may have prohibitively high memory requirements if
Table 4.1: Memory and computational complexity of the minimizer approach (4.8), reverse-mode unrolled approach (4.16), and forward-mode unrolled approach (4.17) to computing $\nabla_\gamma \ell(\gamma; \hat{x}(\gamma))$. Computational costs do not include running the optimization algorithm (typically expensive but often comparable across methods), computing $\nabla_x \ell(\gamma; x(T))$ (typically cheap), or computing $\nabla_\gamma \ell(\gamma; x)$ (frequently zero). Memory requirements do not include storing a single copy of $x$, $A$, $\gamma$, $H$, and $J$, because all methods require those. Recall $x \in \mathbb{F}^N$, $\gamma \in \mathbb{F}^R$, and there are $T$ iterations of the lower-level optimization algorithm for the unrolled method. Hessian-vector products (first row) and Hessian-inverse-vector products (middle row) are listed separately from all other multiplications (last row) as the computational cost of Hessian operations can vary widely; see discussion in Section 4.2.4.

|                  | Minimizer | Unrolled: reverse | Unrolled: forward |
|------------------|-----------|-------------------|-------------------|
| Memory           | $0$       | $\mathcal{O}(TN)$ | $\mathcal{O}(NR)$|
| Hessian-vector products | $0$     | $\mathcal{O}(T)$   | $\mathcal{O}(TR)$   |
| Hessian-inverse vector products | $1$     | $0$               | $0$               |
| Other multiplications | $NR$   | $\mathcal{O}(TNR)$ | $\mathcal{O}(NR)$ |

In all of the approaches considered, the accuracy of the estimated hyperparameter gradient in turn depends on the solution accuracy or number of unrolled iterations of the lower-level cost function. Ref. [109] notes that their translation to a single level approach failed if they did not optimize the lower-level problem to a sufficient accuracy level. However, [108], [109] did not investigate how the solution accuracy of the lower-level problem impacts the gradient estimation.

For the minimizer and unrolled approaches, [105], [106] found that the gradient estimate from the minimizer approach converges to the true gradient faster than the unrolled approach (in terms of computation). To state the bounds, [105], [106] assert conditions on the structure of the bilevel problem. They assume that $\hat{x}(\gamma)$ is the unique minimizer of the lower-level cost function, the Hessian of the lower-level is invertible, the Hessian and Jacobian of $\Phi$ are Lipschitz continuous with respect to $x$, the gradients of the upper-level loss are Lipschitz continuous with respect to $x$, the norm of $x$ is bounded, and the lower-level cost is strongly convex and Lipschitz smooth for every $\gamma$ value. Section 5.1.1 discusses similar investigations that use these conditions, how easy or hard they are to satisfy, and how they apply to (Ex).

Ref. [106] initializes the lower-level iterates for both the unrolled and minimizer approach with the zero vector, $i.e.$, $x^{(0)} = 0$. Under their assumptions, [106] prove that both the unrolled and minimizer gradients converge linearly in the number of lower-level iterations when the lower-level optimization algorithm and conjugate gradient algorithm for the minimizer approach converge linearly. Although the rate of the approaches is the same, the
minimizer approach converges at a faster linear rate and [106] generally recommends the
minimizer approach, though they found empirically that the unrolled approach may be more
reliable when the strong convexity and Lipschitz smooth assumptions on the lower-level
cost do not hold.

Ref. [105] extended the analysis from [106] to consider a warm start initialization for
the lower-level optimization algorithm. They similarly find that the minimizer approach
has a lower complexity than the unrolled approach. Sections 5.1.1 and 5.2.1 further discuss
complexity results after introducing specific bilevel optimization algorithms.

5 Gradient-Based Bilevel Optimization Methods

The previous section discussed different approaches to finding $\nabla \ell(\gamma)$, the gradient of the
upper-level loss function with respect to the learnable parameters. Building on those results,
this section considers approaches for optimizing the bilevel problem. In particular, this
section concentrates on gradient-based algorithms for optimizing the hyperparameters.
While there is some overlap with single-level optimization methods, this section focuses
on the challenges due to the bilevel structure. Therefore, this section does not discuss the
lower-level optimization algorithms in detail; for overviews of lower-level optimization, see,
*e.g.*, [117].

Gradient-based methods for bilevel problems are an alternative to the approaches
described in Section 3, *e.g.*, grid or random search, Bayesian optimization, and trust
region methods. By incorporating gradient information, the methods presented in this
section can scale to problems having many hyperparameters. In fact, the second section
reviews papers that provide bounds on the number of upper-level gradient descent iterations
required to reach a point within some user-defined tolerance of a solution. While the bounds
depend on the regularity of the upper-level loss and lower-level cost functions, they do not
depend on the number of hyperparameters nor the signal dimension. Although having more
hyperparameters will increase computation per iteration, using a gradient-descent approach
means the number of iterations need not scale with the number of hyperparameters, $R$.

Bilevel gradient methods fall into two broad categories. Most gradient-based approaches
to the bilevel problem fall under the first category: double-loop algorithms. These methods
involve (i) optimizing the lower-level cost, either to some convergence tolerance if using a
minimizer approach or for a certain number of iterations if using an unrolled approach, (ii)
calculating $\nabla \ell(\gamma)$, (iii) taking a gradient step in $\gamma$, and (iv) iterating. The first step is itself
an optimization algorithm and may involve many inner iterations, thus the categorization
as a “double-loop algorithm.”

The second category, “single-loop” algorithms, involve one loop, with each iteration
containing one gradient step for both the lower-level optimization variable, $\mathbf{x}$, and the
upper-level optimization variable, $\gamma$. Single-loop algorithms may alternate updates or update the variables simultaneously. Section 4 used $t$ to denote the lower-level iteration counter; this section introduces $u$ as the iteration counter for the upper-level iterations and as the single iteration counter for single-loop algorithms.

## 5.1 Double-Loop Algorithms

After using one of the approaches in Section 4 to compute the hyperparameter gradient $\nabla \ell(\gamma)$, typical double-loop algorithms for bilevel problems run some type of gradient descent on the upper-level loss. Alg. 1 shows an example double-loop algorithm [118].

The final iterate of a lower-level optimizer is only an approximation of the lower-level minimizer. However, the minimizer approach to calculating the upper-level gradient, $\nabla \ell(\gamma)$ (Section 4.2), assumes $\nabla_x \Phi(\hat{x} ; \gamma) = 0$. Any error stemming from not being at an exact critical point can be magnified in the full calculation (4.8), and the resulting hyperparameter gradient will be an approximation of the true gradient, as illustrated in Fig. 5.1. Alternatively, if one uses the unrolled approach with a set number of iterations (4.14), the gradient is accurate for that specific number of iterations, but the lower-level optimization sequence may not have converged and the overall method may not accurately approximate the original bilevel problem.

Due to such inevitable inexactness when computing $\nabla \ell(\gamma)$, one may wonder about the convergence of double-loop algorithms for bilevel problems. Considering the unrolled method of computing $\nabla \ell(\gamma)$, [119] showed that the sequence of hyperparameter values in

**Figure 5.1:** Error in the upper-level gradient, $\nabla \ell(\gamma)$, for various convergence thresholds for the lower-level optimizer. The bilevel problem is (Ex) with a single filter, $c = [c_0 \ c_1]$, $e^{\beta_0} = 0$, $e^{\beta_1} = -5$, and $\phi(z) = z^2$ so there is an analytic solution for $\nabla \ell(\gamma)$. The training data is piece-wise constant 1d signals and the learnable hyperparameters are the filter coefficients. (a) Upper-level loss function, $\ell(\gamma)$. The cost function is low (dark) where $c_1 \approx c_0$, corresponding to approximate finite differences. The star indicates the minimum. (b-d) Error in the estimated gradient angle using the minimizer approach (4.8), defined as the angle between $\hat{\nabla} \ell(\gamma)$ and $\nabla \ell(\gamma)$, when the lower-level optimization is run until $\|\nabla \Phi(x ; \gamma)\| < \epsilon$. 


Algorithm 1: Hyperparameter optimization with approximate gradient (HOAG) from [118]. As written below, the HOAG algorithm is impractical because it uses \( \hat{x}(\gamma^{(u)}) \) in the convergence criteria; however, for strongly convex lower-level problems, the convergence criteria, \( \| \hat{x}(\gamma^{(u)}) - x^{(t)}(\gamma^{(u)}) \| \), is easily upper-bounded.

Pedregosa [118] proved a similar result for the minimizer formula (4.5) using CG to compute (4.10). Specifically, [118] showed that the hyperparameter sequence converges to a stationary point if the sequence of positive tolerances, \( \{\epsilon(u), u = 1, 2, \ldots\} \) in Alg. 1, is summable. The convergence results are for the algorithm version shown in Alg. 1 that uses a Lipschitz constant of \( \nabla \ell(\gamma) \), which is generally unknown. Although [118] discusses various empirical strategies for setting the step size, the convergence theory does not consider those variations. Thus, the double-loop algorithm [118] requires multiple design decisions.

There are four key design decisions for double-loop algorithms:

1. How accurately should one solve the lower-level problem?
2. What upper-level gradient descent algorithm should one use?
3. How does one pick the step size for the upper-level descent step?

4. What stopping criteria should one use for the upper-level iterations?

This section first reviews some (largely heuristic) approaches to these design decisions and presents example bilevel gradient descent methods with no (or few) assumptions beyond those made in Section 4. Without any further assumptions, the answers to the questions above are based on heuristics, with few theoretical guarantees but often providing good experimental results. The following section discusses recent methods with stricter assumptions on the bilevel problem and their theory-backed answers to the above questions.

The first step in a double-loop algorithm is to optimize the lower-level cost, for which there are many optimization approaches. The only restriction is computability of the gradient of the upper-level loss, \( \nabla \ell(\gamma) \), which typically includes a smoothness assumption (see Section 4 for discussion). Many bilevel methods papers use a standard optimizer for the lower-level problem, although others propose new variants, e.g., [22].

The first design decision (how accurately to solve the lower-level problem) involves a trade-off between computational complexity and accuracy. Example convergence criteria are fairly standard to the optimization literature, e.g., the Euclidean norm of the lower-level gradient [42], [120] or the normalized change in the estimate \( \hat{x} \) [121] being less than some threshold. For example, [42] used a convergence criteria of \( \| \nabla_x \Phi(x^{(t)}; \gamma) \|_2 \leq 10^{-3} \) (where the image scale is 0-255). As mentioned above, [118] uses a sequence of convergence tolerances so that the lower-level cost function is optimized more accurately as the upper-level iterations continue.

Many publications do not report a specific threshold or discuss how they chose a convergence criteria or number of lower-level iterations. However, a few note the importance of such decisions. For example, [109] found that their learning method fails if the lower-level optimizer is insufficiently close to the minimizer and [42] stated their results are “significantly better” than [47] because they solve the lower-level problem “with high[er] accuracy.”

After selecting a level of accuracy, finding (an approximation of) \( \hat{x} \), and calculating \( \nabla_\gamma \ell(\gamma; \hat{x}(\gamma)) \) using one of the approaches from Section 4, one must make the second design decision: which gradient-based method to use for the upper-level problem. Many bilevel methods suggest a simple gradient-based method such as plain gradient descent (GD) [20], GD with a line search (see the third design decision), projected GD [113], or stochastic GD [109]. These methods update \( \gamma \) based on only the current upper-level gradient; they do not have memory of previous gradients nor require/estimate any second-order information.

Methods that incorporate some second-order information use more memory and computation per iteration, but may converge faster than basic GD methods. For example, Broyden-Fletcher-Goldfarb-Shanno (BFGS) and L-BFGS (the low-memory version of BFGS) [122] are quasi-Newton algorithms that store and update an approximate Hessian matrix that serves as a preconditioner for the gradient. The \( R \times R \) size of the Hessian grows as the
number hyperparameters increases, but quasi-Newton methods like L-BFGS use practical rank-1 updates with storage $O(R)$. Adam [123] is a popular GD method, especially in the machine learning community, that tailors the step size (equivalently the learning rate) for each hyperparameter based on moments of the gradient. Although Adam requires its own parameters, the parameters are relatively easy to set and the default settings often perform adequately. Example bilevel papers using methods with second-order information include those that use BFGS [19], L-BFGS [42], Gauss-Newton [124], and Adam [22].

Many gradient-based methods require selecting a step size parameter, e.g., one must choose a step size $\alpha$ in classical GD:

$$\gamma^{(u+1)} = \gamma^{(u)} - \alpha \nabla \ell \left( \gamma^{(u)} \right).$$

This choice is the third design decision. Bilevel problems are generally non-convex, and typically a Lipschitz constant is unavailable, so line search strategies initially appear appealing. However, any line search strategy that involves attempting multiple values quickly becomes computationally intractable for large-scale problems. The upper-level loss function in bilevel problems is particularly expensive to evaluate because it requires optimizing the lower-level cost! Further, recall that the upper-level loss is typically an expectation over multiple training samples (UL), so evaluating a single option for $\alpha$ involves optimizing the lower-level cost $J$ times (or using a stochastic approach and selecting a batch size).

Despite these challenges, a line search strategy may be viable if it rarely requires multiple attempts. For example, the backtracking line search in [120] that used the Armijo–Goldstein condition required 57-59 lower-level evaluations (per training example) over 40 upper-level gradient descent steps, so most upper-level steps required only one lower-level evaluation. Other bilevel papers that used backtracking with Armijo-type conditions include [10], [19], [121], whereas [125] used the Barzilai-Borwein method for picking an adaptive step size.

Other approaches to determining the step size are: (i) normalize the gradient by the dimension of the data and pick a fixed step size [109], (ii) pick a value that is small enough based on experience [20], or (iii) adapt the step size based on the decrease from the previous iteration [118].

The fourth design decision needed is the convergence criteria for the upper-level loss. As with the lower-level convergence criteria, few publications include a specific threshold, but most bilevel methods tend to use traditional convergence criteria such as the norm of the hyperparameter gradient falling below some threshold [19], the norm of the change in parameters falling below some threshold [42], and/or reaching a maximum iteration count (many papers). One specific example is to terminate when the normalized change in learned parameters, $\| \gamma^{(u+1)} - \gamma^{(u)} \| / \| \gamma^{(u)} \|$, is below 0.01 [121]. The normalized change bound is convenient because it is unitless and thus invariant to scaling of $\gamma$. 50
5.1.1 Complexity Analysis

A series of recent papers established finite-time sample complexity bounds for stochastic bilevel optimization methods based on gradient descent for the upper-level loss and lower-level cost. Whereas [105], [126] use double-loop approaches; [127], [128] use single-loop algorithms (see Section 5.2.1). Unlike most of the methods discussed in the previous section, these papers make additional assumptions about the upper and lower-level functions then chose the upper and lower-level step sizes to ensure convergence.

In these works, “finite-time sample complexity” refers to big-O bounds on a number of iterations that ensures one reaches a minimizer to within some desired tolerance. In contrast to asymptotic convergence analysis, finite-time bounds provide information about the estimated hyperparameters, \( \gamma^{(a)} \), after a finite number of upper-level iterations. These bounds depend on problem-specific quantities, such as Lipschitz constants, but not on the hyperparameter or signal dimensions.

To summarize the results, this section returns to the notation from the introduction where the upper-level loss may be deterministic or stochastic, e.g., the bilevel problem is

\[
\hat{\gamma} = \arg\min_{\gamma} \ell(\gamma) \quad \text{with} \quad \ell(\gamma) = \begin{cases} 
\ell(\gamma, \hat{x}(\gamma)) & \text{deterministic} \\
\mathbb{E}[\ell(\gamma, \hat{x}(\gamma))] & \text{stochastic.}
\end{cases}
\]  

(5.1)

This section reviews results for double-loop methods [105], [126] for the deterministic setting and then the stochastic case.

The expectation in (5.1) can have different meanings depending on the setting. When one has \( J \) training images with one noise realization per image, one often picks a random subset (“minibatch”) of those \( J \) images for each update of \( \gamma \), corresponding to stochastic gradient descent of the upper-level loss. In this setting, the randomness is a property of the algorithm, not of the upper-level loss, and the expectation reduces to the deterministic case. An interesting opportunity for a stochastic formulation is to add different noise realizations in (1.3), providing an uncountable ensemble of \((x, y)\) training tuples, where the expectation above is over the distribution of noise realizations. Yet another possibility is to have a truly random set of training images \( x^{\text{true}} \) drawn from some distribution. For example, [129] trained a CNN-based CT reconstruction method using an ensemble of images consisting of randomly generated ellipses. Other variations, such as random rotations or warps, have also been used for data augmentation [130]. One could combine such a random ensemble of images with a random ensemble of noise realizations, in which case the expectation in (5.1) would be taken over both the image and noise distributions. We are unaware of any bilevel methods for imaging that exploit this full generality. Future literature on stochastic methods should clearly state what expectation is used and may consider exploiting a more general definition of randomness.

The complexity results (summarized in Tab. 5.2 and Tab. 5.3) are all in terms of finding
\(\gamma_\epsilon\), defined as an \(\epsilon\)-optimal solution. In the (atypical) setting where \(\ell(\gamma)\) is convex, \(\gamma_\epsilon\) is an \(\epsilon\)-optimal solution if it satisfies either \(\ell(\gamma_\epsilon) - \ell(\hat{\gamma}) \leq \epsilon\) [105], [126], [127] or \(||\gamma - \gamma_\epsilon||^2 \leq \epsilon\) [128]. (These conditions are equivalent if \(\ell\) is strongly convex in \(\gamma\), but can differ otherwise.)

In the (common) non-convex setting, \(\gamma_\epsilon\) is typically called an \(\epsilon\)-stationary point if it satisfies \(||\nabla \ell(\gamma_\epsilon)||^2 \leq \epsilon\) [105], [126], [128]. In the stochastic setting, \(\gamma_\epsilon\) must satisfy these conditions in expectation.

References [105], [126]–[128] all make similar assumptions about \(\ell\) and \(\Phi\) to derive theoretical results for their proposed bilevel optimization methods. We first summarize the set of assumptions from [126], and later note any additional assumptions used by the other theoretical results for their proposed bilevel optimization methods. We first summarize the

1. \(\forall \gamma \in \mathbb{F}^m, \nabla_\gamma \ell(\gamma; x)\) and \(\nabla_x \ell(\gamma; x)\) are Lipschitz continuous with respect to \(x\), with corresponding Lipschitz constants \(L_{x, \nabla_\gamma \ell}\) and \(L_{x, \nabla_x \ell}\). (These constants are independent of \(x\) and \(\gamma\).)

2. The gradient with respect to \(x\) is bounded, i.e.,
   \(|\nabla_x \ell(\gamma^*, x)| \leq C_{\nabla_x \ell}, \forall x \in \mathbb{F}^N\).

3. \(\forall x \in \mathbb{F}^N, \nabla_x \ell(\gamma; x)\) is Lipschitz continuous with respect to \(\gamma\), with corresponding Lipschitz constant \(L_{\gamma, \nabla_x \ell}\).

The conditions in [126] on the lower-level function, \(\Phi(x; \gamma)\), are:

1. \(\Phi\) is continuously twice differentiable in \(\gamma\) and \(x\).

2. \(\forall \gamma \in \mathbb{F}^m, \nabla_x \Phi(x; \gamma)\) is Lipschitz continuous with respect to \(x\) with corresponding constant \(L_{x, \nabla_x \Phi}\).

3. \(\forall \gamma \in \mathbb{F}^m, \Phi(x; \gamma)\) is strongly convex with respect to \(x\), i.e., \(\mu_{x, \Phi} I \leq \nabla_x^2 \Phi(\gamma^*; x)\), for some \(\mu_{x, \Phi} > 0\).

4. \(\forall \gamma \in \mathbb{F}^m, \nabla_{xx} \Phi(x; \gamma)\) and \(\nabla_{yx} \Phi(x; \gamma)\) are Lipschitz continuous with respect to \(x\) with Lipschitz constants \(L_{x, \nabla_{xx} \Phi}\) and \(L_{x, \nabla_{yx} \Phi}\).

5. The mixed second gradient of \(\Phi\) is bounded, i.e.,
   \(|\nabla_{yx} \Phi(x; \gamma)| \leq C_{\nabla_{yx} \Phi}, \forall \gamma, x\).

6. \(\forall x \in \mathbb{F}^N, \nabla_{yx} \Phi(x; \gamma)\) and \(\nabla_{xx} \Phi(x; \gamma)\) are Lipschitz continuous with respect to \(\gamma\) with Lipschitz constants \(L_{\gamma, \nabla_{yx} \Phi}\) and \(L_{\gamma, \nabla_{xx} \Phi}\).

To consider how the complexity analysis bounds may apply in practice, Appendix A.2 examines how each of these conditions applies to the running filter learning example (Ex). Although a few of the conditions are easily satisfied, most are not. Appendix A.2 shows that the conditions are met if one invokes box constraints on the variables \(x\) and \(\gamma\). Although imposing box constraints requires modifying the algorithms, e.g., by including a projection
Algorithm 2: Bilevel Approximation (BA) Method from [126]. The differences for the AID-BiO and ITD-BiO methods from [105] are: (1) when $u > 0$, the BiO methods replace line 3 with $x^{(0)} = x^{(T_u - 1)}$, (2) $T_i$ does not vary with upper-level iteration, (3) the upper-level gradient calculation in line 7 can use the minimizer approach (4.8) or backpropagation (4.16), and (4) the hyperparameter update is standard gradient descent, so line 8 becomes $\gamma^{(i+1)} = \gamma^{(i)} - \alpha \ell g$.

Step, the iterates remain unchanged if the constraints are sufficiently generous. However, such generous box constraints are likely to yield large Lipschitz constants and bounds, leading to overly-conservative predicted convergence rates. Further, any differentiable upper-level loss and lower-level cost function would meet the conditions above with such box constraints. Generalizing the following complexity analysis for looser conditions is an important avenue for future work.

**Deterministic**

Ghadimi and Wang [126] were the first to provide a finite-time analysis of the bilevel problem. The authors proposed and analyzed the Bilevel Approximation (BA) method (see Alg. 2). BA uses two nested loops and is very similar to the gradient-based methods described in the previous section. The inner loop minimizes the lower-level cost to some accuracy, determined by the number of lower-level iterations; the more inner iterations, the more accurate the gradient will be, but at the cost of more computation and time. The outer loop is (inexact) projected gradient steps on $\ell$. Ref. [126] used the minimizer result (4.8) (with the IFT perspective for the derivation) to estimate the upper-level gradient.

To bound the complexity of BA, [126] first related the error in the lower-level solution to the error in the upper-level gradient estimate as

$$
\|\nabla_{\gamma} \ell(\gamma, x) - \nabla_{\gamma} \ell(\gamma, \hat{x}(\gamma))\| \leq C_{GW} \|x - \hat{x}(\gamma)\|,
$$

Estimated gradient \quad True gradient \quad Error in lower-level
where $C_{GW}$ is a constant that depends on many of the bounds defined in the assumptions above [126]. Combing the above error bound with known gradient descent bounds for the accuracy of the lower-level problem yields bounds on the accuracy of the upper-level gradient. The standard lower-level bounds can vary by the specific algorithm ([126] uses plain GD), but are in terms of $Q_\Phi = \frac{L_x \nabla_x \Phi}{\mu_x \Phi}$ (the “condition number” for the strongly convex lower-level function) and the distance between the initialization and the minimizer.

Ref. [126] shows that $\hat{x}(\gamma)$ is Lipschitz continuous in $\gamma$ under the above assumptions, which intuitively states that the lower-level minimizer does not change too rapidly with changes in the hyperparameters. Further, $\nabla \ell(\gamma)$ is Lipschitz continuous in $\gamma$ with a Lipschitz constant, $L_{\gamma, \nabla_{\gamma} \ell}$, that depends on many of the constants given above.

The main theorems from [126] hold when the lower-level GD step size is $\alpha_\Phi = \frac{2}{L_x \nabla_x \Phi + \mu_x \Phi}$ and the upper-level step size satisfies $\alpha_\ell \leq \frac{1}{L_{\gamma, \nabla_{\gamma} \ell}}$. Then, the distance between the $u$th loss function value and the minimum loss function value, $\ell(\gamma^{(u)}, \hat{x}(\gamma^{(u)}) - \ell(\hat{y}, \hat{x}(\hat{y}))$, is bounded by a constant that depends on the starting distance from a minimizer (dependent on the initialization of $\gamma$ and $x$), $Q_\Phi$, $C_{GW}$, the number of inner iterations, and the upper-level step size. The bound differs for strongly convex, convex, and possibly non-convex upper-level loss functions. Tab. 5.1 summarizes the sample complexity required to reach an $\epsilon$-optimal point in each of these scenarios.

| $\ell(\gamma)$ | Upper-level gradients | Lower-level gradients |
|----------------|----------------------|----------------------|
| Strongly convex | $\mathcal{O}\left(\log \left(\frac{1}{\epsilon}\right)\right)$ | $\mathcal{O}\left(\log^2 \left(\frac{1}{\epsilon}\right)\right)$ |
| Convex | $\mathcal{O}\left(\frac{1}{\sqrt{\epsilon}}\right)$ | $\mathcal{O}\left(\frac{1}{\sqrt[3/4]{\epsilon}}\right)$ |
| Non-convex | $\mathcal{O}\left(\frac{1}{\epsilon}\right)$ | $\mathcal{O}\left(\frac{1}{\sqrt[5/4]{\epsilon}}\right)$ |

Table 5.1: Sample complexity to reach an $\epsilon$-optimal solution of the deterministic bilevel problem using BA [126], for various assumptions on the upper-level loss function. Usually $\ell(\gamma)$ is non-convex and that case has the worst-case order results. The complexities show the total number of partial gradients of the upper-level loss (equal to the number of lower-level Hessians needed for estimating $\nabla \ell(\gamma)$ using (4.8)) and the partial gradients of the lower-level. The convex results use the accelerated BA method, which uses acceleration techniques similar to Nesterov’s method [131] applied to the upper-level gradient step in Alg. 2.

Ji, Yang, and Liang [105] proposed two methods for Bilevel Optimization that improve on the sample complexities from [126] for non-convex loss functions under similar assumptions. The first, ITD-BiO, uses the unrolled method for calculating the upper-level gradient (see Section 4.4). The second, AID-BiO, uses the minimizer method with the implicit function theory perspective (see Section 4.2). Tab. 5.2 summarizes the sample complexities [105]. Much of the computational advantage of ITD-BiO and AID-BiO is in improving the iteration complexity with respect to the condition number.
One of the main computational advantages of the AID-BiO and IFT-BiO methods in [105] over the BA algorithm Alg. 2 is a warm restart for the lower-level optimization. Although the hyperparameters change every outer iteration, the change is generally small enough that the stopping point of the previous lower-level descent is a better initialization than the noisy data (recall that [126] showed the lower-level minimizer is Lipschitz continuous in $\gamma$). One can account for this warm restart when using automatic differentiation tools (backpropagation) [105]. The caption for Alg. 2 summarizes the other differences between BA and the BiO methods.

|          | Upper-level gradients | Lower-level gradients | Hessian-vector products |
|----------|-----------------------|-----------------------|-------------------------|
| BA       | $\mathcal{O}(\kappa^{-1/\tau})$ | $\mathcal{O}(\kappa^{5/\tau})$ | $\mathcal{O}(\kappa^{-4.1/\tau})^*$ |
| AID-BiO  | $\mathcal{O}(\kappa^{-3/\tau})$ | $\mathcal{O}(\kappa^{-1/\tau})^*$ | $\mathcal{O}(\kappa^{-3.5/\tau})$ |
| ITD-BiO  | $\mathcal{O}(\kappa^{-3/\tau})$ | $\mathcal{O}(\kappa^{-1/\tau})^*$ | $\mathcal{O}(\kappa^{-4.1/\tau})$ |

Table 5.2: A comparison of the finite-time sample complexity to reach an $\epsilon$-solution of the deterministic bilevel problem when the upper-level loss function is non-convex using BA [126], AID-BiO [105], and ITD-BiO [105]. $^*$ = order omits any $\log(\epsilon)^{-1}$ term.

Stochastic

Discussing optimization algorithms for the stochastic bilevel problem requires additional assumptions about the stochastic estimates. The stochastic methods discussed here are all based on the minimizer approach to finding the upper-level gradient. Therefore, the methods require estimates for $\nabla_\gamma \ell(\gamma; x)$, $\nabla_x \ell(\gamma; x)$, $\nabla_\gamma \Phi(x; \gamma)$, $\nabla_{x,\gamma} \Phi(x; \gamma)$, and $\nabla_{x,\gamma} \Phi(x; \gamma)$. We denote the estimates of these gradient using hats, e.g., $\hat{\nabla}_\gamma \ell(\gamma; x)$. Following (4.8), an estimate of the upper-level gradient approximation is thus

$$\hat{\nabla} \ell(\gamma) = \hat{\nabla}_\gamma \ell(\gamma; x) - (\hat{\nabla}_{x\gamma} \Phi(x; \gamma))'(\hat{\nabla}_{xx} \Phi(x; \gamma))^{-1}\hat{\nabla}_x \ell(\gamma, x).$$

As in the deterministic case, we start with the assumptions and results from [126]. In addition to the assumptions above on $\ell$ (assumptions A$\ell$ 1-A$\ell$ 3) and $\Phi$ (assumptions A$\Phi$ 1-A$\Phi$6), the stochastic-specific assumptions are that (i) all estimated gradients are unbiased and (ii) the variance of the estimation errors is bounded by $\sigma_{\nabla_\gamma \ell}^2$, $\sigma_{\nabla_x \ell}^2$, $\sigma_{\nabla_\gamma \Phi}^2$, $\sigma_{\nabla_{x,\gamma} \Phi}^2$, and $\sigma_{\nabla_{x,\gamma} \Phi}^2$. For example, [126] assumes

$$\mathbb{E} \left[ \|\nabla_\gamma \ell(\gamma; x) - \hat{\nabla}_\gamma \ell(\gamma; x)\|^2 \right] \leq \sigma_{\nabla_\gamma \ell}^2 \quad \forall x, \gamma.$$

The Bilevel Stochastic Approximation (BSA) method replaces the lower-level update in BA (see Alg. 2) with standard stochastic gradient descent. The corresponding upper-level step in BSA is a projected gradient step with stochastic estimates of all gradients.
Another difference in the stochastic versions of the BA [126] and BiO [105] methods is that they use an inverse matrix theorem (based on the Neumann series) to estimate the Hessian inverse. Ref. [105] simplifies the inverse Hessian calculation to replace expensive matrix-matrix multiplications with matrix-vector multiplications. This same strategy makes backpropagation more computationally efficient than the forward mode computation for the unrolled gradient; see Section 4.4.

5.2 Single-Loop Algorithms

Unlike the double-loop algorithms, single-loop algorithms take a gradient step in $\gamma$ without optimizing the lower-level problem each step. Two early bilevel method papers [21], [99] proposed single-loop approaches based on solving the system of equations that arises from the Lagrangian. More recently, [127], [128] extended the complexity analysis of [105], [126] to single-loop algorithms that alternate gradient steps in $x$ and $\gamma$. This section discusses the two approaches in turn.

The system of equations approach in [21], [99] closely follows the KKT perspective on the minimizer approach in Section 4.2.2. Recall that the gradient of the lower-level problem is zero at a minimizer, $\hat{x}$, and one can use this equality as a constraint on the upper-level loss function. The corresponding Lagrangian is

$$L(x, \gamma, \nu) = \ell(\gamma; x) + \nu' \nabla_x \Phi(x; \gamma),$$

where $\nu$ is a vector of Lagrange multipliers. For the filter learning example (Ex), the Lagrangian is

$$L(x, \gamma, \nu) = \frac{1}{2} \|x - x^{\text{true}}\|_2^2 +$$

$$\nu' \left( A'(Ax - y) + c^{\beta_0} \sum_{k=1}^{K} c^{\beta_k} \hat{c}_k \otimes \phi_\epsilon(c_k \otimes x; \epsilon) \right).$$

We again consider derivatives of the Lagrangian with respect to the Lagrange multipliers, $x$, and $\gamma$. Here are the general expressions and the specific equations for the filter learning...
example (Ex), when considering the element of $\gamma$ corresponding to $\beta_k$:

$$
\nabla_{\nu} L(x, \gamma, \nu) = \nabla_x \Phi(x ; \gamma)
= A'(Ax - y) + e^{\beta_0} \sum_{k=1}^K e^{\beta_k} \tilde{c}_k \otimes \phi'(c_k \otimes x; \epsilon)
$$

$$
\nabla_x L(x, \gamma, \nu) = \nabla_x \ell(\gamma ; x) + \nabla_{xx} \Phi(x ; \gamma) \nu
= x - x^{\text{true}}
$$

$$
\nabla_{\gamma} L(x, \gamma, \nu) = \nabla_{\gamma} \ell(\gamma ; x) + \nu' \nabla_{\gamma x} \Phi(x ; \gamma)
$$

$$
\nabla_{\beta_k} L(x, \gamma, \nu) = \nu' \left( e^{\beta_0} e^{\beta_k} \tilde{c}_k \otimes \phi'(c_k \otimes \hat{x}) \right)
.$$
Algorithm 3: Two-Timescale Stochastic Approximation (TTSA) Method from [127]. TTSA includes a possible projection of the hyperparameter after each gradient step onto a constraint set, not shown here. The tildes denote stochastic approximations for the corresponding gradients.

The second main challenge is in estimating the upper-level gradient, even given stochastic estimates of $\nabla x x \Phi$ and $\nabla x y \Phi$, because the minimizer equation (4.8) is nonlinear. The theoretical results about TTSA are built on the assumption that the upper-level gradient is biased due to this nonlinearity. In contrast, STABLE uses recursion to update estimates of the gradients and thus reduce variance. The remainder of this section goes into more detail about both algorithms.

Alg. 3 summarizes the single-loop algorithm TTSA [127]. The analysis of TTSA uses the same lower-level cost function assumptions as mentioned above for BSA [126] and one additional upper-level assumption: that $\ell$ is weakly convex with parameter $\mu_\ell$, i.e.,

$$\ell(\gamma + \delta) \geq \ell(\gamma) \langle \nabla \ell(\gamma), \delta \rangle + \mu_\ell \| \delta \|^2, \quad \forall \gamma, \delta \in \mathbb{R}^R.$$  

TTSA assumes the lower-level gradient estimate is still unbiased and that its variance is now bounded as

$$\mathbb{E} \left[ \| \nabla_x \Phi(x, \gamma) - \tilde{\nabla}_x \Phi(x, \gamma) \|^2 \right] \leq \sigma_\Phi^2 \| \nabla_x \Phi(x, \gamma) \|^2.$$  

Further, the stochastic upper-level gradient estimate, $\tilde{\nabla}_y \ell(\gamma^{(u)}, x^{(u+1)})$, includes a bias $B_u \leq b_u$, that stems from the nonlinear dependence on the lower-level Hessian. This bias decreases as the batch size increases.

The “two-timescale” part of TTSA comes from using different upper and lower step size sequences. The lower-level step size is larger and bounds the tracking error (the distance between $\hat{x}$ and the $x$ iterate) as the hyperparameters change (at the upper-level loss’s relatively slower rate). Thus, [127] chose step-sizes such that $\alpha_\ell(u)/\alpha_\Phi(u) \to 0$. Specifically, if $\ell$ is strongly convex, then $\alpha_\ell$ is $O(u^{-1})$ and $\alpha_\Phi$ is $O(u^{-2/3})$. If $\ell$ is convex, then $\alpha_\ell$ is $O\left(u^{-3/4}\right)$ and $\alpha_\Phi$ is $O\left(u^{-1/2}\right)$.

Chen, Sun, and Yin [128] improved the sample complexity of TTSA. By using a single timescale, their algorithm, STABLE, achieves the “same order of sample complexity as
the stochastic gradient descent method for the single-level stochastic optimization” [128]. However, the improved sample complexity comes at the cost of additional computation per iteration as STABLE can no longer trade a matrix inversion (of size $R \times R$) for matrix-vector products, as done in the Ref. [105]. Ref. [128] therefore recommended STABLE when sampling is more costly than computation or when $R$ is relatively small.

The analysis of STABLE uses the same upper-level loss and lower-level cost function assumptions as listed above for BSA. Additionally, STABLE assumes that, $\forall x$, $\nabla_\gamma \ell(y; x)$ is Lipschitz continuous in $\gamma$. This condition is easily satisfied as many upper-level loss functions do not regularize $\gamma$. Further, those that do often use a squared 2-norm, i.e., Tikhonov-style regularization, that has a Lipschitz continuous gradient. Additionally, rather than bounding the gradient norms as in assumptions A\ell^2 and A\Phi^5, [127] assumes the following moments are bounded:

- the second and fourth moment of $\nabla_\gamma \ell(y; x)$ and $\nabla_x \ell(y; x)$ and
- the second moment of $\nabla_x \Phi(x; \gamma)$ and $\nabla_{xx} \Phi(x; \gamma)$,

ensuring that the upper-level gradient is Lipschitz continuous.

Like the previous algorithms discussed, STABLE evaluates the minimizer result (4.8) at non-minimizer lower-level iterates, $x^{(u)}$, to estimate the hyperparameter gradient. However, it differs in how it estimates and uses the gradients. STABLE replaces the upper-level gradient in TTSA line 4 with

$$
g = \nabla_{\gamma} \ell^{(u)} - (\Delta_{xy}^{(u)})^T (\Delta_{xx}^{(u)})^{-1} \nabla_x \ell^{(u)}. \tag{5.3}
$$

The newly defined gradient matrices are recursively updated as

$$
\Delta_{xy}^{(u)} = \mathcal{P}_{\|\Delta\| \leq C_{\nabla_{xy} \Phi}} \left( (1 - \tau_u) \left( \Delta_{xy}^{(u-1)} - \tilde{\nabla}_{xy} \Phi^{(u-1)} \right) + \tilde{\nabla}_{xy} \Phi^{(u)} \right),
$$

$$
\Delta_{xx}^{(u)} = \mathcal{P}_{\Delta_{\preceq \mu_x, \Phi} I} \left( (1 - \tau_u) \left( \Delta_{xx}^{(u-1)} - \tilde{\nabla}_{xx} \Phi^{(u-1)} \right) + \tilde{\nabla}_{xx} \Phi^{(u)} \right).
$$

In the $\Delta_{xy}^{(u)}$ update, the projection onto the set of matrices with a maximum norm helps ensure stability by not allowing the gradient to get too large. The projection in the $\Delta_{xx}^{(u)}$ update is an eigenvalue truncation that ensures positive definiteness of the estimated Hessian in this Newton-based method. After computing the gradient $g$ (5.3), the upper-level update is a standard descent step as in Alg. 3 line 5.

STABLE also uses the recursively estimated gradient matrices in the lower-level cost function descent. It replaces the standard gradient descent step in Alg. 3 line 3 with one
that uses second order information:

\[ x^{(u+1)} = x^{(u)} - \alpha \Phi(u) \nabla_x \Phi(x^{(u)}, \gamma^{(u)}) - \left( \nabla^{(u)}_{xx} \right)^{-1} \left( \nabla^{(u)}_{yx} \right)' \left( x^{(u+1)} - x^{(u)} \right). \]

With these changes, STABLE is able to reduce the iteration complexity relative to TTSA as summarized in Tab. 5.3.

|       | Upper-level gradients | Lower-level gradients |
|-------|-----------------------|-----------------------|
| BA    | \( O \left( \log \left( \frac{1}{\epsilon} \right) \right) \) | \( O \left( \log \left( \frac{1}{\epsilon} \right) \right) \) |
| stocBiO | \( O \left( \frac{1}{\epsilon^2} \right) \) | \( O \left( \frac{1}{\epsilon^2} \right) \) |
| TTSA  | \( O \left( \frac{1}{\epsilon^{2.5}} \right) \) | \( O \left( \frac{1}{\epsilon^{2.5}} \right) \) |
| STABLE | \( O \left( \frac{1}{\epsilon^2} \right) \) | \( O \left( \frac{1}{\epsilon^2} \right) \) |

**Table 5.3:** Finite-time sample complexities for the stochastic bilevel problem in the common scenario where \( \ell \) is non-convex when using BA [126], stocBiO [105], TTSA [127], and STABLE [128]. When \( \ell \) is strongly convex, the sample complexity of STABLE is \( O \left( \frac{1}{\epsilon^2} \right) \) (for the upper- and lower-level gradients), which is the same as single level stochastic gradient algorithms. See cited papers for other complexity results when \( \ell \) is strongly convex.

### 5.3 Summary of Methods

There are many variations of gradient-based methods for optimizing bilevel problems, especially when one considers that many of the upper-level descent strategies can work with either the minimizer or unrolled approach discussed in Section 4. There is no clear single “best” algorithm for all applications; each algorithm involves trade-offs.

Building on the minimizer and unrolled methods for finding the upper-level gradient with respect to the hyperparameters, \( \nabla \ell(\gamma) \), double-loop algorithms are an intuitive approach. Although optimizing the lower-level problem every time one takes a gradient step in \( \gamma \) is computationally expensive, the lower-level problem is embarrassingly parallelizable across samples. Specifically, one can optimize the lower-level cost for each training sample independently before averaging the resulting gradients to take an upper-level gradient step. In the typical scenario when training is performed offline, training wall-time can therefore be dramatically reduced by using multiple processors.

Single-loop algorithms remove the need to optimize the lower-level cost function multiple times. The single-loop algorithms that consider a system of equations often accelerate convergence using Newton solvers [10], [99]. However, the optimality system grows quickly when there are multiple training images, and may become too computationally expensive as \( J \) increases [42].
The methods that alternate gradient steps in $x$ and $\gamma$ \cite{127}, \cite{128} seem particularly promising for bilevel problems with large training datasets and many hyperparameters. Both methods improve on the finite-time sample complexity of the double-loop algorithms proposed in \cite{105}, \cite{126}. These alternating gradient step methods are relatively recent and have yet to appear in many other papers.

This section organized algorithms based on the number of for-loops; double-loop algorithms have two loops while single-loop algorithms have one. However, not all methods fall cleanly into one group. One such example is the Penalty method \cite{132}. The Penalty method forms a single-level, constrained optimization problem, with the constraint that the gradient of the lower-level cost function should be zero, $\nabla_x \Phi(x; \gamma) = 0$. (This step is similar to the derivation of the minimizer approach via KKT conditions; see Section 4.2.2.) Rather than forming the Lagrangian as in (5.2), \cite{132} penalizes the norm of the gradient, with increasing penalties as the upper iterations increase. Thus, the Penalty cost function\footnote{This is a simplification; \cite{132} allows for constraints on $x$ and $\gamma$.} at iteration $u$ is

$$p(\gamma, x) = \ell(\gamma; \hat{x}(\gamma)) + \lambda^{(u)} \|\nabla_x \Phi(x; \gamma)\|^2.$$ 

The penalty variable sequence, $\lambda^{(u)}$, must be positive, non-decreasing, and divergent ($\lambda^{(u)} \to \infty$).

Penalty \cite{132} incorporates elements of both double-loop and single-loop algorithms. Similar to the double-loop algorithms, Penalty takes multiple gradient descent steps in the lower-level optimization variable, $x$, before calculating and updating the hyperparameters. However, Penalty forms a single-level optimization problem that could be optimized using techniques such as those used in the single-loop algorithms.

The finite-time complexity analyses \cite{105}, \cite{126}–\cite{128} justify the use of gradient-based bilevel methods for problems with many hyperparameters, as none of the sample complexity bounds involved the number of hyperparameters. This is in stark contrast with the hyperparameter optimization strategies in Section 3. However, the per-iteration cost for bilevel methods is still large and increasing with the hyperparameter dimension. Further, the conditions on the lower-level cost function $A\Phi_1$-$A\Phi_6$ seem restrictive and may not be satisfied in practice. Complexity analysis based on more relaxed conditions could be very valuable.

Gradient-based and other hyperparameter optimization methods are active research areas, and the trade-offs continue to evolve. Although it currently seems that gradient-based bilevel methods make sense for problems with many hyperparameters, new methods may overtake or combine with what is presented here. For example, most bilevel methods (and convergence analyses thereof) use classical gradient descent for the lower-level optimization algorithm, whereas \cite{133} showed that the Optimized Gradient Method (OGM) has better convergence guarantees and is optimal among first-order methods for smooth convex
problems [134]. These advances provide opportunities for further acceleration of bilevel methods.

6 Survey of Applications

Bilevel methods have been used in many image reconstruction applications, including 1D signal denoising [20], image denoising (see following sections), compressed sensing [22], spectral CT image reconstruction [121], and MRI image reconstruction [22]. This section discusses trends and highlights specific applications to provide concrete examples of bilevel methods for image reconstruction.

Many papers present or analyze bilevel optimization methods for general upper-level loss functions and lower-level cost functions, under some set of assumptions about each level. Sections 4 and 5 summarized many of these methods. Although there are cases when the choice of a loss function and/or cost impacts the optimization strategy, many bilevel problems could use any optimization method. Thus, this section concentrates on the specific applications, rather than methodology.

This section is split into a discussion of lower-level cost and upper-level loss functions. (Lower-level cost functions that involve CNNs are discussed separately; see Section 7.1.) The conclusion section discusses examples where the loss function is tightly connected to the cost function.

6.1 Lower-level Cost Function Design

Once a bilevel problem is optimized to find \( \hat{\gamma} \), the learned parameters are typically deployed in the same lower-level problem as used during training but with new, testing data. Thus, it is the lower-level cost function that specifies the application of the bilevel problem, e.g., CT image reconstruction or image deblurring.

Denoising applications consider the case where the forward model is an identity operator (\( A = I \)). This case has the simplest possible data-fit term in the cost function and requires the least amount of computation when computing gradients or evaluating \( \Phi \). Because bilevel methods are generally already computationally expensive, it is unsurprising that many papers focus on denoising, even if only as a starting point towards applying the proposed bilevel method to other applications.

More general image reconstruction problems consider non-identity forward models. Few papers learn parameters for image reconstruction in the fully task-based manner described in (UL), likely due to the additional computational cost. Some papers, e.g., [9], [23], [42] consider learning parameters for denoising, and then apply \( \hat{\gamma} \) in a reconstruction problem with the same regularizer but introducing the new \( A \) to the data-fit term. These “crossover
experiments" [9] test the generalizability of the learned parameters, but they sacrifice the specific task-based nature of the bilevel method.

Many bilevel methods, especially in image denoising [20], [42], [47], [99], [124], but also in image reconstruction [19], use the same or a very similar lower-level cost as the running example in this review. From Section 1.2, the running example cost function is:

$$\hat{x}(\gamma, y) = \arg\min_x \Phi(x; \gamma) + R(x; \gamma)$$

The learned hyperparameters, $\gamma$, include the tuning parameters, $\beta_k$ and/or the filter coefficients, $c_k$. The image reconstruction example in [19] generalized (6.1) for implicitly defined forward models by using a different data-fit term, as given in (4.7). Their two examples learn parameters to estimate the diffusion coefficient or forcing function in a second-order elliptic partial differential equation.

Two common variations among applications using (6.1) are (1) the choice of which tuning parameters to learn and (2) what sparsifying function, $\phi$, to use. Some methods [19], [99], [124] learn only the tuning parameters; these methods typically use finite differencing filters or discrete cosine transform (DCT) filters (excluding the DC filter) as the $c_k$'s. Other methods learn only filter coefficients [20]. A slight variation for learning the filter coefficients is to learn coefficients for a linear combination of filter basis elements [42], [47], i.e., learning $a_{k,i}$ where

$$c_k = \sum_i a_{k,i} b_i,$$

for some set of basis filter elements, $b_i$. One benefit of imposing a filter basis is the ability to ensure the filters lie in a given subspace. For example, [42], [47] use the DCT as a basis and remove the constant filter so that all learned filters are guaranteed to have zero-mean.

In terms of sparsifying functions, [20], [124] used the same corner rounded 1-norm as in (CR1N), [47] used $\phi = \log(1 + z^2)$ to relate their method to the Field of Experts framework [45], [19] used a quadratic penalty, and [42], [99] both consider multiple $\phi$ options to examine the impact of non-convexity in $\phi$. Ref. [99] compared $p$-norms, $\|c_k \odot x\|_p$, for $p \in \{\frac{1}{2}, 1, 2\}$, where the $p = \frac{1}{2}$ and $p = 1$ cases are corner-rounded to ensure $\phi$ is smooth. (The $p = \frac{1}{2}$ case is non-convex.) Ref. [42] compared the convex corner-rounded 1-norm in (CR1N) with two non-convex choices: the log-sum penalty $\log(1 + z^2)$, and the Student-t function $\log\left(10e + \sqrt{z^2 + e^2}\right)$.

Both [42], [99] found that non-convex penalty functions led to denoised images with better (higher) PSNR. They hypothesize that the improvement is due to the non-convex penalty functions better matching the heavy-tailed distributions in natural images. As
further evidence of the importance of non-convexity, [42] found that untrained 7 × 7 DCT filters (excluding the constant filter) with learned tuning parameters and a non-convex φ outperformed learned filter coefficients with a convex φ, despite the increased data adaptability when learning filter coefficients. The trade-off for using non-convex penalty functions is the possibility of local minimizers of the lower-level cost.

Chen, Ranftl, and Pock [42] also investigated how the number of learned filters and the size of the filters impacted denoising PSNR. They concluded that increasing the number of filters to achieve an over-complete filter set may not be worth the increased computational expense and that increasing the filter size past 11 × 11 is unlikely to improve PSNR. Using 48 filters of size 7 × 7 and the log-sum penalty function, [42] achieved denoising results on natural images comparable to algorithms such as BM3D [135]. Although results will vary between applications and training data sets, the results from [42] provide motivation for filter learning and an initial guide for designing bilevel methods.

In addition to variations on the running example for Φ (6.1), a common regularizer for the lower-level cost is Total Generalized Variation (TGV) [136]. Whereas TV encourages images to be piece-wise constant, TGV is a generalization of TV designed for piece-wise linear images. Another generalization of TV for piece-wise linear images is Infimal Convolutional Total Variation (ICTV) [137]. Bilevel papers that investigate ICTV include [10], [11]; these papers also investigate TGV. See [138] for a comparison of the two.

TGV cost functions are typically expressed in the continuous domain, at least initially, but then discretized for implementation, e.g., [139], [140]. One discrete approximation of the TGV regularizer is:

\[
R_{TGV}(x) = \min_z e^{\beta_1} \| c_{TV} \ast x - z \|_1 + e^{\beta_2} \| \partial z \|_1,
\]

where \( c_{TV} \) is a filter that takes finite differences and \( \partial \) is a filter that approximates a symmetrized gradient. In TV, one usually thinks of \( z \) as a sparse vector; here \( z \) is a vector whose finite differences are sparse, so \( z \) is approximately piece-wise constant. Encouraging \( z \) to be piece-wise constant in turn makes \( x \) approximately piece-wise constant, since \( c_{TV} \ast x \approx z \) from the first term. Bilevel methods for learning \( \beta_1 \) and \( \beta_2 \) for the TGV regularizer include [10], [11]. An extension to the TGV regularizer model is to learn a space-varying tuning parameter [120].

As an example of how the regularizer should be chosen based on the application, [120] found that standard TV with a learned tuning parameter performed best (in terms of SSIM) for approximately piece-wise constant images while TGV with learned tuning parameters performed best for approximately piece-wise linear images.
6.2 Upper-Level Loss Function Design

From some of the earliest bilevel methods, e.g., [20], [21], to some of the most recent bilevel methods, e.g., [23], [113], mean square error (MSE) remains the most common upper-level loss function. In the unsupervised setting, [82], [83] used SURE (an estimate of the MSE, see Section 3.1) as the upper-level loss function. Unlike many perceptually motivated image quality measures, MSE is convex in $x$ and it is easy to find $\nabla_x \ell(\gamma; \hat{x}(\gamma))$. However, MSE does not capture perceptual quality nor image utility (see Section 3.1). This section discusses a few bilevel methods that used different loss functions.

Ref. [11] compared a MSE upper-level loss function with a Huber (corner rounded 1-norm) loss function. The corresponding lower-level problem was a denoising problem with a standard 2-norm data-fit term and three different options for a regularizer: TV, TGV, and ICTV. The authors learned tuning parameters for a natural image dataset using both upper-level loss function options for each of the lower-level regularizers.

Since SNR is equivalent to MSE, the MSE loss will always perform the best according to any SNR metric (assuming the bilevel model is properly trained). However, [11] found the tuning parameters learned using the Huber loss yielded denoised images with better qualitative properties and better SSIM, especially at low noise levels. Like MSE, the Huber loss operates point-wise and is easy to differentiate. Thus, the authors conclude that the Huber loss is a good trade-off between tractability and improving on MSE as an image quality measure.

A set of loss functions in [120], [121], [124] consider the unsupervised or “blind” bilevel setting, where one wishes to reconstruct an image without clean samples. Therefore, rather than using an image quality metric that compares a reconstructed image, $\hat{x}$, to some true image, $x_{true}$, these loss function consider the estimated residual,

$$\hat{n} = \hat{n}(\gamma) = A\hat{x}(\gamma) - y,$$

where $\gamma$ is learned using only noisy data. Unsupervised bilevel methods may be beneficial when there is no clean data and one has more knowledge of noise properties than of expected image content. All three methods [120], [121], [124] assume the noise variance, $\sigma^2$, is known.

The earliest example [124], learned tuning parameters $\gamma$ such that $\hat{n}$ matched the second moment of the assumed Gaussian distribution for the noise. Their lower-level cost is comparable to (Ex), but re-written in terms of $n$ and with pre-defined finite differencing or $5 \times 5$ DCT filters, i.e., they learn only the tuning parameters, $\beta_k$. Their upper-level loss encourages the empirical variances of the noise in different frequency bands to match the
expected variances:

\[
\ell(\gamma; n(\gamma)) = \frac{1}{2} \sum_i \left( \frac{(\|f_i \odot n\|_2^2 - \mu_i)^2}{v_i} \right)
\]

\[
\mu_i = \mathbb{E}\left[\|f_i \odot n\|_2^2\right] \quad \text{and} \quad v_i = \text{Var}\left[\|f_i \odot n\|_2^2\right],
\]

where \(f_i\) are predetermined filters that select specific frequency components. By using bandpass filters that partition Fourier space, the corresponding means and variances of the second moments of the filtered noise are easily computed, with

\[
\mu_i = N\sigma^2 \|f_i\|^2 \quad \text{and} \quad v_i = N\sigma^4 \|f_i\|^4.
\]

Although the experimental results are promising, [124] does not claim state-of-the-art results since their lower-level denoiser is relatively simple.

As an alternative to the Gaussian-inspired approach in [124], [120] and [121] use loss functions that penalize noise outside a set “noise corridor.” Both methods learn space-varying tuning parameters, and the upper-level loss consists of a data-fit term (that measures noise properties) and a regularizer on \(\gamma\). The data-fit term in the upper-level loss function in [124] defines the noise corridor between a maximum variance, \(\bar{\sigma}^2\), and a minimum variance, \(\sigma^2\):

\[
1'F. (w \odot (n(\gamma) \odot n(\gamma))) \quad \text{for} \quad F(n) = \frac{1}{2} \max(n - \bar{\sigma}^2, 0)^2 + \frac{1}{2} \min(n - \sigma^2, 0)^2,
\]

where \(w\) is a predetermined weighting vector. The noise corridor function, \(F(n)\), penalizes any noise outside of the expected range as shown in Fig. 6.1. Ref. [121] uses the same noise corridor function, but extends the bilevel method for images with Poisson noise; [121] thus estimates the noisy image using the Kullback-Leibler distance. In addition to the noise corridor function as the data-fit component of the upper-level loss function, [120], [121] include a smoothness-promoting regularizer on \(\gamma\), which is a spatially varying tuning parameter vector in both methods.

The task-based nature of bilevel typically makes regularizers or constraints on \(\gamma\) unnecessary (see Section 7.3 for common options for other forms of learning). However, there are two general cases where a regularizer on \(\gamma\) is useful in the upper-level loss function. First, a regularizer can help avoid over-fitting when the amount of training data is insufficient for the number of learnable hyperparameters. This is often the case when learning space-varying parameters that have similar dimensions as the input data, e.g., [21], [120], [121], [141]. In such cases, the regularization often takes the form of a 2-norm on the learned hyperparameters, \(\|\gamma\|_2^2\).

Second, some problems require application-specific constraints. Many such constraints do not require a regularization term. For example, non-negativity constraints on tuning
parameters are easily handled by redefining the tuning parameter in terms of an exponential, as in (Ex), and box constraints are common and easy to incorporate with a projection step if using a gradient-based method. Constraints that require sparsity on the learned parameters may benefit from regularization in the upper-level loss function.

An example of an application-specific constraint is found in [95], [142]. The application in both papers is MRI reconstruction with a data-fit term and a variational regularizer. Both papers extend the bilevel model in (Ex) to include part of the forward model in the learnable parameters, \( \gamma \). Specifically, [95], [142] learned the sparse sampling matrix for MRI. (Ref. [142] additionally learns tuning parameters for predetermined filters, whereas [95] sets the tuning parameters and filters and learns only the sampling matrix.) Here, the forward model is

\[
A = \text{diag}(s_1, s_2, \ldots, s_M) F, 
\]

where \( F \) is the DFT matrix and \( s_i \) are learned binary values that specify whether a frequency location should be sampled.

The motivation for learning a sparse sampling matrix comes from the lower-level MRI reconstruction problem; designing more effective sparse sampling patterns in MRI can decrease scan time and thus improve patient experience, decrease cost, and decrease artifacts from patient movement. This goal requires the learned parameters, \( s_i \), to be binary, which in turn influences the upper-level loss function design. Thus, [95], [142] include regularization in the upper-level to encourage \( s \) to be sparse, e.g., [142] uses an upper-level loss with a MSE term and regularizer on \( s \):

\[
\ell(\gamma; \hat{x}(\gamma)) = \| \hat{x}(\gamma) - x^{\text{true}} \|^2_2 + \lambda \sum_i (s_i + s_i(1 - s_i)),
\]

where \( \lambda \) is a upper-level tuning parameter that one must set manually. (In experiments, they thresholded the learned \( s_i \) values to be exactly binary.) An alternative approach is to
constrain the number of samples [143], though that formulation requires other optimization methods.

6.3 Conclusion

This section split the discussion of lower-level cost and upper-level loss functions to discuss trends in both areas. However, when designing a bilevel problem, design decisions can impact both levels. For example, the unsupervised nature of [121], [124] clearly impacted their choice of upper-level loss function to use noise statistics rather than MSE calculated with ground-truth data. Since it can be challenging to learn many good parameters from noisy training data, the unsupervised nature also likely impacted the authors’ decision to learn only tuning parameters and set the filters manually. Another example of coupling between lower-level and upper-level design is when one enforces application-specific constraints on the learned parameters, e.g., using a regularizer like (6.3) in the upper-level loss to promote sparsity of the MRI sampling matrix [95], [142].

In addition to design decisions influencing both levels, bilevel methods may adopt common techniques for the upper-level loss function and lower-level cost function. For example, a common theme is the tendency to use smooth functions, such as replacing the 1-norm with a corner-rounded 1-norm. This approach requires setting a smoothing parameter, e.g., $\epsilon$ in (CR1N), which in turn impacts the Lipschitz constant and optimization speed. More accurate approximations generally lead to larger Lipschitz constants and slower convergence. One approach to trading-off the accuracy of the smoothing with optimization speed is to use a graduated approach and approximate the non-smooth term more and more closely as the optimization progresses [22].

The prevalence of smoothing is unsurprising considering that this review focuses on gradient-based bilevel methods. A rare exception is [109], which used the (not corner-rounded) one-norm to define $\phi$ to learn convolutional filters using the translation to a single level approach described in Section 4.3. The impact of smoothing and how accurately one should approximate a non-differentiable point remains an open question.

From an image quality perspective, ideally one would design independently the lower-level cost function and upper-level training loss. The lower-level cost would depend on the imaging physics and would incorporate regularizers that expected to provide excellent image quality when tuned appropriately, and the upper-level loss would use terms that are meaningful for the imaging tasks of interest. As we have seen, in practice one often makes compromises to facilitate optimization and reduce computation time.
Increasing reliance on prior knowledge

Increasing reliance on learning from training data

Learned Networks

Inference-aided Networks

Learning-aided Inference Methods

Inference Methods

Bilevel-inspired unrolled networks

Bilevel models following (UL)

Adapts to training data

No need for forward model or image model

More expressive representation

Uses prior knowledge

Does not require large training datasets

Often has theoretical guarantees

**Figure 7.1:** Spectrum of learning to inference-based methods from [144].

### 7 Connections and Future Directions

This final section connects bilevel methods with related approaches and mentions some additional future directions beyond those already described in previous sections.

Shlezinger *et al.* [144] recently proposed a framework, summarized in Fig. 7.1, for categorizing learning-based approaches that combine inferences, or prior knowledge\(^1\), and deep learning. Inferences can include information about the structure of the forward model, \(A\), or about the object \(x\) being imaged. For example, any known statistical properties of the object of interest could be used to design a regularizer that encourages the minimizer \(\hat{x}\) to be compatible with that prior information. At one extreme, inference-based approaches rely on a relatively small number of handcrafted regularizers with a few, if any, tuning parameters learned from training data. At the other extreme, fully learned approaches assume no information about the application or data and learn all hyperparameters from training data.

Ref. [144] proposes two general categories for methods that mix elements of inference-based and learning-based methods. The first category, inference-aided networks, includes DNNs with architectures based on an inference-based method. For example, in deep unrolling, one starts with a fixed number of iterations of an optimization algorithm derived from a cost function and then learns parameters that may vary between iterations, or “layers,” or may be shared across such iterations. Section 7.1 further discusses unrolling, which is a common inference-aided network design strategy, and the connection to the bilevel unrolling method described in Section 4.4.

\(^1\)Ref. [144] uses the term “model-based”, but this review uses “inferences” to differentiate from other definitions of model-based learning in the literature.
The second general category is DNN-aided inference methods [144]. These methods incorporate a deep learning component into traditional inference-based techniques (typically a cost function in image reconstruction). The learned DNN component(s) can be trained separately for each iteration or end-to-end. Because prior knowledge takes a larger role than in the inference-aided networks, these methods typically require smaller training datasets, with the amount of training data required varying with the number of hyperparameters. Section 7.2 discusses how bilevel methods compare to Plug-and-Play, which is an example DNN-aided inference model.

While [144] focused on DNNs due to their highly expressive nature and the abundance of interest in them, the idea of trading off prior knowledge and learning components applies to machine learning more broadly. Sections 7.1 and 7.2 describe how bilevel methods fit into the framework from [144] and relates bilevel methods to other methods in the framework. Although not covered in the above framework, Section 7.3 also compares bilevel methods to a third general category: “single-level” hyperparameter learning methods. Like bilevel methods, single-level methods learn hyperparameters in a supervised manner. However, they generally learn parameters that sparsify the training images, $\{x_{j}^{\text{true}}\}$, and do not use the noisy data, $\{y_{j}\}$. This last comparison demonstrates the benefit of task-based approaches. Of course, there is variety among bilevel methods; this discussion is meant to provide perspective and general relations to increase understanding, rather than to narrow the definition or application of any method.

### 7.1 Connection: Learnable Optimization Algorithms

Learning parameters in unrolled optimization algorithms to create an inference-aided network, often called a Learnable Optimization Algorithm (LOA), is a quickly growing area of research [145]. The first such instance was a learned version of the Iterative Shrinkage and Thresholding Algorithm (ISTA), called LISTA [146]. Similar to the bilevel unrolling method, a LOA typically starts from a traditional, inference-based optimization algorithm, unrolls multiple iterations, and then learns parameters using end-to-end training.

There are many other unrolled methods for image reconstruction [145]. Two examples that explicitly state the bilevel connection are [22], [147]; both set-up a bilevel problem with a DNN as a regularizer and then allow the parameters to vary by iteration, i.e., learning $c_{k}^{(t)}$ where $t$ denotes the lower-level iteration. Ref. [147] motivated the use of an unrolled DNN over more inference-based methods by the lack of an accurate forward model, specifically coil sensitivity maps, for MRI reconstruction. Other examples of unrolled networks are [148], which unrolls the Field of Experts model [45] (see Sections 2.3 and 6.1 for how the Field of Experts model has inspired many bilevel methods); [149], which unrolls the convolutional analysis operator model [51] (see Section 2.3); and [119], which discusses the connection to
meta-learning.

Unlike the unrolled approach to bilevel learning described in Section 4.4, many LOAs depart from their base cost function and “only superficially resemble the steps of optimization algorithms” [22]. For example, unrolled algorithms may “untie” the gradient from the original cost function, e.g., using \( \hat{A}'(Ax - y) \), instead of \( A'(Ax - y) \) for the gradient of the common 2-norm data-fit term, where \( \hat{A}' \) is learned or otherwise differs from the adjoint of \( A \). LOAs that allow the learned parameters to vary every unrolled iteration or learn step size and momentum parameters further depart from a cost function perspective.

In addition to selecting which variables to learn, one must decide how many iterations to unroll for both bilevel unrolled approaches and LOAs. Most methods pick a set number of iterations in advance, perhaps based on previous experience, initial trials, or the available computational resources. Using a set number of iterations yields an algorithm with predictable run times and allows the learned parameters to adapt to the given number of iterations. Further, picking a small number of iterations can act as implicit regularization, comparable to early stopping in machine learning, which may be helpful when the amount of training data is small relative to the number of hyperparameters in the unrolled algorithm [119].

One can also use a convergence criteria to determine the number of iterations to evaluate, rather than selecting a number in advance [113]. This convergence-based method more closely follows classic inference-based optimization algorithms. A benefit of running the lower-level optimization algorithm until convergence is that one could switch optimization algorithms between training and testing, especially for strictly convex lower-level cost functions, and still expect the learned parameters to perform similarly. This ability to switch optimization algorithms means one could use faster, but not differentiable, algorithms at test-time, such as accelerated gradient descent methods with adaptive restart [110]. We are unaware of any bilevel methods that have exploited this possibility.

A subtle point in unrolling gradient-based methods for the lower-level cost function is that the Lipschitz constant of its gradient is a function of the hyperparameters, so the step size range that ensures convergence cannot be pre-specified. Most LOA methods use some fixed step size and allow the learned parameters to adapt to it or learn the step size. An alternative approach is to compute a new step-size as a function of the current parameters, \( \gamma(u) \), every upper-level iteration. For example, given a set value of the tuning parameters and filter coefficients, a Lipschitz constant of the lower-level gradient for (Ex) is

\[
L = \sigma_1^2(A) + e^{\beta_0} L_{\phi} \sum_k e^{\beta_k} \| c_k \|^2,
\]

so a reasonable step size for the classical gradient descent method would be \( 1/L \).

The adaptive approach to setting the step size ensures that any theoretical guarantees of the lower-level optimizer hold. This approach may be beneficial when using a convergence
criteria for the lower-level optimization algorithm or when running sufficiently many lower-level iterations to essentially converge. However, updating the step-size every upper-level iteration is incompatible with fixing the number of unrolled iterations. To illustrate, consider an upper-level iteration where the tuning parameters increase, leading to a larger $L$ and a smaller step size. In a fixed number of iterations, the smaller step size means the lower-level optimization algorithm will be further from convergence, and the estimated minimizer, $\hat{x}(\gamma^{(u+1)})$, may be worse (as judged by the upper-level loss function) than $\hat{x}(\gamma^{(u)})$, even if the updated hyperparameters are better when evaluated with the previous (larger) step-size or more lower-level iterations.

Even within the unrolling methodology, one must make several design decisions. To remain most closely tied to the original optimization algorithm, an unrolled method might fix a large number of iterations or run the optimization algorithm until convergence, use the same parameters every layer, and calculate the step size based on the Lipschitz constant every upper-level iteration. Like all design decisions, there are trade-offs and the literature shows many successful methods that benefit from the increased generality of designing LOAs that are further removed from their cost function roots [145]. Echoing the ideas from [144], the design should be based on the specific application and relative availability, reliability, and importance of prior knowledge and training data.

This survey focuses on unrolled methods that are closely tied to the original bilevel formulation; [145] reviews LOAs more broadly. A benefit of maintaining the connection to the original cost function and optimization algorithm is that, once trained, the lower-level problem in an unrolled bilevel method inherits any theoretical and convergence results from the corresponding optimization method. The corresponding benefit for LOAs is increased flexibility in network architecture.

### 7.2 Connection: Plug-and-play Priors

The Plug-and-Play (PNP) framework [150] is an example of a DNN-aided inference method. It is similar to bilevel methods in its dependence on the forward model. However, unlike bilevel methods, the PNP framework need not be connected to a specific lower-level cost function and it leverages pre-trained denoisers rather than training them for a specific task.

As a brief overview of the PNP framework, consider rewriting the generic data-fit plus regularizer optimization problem (2.1) with an auxiliary variable:

$$\hat{x} = \arg\min_{x \in F} \phi(x; \gamma) + \beta \underbrace{R(z; \gamma)}_{\text{Regularizer}}$$

Using ADMM [151] to solve this constrained optimization problem and rearranging variables
yields the following iterative optimization approach for (7.2):

\[ x^{(u+1)} = \arg\min_x d(x; y) + \frac{\lambda}{2} \| x - (z^{(u)} - u^{(u)}) \|^2_2 = \text{prox}_{\frac{1}{\lambda} d(x; y)}(\tilde{x}) \]

\[ z^{(u)} = \arg\min_z \beta R(z; y) + \frac{\lambda}{2} \| z - (x^{(u)} + u^{(u)}) \|^2_2 = \text{prox}_{\frac{\beta}{\lambda} R(z; y)}(\tilde{z}) \]

\[ u^{(u+1)} = u^{(u)} + (x^{(u+1)} - y^{(u+1)}) \]

where \( \lambda \) is an ADMM penalty parameter that effects the convergence rate (but not the limit, for convex problems). The first step is a proximal update for \( x \) that uses the forward model but does not depend on the regularizer. Conversely, the second step is proximal update for the split variable \( z \) that depends on the regularizer, but is agnostic of the forward model. This step acts as a denoiser. The final step is the dual variable update and encourages \( x^{(u)} \approx z^{(u)} \) as \( u \to \infty \).

The key insight from [150] is that the above update equations separate the forward model and denoiser. Thus, one can substitute, or “plug in,” a wide range of denoisers for the \( z \) update, in place of its proximal update, while keeping the data-fit update independent.

Whereas in the original ADMM approach, the parameter \( \lambda \) has no effect on the final image for convex cost functions, in the PNP framework that parameter does affect image quality. Thus, one could also use training data to tune the \( \lambda \) in a bilevel manner. Although PNP allows one to substitute a pre-trained denoiser, one could additionally tune the parameters in the denoiser. Ref. [152] provides one such example of starting from a PNP framework then learning denoising parameters and \( \lambda \) that vary by iteration.

A large motivation for the PNP framework is the abundance of advanced denoising methods, including ones that are not associated with an optimization problem such as BM3D [135]. However, using existing denoisers sacrifices the ability to learn parameters to work well with the specific forward model, as is done in task-based methods. As a simple example of how learned parameters may differ when \( A \) changes, Chambolle and Pock [9] found that different (synthesis) filters worked better for image denoising versus image inpainting. A more complicated example is using bilevel methods to learn some aspect of \( A \) alongside some aspect of the regularizer, e.g., [142] learned a sparse sampling matrix and tuning parameter for MRI that are adaptive to the regularization for the image reconstruction problem.

### 7.3 Connection: Single-Level Parameter Learning

Section 2.3 briefly discussed some approaches to learning analysis operators. This section further motivates the task-based bilevel set-up by discussing the filter learning constraints imposed in single-level hyperparameter learning methods.
As summarized in Section 2.3, the earliest methods for learning analysis regularizers had no constraints on the analysis operators. Those approaches learned filters from training data to make a prior distribution match the observed data distribution. In contrast, more recent approaches to filter learning minimize a cost function that requires either a penalty function or constraint on the operators to ensure filter diversity. For reference, the cost functions mentioned in Section 2.3 were:

\[
\text{AOL} : \arg\min_{\Omega, X} \|\Omega X\|_1 + \frac{\beta}{2} \|Y - X\|^2 \quad \text{s.t.} \quad \Omega \in S,
\]

\[
\text{TL} : \arg\min_{\Omega \in \mathbb{R}^{3 \times 5}, X} \|\Omega Y - X\|^2 + R(\Omega) \quad \text{s.t.} \quad \|X_i\|_0 \leq \alpha \forall i,
\]

\[
\text{CAOL} : \arg\min_{[c_1, \ldots, c_K]} \sum_{k=1}^{K} \frac{1}{2} \|c_k \odot x - z\|^2 + \beta \|z_k\|_0 \quad \text{s.t.} \quad [c_1, \ldots, c_K] \in S,
\]

where AOL is analysis operator learning [50], TL is transform learning [40], and CAOL is convolutional analysis operator learning [51]. In the following discussion of constraint sets, the equivalent filter matrix for CAOL has the convolutional kernels as rows:

\[
\Omega_{\text{CAOL}} = \begin{bmatrix} c'_1 \\ \vdots \\ c'_K \end{bmatrix}
\]

While there are many other proposed cost functions in the literature, using different norms or including additional variables, these three examples capture the most common structures for filter learning.

In all the above cost functions, if one removed the constraint or regularizer, then the trivial solution would be to learn zero filters for \(\Omega\). Furthermore, a simple row norm constraint on \(\Omega\) would be insufficient, as then the minimizer would contain a single filter that is repeated many times. (In contrast, a unit norm constraint typically suffices for dictionary learning.) A row norm constraint plus a full rank constraint is also insufficient because \(\Omega\) can have full rank while being arbitrarily close to the rank-1 case of having a single repeated row.

The choice of constraint set \(S\) is important in single-level learning. Many methods constrain analysis operators to satisfy a tight frame constraint. A matrix \(A\) is a tight frame if there is a positive constant, \(\alpha\), such that

\[
\|A'x\|^2_2 = \sum_i |\langle q_i, x \rangle|^2 = \alpha \|x\|^2_2, \quad \forall x
\]

where \(q_i\) is the \(i\)th column of \(A\). This tight frame condition is equivalent to \(AA' = \alpha I\) for some positive constant \(\alpha\). Most analysis operators are defined with filters in their rows, so a tight frame requirement on the filters appears as the constraint \(\Omega'\Omega = \alpha I\).
Under the tight frame constraint for the filters, $\Omega$ must be square or tall, so the filters are complete or over-complete. However, [50] found that the frame constraint was insufficient when learning over-complete operators, as the “excess” rows past full-rank tended to be all zeros. Therefore, [50] imposed a uniformly-normalized tight frame constraint: each row of the $\Omega$ had to have unit norm and the filters had to form a tight frame.

Ref. [39] similarly constrained $\Omega$ to have unit-norm rows with the filters forming a frame (though not tight). Such loosening of the tight frame constraint to a frame constraint could lead to the problem of learning almost identical rows, as discussed above. To prevent this issue, [39] additionally included a penalty that encourages distinct rows:

$$-\sum_k \sum_{k<k} \log \left( 1 - (\omega_k^T \omega_k)^2 \right).$$

\hspace{1cm} (7.3)

One possible concern with a tight frame constraint is that it requires the filters to span all of $\mathbb{F}^N$, so every spatial frequency can pass through at least one filter. However, most images are not zero-mean and have piece-wise constant regions, so the zero frequency component is not sparse. Ref. [50] modified the tight-frame constraint to require $\Omega$ to span some space (e.g., the space orthogonal to the zero frequency term). Likewise, [153] extended the CAOL algorithm to include handcrafted filters, such as a zero frequency term, that can then be used or discarded when reconstructing images. In the bilevel literature, [42], [47] similarly ensured that learned filters had no zero frequency component by learning coefficients for a linear combination of filter basis vectors, rather than learning the filters directly; see Section 6.1.

As an alternative to imposing a strict constraint on the filters, one can penalize $\Omega$ to encourage filter diversity, as in (7.3). Using a penalty has the advantage of being able to learn any size (under- or over-complete) $\Omega$ and not requiring the filters to represent all frequencies. For example, as an alternative to the tight frame constraint, [51] proposed a version of CAOL using the following regularizer (to within scaling constants)

$$R(\Omega) = \beta \|\Omega \Omega^T - I\|^2$$

and a unit norm constraint on the filters. Ref. [43] included a similar penalty to (7.3), but with the inner product being divided by the norm of the filters as the filters were not constrained to unit norm. All such variations on this penalty are to encourage filter diversity.

To ensure a square $\Omega$ is full rank, while also encouraging it to be well-conditioned, [40] used a regularizer that includes a term of the form

$$R(\Omega) = \beta \log |\Omega|.$$

The log determinant term is known as a log barrier; it forces $\Omega$ to have full rank because of the asymptote of the log function. Ref. [43] includes a similar log barrier regularization term in terms of the eigenvalues of $\Omega$ to ensure it is left-invertible.
As another example of a filter penalty regularizer, both [40] and [43], include the following regularization term

\[ R(\Omega) = \beta_2 \|\Omega\|^2_F, \]

rather than constraining the norm of the filters. This Frobenius norm addresses the scale ambiguity in the analysis and transform formulations and ensures the filter coefficients do not grow too large in magnitude.

Yet another approach to encouraging filter diversity is to consider the frequency response of the set of filters. Pfister and Bresler [43] discusses different constraint options for filter banks based on convolution strides to ensure perfect reconstruction. When the stride is one and one considers circular boundary conditions, the filters can perfectly reconstruct any signal as long as they pass the \( N \) discrete Fourier transform frequencies. Tight frames satisfy this constraint, but the constraint is more relaxed than a tight frame constraint.

Section 6 discussed some (relatively rare) bilevel problems with penalties on the learned hyperparameters, but, notably, there are no constraints nor penalties on the filters in the bilevel method (Ex)! Because of its task-based nature, filters learned via the bilevel method should be those that are best for image reconstruction. Thus, one should not have to worry about redundant filters, zero filters, or filters with excessively large coefficients. This property is one of the key benefits of bilevel methods.

### 7.4 Future Directions

Throughout this review, we mentioned a few areas for future work on bilevel methods. This section highlights some of the avenues that we think are particularly promising.

Advancing upper-level loss function design is identified as future work in many bilevel papers. Despite the abundance of research on image quality metrics (see Section 3.1), most bilevel methods use mean square error for the upper-level loss function (see Section 6.2 for exceptions). Using loss functions that better match the end-application of the images is a clear future direction for bilevel methods that nicely aligns with their task-based nature.

For example, in the medical imaging field there is a large literature on objective measures of image quality [154], often based on mathematical observers designed to emulate human performance on signal detection tasks, for example for situations where a lesion’s location is unknown [155]. To our knowledge, there has been little if any work to date on using such mathematical observers to define loss functions for bilevel methods or for training CNN models, though there has been work on CNN-based observers [156]. Using task-based metrics for bilevel methods and CNN training is a natural direction for future work that could bridge the extensive literature on such metrics with the image reconstruction field.

Unsupervised bilevel problems are exceptions to the trend of using MSE for the upper-level loss function. Section 6.1 considered a few unsupervised bilevel methods that use noise
statistics to estimate the quality of the reconstructed images, e.g., [120], [121], [124] [82], [83]. One extension to the unsupervised setting is the semi-supervised setting, where one might have access to a few clean training samples and additional, noisy training samples. An alternative extension would be to consider a two-stage bilevel problem, where perhaps filters and initial tuning parameters are learned offline from a training dataset and then the tuning parameters are further adjusted during the reconstruction in an unsupervised manner to account for possible differences between the specific image being reconstructed and the training dataset.

Just as considering more advanced image quality metrics for the upper-level loss function is a promising area for future work, bilevel methods can likely be improved by using more advanced lower-level cost functions. Perhaps due to the already challenging and non-convex nature of bilevel problems, most methods consider convex lower-level cost functions. Papers that examine non-convex regularizers, e.g., [42], [99], conclude that non-convex regularizers lead to more accurate image reconstructions, likely due to better matching the statistics of natural images. This observation aligns with the simple denoising experimental results in [157], where learned filters with (CR1N) as the regularizer denoised signals worse than a hand-crafted filter with the non-convex 0-norm regularizer. In other words, the structure of the regularizer matters in addition to how one learns the filters.

In addition to non-convexity, future bilevel methods could consider non-smooth cost functions. Many bilevel methods require the lower-level cost to be smooth. Exceptions include, the translation to a single level approach (Section 4.3), which uses the 1-norm as the lower-level regularizer and unrolled methods (Section 4.4), which can be applied to non-smooth cost functions as long as the optimization algorithm has smooth updates. As an example of a non-smooth cost with a smooth update, [158] proved that a cost function with a 1-norm plus a $p$-norm regularizer has a smooth proximal operator for $p \in (1, 2)$, despite including a non-smooth 1-norm. The impact of smoothing the cost function on the perceptual quality of the reconstructed image is largely unknown.

Another avenue for future work is based on the fact that $x_{\text{true}}$ is really a continuous-space function. A few methods, e.g., [10], [11], develop bilevel methods in continuous-space. However, the majority of methods use discretized forward models without considering the impact of this simplification (as done in this review paper). Future investigations of bilevel methods should strive to avoid the “inverse crime” [159] implicit in (1.3) where the data is synthesized using the same discretization assumed by the reconstruction method.

### 7.5 Summary of Advantages and Disadvantages

Like the methods described in [144], bilevel methods for computational imaging involve mixing inference-based optimization approaches with learning-based approaches to leverage
benefits of both techniques.

Inference-based approaches use prior knowledge, usually in the form of a forward model and an object model, to reconstruct images. Typically the forward model, $A$, is under-determined, so some form of regularization based on the object model is essential. Regularizers always involve some number of adjustable parameters; traditionally inference-based methods select such parameters empirically or using basic image properties like resolution and noise [101], [160]. The regularization parameters may also be learned from training to maximize SNR [161] or detection task performance [162] in a bilevel manner (often using a grid or random search due to the relatively small number of learnable parameters). When the forward model and object model are well-known and easy to incorporate in a cost function, inference-based methods can yield accurate reconstructions without the need for large datasets of clean training data.

Learning-based approaches use training datasets to learn a prior. Recently, learning-based approaches have achieved remarkable reconstruction accuracy in practice, largely due to the increased availability in computational resources and larger, more accessible training datasets [4], [5]. However, many (deep) learning methods lack theoretical guarantees and explainability and finding sufficient training data is still challenging in many applications. Both of these challenges may impede adoption of learning-based methods in clinical practice for some applications, such as medical image reconstruction [163]. Some deep learning methods for CT image reconstruction were approved for clinical use in 2019 [164]; early studies have shown such methods can significantly reduce noise but may also compromise low-contrast spatial resolution [165].

Combining inference-based and learning-based approaches allows the integration of learning from training data while using smaller training datasets by incorporating prior knowledge. Such mixed methods often maintain interpretability from the inference-based roots while using learning to provide adaptive regularization. Thus, the benefits of bilevel methods in this review’s introduction are generally shared among the methods described in [144]: theoretical guarantees, competitive performance in terms of reconstruction accuracy, and similar performance to learned networks with a fraction of the free parameters, e.g., [22], [23].

What distinguishes bilevel methods from the other methods in the inference-based to learning-based spectrum in Fig. 7.1? While one can argue that the conventional CNN and deep learning approach is always bilevel in the sense that the hyperparameters are trained to minimize a loss function, this review considered bilevel methods with the cost function structure (LL). The regularization term in (LL) could be based on a DNN [22], but we followed the bilevel literature that focuses on priors/regularizers, such as in (Ex), maintaining a stronger connection to traditional cost function design.

Another lens for understanding bilevel methods is extending single-level hyperparameter optimization approaches to be task-based, bilevel approaches. Single-level approaches to
image reconstruction, such as those using dictionary learning [69], convolutional analysis operator learning [51], and convolutional dictionary learning [166], [167], generally aim to learn characteristics of a training dataset, with the idea that these characteristics can then be used in a prior for an image reconstruction task. While such an approach may learn more general information, [109], [157] showed that a common single-level optimization strategy resulted in learning a regularizer that was suboptimal for the simple task of image denoising.

As further evidence of the benefit of task-based learning, [109] found that the lack of constraints in the bilevel filter learning problem is important; the learned filters used the flexibility of the model and were not orthonormal, whereas orthonormality is a constraint often imposed in single-level models (see Section 7.3). Ref. [99] showed how the task-based nature adapts to training data; total variation based regularization works well for piece-wise constant images but less so for natural images. Beyond adapting to the training dataset, bilevel methods are task-based in terms of adapting to the level of noise; [95] found the learned tuning parameters for image denoising go to 0 as the noise goes to 0, since no regularization is needed in the absence of noise for well-determined problems.

A primary disadvantage cited for most bilevel methods is the computational cost compared to single-level hyperparameter optimization methods or other methods with a smaller learning component. In turn, the main driver behind the large computational cost of gradient-descent based bilevel optimization methods is that one typically has to optimize the lower-level cost function many times, either to some tolerance or for a certain number of iterations. The computational cost involves a trade-off because how accurately one optimizes the lower-level problem can impact the quality of the learned parameters. For example, [42], [99] both claim better denoising accuracy than [47] because they optimize the lower-level problem more accurately. Similarly, [109] notes that learning will fail if the lower-level cost is not optimized to sufficient accuracy.

There are various strategies to decrease the computational cost for bilevel methods. Some are relatively intuitive and applicable to a wide range of problems in machine learning. For example, [109] used larger batch size as the iterations continue, [10] increased the batch size if a gradient step in $\gamma$ does not sufficiently improve the loss function, and [95] tightened the accuracy requirement for the gradient estimation over iterations. These strategies all save computation by starting with rougher approximations near the beginning of the optimization method, when $\gamma^{(u)}$ is likely far from $\hat{\gamma}$, while using a relatively accurate solution by the end of the algorithm.

Another disadvantage of bilevel methods is that, while the optimization algorithm for the lower-level problem often has theoretical convergence guarantees, and the lower-level cost is often designed to be strictly convex, the full bilevel problem (UL) is usually non-convex, so the quality of the learned hyperparameters can depend on initialization. Thus, in practice, one need a strategy for initializing $\gamma$. For example, for (Ex), one may decide to
use a single-level filter learning technique such as the Field of Experts [45] to initialize the hyperparameters. Or, one can use a handcrafted set of filters, such as the DCT filters (or a subset thereof). Other hyperparameters often have similar warm start options. Despite the non-convexity, papers that tested multiple initializations generally found similarly good solutions surprisingly often, e.g., [42], [95], [120].

There is no one correct answer for how much a method should use prior information or learning techniques, and it is unlikely that any single approach can be the best for all image reconstruction applications. Like most engineering problems, the trade-off is application-dependent. One should (minimally) consider the amount of training data available, how representative the training data is of the test data, how under-determined the forward model is (i.e., how strong of regularization is needed), how well-known the object model is, the importance of theoretical guarantees and explainability, and the available computational resources at training time and at test time. Bilevel methods show particular promise for applications where training data is limited and/or explainability is highly valued, such as in medical imaging.

Acknowledgements

This work was made possible in part due to the support of NIH grant R01 EB023618, NSF grant IIS 1838179, and the Rackham Predoctoral Fellowship. The authors would like to thank Lindon Roberts for helpful email discussion of [95].

A Additional Running Example Results

This appendix derives some results that are relevant to the running example used throughout the survey.

A.1 Derivatives for Convolutional Filters

This section proves the result

$$\nabla_{c_s} (\hat{c}_k \odot f.(c_k \odot x)) = f.((c_k \odot z^{(s)}) + \hat{c}_k \odot (\hat{f}.(c_k \odot x) \odot x^{(-s)})),$$

(A.1)

when considering $F = \mathbb{R}$. This equation is key to finding derivatives of the lower-level cost function in (Ex) with respect to the filter coefficients.

To simplify notation, we drop the indexing over $k$, so $c$ is a single filter and $c_s$ denotes the $s$th element in the filter for $s \in \mathbb{Z}^D$. Here, $s$ indexes every dimension of $c$, e.g., for a two-dimensional filter, we could equivalently write $s$ as $(s_1, s_2)$. Recall that the notation $\hat{c}$ signifies a reversed version of $c$, as needed for the adjoint of convolution.
Define the notation $\mathbf{x}^{(i)}$ as the vector $\mathbf{x}$ circularly shifted according to the index $i$. Thus, if $\mathbf{x}$ is 0-indexed and we use circular indexing,

$$ (\mathbf{x}^{(s)})_i = \mathbf{x}_{s+i}. $$

As two concrete examples,

$$ \mathbf{x}^{(-1)} = \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_N \\ x_1 \end{bmatrix}, $$

and, in two dimensions, if $\mathbf{X} \in \mathbb{F}^{M \times N}$

$$ \mathbf{X}^{(1,2)} = \begin{bmatrix} x_{M,N-1} & x_{M,N} & x_{M,1} & \cdots & x_{M,3} \\ x_{1,N-1} & x_{1,N} & x_{1,1} & \cdots & x_{1,3} \\ x_{2,N-1} & x_{2,N} & x_{2,1} & \cdots & x_{2,3} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{M-1,N-1} & x_{M-1,N} & x_{M-1,1} & \cdots & x_{M-1,3} \end{bmatrix}. $$

Notice the element $x_{1,1}$ appears in the second row and third column of $\mathbf{X}^{(1,2)}$, since

$$ \left(\overline{(1,2)} \atop \overline{s}\overline{i} \right) = \langle 2, 3 \rangle. $$

This circular shift notation is useful in the derivation and statement of the desired gradient.

Define $\mathbf{z} = \mathbf{c} \ast \mathbf{x}$, where $\mathbf{c}$ and $\mathbf{x}$ are both $N$-dimensional. By the definition of convolution, $\mathbf{z}$ is given by

$$ \mathbf{z} = \sum_{i_1} \cdots \sum_{i_N} h_{i_1,\ldots,i_N} \mathbf{x}^{(i_1,\ldots,i_N)} := \sum_{i_1,\ldots,i_N} h_{i_1,\ldots,i_N} \mathbf{x}^{(i)}, $$

where, for each sum, the indexing variable $i_n$ iterates over the size of $\mathbf{c}$ in the $i$th dimension and we simplify the index for circularly shifting vectors, $i_1,\ldots,i_N$, as simply $\langle i \rangle$. This expression shows that the derivative of $\mathbf{c} \ast \mathbf{x}$ with respect to the $s$th filter coefficient is the $-s$th coefficient in $\mathbf{x}$, i.e.,

$$ \nabla_{c_s} (\mathbf{c} \ast \mathbf{x}) = \mathbf{x}^{(-s)}. \quad (A.2) $$
We can now find the gradient of interest:
\[
\tilde{c} \odot f.(z) = \sum_{i_1, \ldots, i_N} (\tilde{c})_{i_1, \ldots, i_N} f.(z)_{(i)}
\]
by the convolution formula
\[
= \sum_{i_1, \ldots, i_N} [\tilde{c}]_{i_1, \ldots, i_N} f.(z)_{(i)}
\]
since \( f \) operates point-wise
\[
= \sum_{i_1, \ldots, i_N} c_{i_1, \ldots, i_N} f.(z)_{(i)}
\]
by definition of \( \tilde{c} \)
\[
= \sum_{i_1, \ldots, i_N} c_{i_1, \ldots, i_N} f.(z)_{(i)}
\]
reverse summation order.

Recall that \( z \) is a function of \( c_s \). Therefore, using the chain rule to take the derivative,
\[
\nabla_{c_s} (\tilde{c} \odot f.(z)) = f.(z\langle s \rangle) + \sum_{i_1} \cdots \sum_{i_N} c_{i_1, \ldots, i_N} f.(z\langle i_1, \ldots, i_N \rangle) \odot \nabla_{c_s} (z^{(i)})
\]
where the second equality follows from (A.2). Recognizing the convolution formula in the second summand, the expression can be simplified to
\[
f.(z\langle s \rangle) + \tilde{c} \odot (f.(z) \odot x^{(-s)}).
\]
This proves the claim. Note that the provided formula is for a single element in \( c \). One can concatenate the result for each value of \( s \) to get the gradient for all elements in \( c \).

### A.2 Evaluating Conditions for the Running Example

To better understand the conditions in Section 5.1.1, this section illustrates how they apply to the filter learning example (Ex).

#### A.2.1 Upper-level Loss Conditions

Recall the upper-level loss function is mean square error:
\[
\ell(\gamma; x) = \frac{1}{2} \| x - x^{\text{true}} \|_2^2,
\]
where \( \ell \) is typically evaluated at \( x = \hat{x}(\gamma) \).

The loss function (A.3) satisfies \( A\ell \). Because there is no dependence on \( \gamma \) in the upper-level, \( L_{x, \nabla_{\gamma} \ell} = 0 \). The gradient with respect to \( x \) is \( \nabla_x \ell(\gamma; x) = x - x^{\text{true}} \), so \( L_{x, \nabla_x \ell} = 1 \).
The norm of the gradient with respect to $x$, $\|\nabla_x \ell(\gamma; x)\| = \|x - x^{\text{true}}\|$, can grow arbitrarily large, so condition $A\ell 2$ is not met in general. However, in most applications, one can assume an upper bound (possibly quite large) on the elements of $x^{\text{true}}$ and impose that bound as a box constraint when computing $\hat{x}$. Then the triangle inequality provides a bound on $\|x - x^{\text{true}}\|$ for all $x$ within the constraint box.

Finally, $A\ell 3$ is met by any loss function, including (A.3), that lacks cross terms between $x$ and $\gamma$. We are unaware of any bilevel method papers using such cross terms.

### A.2.2 Lower-level Cost Conditions

One property used below in many of the bounds for the lower-level cost function is that

$$\sigma_1(C_k) = \|c_k\|_2,$$

where $\sigma_1(\cdot)$ is a function that returns the first singular value of a matrix argument. To derive this equality, we use the circular boundary assumption on the convolutional filters. By the DFT convolution property, multiplication by the convolutional matrix $C_k$ is equivalent to multiplication by the orthogonal discrete Fourier Transform (DFT) matrix, $F'$, filtering in the Fourier transform domain, and multiplication with the inverse DFT matrix. This identity provides an eigenvalue decomposition for the convolutional matrix,

$$C_k = F A F',$$

where the eigenvalues of $C_k$ are simply the DFT of the first column of $C_k$:

$$\text{eig}(C_k) = \left\{ \sum_{n=0}^{N-1} (c_k)_n e^{-i2\pi nm/N} \right\} \text{ for } m \in \{0, \ldots, N-1\}.$$

The largest singular value of $C_k' C_k$ is the squared magnitude of the largest eigenvalue of $C_k$, which we can simplify using the property that the magnitude of any complex exponential is bounded by 1:

$$\sigma_1(C_k' C_k) = \max_{m=\{0,\ldots,N-1\}} \left( \sum_{n=0}^{N-1} (c_k)_n e^{-i2\pi nm/N} \right)^2 \leq \left( \sum_{n=0}^{N-1} (c_k)_n \right)^2 = \|c_k\|_2^2.$$

This inequality proves the claim.

As with the upper-level conditions, (Ex) meets most of the conditions if we impose additional constraints on the maximum norm of variables. In addition to bounding the elements in $x$, as we did to ensure $A\ell 2$, imposing bounds on $\|c_k\|$ and $|\beta_k|$ is sufficient to meet all the lower-level conditions $A\Phi 1$-$A\Phi 6$. We now examine each condition individually.

Recall from (Ex) that the example lower-level cost function is

$$\hat{x}(\gamma) = \arg\min_{x \in F^N} \frac{1}{2} \|Ax - y\|_2^2 + e^{\beta_0 K} \sum_{k=1}^{K} e^{\beta_k} \phi_k (c_k \otimes x; \epsilon),$$
where $\phi$ is a corner-rounded 1-norm (CR1N).

As described in Section 4.2, the minimizer approach requires $\Phi$ to be twice differentiable. Thus, $\Phi$ satisfies $A\Phi_1$. This condition limits the choices of $\phi$ to twice differentiable functions.

Considering $A\Phi_2$, the gradient of $\Phi$ with respect to $x$ is Lipschitz continuous in $x$ if the norm of the Hessian, $\|\nabla_{xx}\Phi(x ; \gamma)\|_2$, is bounded. Using (4.9) and assuming the Lipschitz constant of the derivative of $\phi$ is $L_{\phi}$ (for (CR1N), $L_{\phi} = \frac{1}{\epsilon}$), a Lipschitz constant for $\nabla_x \Phi$ is

$$L_{x,\nabla_x \Phi} = \sigma_1^2(A) + L_{\phi} e^{\beta_0} \sum_k e^{\beta_k} \sigma_1 C_k' C_k$$

$$= \sigma_1^2(A) + L_{\phi} e^{\beta_0} \sum_k e^{\beta_k} \|c_k\|^2$$

by (A.4).

The Lipschitz constant $L_{x,\nabla_x \Phi}$ depends on the values in $\gamma$ and therefore does not strictly satisfy $A\Phi_2$. Here if $\beta_0, \beta_k, \text{ and } c_k$ have upper bounds, then one can upper bound $L_{x,\nabla_x \Phi}$. All of the bounds below have similar considerations.

To consider the strong convexity condition in $A\Phi_3$, we again consider the Hessian,

$$\nabla_{xx} \Phi(x ; \gamma) = A'A + e^{\beta_0} \sum_k e^{\beta_k} C_k' \text{diag}(\tilde{\phi}(c_k \oplus x)) C_k.$$  \hspace{1cm} (A.5)

We assume that $\tilde{\phi}(z) \geq 0 \forall z$, as is the case for the corner rounded 1-norm. If $A'A$ is positive-definite with $\sigma_N(A'A) > 0$ (this is equivalent to $A$ having full column rank), then the Hessian is positive-definite and $\mu_{x,\Phi} = \sigma_1^2(A)$ suffices as a strong convexity parameter.

In applications like compressed sensing, $A$ does not have full column rank. In such cases, $\sigma_N(A'A) = 0$ and as $e^{\beta_0} \to 0$ the regularizer term vanishes, so there does not exist any universal $\mu_{x,\Phi} > 0$ for all $\gamma \in \mathbb{R}^R$, so the strong convexity condition $A\Phi_3$ is not satisfied. However, the condition is likely to hold in practice for “good” values of $\gamma$. Specifically, the condition will hold for any value of $\gamma$ such that the null-space of the regularization term is disjoint from the null-space of $A$ and the regularization parameters are sufficiently large. To interpret this condition, recall that regularization helps compensate for the under-determined nature of $A$ (Section 2.1). Values of $\gamma$ that do not sufficiently “fill-in” the null-space of $A$ will leave the lower-level cost function under-determined; the task-based nature of the bilevel problem should discourage these “bad” values. Nevertheless, how to adapt the complexity theory to rigorously address these subtleties is an open question.

The fourth condition, $A\Phi_4$, is that $\nabla_{xx} \Phi(x ; \gamma)$ and $\nabla_{yx} \Phi(x ; \gamma)$ are Lipschitz continuous with respect to $x$ for all $\gamma$. For the first part part, a Lipschitz constant results from bounding the difference in the Hessian evaluated at two points, $x^{(1)}$ and $x^{(2)}$:

$$\left\|\nabla_{xx} \Phi(x^{(1)} ; \gamma) - \nabla_{xx} \Phi(x^{(2)} ; \gamma)\right\|_2$$

$$= \left\|e^{\beta_0} \sum_k e^{\beta_k} C_k' \text{diag}(\tilde{\phi}(c_k \oplus x^{(1)}) - \tilde{\phi}(c_k \oplus x^{(2)})) C_k\right\|_2.$$
Since every element of $\hat{\phi}$ is bounded in $(0, L_{\hat{\phi}})$, the difference between any two evaluations of $\hat{\phi}$ is at most $L_{\hat{\phi}}$. Thus
\[
\|\nabla_{xx} \Phi(x^{(1)}; \gamma) - \nabla_{xx} \Phi(x^{(2)}; \gamma)\|_2 \leq e^{\beta_0} L_{\hat{\phi}} \sum_k e^{\beta_k} \|C'_k C_k\|_2 \\
\leq e^{\beta_0} L_{\hat{\phi}} \sum_k e^{\beta_k} \|c_k\|_2^2.
\]

The final simplification again uses (A.4). Thus, $L_{\Phi} = e^{\beta_0} L_{\hat{\phi}} \sum_k e^{\beta_k} \|c_k\|_2^2$.

For the second part of A.\Phi4, we must look at the tuning parameters and filter coefficients separately. When considering learning a tuning parameter, $\beta_k$, 
\[
\nabla_{\beta_k} \Phi(x; \gamma) = e^{\beta_0 + \beta_k} C'_k \hat{\phi}(C_k x).
\]

To find a Lipschitz constant, consider the gradient with respect to $x$:
\[
\nabla_x (\nabla_{\beta_k} \Phi(x; \gamma)) = e^{\beta_0 + \beta_k} C'_k \text{diag}(\hat{\phi}(C_k x)) C_k.
\]

A Lipschitz constant of $\nabla_{\beta_k} \Phi(x; \gamma)$ is given by the bound on the norm of this matrix (we chose to use the matrix 2-norm, also called the spectral norm). Using similar steps as above to simplify the expression, $L_{\Phi} = e^{\beta_0 + \beta_k} L_{\hat{\phi}} \|c_k\|_2^2$.

When considering learning the $s$th element of the $k$th filter,
\[
\nabla_{c_{k,s}} \Phi(x; \gamma) = e^{\beta_0 + \beta_k} \left( \hat{\phi}(\beta_k x^{(s)}) + C'_k \left( \hat{\phi}(C_k x) \odot x^{(s)} \right) \right)
\]
\[
= e^{\beta_0 + \beta_k} \left( \hat{\phi}(R C_k x) + C'_k \left( \hat{\phi}(C_k x) \odot R x \right) \right) \in \mathbb{F}^N,
\]

where $R$ is a rotation matrix that depends on $s$ such that $R x = x^{(s)}$. For taking the gradient, it is convenient to note that the last term can be expressed in multiple ways:
\[
\hat{\phi}(C_k x) \odot x^{(s)} = \text{diag}(\hat{\phi}(C_k x)) x^{(s)} = \text{diag}(x^{(s)}) \hat{\phi}(C_k x).
\]

Using the alternate expressions to perform the chain rule with respect to the $x$ that is not in the diag(·) statement, the gradient with respect to $x$ is:
\[
\nabla_x \left( \nabla_{c_{k,s}} \Phi(x; \gamma) \right) = e^{\beta_0 + \beta_k} \left( C'_k R^t \text{diag}(\hat{\phi}(R C_k x)) \right)
\]
\[
+ C'_k \text{diag}(\hat{\phi}(C_k x))
\]
\[
+ C'_k \text{diag}(\hat{\phi}(C_k x)) \text{diag}(R x) C_k.
\]
The bound on the spectral norm of the first and second expressions are both \( \sigma_1(C_k)L_\phi \) because, for any \( z \in \mathbb{F}^N \),

\[
\left\| \text{diag} (\dot{\phi} (z)) \right\|_2 \leq \max_z |\dot{\phi}(z)| = L_\phi.
\]

The third expression is bounded by \( \sigma_1^2(C_k) \|x\| L_\phi \), which requires a bound on the norm of \( x \), similar to \( A_f \). Summing the three expressions and including the tuning parameters gives the final Lipschitz constant

\[
L_{x, \nabla c_k,s \Phi} = e^{\beta_0} + \beta_k \sigma_1(C_k)(2L_\phi + \sigma_1(C_k)L_\phi \|x\|).
\]

(A.6)

The fifth assumption, \( A\Phi_5 \) states that the mixed second gradient of \( \Phi \) is bounded. For the tuning parameters, the mixed second gradient is given in (4.9) as

\[
\nabla_{\beta k} \Phi(\hat{x}; y) = e^{\beta_0} e^{\beta_k} \bar{c}_k \circ \dot{\phi} (c_k \circ \hat{x}).
\]

The bound given in Section 5.1.1 follows easily by considering that

\[
\left\| \text{diag} (\dot{\phi} (c_k \circ \hat{x})) \right\|_2 \leq \max_z |\dot{\phi}(z)| = L_\phi.
\]

For a filter coefficient, the mixed second gradient is more complicated:

\[
\nabla_{c_k,s} \Phi(\hat{x}; y) = e^{\beta_0} + \beta_k \left( \dot{\phi} (c_k \circ \hat{x}) \right) + \bar{c}_k \circ \left( \dot{\phi} (c_k \circ \hat{x}) \right).
\]

Under the assumption that the bounds \( L_\Phi \) and \( L_{\beta} \) exist (they are 1 and \( \frac{1}{\epsilon} \) respectively for (CR1N)), a bound on the norm of the mixed gradient is

\[
\left\| \nabla_{c_k,s} \Phi(\hat{x}; y) \right\| \leq e^{\beta_0} + \beta_k \left( L_\Phi + L_\Phi \|c_k\| \|x\| \right).
\]

The sixth assumption, \( A\Phi_6 \), is that \( L_{\gamma, \nabla y \Phi} \) and \( L_{\gamma, \nabla x \Phi} \) exist. Lipschitz constants for the tuning parameters are

\[
L_{\beta k, \nabla_{\beta k} \Phi} = e^{\beta_0} + \beta_k \|c_k\|_2 L_\phi \quad \text{and} \quad L_{\beta k, \nabla_{x} \Phi} = e^{\beta_0} + \beta_k \|c_k\| L_\phi^2.
\]

Using similar derivations as shown above, corresponding Lipschitz constants for the filter coefficients are

\[
L_{c_k,s, \nabla_{c_k,s} \Phi} = e^{\beta_0} + \beta_k \left( L_\phi + \|x\| \left( L_\phi + L_\phi \|c_k\| \|x\| \right) \right)
\]

\[
L_{c_k,s, \nabla_{x} \Phi} = e^{\beta_0} + \beta_k \left( 2L_\phi \|c_k\| + L_\phi \|c_k\|^2 \|x\| \right).
\]

This is the last lower-level condition.
References

[1] H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of inverse problems*. Dordrecht: Kluwer, 1996.

[2] C. H. McCollough, A. C. Bartley, R. E. Carter, B. Chen, T. A. Drees, P. Edwards, D. R. Holmes, A. E. Huang, F. Khan, S. Leng, K. L. McMillan, G. J. Michalak, K. M. Nunez, L. Yu, and J. G. Fletcher, “Low-dose CT for the detection and classification of metastatic liver lesions: Results of the 2016 Low Dose CT Grand Challenge,” e339–52, *Med. Phys.*, vol. 44, no. 10, Oct. 2017. doi: 10.1002/mp.12345.

[3] Y. Eldar and G. Kutyniok, *Compressed sensing: Theory and applications*. Cambridge, 2012. doi: 10.1017/CBO9780511794308.

[4] G. Wang, “A perspective on deep imaging,” 8914–24, *IEEE Access*, vol. 4, Nov. 2016. doi: 10.1109/ACCESS.2016.2624938.

[5] K. Hammernik and F. Knoll, “Machine learning for image reconstruction,” in *Handbook of Medical Image Computing and Computer Assisted Intervention*, Elsevier, 2020, pp. 25–64. doi: 10.1016/B978-0-12-816176-0.00007-7.

[6] M. T. McCann and M. Unser, “Biomedical image reconstruction: From the foundations to deep neural networks,” pp. 283–359, *Foundation and Trends in Signal Processing*, vol. 13, no. 3, 2019. doi: 10.1561/2000000101.

[7] S. Dempe and A. Zemkoho, Eds., *Bilevel Optimization: Advances and next Challenges*, vol. 161, ser. Springer Optimization and Its Applications. Springer International Publishing, 2020. doi: 10.1007/978-3-030-52119-6.

[8] J. Mairal, F. Bach, and J. Ponce, “Task-driven dictionary learning,” pp. 791–804, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, Apr. 2012. doi: 10.1109/TPAMI.2011.156.

[9] A. Chambolle and T. Pock, “Learning consistent discretizations of the total variation,” pp. 778–813, vol. 14, no. 2, 2021. doi: 10.1137/20MI177199.

[10] L. Calatroni, C. Chung, J. C. D. L. Reyes, C.-B. Schönlieb, and T. Valkonen, “Bilevel approaches for learning of variational imaging models,” in *Variational Methods in Imaging and Geometric Control*, ser. Radon Series on Computational and Applied Mathematics, vol. 18, De Gruyter, 2017. [Online]. Available: http://arxiv.org/abs/1505.02120.

[11] J. C. De los Reyes, C.-B. Schönlieb, and T. Valkonen, “Bilevel parameter learning for higher-order total variation regularisation models,” pp. 1–25, *Journal of Mathematical Imaging and Vision*, vol. 57, no. 1, Jan. 2017. doi: 10.1007/s10851-016-0662-8.

[12] P. Ochs, R. Ranftl, T. Brox, and T. Pock, “Bilevel optimization with nonsmooth lower level problems,” in *International Conference on Scale Space and Variational Methods in Computer Vision*, vol. 9087, pp. 654–665, 2015. doi: 10.1007/978-3-319-18461-6_52.
[13] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” 65–98, SIAM Review, vol. 59, no. 1, 2017. DOI: 10.1137/141000671.

[14] M. Stone, “Cross-validation: A review,” 127–139, Math Oper Stat Ser Stat., vol. 9, no. 1, 1978. DOI: 10.1080/02331887808801414.

[15] G. H. Golub, M. Heath, and G. Wahba, “Generalized cross-validation as a method for choosing a good ridge parameter,” 215–23, Technometrics, vol. 21, no. 2, May 1979. [Online]. Available: http://www.jstor.org/stable/1268518.

[16] D. L. Phillips, “A technique for the numerical solution of certain integral equations of the first kind,” 84–97, J. Assoc. Comput. Mach., vol. 9, no. 1, Jan. 1962. DOI: 10.1145/321105.321114.

[17] S. S. Saquib, C. A. Bouman, and K. Sauer, “ML parameter estimation for Markov random fields, with applications to Bayesian tomography,” 1029–44, IEEE Trans. Im. Proc., vol. 7, no. 7, Jul. 1998. DOI: 10.1109/83.701163.

[18] W. P. Segars, G. Sturgeon, S. Mendonca, J. Grimes, and B. M. W. Tsui, “4D XCAT phantom for multimodality imaging research,” pp. 4902–15, Medical Physics, vol. 37, no. 9, Aug. 2010. DOI: 10.1118/1.3480985.

[19] G. Holler, K. Kunisch, and R. C. Barnard, “A bilevel approach for parameter learning in inverse problems,” p. 115012, Inverse Problems, vol. 34, no. 11, Nov. 1, 2018. DOI: 10.1088/1361-6420/aade77.

[20] G. Peyré and J. M. Fadili, “Learning analysis sparsity priors,” in IEEE Intl. Conf. on Sampling Theory and Appl. (SampTA), 2011. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00542016/document.

[21] E. Haber and L. Tenorio, “Learning regularization functionals a supervised training approach,” pp. 611–626, Inverse Problems, vol. 19, no. 3, Jun. 1, 2003. DOI: 10.1088/0266-5611/19/3/309.

[22] Y. Chen, H. Liu, X. Ye, and Q. Zhang. (Jul. 22, 2020). “Learnable descent algorithm for nonsmooth nonconvex image reconstruction.” arXiv: 2007.11245 [cs].

[23] E. Kobler, A. Effland, K. Kunisch, and T. Pock. (Jun. 15, 2020). “Total Deep Variation: A Stable Regularizer for Inverse Problems.” arXiv: 2006.08789 [cs, math].

[24] R. M. Lewitt and S. Matej, “Overview of methods for image reconstruction from projections in emission computed tomography,” 1588–611, Proc. IEEE, vol. 91, no. 10, Oct. 2003. DOI: 10.1109/JPROC.2003.817882.

[25] M. Elad, P. Milanfar, and R. Rubinstein, “Analysis versus synthesis in signal priors,” pp. 947–68, Inverse Problems, vol. 23, no. 3, Jun. 2007. DOI: 10.1088/0266-5611/23/3/007.
[26] C. Guillemot and O. Le Meur, “Image inpainting: Overview and recent advances,” 127–44, IEEE Sig. Proc. Mag., vol. 31, no. 1, Jan. 2014. doi: 10.1109/MSP.2013.2273004.

[27] J. A. Fessler, “Model-based image reconstruction for MRI,” 81–9, IEEE Sig. Proc. Mag., vol. 27, no. 4, Jul. 2010. doi: 10.1109/MSP.2010.936726.

[28] G. H. Golub and C. F. Van Loan, “An analysis of the total least squares problem,” 883–93, SIAM J. Numer. Anal., vol. 17, no. 6, Dec. 1980. doi: 10.1137/0717073.

[29] L. Ying and J. Sheng, “Joint image reconstruction and sensitivity estimation in SENSE (JSENSE),” 1196–1202, Mag. Res. Med., vol. 57, no. 6, Jun. 2007. doi: 10.1002/mrm.21245.

[30] S. Ravishankar, J. C. Ye, and J. A. Fessler, “Image reconstruction: From sparsity to data-adaptive methods and machine learning,” 86–109, Proc. IEEE, vol. 108, no. 1, Jan. 2020. doi: 10.1109/JPROC.2019.2936204.

[31] S. Nam, M. Davies, M. Elad, and R. Gribonval, “The cosparse analysis model and algorithms,” pp. 30–56, Applied and Computational Harmonic Analysis, vol. 34, no. 1, Jan. 2013. doi: 10.1016/j.jacha.2012.03.006.

[32] M. Elad, Sparse and redundant representations: from theory to applications in signal and image processing. Berlin: Springer, 2010. doi: 10.1007/978-1-4419-7011-4.

[33] G. Peyre, “A review of adaptive image representations,” 896–911, IEEE J. Sel. Top. Sig. Proc., vol. 5, no. 5, Sep. 2011. doi: 10.1109/JSTSP.2011.2120592.

[34] E. Candes, J. Romberg, and T. Tao, “Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information,” pp. 489–509, IEEE Transactions on Information Theory, vol. 52, no. 2, Feb. 2006. doi: 10.1109/TIT.2005.862083.

[35] R. Tibshirani, “Regression shrinkage and selection via the Lasso,” pp. 267–288, Journal of the Royal Statistical Society: Series B (Methodological), vol. 58, no. 1, Jan. 1996. doi: 10.1111/j.2517-6161.1996.tb02080.x.

[36] P. Zhou, C. Zhang, and E. Fatemi, “Nonlinear total variation based noise removal algorithm,” 259–68, Physica D, vol. 60, no. 1-4, Nov. 1992. doi: 10.1016/0167-2789(92)90242-F.

[37] E. J. Candès, Y. C. Eldar, D. Needell, and P. Randall, “Compressed sensing with coherent and redundant dictionaries,” pp. 59–73, Applied and Computational Harmonic Analysis, vol. 31, no. 1, Jul. 2011. doi: 10.1016/j.jacha.2010.10.002.

[38] S. Hawe, M. Kleinsteuber, and K. Diepold, “Analysis operator learning and its application to image reconstruction,” pp. 2138–2150, IEEE Transactions on Image Processing, vol. 22, no. 6, Jun. 2013. doi: 10.1109/TIP.2013.2246175.
S. Ravishankar and Y. Bresler, “Learning sparsifying transforms,” pp. 1072–1086, *IEEE Transactions on Signal Processing*, vol. 61, no. 5, Mar. 2013. DOI: 10.1109/TSP.2012.2226449.

J. A. Fessler, “Optimization methods for MR image reconstruction,” 33–40, *IEEE Sig. Proc. Mag.*, vol. 37, no. 1, Jan. 2020. DOI: 10.1109/MSP.2019.2943645.

Y. Chen, R. Ranftl, and T. Pock, “Insights into analysis operator learning: From patch-based sparse models to higher order MRFs,” pp. 1060–1072, *IEEE Transactions on Image Processing*, vol. 23, no. 3, Mar. 2014. DOI: 10.1109/TIP.2014.2299065.

L. Pfister and Y. Bresler, “Learning filter bank sparsifying transforms,” pp. 504–519, *IEEE Transactions on Signal Processing*, vol. 67, no. 2, Jan. 2019. DOI: 10.1109/TSP.2018.2883021.

B. M. Afkham, J. Chung, and M. Chung. (Apr. 13, 2021). “Learning regularization parameters of inverse problems via deep neural networks.” arXiv: 2104.06594.

S. Roth and M. Black, “Fields of experts: A framework for learning image priors,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 2, pp. 860–867, 2005. DOI: 10.1109/CVPR.2005.160.

M. F. Tappen, C. Liu, E. H. Adelson, and W. T. Freeman, “Learning gaussian conditional random fields for low-level vision,” in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, Jun. 2007. DOI: 10.1109/CVPR.2007.382979.

K. G. G. Samuel and M. F. Tappen, “Learning optimized MAP estimates in continuously-valued MRF models,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 477–484, Jun. 2009. DOI: 10.1109/CVPR.2009.5206774.

B. Ophir, M. Elad, N. Bertin, and M. D. Plumbley, “Sequential minimal eigenvalues - an approach to analysis dictionary learning,” pp. 1465–1469, *19th European Signal Processing Conference*, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/7074010.

M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, “Analysis operator learning for overcomplete cosparse representations,” presented at the 2011 19th European Signal Processing Conference, pp. 1470–1474, IEEE, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/7074220.

M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, “Constrained overcomplete analysis operator learning for cosparse signal modelling,” pp. 2341–2355, *IEEE Transactions on Signal Processing*, vol. 61, no. 9, May 2013. DOI: 10.1109/TSP.2013.2250968.

I. Y. Chun and J. A. Fessler, “Convolutional analysis operator learning: Acceleration and convergence,” pp. 2108–2122, *IEEE Transactions on Image Processing*, vol. 29, 2020. DOI: 10.1109/TIP.2019.2937734. arXiv: 1802.05584.

S. Haykin, “Neural networks expand SP’s horizons,” 24–49, *IEEE Sig. Proc. Mag.*, vol. 13, no. 2, Mar. 1996. DOI: 10.1109/79.487040.
[53] J.-N. Hwang, S.-Y. Kung, M. Niranjan, and J. C. Principe, “The past, present, and future of neural networks for signal processing,” 28–48, IEEE Sig. Proc. Mag., vol. 14, no. 6, Nov. 1997. DOI: 10.1109/79.637299.

[54] A. Lucas, M. Iliadis, R. Molina, and A. K. Katsaggelos, “Using deep neural networks for inverse problems in imaging: Beyond analytical methods,” 20–36, IEEE Sig. Proc. Mag., vol. 35, no. 1, Jan. 2018. DOI: 10.1109/msp.2017.2760358.

[55] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in Medical Image Computing and Computer-Assisted Intervention, 234–41, 2015. DOI: 10.1007/978-3-319-24574-4_28.

[56] J. C. Ye, Y. Han, and E. Cha, “Deep convolutional framelets: A general deep learning framework for inverse problems,” 991–1048, SIAM J. Imaging Sci., vol. 11, no. 2, Jan. 2018. DOI: 10.1137/17m1141771.

[57] B. Wen, S. Ravishankar, L. Pfister, and Y. Bresler, “Transform learning for magnetic resonance image reconstruction: From model-based learning to building neural networks,” 41–53, IEEE Sig. Proc. Mag., vol. 37, no. 1, Jan. 2020. DOI: 10.1109/MSP.2019.2951469.

[58] M. Feurer and F. Hutter, “Chapter 1: Hyperparameter optimization,” in Automated Machine Learning: Methods, Systems, Challenges, ser. The Springer Series on Challenges in Machine Learning, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., Springer International Publishing, 2019, pp. 3–33. DOI: 10.1007/978-3-030-05318-5.

[59] L. A. Shepp and B. F. Logan, “The Fourier reconstruction of a head section,” 21–43, IEEE Trans. Nuc. Sci., vol. 21, no. 3, Jun. 1974. DOI: 10.1109/TNS.1974.6499235.

[60] J. A. Fessler, MIRT-demo: 01-recon, Jul. 25, 2020. [Online]. Available: https://github.com/JeffFessler/mirt-demo/blob/master/isbi-19/01-recon.jl.

[61] C. You, Q. Yang, H. Shan, L. Gjestebly, G. Li, S. Ju, Z. Zhang, Z. Zhao, Y. Zhang, W. Cong, and G. Wang, “Structure-sensitive multi-scale deep neural network for low-dose CT denoising,” pp. 41 839–41 855, IEEE Access, vol. 6, 2018. DOI: 10.1109/ACCESS.2018.2858196.

[62] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” 600–12, IEEE Trans. Im. Proc., vol. 13, no. 4, Apr. 2004. DOI: 10.1109/TIP.2003.819861.

[63] Z. Wang and A. Bovik, “Reduced- and no-reference image quality assessment,” pp. 29–40, IEEE Signal Processing Magazine, vol. 28, no. 6, Nov. 2011. DOI: 10.1109/MSP.2011.942471.

[64] L. Zhang, L. Zhang, X. Mou, and D. Zhang, “A comprehensive evaluation of full reference image quality assessment algorithms,” in 2012 19th IEEE International Conference on Image Processing, pp. 1477–1480, Sep. 2012. DOI: 10.1109/ICIP.2012.6467150.
[65] W. Zhang, K. Ma, J. Yan, D. Deng, and Z. Wang, “Blind image quality assessment using a deep bilinear convolutional neural network,” pp. 36–47, IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 1, Jan. 2020. DOI: 10.1109/TCSVT.2018.2886771.

[66] A. Mason, J. Rioux, S. E. Clarke, A. Costa, M. Schmidt, V. Keough, T. Huynh, and S. Beyea, “Comparison of objective image quality metrics to expert radiologists’ scoring of diagnostic quality of MR images,” pp. 1064–1072, IEEE Transactions on Medical Imaging, vol. 39, no. 4, Apr. 2020. DOI: 10.1109/TMI.2019.2930338.

[67] M. Gholizadeh-Ansari, J. Alirezaie, and P. Babyn, “Deep learning for low-dose CT denoising using perceptual loss and edge detection layer,” 504–15, J. Digital Im., vol. 33, no. 2, 2020. DOI: 10.1007/s10278-019-00274-4.

[68] G. Seif and D. A., “Edge-based loss function for single image super-resolution,” in Proc. IEEE Conf. Acoust. Speech Sig. Proc., 1468–72, 2018. DOI: 10.1109/ICASSP.2018.8461664.

[69] S. Ravishankar and Y. Bresler, “MR image reconstruction from highly undersampled k-space data by dictionary learning,” pp. 1028–1041, IEEE Transactions on Medical Imaging, vol. 30, no. 5, May 2011. DOI: 10.1109/TMI.2010.2090538.

[70] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” pp. 600–612, IEEE Transactions on Image Processing, vol. 13, no. 4, Apr. 2004. DOI: 10.1109/TIP.2003.819861.

[71] Z. Wang, E. Simoncelli, and A. Bovik, “Multiscale structural similarity for image quality assessment,” in The Thirteenth Asilomar Conference on Signals, Systems & Computers, 2003, pp. 1398–1402, IEEE, 2003. DOI: 10.1109/ACSSC.2003.1292216.

[72] G. P. Renieblas, A. T. Nogués, A. M. González, N. G. León, and E. G. . Castillo, “Structural similarity index family for image quality assessment in radiological images,” p. 035501, J. Med. Im., vol. 4, no. 3, Jul. 2017. DOI: 10.1117/1.JMI.4.3.035501.

[73] S. Bosse, D. Maniry, K.-R. Muller, T. Wiegand, and W. Samek, “Deep neural networks for no-reference and full-reference image quality assessment,” pp. 206–219, IEEE Transactions on Image Processing, vol. 27, no. 1, Jan. 2018. DOI: 10.1109/TIP.2017.2760518.

[74] G. W. Lindsay, “Convolutional neural networks as a model of the visual system: Past, present, and future,” pp. 1–15, Journal of Cognitive Neuroscience, Feb. 6, 2020. DOI: 10.1162/jocn_a_01544.

[75] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in International Conference on Learning Representations, May 2015. [Online]. Available: http://arxiv.org/abs/1409.1556.
[76] T. J. Hebert and R. Leahy, “Statistic-based MAP image reconstruction from Poisson data using Gibbs priors,” 2290–303, *IEEE Trans. Sig. Proc.*, vol. 40, no. 9, Sep. 1992. doi: 10.1109/78.157228.

[77] C. M. Stein, “Estimation of the mean of a multivariate normal distribution,” *The Annals of Statistics*, vol. 9, no. 6, Nov. 1, 1981. doi: 10.1214/aos/1176345632.

[78] S. Ramani, T. Blu, and M. Unser, “Monte-carlo sure: A black-box optimization of regularization parameters for general denoising algorithms,” pp. 1540–1554, *IEEE Transactions on Image Processing*, vol. 17, no. 9, Sep. 2008. doi: 10.1109/TIP.2008.2001404.

[79] S. Soltanayev and S. Y. Chun, “Training deep learning based denoisers without ground truth data,” *Neural Information Processing Systems*, 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/c0560792e4a3c79e62f76cbf9fb277dd-Paper.pdf.

[80] K. Kim, S. Soltanayev, and S. Y. Chun, “Unsupervised training of denoisers for low-dose CT reconstruction without full-dose ground truth,” 1112–25, *IEEE J. Sel. Top. Sig. Proc.*, vol. 14, no. 6, Oct. 2020. doi: 10.1109/JSTSP.2020.3007326.

[81] M. Zhussip, S. Soltanayev, and S. Y. Chun, “Training deep learning based image denoisers from undersampled measurements without ground truth and without image prior,” in *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, 10247–56, 2019. doi: 10.1109/CVPR.2019.01050.

[82] H. Zhang, X. Chen, X. Zhang, and X. Zhang, “A bi-level nested sparse optimization for adaptive mechanical fault feature detection,” pp. 19767–19782, *IEEE Access*, vol. 8, 2020. doi: 10.1109/ACCESS.2020.2968726.

[83] C.-A. Deledalle, S. Vaiter, J. Fadili, and G. Peyré, “Stein Unbiased GrAdient estimator of the Risk (SUGAR) for multiple parameter selection,” pp. 2448–2487, *SIAM Journal on Imaging Sciences*, vol. 7, no. 4, Jan. 2014. doi: 10.1137/140968045.

[84] Y. Eldar, “Generalized SURE for exponential families: Applications to regularization,” pp. 471–481, *IEEE Transactions on Signal Processing*, vol. 57, no. 2, Feb. 2009. doi: 10.1109/TSP.2008.2008212.

[85] A. Mittal, R. Soundararajan, and A. C. Bovik, “Making a “completely blind” image quality analyzer,” pp. 209–212, *IEEE Signal Processing Letters*, vol. 20, no. 3, Mar. 2013. doi: 10.1109/LSP.2012.2227726.

[86] L. Kang, P. Ye, Y. Li, and D. Doermann, “Convolutional neural networks for no-reference image quality assessment,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1733–1740, Jun. 2014. doi: 10.1109/CVPR.2014.224.

[87] T. U. of Texas at Austin: Laboratory for Image and Video Engineering. (). “Image & video quality assessment at LIVE,” [Online]. Available: http://live.ece.utexas.edu/research/quality/.

93
[102] S. Dempe, “Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints,” pp. 333–359, Optimization, vol. 52, no. 3, Jun. 2003. DOI: 10.1080/0233193031000149894.

[103] S. Scholtes and M. Stöhr, “How stringent is the linear independence assumption for mathematical programs with complementarity constraints?” Pp. 851–863, Mathematics of Operations Research, vol. 26, no. 4, Nov. 2001. DOI: 10.1287/moor.26.4.851.10007.

[104] K. B. Petersen and M. S. Pedersen, The Matrix Cookbook. Technical University of Denmark, Nov. 2012. [Online]. Available: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=3274.

[105] K. Ji, J. Yang, and Y. Liang, “Bilevel optimization: Convergence analysis and enhanced design,” in Proceedings of the 38th International Conference on Machine Learning, pp. 4882–4892, Jul. 2021. [Online]. Available: https://proceedings.mlr.press/v139/ji21c.html%5C.

[106] R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo, “On the iteration complexity of hypergradient computation,” in Proceedings of the 37th International Conference on Machine Learning, p. 11, 2020. [Online]. Available: http://proceedings.mlr.press/v119/grazzi20a.html.

[107] C. B. Do, C.-S. Foo, and A. Y. Ng, “Efficient multiple hyperparameter learning for log-linear models,” in Neural Information Processing Systems, pp. 377–384, Dec. 3, 2007. [Online]. Available: https://proceedings.neurips.cc/paper/2007/file/851dd5f5058cf22df63d3344ad89919cf-Paper.pdf.

[108] P. Sprechmann, R. Litman, T. B. Yakar, A. M. Bronstein, and G. Sapiro, “Supervised sparse analysis and synthesis operators,” in Neural Information Processing Systems, pp. 908–916, 2013. [Online]. Available: http://papers.nips.cc/paper/5002-supervised-sparse-analysis-and-synthesis-operators.pdf.

[109] M. T. McCann and S. Ravishankar, “Supervised learning of sparsity-promoting regularizers for denoising,” arXiv Computing Research Repository, Jun. 9, 2020. arXiv: 2006.05521.

[110] D. Kim and J. A. Fessler, “Adaptive restart of the optimized gradient method for convex optimization,” 240–63, J. Optim. Theory Appl., vol. 178, no. 1, Jul. 2018. DOI: 10.1007/s10957-018-1287-4.

[111] S. Boyd and L. Vandenberghe, Convex optimization. UK: Cambridge, 2004. [Online]. Available: http://web.stanford.edu/~boyd/cvxbook.html.

[112] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, “Forward and reverse gradient-based hyperparameter optimization,” in Proceedings of the International Conference on Machine Learning, pp. 1165–1173, PMLR, Dec. 12, 2017. [Online]. Available: http://proceedings.mlr.press/v70/franceschi17a.html.
[113] H. Antil, Z. Di, and R. Khatri, “Bilevel optimization, deep learning and fractional laplacian regularization with applications in tomography,” *Inverse Problems*, Mar. 18, 2020. DOI: 10.1088/1361-6420/ab80d7.

[114] V. Monga, Y. Li, and Y. C. Eldar, “Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing,” 18–44, *IEEE Sig. Proc. Mag.*, vol. 38, no. 2, Mar. 2021. DOI: 10.1109/MSP.2020.3016905.

[115] B. Dauvergne and L. Hascoet, “The data-flow equations of checkpointing in reverse automatic differentiation,” in *International Conference on Computational Science*, pp. 566–573, 2006. DOI: 10.1007/11758549_78.

[116] M. Kellman, K. Zhang, E. Markley, J. Tamir, E. Bostan, M. Lustig, and L. Waller, “Memory-efficient learning for large-scale computational imaging,” pp. 1403–1414, *IEEE Transactions on Computational Imaging*, vol. 6, 2020. DOI: 10.1109/TCI.2020.3025735.

[117] D. P. Palomar and Y. C. Eldar, *Convex optimization in signal processing and communications*. Cambridge, 2011. DOI: 10.1017/CBO9780511804458.

[118] F. Pedregosa, “Hyperparameter optimization with approximate gradient,” in *Proceedings International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48, pp. 737–46, PMLR, Jun. 20–22, 2016. [Online]. Available: http://proceedings.mlr.press/v48/pedregosa16.html.

[119] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, “Bilevel programming for hyperparameter optimization and meta-learning,” in *International Conference on Machine Learning*, Jul. 3, 2018. arXiv: 1806.04910.

[120] M. Hintermüller, K. Papafitsoros, C. N. Rautenberg, and H. Sun. (Feb. 13, 2020). “Dualization and automatic distributed parameter selection of total generalized variation via bilevel optimization.” arXiv: 2002.05614.

[121] B. Sixou, “Adaptative regularization parameter for poisson noise with a bilevel approach: Application to spectral computerized tomography,” pp. 1–18, *Inverse Problems in Science and Engineering*, Dec. 22, 2020. DOI: 10.1080/17415977.2020.1864348.

[122] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, “A limited memory algorithm for bound constrained optimization,” 1190–208, *SIAM J. Sci. Comp.*, vol. 16, no. 5, 1995. DOI: 10.1137/0916069.

[123] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations*, vol. abs/1412.6980, May 2015. arXiv: 1412.6980.

[124] J. Fehrenbach, M. Nikolova, G. Steidl, and P. Weiss, “Bilevel image denoising using gaussianity tests,” in *International Conference on Scale Space and Variational Methods in Computer Vision*, vol. 9087, pp. 117–128, 2015. DOI: 10.1007/978-3-319-18461-6_10.
[125] B. Lecouat, J. Ponce, and J. Mairal, “A flexible framework for designing trainable priors with adaptive smoothing and game encoding,” in Advances in Neural Information Processing Systems, vol. 33, pp. 15664–15675, 2020. [Online]. Available: https://papers.nips.cc/paper/2020/hash/b4edda67f057e218a8e766927e3e5c5-Abstract.html.

[126] S. Ghadimi and M. Wang. (Feb. 6, 2018). “Approximation methods for bilevel programming.” arXiv: 1802.02246.

[127] M. Hong, H.-T. Wai, Z. Wang, and Z. Yang. (Dec. 20, 2020). “A Two-Timescale Framework for Bilevel Optimization: Complexity Analysis and Application to Actor-Critic.” arXiv: 2007.05170 [cs, math], [Online]. Available: http://arxiv.org/abs/2007.05170.

[128] T. Chen, Y. Sun, and W. Yin. (Feb. 22, 2021). “A single-timescale stochastic bilevel optimization method.” arXiv: 2102.04671.

[129] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, “Deep convolutional neural network for inverse problems in imaging,” 4509–22, IEEE Trans. Im. Proc., vol. 26, no. 9, Sep. 2017. DOI: 10.1109/TIP.2017.2713099.

[130] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” p. 60, J. Big Data, vol. 6, no. 1, Jul. 2019. DOI: 10.1186/s40537-019-0197-0.

[131] Y. Nesterov, “A method of solving a convex programming problem with convergence rate $O(1/k^2)$,” 372–76, Soviet Math. Dokl., vol. 27, no. 2, 1983.

[132] A. Mehra and J. Hamm. (Jun. 28, 2020). “Penalty method for inversion-free deep bilevel optimization.” arXiv: 1911.03432.

[133] D. Kim and J. A. Fessler, “On the convergence analysis of the optimized gradient method,” pp. 187–205, Journal of Optimization Theory and Applications, vol. 172, no. 1, Jan. 2017. DOI: 10.1007/s10957-016-1018-7.

[134] Y. Drori, “The exact information-based complexity of smooth convex minimization,” p. 1–16, J. Complexity, vol. 39, Apr. 2017. DOI: 10.1016/j.jco.2016.11.001.

[135] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, “Image denoising by sparse 3-D transform-domain collaborative filtering,” pp. 2080–2095, IEEE Transactions on Image Processing, vol. 16, no. 8, Aug. 2007. DOI: 10.1109/TIP.2007.901238.

[136] K. Bredies, K. Kunisch, and T. Pock, “Total generalized variation,” pp. 492–526, SIAM Journal on Imaging Sciences, vol. 3, no. 3, Jan. 2010. DOI: 10.1137/090769521.

[137] A. Chambolle and P.-L. Lions, “Image recovery via total variation minimization and related problems,” pp. 167–188, Numerische Mathematik, vol. 76, no. 2, Apr. 1, 1997. DOI: 10.1007/s002110050258.

[138] M. Benning, C. Brune, M. Burger, and J. Müller, “Higher-order TV methods—Enhancement via Bregman iteration,” pp. 269–310, Journal of Scientific Computing, vol. 54, no. 2-3, Feb. 2013. DOI: 10.1007/s10915-012-9650-3.
[139] F. Knoll, K. Bredies, T. Pock, and R. Stollberger, “Second order total generalized variation (TGV) for MRI,” 480–91, Mag. Res. Med., vol. 65, no. 2, 2011. doi: 10.1002/mrm.22595.

[140] S. Setzer, G. Steidl, and T. Teuber, “Infimal convolution regularizations with discrete $\ell_1$-type functionals,” 797–827, Comm. Math. Sci., vol. 9, no. 3, 2011. doi: 10.4310/CMS.2011.v9.n3.a7.

[141] M. D’Elia, J. C. D. los Reyes, and A. M. Trujillo. (Apr. 29, 2020). “Bilevel parameter optimization for learning nonlocal image denoising models.” arXiv: 1912.02347 [math].

[142] F. Sherry, M. Benning, J. C. De los Reyes, M. J. Graves, G. Maierhofer, G. Williams, C.-B. Schonlieb, and M. J. Ehrhardt, “Learning the sampling pattern for MRI,” pp. 4310–4321, IEEE Transactions on Medical Imaging, vol. 39, no. 12, Dec. 2020. doi: 10.1109/TMI.2020.3017353.

[143] B. Gozcu, R. K. Mahabadi, Y.-H. Li, E. Ilicak, T. Cukur, J. Scarlett, and V. Cevher, “Learning-based compressive MRI,” 1394–406, IEEE Trans. Med. Imag., vol. 37, no. 6, Jun. 2018. doi: 10.1109/TMI.2018.2832540.

[144] N. Shlezinger, J. Whang, Y. C. Eldar, and A. G. Dimakis. (Dec. 15, 2020). “Model-based deep learning.” arXiv: 2012.08405 [cs, eess].

[145] V. Monga, Y. Li, and Y. C. Eldar, “Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing,” pp. 18–44, IEEE Signal Processing Magazine, vol. 38, no. 2, Mar. 2021. doi: 10.1109/MSP.2020.3016905.

[146] K. Gregor and Y. LeCun, “Learning fast approximations of sparse coding,” in Proc. Intl. Conf. Mach. Learn, 2010. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/gregor-icml-10.pdf.

[147] W. Bian, Y. Chen, and X. Ye, “Deep parallel MRI reconstruction network without coil sensitivities,” Aug. 18, 2020. arXiv: 2008.01410 [cs, eess].

[148] K. Hammernik, T. Klatzer, E. Kobler, M. P. Recht, D. K. Sodickson, T. Pock, and F. Knoll, “Learning a variational network for reconstruction of accelerated MRI data,” pp. 3055–3071, Magnetic Resonance in Medicine, vol. 79, no. 6, 2018. doi: 10.1002/mrm.26977.

[149] H. Lim, I. Y. Chun, Y. K. Dewaraja, and J. A. Fessler, “Improved low-count quantitative PET reconstruction with an iterative neural network,” pp. 3512–3522, IEEE Transactions on Medical Imaging, vol. 39, no. 11, Nov. 2020. doi: 10.1109/TMI.2020.2998480.

[150] S. V. Venkatkrishnan, C. A. Bouman, and B. Wohlberg, “Plug-and-Play priors for model based reconstruction,” in 2013 IEEE Global Conference on Signal and Information Processing, pp. 945–948, IEEE, Dec. 2013. doi: 10.1109/GlobalSIP.2013.6737048.
[151] J. Eckstein and D. P. Bertsekas, “On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators,” 293–318, Mathematical Programming, vol. 55, no. 1-3, Apr. 1992. doi: 10.1007/BF01581204.

[152] J. He, Y. Yang, Y. Wang, D. Zeng, Z. Bian, H. Zhang, J. Sun, Z. Xu, and J. Ma, “Optimizing a parameterized plug-and-play ADMM for iterative low-dose CT reconstruction,” pp. 371–382, IEEE Transactions on Medical Imaging, vol. 38, no. 2, Feb. 2019. doi: 10.1109/TMI.2018.2865202.

[153] C. Crockett, D. Hong, I. Y. Chun, and J. A. Fessler, “Incorporating handcrafted filters in convolutional analysis operator learning for ill-posed inverse problems,” in 2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), pp. 316–320, Dec. 2019. doi: 10.1109/CAMSAP45676.2019.9022669.

[154] H. H. Barrett, “Objective assessment of image quality: Effects of quantum noise and object variability,” 1266–1278, J. Opt. Soc. Am. A, vol. 7, no. 7, Jul. 1990. doi: 10.1364/JOSAA.7.001266.

[155] A. Yendiki and J. A. Fessler, “Analysis of observer performance in unknown-location tasks for tomographic image reconstruction,” B99–109, J. Opt. Soc. Am. A, vol. 24, no. 12, Dec. 2007. doi: 10.1364/JOSAA.24.000B99.

[156] F. K. Kopp, M. Catalano, D. Pfeiffer, A. A. Fingerle, E. J. Rummeny, and P. B. Noel, “CNN as model observer in a liver lesion detection task for x-ray computed tomography: A phantom study,” 4439–47, Med. Phys., vol. 45, no. 10, Oct. 2018. doi: 10.1002/mp.13151.

[157] C. Crockett and J. A. Fessler, “Motivating bilevel approaches to filter learning: A case study,” in 2021 IEEE International Conference on Image Processing (ICIP), pp. 2803–2807, IEEE, Sep. 19, 2021. doi: 10.1109/ICIP42928.2021.9506489.

[158] C. Christof, “Gradient-based solution algorithms for a class of bilevel optimization and optimal control problems with a nonsmooth lower level,” pp. 290–318, SIAM Journal on Optimization, vol. 30, no. 1, Jan. 2020. doi: 10.1137/18M1225707.

[159] J. Kaipioa and E. Somersalo, “Statistical inverse problems: Discretization, model reduction and inverse crimes,” 493–504, J. Comp. Appl. Math., vol. 198, no. 2, Jan. 2007. doi: 10.1016/j.cam.2005.09.027.

[160] J. A. Fessler and W. L. Rogers, “Spatial resolution properties of penalized-likelihood image reconstruction methods: Space-invariant tomographs,” 1346–58, IEEE Trans. Im. Proc., vol. 5, no. 9, Sep. 1996. doi: 10.1109/83.535846.

[161] J. Qi and R. H. Huesman, “Penalized maximum-likelihood image reconstruction for lesion detection,” 4017–30, Phys. Med. Biol., vol. 51, no. 16, Aug. 2006. doi: 10.1088/0031-9155/51/16/009.
[162] L. Yang, J. Zhou, A. Ferrero, R. D. Badawi, and J. Qi, “Regularization design in penalized maximum-likelihood image reconstruction for lesion detection in 3D PET,” 403–20, *Phys. Med. Biol.*, vol. 59, no. 2, Jan. 2014. DOI: 10.1088/0031-9155/59/2/403.

[163] B. Sahiner, A. Pezeshk, L. M. Hadjiiski, X. Wang, K. Drukker, K. Cha, R. Summers, and M. L. Giger, “Deep learning in medical imaging and radiation therapy,” *Medical Physics*, Nov. 2018. DOI: 10.1002/mp.13264.

[164] FDA, 510k premarket notification of Deep Learning Image Reconstruction (GE Medical Systems), 2019. [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfpmn/pmn.cfm?ID=K183202.

[165] J. Solomon, P. Lyu, D. Marin, and E. Samei, “Noise and spatial resolution properties of a commercially available deep learning-based CT reconstruction algorithm,” 3961–71, *Med. Phys.*, vol. 47, no. 9, 2020. DOI: 10.1002/mp.14319.

[166] C. Garcia-Cardona and B. Wohlberg, “Convolutional dictionary learning: A comparative review and new algorithms,” pp. 366–381, *IEEE Transactions on Computational Imaging*, vol. 4, no. 3, Sep. 2018. DOI: 10.1109/TCI.2018.2840334.

[167] I. Y. Chun and J. A. Fessler, “Convolutional dictionary learning: Acceleration and convergence,” 1697–712, *IEEE Trans. Im. Proc.*, vol. 27, no. 4, Apr. 2018. DOI: 10.1109/TIP.2017.2761545.