CrossMark

Techniques for Gradient-Based Bilevel Optimization with Non-smooth Lower Level Problems

Peter Ochs[1,3] · René Ranftl[2] · Thomas Brox[3] · Thomas Pock[4,5]

Abstract We propose techniques for approximating bilevel optimization problems with non-smooth lower level problems that can have a non-unique solution. To this end, we substitute the expression of a minimizer of the lower level minimization problem with an iterative algorithm that is guaranteed to converge to a minimizer of the problem. Using suitable non-linear proximal distance functions, the update mappings of such an iterative algorithm can be differentiable, notwithstanding the fact that the minimization problem is non-smooth.

1 Introduction

We consider numerical methods for solving bilevel optimization problems of the form:

✉ Peter Ochs
  ochs@mia.uni-saarland.de; ochs@cs.uni-freiburg.de

[1] Mathematical Image Analysis Group, University of Saarland, Saarbrücken, Germany

[2] Visual Computing Lab, Intel Labs, Santa Clara, CA, USA

[3] Computer Vision Group, University of Freiburg, Freiburg im Breisgau, Germany

[4] Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria

[5] Digital Safety & Security Department, AIT Austrian Institute of Technology GmbH, 1220 Vienna, Austria

$$\min_{\vartheta} \ \mathcal{L}(x^*(\vartheta), \vartheta)$$
$$s.t. \ x^*(\vartheta) \in \arg\min_{x \in \mathbb{R}^N} E(x, \vartheta), \tag{1}$$

where $\mathcal{L}$ (denoted *loss function*) is a function penalizing the differences between the output of the lower level problem $x^*(\vartheta)$ and some given ground truth data. In addition, $\mathcal{L}$ can also contain a regularizer on the parameter vector ϑ, e.g., a sparsity prior. The mapping $x^*(\vartheta)$ is the solution of an optimization problem (parametrized by ϑ) that solves a specific task, e.g., multi-label segmentation.

In the general bilevel literature, (1) is often presented as a leader–follower problem. The leader (upper level problem) tries to optimize the next move (minimization of the upper level problem) under consideration of the move of an opponent, the follower. Given some information ϑ to the follower, the leader tries to anticipate the follower's next move (minimization of the lower level problem).

In this paper, we focus on a class of problems that allows for non-smooth convex functions $x \mapsto E(x, \vartheta)$ in the lower level problem, e.g., sparse models based on the ℓ_1-norm. Such models have become very popular in the computer vision, image processing, and machine learning communities since they are robust with respect to noise and outliers in the input data.

Due to the possibly high dimensionality of the parameter vector, we pursue the minimization of the bilevel problem (1) using gradient-based methods. Hence, a descent direction of $\mathcal{L}$ with respect to ϑ must be determined. Its estimation involves the Jacobian of the solution map $x^*(\vartheta)$ with respect to the parameter vector ϑ, which causes three kinds of problems:

(i) The solution mapping $x^*(\vartheta)$ is only defined implicitly (as a minimizer of the lower level problem).

(ii) The lower level's solution is not unique.[1]

(iii) The lower level problem is non-smooth.

(i) A reduction to a single-level problem by explicitly solving the lower level problem is not always possible. Nevertheless, if the lower level problem is sufficiently smooth, sometimes, it can be replaced by its optimality condition, and the implicit function theorem (cf. Sect. 4.1) provides an explicit formula for the derivative of the solution map. This approach does not work for non-smooth lower level problems.

(ii) Consider the example

$$
\begin{aligned}
&\min_{\vartheta \in \mathbb{R}} (x^*(\vartheta) - 1)^2 \\
&s.t. \, x^*(\vartheta) \in \arg \min_{x \in [0,1]} \vartheta x,
\end{aligned}
\tag{2}
$$

which reduces to minimization of a step function

$$
\min_{\vartheta \in \mathbb{R}} \mathcal{L}(x^*(\vartheta)), \quad \mathcal{L}(x^*(\vartheta)) = \begin{cases} 1, & \text{if } \vartheta > 0; \\ 0, & \text{if } \vartheta < 0; \\ [0,1], & \text{if } \vartheta = 0. \end{cases}
$$

A gradient-based method will get stuck almost everywhere, as the derivative vanishes for all $\vartheta \neq 0$. Similar situations arise for robust models in the lower level problem. By definition, the solution is not affected by small perturbations of the input data (or the parameter ϑ). For instance in the multi-label segmentation problem, small changes in the pixel likelihoods do not change the segmentation result; the energy landscape of the loss function will have the form of a high-dimensional step function.

(iii) Due to the non-smoothness of the lower level problem, standard calculus cannot be applied. In variational (non-smooth) analysis, there are many generalizations of derivatives, such as the convex subdifferential, the Fréchet subdifferential, or the limiting subdifferential. However, they are often set-valued and generalizations of the chain rule and rely on constraint qualifications that are sometimes quite restrictive and often hard to verify.

In the conference version of this paper [30], we introduced an approach to overcome the smoothness restriction in some cases of practical interest. The idea is to replace the lower level problem by an iterative algorithm that is guaranteed to converge to a solution of the problem. If the update mapping of the algorithm is a smooth function, the chain rule can be applied to the composition of these update mappings recursively and the exact derivatives with respect to the parameter vector ϑ can be computed. Algorithms based on

Bregman distances are key for this development. The number of iterations of the iterative algorithm steers the approximation quality of the lower level problem.

The iterative algorithm that replaces the lower level is stopped after a small number of iterations. However, once the algorithm and the number of iterations are fixed, the resulting bilevel optimization problem seeks for optimal ϑ for exactly this algorithm and this (fixed) number of iterations. Numerically, the derivative that is involved in gradient-based minimization is exact: the number of chain rule recursions is finite. This is in contrast to an approach based on the optimality condition of a smooth approximation of the lower level problem. In this case, the descent direction is based on the derivative of the optimality condition evaluated at the minimizer of the lower level problem, which is only determined approximately.

Beyond the analysis of the conference paper, we discuss approximations to the derivative evaluation that reduce the memory requirements and the computational cost significantly. We extend the class of problems that can be used in our framework and give some more details about the general implementation of our approach. Moreover, we consider the limiting case, i.e., the fixed point equation of an iterative algorithm in the lower level problem.

We point out several applications of our approach and evaluate it for a multi-label segmentation problem coupled with a convolutional neural network.

2 Related Work

We propose a simple approximation of the lower level problem that naturally addresses *non-smoothness* and *non-uniqueness*.

For a non-unique solution map (a set-valued mapping) of the lower level problem (1) is not even well defined (cf. Remark 1). Dempe et al. [14] describe three possible options to cope with this problem. The *optimistic bilevel optimization problem* assumes a cooperative strategy of leader and follower, i.e., in case of multiple solutions the follower tries to minimize the upper level objective. The *pessimistic bilevel problem* is the other extreme, where the leader tries to bound the damage that the follower could cause by its move. The *selection function approach* assumes that the leader can always predict the follower's choice. Of course, these three approaches are the same for lower level problems with a unique output.

Our approach does not fall into any of the three cases; however, the selection function approach is the closest. The difference is that our approximation changes the output also at (originally) unique points. Our solution strategy reduces the solution map to be single-valued, similar to the approaches mentioned above.

---

[1] Note that the bilevel problem as in (1) is not well-defined in this case. We discuss some details in Sect. 3.

Dempe et al. [14] classify the following optimality conditions.[2] The *primal Karush–Kuhn–Tucker (KKT) transformation* replaces the lower level problem by the necessary and sufficient optimality condition for a convex function. The equivalence to the original problem is shown in [15]. The *classical KKT transformation* substitutes the lower level problem with the classical KKT conditions. Due to the extra variable, the problems are not fully equivalent anymore (see [14]). This approach, which leads to a non-smooth mathematical problem with complementary constraints (MPEC), is the most frequently used one. The third approach is the *optimal value transform*, which introduces a constraint that bounds the lower level objective by the optimal value function.

Our approach is—in the limit—motivated by the first class of the primal KKT transformation. We consider the fixed point equation of an algorithm, which represents the optimality condition without introducing additional variables, and approximate this situation with finitely many iterations of the algorithm.

We focus on gradient-based methods, such as gradient descent, L-BFGS [25], non-linear conjugate gradient [1,19], Heavy-ball method [40], iPiano [29], and others, for solving the bilevel optimization problem. In particular, this paper focuses on the estimation of descent directions. As one option, the gradient can be approximated numerically with finite differences such as in [16]. We rather pursue what is known as algorithmic/automatic differentiation. It is based on the idea to decompose the derivative evaluation into small parts by means of a chain rule, where the analytic derivative of each part is known. A whole branch of research deals with this technique [21]. Obviously, the idea to differentiate an algorithm in the lower level problem is not new [17,37]. The difference is that our algorithm has a smooth update mapping while actually minimizing a non-smooth objective function. Another idea to approach a non-smooth problem with an iterative algorithm is presented in [12], where a chain rule for weak derivatives is used (cf. Sect. 4.4).

The special case of a lower level problem that depends linearly on the parameters is treated by structured output support vector machines [38]. The linear structure of the lower level problem allows the construction of an upper bound of the upper level objective function, which needs to be minimized. In general, this approach is only an approximation to the bilevel problem in (1), which can be solved using subgradient descent.

There are several practical examples of bilevel optimization in the computer vision and machine learning. Bilevel optimization was considered for task-specific sparse analysis prior learning [32] and applied to signal restoration. In [10,11,23], a bilevel approach was used to learn a model of natural image statistics, which was then applied to various image restoration tasks. A variational formulation for learning a good noise model was addressed in [35] in a PDE-constrained optimization framework, with some follow-up works [6,7,34]. In machine learning, bilevel optimization was used to train an SVM [4] and other techniques [27]. Recently, it was used for the end-to-end training of a Convolutional Neural Network (CNN) and a graphical model for binary image segmentation [33] (cf. Sect. 8).

Finally, we refer to [13] for an annotated bibliography with many references regarding the theoretical and practical development in bilevel optimization.

Preliminaries

We work in a Euclidean vector space $\mathbb{R}^N$ of dimension N equipped with the standard Euclidean norm $\| \cdot \| := \sqrt{\langle \cdot, \cdot \rangle}$ that is induced by the standard inner product. We use the notation $\overline{\mathbb{R}} := \mathbb{R} \cup \{\infty\}$ to denote the extended real numbers.

We use the notation $[x * a]$ for $x, a \in \mathbb{R}^N$ to denote the set $\{x \in \mathbb{R}^N \,|\, \forall i: \; x_i * a_i\}$, where $* \in \{<, \leq, =, \geq, >\}$ is a binary relation on $\mathbb{R} \times \mathbb{R}$. For example, $[x \geq 0]$ denotes the non-negative orthant in $\mathbb{R}^N$.

3 The Bilevel Problem

We consider bilevel optimization problems of the form:

$$
\begin{aligned}
&\min_{\vartheta \in \mathbb{R}^P} \; \mathcal{L}(x^*(\vartheta), \vartheta) + \ell(\vartheta) \\
&s.t. \; x^*(\vartheta) \in \arg\min_{x \in \mathbb{R}^N} E(x, \vartheta).
\end{aligned}
\tag{3}
$$

The function $\ell \colon \mathbb{R}^P \to \overline{\mathbb{R}}$ is assumed to be proper, lower semi-continuous, convex, and "prox-friendly"[3] and the function $\mathcal{L} \colon \mathbb{R}^N \times \mathbb{R}^P \to \mathbb{R}$ to be continuously differentiable on dom ℓ. The optimization variable is the (parameter) vector $\vartheta \in \mathbb{R}^P$. It appears implicitly and explicit in the upper level problem. It is implicit via the solution mapping $x^*(\vartheta) \in \mathbb{R}^N$ of the lower level problem and explicit in ℓ and in the second argument of $\mathcal{L}$. The lower level is a minimization problem in the first variable of a proper, lower semi-continuous function $E \colon \mathbb{R}^N \times \mathbb{R}^P \to \overline{\mathbb{R}}$. For each $\vartheta \in \mathbb{R}^P$, the objective function (energy) $x \mapsto E(x, \vartheta)$ is assumed to be convex.

Note that our formulation includes constrained optimization problems in the upper and lower level problems. The functions ℓ and E are defined as extended-valued (real) functions. Of course, in order to handle the constraints efficiently in the algorithm, the constraint sets should not be too complicated.

---

[2] The classification in [14] applies to the optimistic bilevel problem.

[3] The associated proximity operator has a closed-form solution or the solution may be determined efficiently numerically.

In order to solve the optimization problem in (3), we can apply iPiano [29], a gradient-based algorithm that can handle the non-smooth part $\ell(\vartheta)$. The extension of iPiano in [28, Chapter 6] allows for a prox-bounded (non-convex, non-smooth) function $\ell(\vartheta)$. Informally, the update step of this algorithm (for the parameter vector ϑ) reads

$$\vartheta^{k+1} \in \text{prox}_{\alpha_k \ell} \left(\vartheta^k - \alpha_k \nabla_\vartheta \mathcal{L}(x^*(\vartheta^k), \vartheta^k) \right.$$
$$\left. + \beta_k(\vartheta^k - \vartheta^{k-1}) \right), \qquad (4)$$

where $\text{prox}_{\alpha_k \ell}$ denotes the proximity operator of the function ℓ, α_k is a step size parameter, and β_k steers the so-called inertial effect of the algorithm (usually $\beta_k \in [0, 1]$). For details about α_k and β_k, we refer to [28,29], where convergence to a stationary point (a zero in the limiting subdifferential) is proved under mild assumptions. We could also apply proximal gradient descent (forward–backward splitting) [2] ($\beta_k = 0$). In our experiments, iPiano was usually faster and less sensitive to local optima, however. If the non-smooth term is not present, several gradient-based solvers can be used [1,19,25,40].

The structure of the update step in (4) points out that the main aspect in applying such gradient-based algorithm is the evaluation of the gradient $\nabla_\vartheta \mathcal{L}(x^*(\vartheta), \vartheta)$. The remainder of this paper deals with exactly this problem: compute $\nabla_\vartheta \mathcal{L}(x^*(\vartheta), \vartheta)$ with a solution mapping $\vartheta \mapsto x^*(\vartheta)$ of a possibly non-smooth objective function in the lower level. Note that the following approximations naturally yield or require a unique solution of the lower level problem.

Remark 1 The formulation (3) of a bilevel optimization problem only makes sense when $\arg\min_{x \in \mathbb{R}^N} E(x, \vartheta)$ yields a unique minimizer. In that case, optimality of the bilevel problem can be derived from standard optimality conditions in non-linear programming. If the lower level problem does not provide a unique solution, the loss function $\mathcal{L}$ must be defined on the power set of $\mathbb{R}^N$ and a different notion of optimality must be introduced. Since this results in problems beyond the scope of this paper, we refer to [14]. A common circumvention is to consider the corresponding optimistic bilevel problem.

4 Computing Descent Directions

For a given parameter value $\vartheta \in \mathbb{R}^P$, we would like to compute a descent direction of $\mathcal{L}$ in (3) with respect to ϑ to find a numerical solution using some gradient-based method. Obviously, we need the derivative of the solution map $x^*(\vartheta)$ with respect to ϑ. In the following, we present strategies to approximate the (possibly non-smooth) lower level problem and to compute a descent direction.

4.1 Derivative of a Smoothed Lower Level Problem

If the objective function of the lower level problem of (3) can be approximated well with a twice continuously differentiable function (again denoted E), we can make use of the implicit function theorem to find the derivative of the solution map with respect to ϑ. The optimality condition of the lower level problem is $\nabla_x E(x, \vartheta) = 0$, which under some conditions implicitly defines a function $x^*(\vartheta)$. As we assume that the problem $\min_x E(x, \vartheta)$ has a solution, there is (x^*, ϑ) such that $\nabla_x E(x^*, \vartheta) = 0$. Then, under the conditions that $\nabla_x E(x^*, \vartheta)$ is continuously differentiable and $(\partial(\nabla_x E)/\partial x)(x^*, \vartheta)$ is invertible, there exists an explicit function $x^*(\vartheta)$ defined on a (open) neighborhood of x^*. Moreover, the function $x^*(\vartheta)$ is continuously differentiable at ϑ and it holds that

$$\frac{\partial x^*}{\partial \vartheta}(\vartheta) = \left(-\frac{\partial(\nabla_x E)}{\partial x}(x^*(\vartheta), \vartheta) \right)^{-1} \frac{\partial(\nabla_x E)}{\partial \vartheta}(x^*(\vartheta), \vartheta).$$

Using the Hessian $H_E(x^*(\vartheta), \vartheta) := \frac{\partial^2 E}{\partial x^2}(x^*(\vartheta), \vartheta)$ yields

$$\frac{\partial x^*}{\partial \vartheta}(\vartheta) = -(H_E(x^*(\vartheta), \vartheta))^{-1} \frac{\partial^2 E}{\partial \vartheta \partial x}(x^*(\vartheta), \vartheta). \qquad (5)$$

The requirement for using (5) from the implicit function theorem is the continuous differentiability of $\partial E/\partial x$ and the invertibility of H_E. Application of the chain rule yields the total derivative of the loss function $\mathcal{L}$ of (3) w.r.t. ϑ

$$\frac{d\mathcal{L}}{d\vartheta} = -\left[\frac{\partial \mathcal{L}}{\partial x} H_E^{-1} \right] \frac{\partial^2 E}{\partial \vartheta \partial x} + \frac{\partial \mathcal{L}}{\partial \vartheta}, \qquad (6)$$

where the function evaluation at $(x^*(\vartheta), \vartheta)$ is dropped for brevity. A clever way of setting parentheses, as it is indicated by the squared brackets, avoids explicit inversion of the Hessian matrix. However, for large problems iterative solvers are required.

4.2 Derivative of Iterative Algorithms

We can replace the minimization problem in the lower level of (3) by an algorithm that solves this problem, i.e., the lower level problem is replaced by an equality constraint. This approach shows three advantages: (i) after approximating the lower level of (3) by an algorithm, the approach is exact; and (ii) the update step of the algorithm can be smooth without the lower level problem to be smooth; (iii) the output is always unique (for a fixed initialization), which circumvents the critical issue of a non-unique lower level solution.

Let $\mathcal{A}$ and $\mathcal{A}^{(n)} \colon X \times \mathbb{R}^P \to X$ describe one or n iterations, respectively, of algorithm $\mathcal{A}$ for minimizing E in (3). For simplicity, we assume that the feasible set mapping

$\vartheta \mapsto \{x \in \mathbb{R}^N \mid (x, \vartheta) \in \mathrm{dom}\, \mathcal{L}\}$ is constant,[4] i.e., the same X is assigned to all $\vartheta \in \mathbb{R}^P$. Note that $X = \mathbb{R}^N$ is permitted.

For fixed $n \in \mathbb{N}$, we replace (3) by

$$\min_{\vartheta} \; \mathcal{L}(x^*(\vartheta), \vartheta) + \ell(\vartheta)$$
$$s.t. \; x^*(\vartheta) = \mathcal{A}^{(n+1)}(x^{(0)}, \vartheta), \tag{7}$$

where $x^{(0)}$ is some initialization of the algorithm. The solution map of the lower level problem $x^*(\vartheta)$ is the output of the algorithm $\mathcal{A}$ after $n + 1$ iterations. If we write down one iteration of the algorithm, i.e., $x^{(n+1)}(\vartheta) = \mathcal{A}(x^{(n)}(\vartheta), \vartheta)$, we have to assume that $x^{(n)}$ depends on the choice of ϑ. However, this dependency can be dropped for the first iterate, which emerges from the initialization.

A suitable algorithm has the properties that $x^{(n)}(\vartheta)$ converges pointwise (i.e., for each ϑ) to a solution of the lower level problem as n goes to infinity and $E(x^{(n)}, \vartheta) = E(\mathcal{A}^{(n)}(x^{(0)}, \vartheta), \vartheta) \to \min_x E(x, \vartheta)$ for $n \to \infty$. Note that for Bregman proximity functions in algorithm $\mathcal{A}$, the solution for $n \to \infty$ could lie on $\mathrm{bdry}(X)$, despite $x^{(n)} \in \mathrm{int}(X)$ for all n. However, this matters only for an asymptotic analysis.

If $\mathcal{A}$ is (totally) differentiable with respect to ϑ, then, by the standard chain rule, $\mathcal{A}^{(n)}$ is differentiable with respect to ϑ as well. This way, we obtain a totally differentiable approximation to the lower level problem of (3), where the approximation quality can simply be controlled by the number of iterations. For the so-called descent algorithms, it holds that

$$E(x^{(n+1)}, \vartheta) - \min_x E(x, \vartheta) \le E(x^{(n)}, \vartheta) - \min_x E(x, \vartheta).$$

A large number of iterations usually approximate the minimum of E better than a small number of iterations.

Nevertheless, also the small number of iterations is interesting for our approach. Once the certain number of iterations is fixed, the bilevel optimization problem seeks for an optimal performance with exactly this chosen number of iterations. Solving the bilevel optimization problem accurately with a small number of iterations n of the lower level algorithm can result in a better performance than a poorly solved bilevel problem with a large number of iterations in the lower level.

Our approach is well suited for minimizing the bilevel problem using gradient-based methods. The differentiation of $\mathcal{L}$ with respect to ϑ in (7) is exact; once an algorithm is selected, no additional approximation is required for computing the derivatives. In contrast, the smoothing approach from Sect. 4.1 requires the minimization of a smooth objective function, the solution of which can be found only approxi-

mately. Therefore, the descent direction, which is based on the optimality condition, is always erroneous.

The "smoothing parameter" in our approach is the number of iterations of the algorithm that replaces the lower level problem. Since the algorithm's update mapping is assumed to be smooth, in particular, locally Lipschitz continuous, which formally means that

$$\|\mathcal{A}(x, \vartheta) - \mathcal{A}(y, \vartheta)\| \le \mathrm{const.}\, \|x - y\|$$

holds in a neighborhood of the initial point, the variation of the output after one iteration is limited. Therefore, intuitively, for a large number of iterations n, less smoothness of $\mathcal{A}^{(n)}$ can be expected.

In order to obtain the derivative of the lower level problem of (7), there are two prominent concepts: forward mode and backward mode. For any vector $\xi \in \mathbb{R}^N$, the *forward mode* corresponds to evaluating the derivative as

$$\xi^\top \frac{dx^{(n+1)}}{d\vartheta}(\vartheta)$$
$$= \xi^\top \left[\frac{\partial \mathcal{A}}{\partial x}(x^{(n)}, \vartheta) \frac{dx^{(n)}}{d\vartheta}(\vartheta) \right] + \xi^\top \frac{\partial \mathcal{A}}{\partial \vartheta}(x^{(n)}, \vartheta), \tag{8}$$

whereas the *backward mode/reverse mode* evaluates the derivative as

$$\left(\frac{dx^{(n+1)}}{d\vartheta}(\vartheta) \right)^\top \xi$$
$$= \left(\frac{dx^{(n)}}{d\vartheta}(\vartheta) \right)^\top \left[\left(\frac{\partial \mathcal{A}}{\partial x}(x^{(n)}, \vartheta) \right)^\top \xi \right]$$
$$+ \left(\left(\frac{\partial \mathcal{A}}{\partial \vartheta}(x^{(n)}, \vartheta) \right)^\top \xi \right), \tag{9}$$

where the squared brackets symbolize the different orders of evaluating the terms. In both approaches, replacing and evaluating the term $dx^{(n)}/d\vartheta$ using the preceding iterate $(n-1)$ is done in the respective order.

Mathematically both concepts result in the same solution. However, numerically the approaches are very different. The reverse mode is usually more efficient when the optimization variable ϑ is high dimensional (i.e., P is large) and the range of the objective function $\mathcal{L}$ is low dimensional—it is always 1 in our setting. This corresponds to ξ being a column vector instead of a derivative matrix. The forward mode is often easier to implement, since it is executed in the same order as the optimization algorithm itself and can be computed online, i.e., during the iteration of the algorithm. As a downside, each partial derivative must be initialized and propagated through the iterations. Therefore, the memory requirement is vastly increasing with the dimension P. We

---

[4] More generally, the concept of outer semi-continuity of the feasible set mapping is needed, otherwise a gradient-based method could converge to a non-feasible point.

focus on the reverse mode for evaluating the derivatives, due to its computationally more appealing nature.

The backward mode is executed in the reverse order of the iterations of the algorithm and needs the optimum x^*, which is $x^{(n+1)}$ in our case, for executing the first matrix vector multiplication. All intermediate results toward the optimum must be available. The implementation of the backward mode (9) is shown in Algorithm 1.

Algorithm 1 *Derivative of an abstract algorithm*

- Assumptions: $\mathcal{A}$ *is totally differentiable.*
- Initialization at $n + 1$:

$$z^{(n+1)} := \left(\frac{\partial \mathcal{L}}{\partial x}(x^*(\vartheta), \vartheta) \right)^\top \in \mathbb{R}^N \quad and \quad w^{(n+1)} := 0 \in \mathbb{R}^P$$

- Iterations ($n \geq 0$): *Update*

    for n to 0 :

$$w^{(n)} = w^{(n+1)} + \left(\frac{\partial \mathcal{A}}{\partial \vartheta}(x^{(n)}, \vartheta) \right)^\top z^{(n+1)}$$

$$z^{(n)} = \left(\frac{\partial \mathcal{A}}{\partial x^{(n)}}(x^{(n)}, \vartheta) \right)^\top z^{(n+1)}$$

- Final derivative of $\mathcal{L}$ in (7) w.r.t. ϑ:

$$\frac{d\mathcal{L}}{d\vartheta}(x^*(\vartheta), \vartheta) = (w^{(0)})^\top + \frac{\partial \mathcal{L}}{\partial \vartheta}(x^*(\vartheta), \vartheta).$$

This approach is quite expensive. But, for a reasonable number of iterations, it is still practical. It is still faster than the inversion of the Hessian matrix in Sect. 4.1; see (6). In the following section, we present approximations that reduce the cost significantly.

4.3 Derivative of Fixed Point Equations

We generalize the result from Sect. 4.1, where the lower level problem of (3) is replaced by the first-order optimality condition of a smooth approximation. The idea is to consider a different optimality condition. A point is optimal, if it satisfies the fixed point equation of an algorithm $\mathcal{A}: X \times \mathbb{R}^P \to X$ solving the original lower level problem, i.e., we address the bilevel problem:

$$\min_{\vartheta} \; \mathcal{L}(x^*(\vartheta), \vartheta) \tag{10}$$
$$s.t. \; x^*(\vartheta) = \mathcal{A}(x^*(\vartheta), \vartheta),$$

where $X \subset \mathbb{R}^N$ is as in Sect. 4.2 and we have a fixed point x^*. This approach is more general than the one in Sect. 4.1, since we could actually first smoothly approximate the lower level problem and then consider the fixed point equation. For

many algorithms, both approaches are equivalent, because optimization algorithms are often derived from the first-order optimality condition.

Following the idea of Sect. 4.2, we can consider a differentiable fixed point equation without the lower level problem to be differentiable. An algorithm that has a differentiable update rule yields a differentiable fixed point equation.

Assume that (x^*, ϑ) solves the fixed point equation. By differentiating the fixed point equation, we obtain

$$\frac{\mathrm{d}x}{\mathrm{d}\vartheta}(\vartheta) = \frac{\partial \mathcal{A}}{\partial x}(x^*(\vartheta), \vartheta) \frac{dx}{d\vartheta}(\vartheta) + \frac{\partial \mathcal{A}}{\partial \vartheta}(x^*(\vartheta), \vartheta),$$

which can be rearranged to yield

$$\frac{\mathrm{d}x}{\mathrm{d}\vartheta}(\vartheta) = \left(I - \frac{\partial \mathcal{A}}{\partial x}(x^*(\vartheta), \vartheta) \right)^{-1} \frac{\partial \mathcal{A}}{\partial \vartheta}(x^*(\vartheta), \vartheta). \tag{11}$$

Assuming that the spectral radius of $(\partial \mathcal{A}/\partial x)(x^*(\vartheta), \vartheta)$ is smaller than 1, we can approximate the inversion using the geometric series:

$$\frac{\mathrm{d}x}{\mathrm{d}\vartheta}(\vartheta) = \sum_{n=0}^{\infty} \left(\frac{\partial \mathcal{A}}{\partial x}(x^*(\vartheta), \vartheta) \right)^n \frac{\partial \mathcal{A}}{\partial \vartheta}(x^*(\vartheta), \vartheta),$$

where $((\partial \mathcal{A}/\partial x)(x^*(\vartheta), \vartheta))^n$ means the n-fold matrix product with itself. Let us approximate this term with a finite summation of $0, \ldots, n_0$. Then by a simple rearrangement, for $\xi \in \mathbb{R}^N$, we have (by abbreviating $(\partial \mathcal{A}/\partial x)(x^*(\vartheta), \vartheta)$ by $\partial \mathcal{A}/\partial x$; the same for $\partial \mathcal{A}/\partial \vartheta$)

$$\xi^\top \frac{\mathrm{d}x}{\mathrm{d}\vartheta}(\vartheta) \approx \xi^\top \sum_{n=0}^{n_0} \left(\frac{\partial \mathcal{A}}{\partial x}\right)^n \frac{\partial \mathcal{A}}{\partial \vartheta}$$

$$= \xi^\top \frac{\partial \mathcal{A}}{\partial x}\left(\frac{\partial \mathcal{A}}{\partial \vartheta} + \frac{\partial \mathcal{A}}{\partial x}\left(\frac{\partial \mathcal{A}}{\partial \vartheta} + \cdots\right)\right)$$

$$= \xi^\top \left[\frac{\partial \mathcal{A}}{\partial x^{(n_0)}}\frac{dx^{(n_0)}}{d\vartheta}\right] + \xi^\top \frac{\partial \mathcal{A}}{\partial \vartheta}.$$

The difference between the last line in this equation and (8) and (9) is the evaluation point of the terms. While in (8) and (9) the terms for $dx^{(n+1)}/d\vartheta$ are evaluated at $(x^{(n)}(\vartheta), \vartheta)$, here, all terms are evaluated at $(x^*(\vartheta), \vartheta)$. Although the condition on the spectral radius is rarely met in practice, this approximation works well empirically and needs to store only the optimum of the algorithm. This leads to an immense reduction of the memory requirements.

4.4 Weak Differentiation of Iterative Algorithms

The approach in [12] also considers an algorithm replacing the non-smooth lower level problem. Their underlying methodology, however, is based on weak differentiability, which can be guaranteed for Lipschitz continuous mappings thanks to Rademacher's theorem. If all iteration mappings are Lipschitz continuous with respect to the iteration variable and the parameter ϑ, weak differentiability follows from the chain rule for Lipschitz mappings [18, Theorem 4]. For details, we refer to [12], in particular Sect. 4.

5 Explicit Derivatives for Exemplary Algorithms

The framework of Bregman proximity functions is key for the idea to approximate a non-smooth optimization problem by an algorithm with smooth update mappings. In this section, we instantiate two such algorithms. Details and examples of Bregman proximity functions are postponed to Sect. 6.1. For understanding this section, it suffices to know that $D_\psi(x, \bar{x})$ provides a distance measure between two points x and $\bar{x}$, and it can be used to define a Bregman proximity operator prox^ψ which generalizes the common proximity operator that is based on the Euclidean distance.

5.1 Derivative of Forward–Backward Splitting

Let us consider forward–backward splitting [24,31] with Bregman proximity function D_ψ (e.g., [3]). It applies to minimization problems of the form:

$$\min_{x \in \mathbb{R}^N} f(x) + g(x),$$

where $f : \mathbb{R}^N \to \mathbb{R}$ is a continuously differentiable, convex function with Lipschitz continuous gradient and $g : \mathbb{R}^N \to \overline{\mathbb{R}}$ is a proper, lower semi-continuous, convex function with a (Bregman) proximity operator that is easy to evaluate. The update rule of the forward–backward splitting we consider is

$$
\begin{aligned}
x^{(n+1)} &= \arg\min_{x \in \mathbb{R}^N} g(x; \vartheta) + f(x^{(n)}; \vartheta) \\
&\quad + \left\langle \nabla f(x^{(n)}; \vartheta), x - x^{(n)}\right\rangle + \frac{1}{\alpha} D_\psi(x, x^{(n)}) \\
&=: \mathrm{prox}_{\alpha g}^\psi\left(\nabla \psi(x^{(n)}) - \alpha \nabla f(x^{(n)}; \vartheta); \vartheta\right) \\
&=: \mathrm{prox}_{\alpha g}^\psi\left(y^{(n)}(x^{(n)}; \vartheta); \vartheta\right),
\end{aligned}
\tag{12}
$$

where we denote $y^{(n)}(x^{(n)}; \vartheta) := \nabla \psi(x^{(n)}) - \alpha \nabla f(x^{(n)}; \vartheta)$, the intermediate result after the forward step. The implementation of the reverse mode for determining the derivative of the solution map of the lower level problem with respect to ϑ is given in Algorithm 2.

Algorithm 2 *Derivative of a forward–backward splitting algorithm*

– Assumptions*: $\mathrm{prox}_{\alpha g}^{\psi}$ and $\mathrm{id} + \alpha \nabla f$ are totally differentiable.*
– Initialization at $n + 1$*:*

$$z^{(n+1)} := \left(\frac{\partial \mathcal{L}}{\partial x}(x^*(\vartheta), \vartheta) \right)^{\top} \in \mathbb{R}^N \quad and \quad w^{(n+1)} := 0 \in \mathbb{R}^P$$

– Iterations ($n \geq 0$)*: Update (where derivatives of $\mathrm{prox}_{\alpha g}^{\psi}$ are evaluated at $(y^{(n)}, \vartheta)$ and derivatives of ∇f at $(x^{(n)}; \vartheta)$)*

```
for n to 0:
```
$$\left| \begin{array}{l} w^{(n)} = w^{(n+1)} + \left(\left(\frac{\partial \, \mathrm{prox}_{\alpha g}^{\psi}}{\partial \vartheta} \right)^{\top} + \left(-\alpha \frac{\partial (\nabla f)}{\partial \vartheta} \right)^{\top} \left(\frac{\partial \, \mathrm{prox}_{\alpha g}^{\psi}}{\partial y} \right)^{\top} \right) z^{(n+1)} \\[3ex] z^{(n)} = \left(\mathrm{id} - \alpha \frac{\partial (\nabla f)}{\partial x} \right)^{\top} \left(\frac{\partial \, \mathrm{prox}_{\alpha g}^{\psi}}{\partial y} \right)^{\top} z^{(n+1)} \end{array} \right.$$

– Final derivative of $\mathcal{L}$ in (7) w.r.t. ϑ*:*

$$\frac{d\mathcal{L}}{d\vartheta}(x^*(\vartheta), \vartheta) = (w^{(0)})^{\top} + \frac{\partial \mathcal{L}}{\partial \vartheta}(x^*(\vartheta), \vartheta).$$

5.2 Derivative of Primal–Dual Splitting

Since the primal–dual algorithm with Bregman proximity functions from [9] provides us with a flexible tool, we specify the implementation of the reverse mode for this algorithm. It applies to the convex–concave saddle-point problem

$$\min_x \max_y \langle Kx, y \rangle + f(x) + g(x) - h^*(y),$$

which is derived from $\min_x f(x) + g(x) + h(Kx)$, where f is convex and has a Lipschitz continuous gradient and g, h are proper, lower semi-continuous convex functions with simple proximity operator for g and for the convex conjugate h^*.

Let the forward iteration of the primal–dual algorithm with variables $x^{(n)} = (u^{(n)}, p^{(n)}) \in \mathbb{R}^{N_u + N_p}$ be given as

$$\begin{aligned} u^{(n+1)} &= \mathcal{PD}_u(u^{(n)}, p^{(n)}, \vartheta) \\ &:= \arg\min_u \left\langle \nabla f(u^{(n)}), u - u^{(n)} \right\rangle + g(u) \\ &\quad + \left\langle Ku, p^{(n)} \right\rangle + \tfrac{1}{\tau} D_u(u, u^{(n)}) \\ p^{(n+1)} &= \mathcal{PD}_p(2u^{(n+1)} - u^{(n)}, p^{(n)}, \vartheta) \\ &:= \arg\min_p h^*(p) - \left\langle K(2u^{(n+1)} - u^{(n)}), p \right\rangle \\ &\quad + \tfrac{1}{\sigma} D_p(p, p^{(n)}), \end{aligned} \tag{13}$$

where f, g, h, K can depend on ϑ. The step size parameter τ and σ must be chosen according to $(\tau^{-1} - L_f)\sigma^{-1} \geq L^2$, where $L = \|K\|$ is the operator norm of K and L_f is the Lipschitz constant of ∇f.

To illustrate the application of the chain rule throughout the primal–dual algorithm, we show a graphical representation of the information flow in Fig. 1, where we use the following abbreviations (analogously for $\mathcal{PD}_p$):

$$\mathcal{PD}_u^{(n)} := \mathcal{PD}_u(u^{(n)}, p^{(n)}, \vartheta);$$

$$\mathcal{PD}_p^{(n)} := \mathcal{PD}_p(2u^{(n+1)} - u^{(n)}, p^{(n)}, \vartheta);$$

$$\partial_u \mathcal{PD}_u := \frac{\partial \mathcal{PD}_u}{\partial u}; \; \partial_p \mathcal{PD}_u := \frac{\partial \mathcal{PD}_u}{\partial p}; \; \partial_\vartheta \mathcal{PD}_u := \frac{\partial \mathcal{PD}_u}{\partial \vartheta}.$$

Remark 2 In Sect. 5.1, we evaluated the forward and the backward steps separately using the chain rule. Of course, this could be done here as well.

Based on this graphical representation, it is easy to derive Algorithm 3.

Fig. 1 The graph shows how the information is backprogated to estimate the derivatives in Algorithm 3. The derivatives at the *nodes* show what derivative is to be evaluated from this point downwards through the graph. The *edges* represent multiplicative (transposed) factors. The final derivative is the sum over all leaf nodes

---
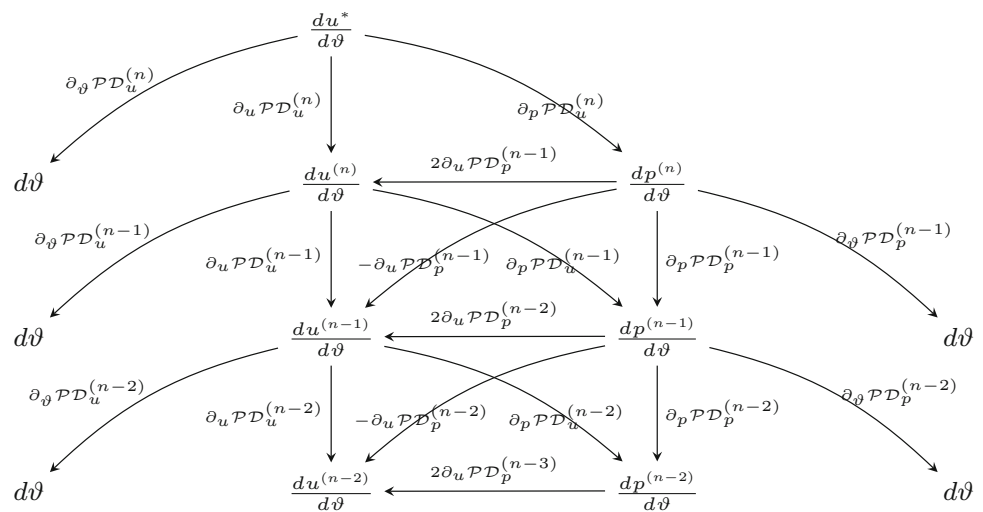
Algorithm 3 *Derivative of a primal–dual algorithm*

– Assumptions*: $\mathcal{PD}_u$ and $\mathcal{PD}_p$ are totally differentiable.*
– Initialization at $n + 1$*:*

$$z^{(n+1)} := \left(\frac{\partial \mathcal{L}}{\partial u}(u^*(\vartheta), \vartheta)\right)^{\top} \in \mathbb{R}^{N_u}, \quad q^{(n+1)} := 0 \in \mathbb{R}^{N_p}$$

and $\quad w^{(n+1)} := 0 \in \mathbb{R}^{P}$

– Iterations ($n \geq 0$)*: Update*

```
for n to 0 :
```
$$w^{(n)} = w^{(n+1)} + \left(\frac{\partial \mathcal{PD}_u^{(n)}}{\partial \vartheta}\right)^{\top} z^{(n+1)} + \left(\frac{\partial \mathcal{PD}_p^{(n)}}{\partial \vartheta}\right)^{\top} q^{(n+1)}$$

$$q^{(n)} = \left(\frac{\partial \mathcal{PD}_u^{(n)}}{\partial p}\right)^{\top} z^{(n+1)} + \left(\frac{\partial \mathcal{PD}_p^{(n)}}{\partial p}\right)^{\top} q^{(n+1)}$$

$$z^{(n)} = \left(\frac{\partial \mathcal{PD}_u^{(n)}}{\partial u}\right)^{\top} z^{(n+1)} + 2\left(\frac{\partial \mathcal{PD}_p^{(n-1)}}{\partial u}\right)^{\top} q^{(n)} - \left(\frac{\partial \mathcal{PD}_p^{(n)}}{\partial u}\right)^{\top} q^{(n+1)}$$

– Final derivative of $\mathcal{L}$ in (7) with $\mathcal{A} = (\mathcal{PD}_u, \mathcal{PD}_p)$ w.r.t. ϑ*:*

$$\frac{d\mathcal{L}}{d\vartheta}(u^*(\vartheta), \vartheta) = (w^{(0)})^{\top} + \frac{\partial \mathcal{L}}{\partial \vartheta}(u^*(\vartheta), \vartheta).$$

A running average is used to implement the ergodic primal–dual algorithm whose output is the average of all iterates, i.e., $u^* = \frac{1}{n+1} \sum_{i=0}^{n} u^{(i)}$: denote $s_u^{(n)} := \frac{1}{n+1} \sum_{i=0}^{n} u^{(i)}$, then $s_u^{(n+1)} = \frac{1}{n+2} u^{(n+1)} + \frac{n+1}{n+2} s_u^{(n)}$. Since the derivative is a linear operator, we can estimate the derivative for the ergodic primal–dual sequence by averaging all $w^{(n)}$. These can be computed as a running average in the loop of Algorithm 3.

6 "Smoothing" Using Bregman Proximity

Splitting-based techniques like those in Sect. 5 usually handle non-smooth terms in the objective function via a (non-linear/Bregman) proximal step. Convex conjugation makes terms in the objective amenable for simple and differentiable proximal mappings. Adding the possibility of considering a primal, primal–dual, or dual formulation yields many examples of practical interest.

In the following, we introduce the class of Bregman functions that can be used in combination with the algorithms in Sect. 5. Then, we discuss a few examples that allow the reformulation of several non-smooth terms arising in applications.

6.1 Bregman Proximity Functions

We consider Bregman proximity functions [5] with the following properties: Let $\psi \colon \mathbb{R}^N \to \overline{\mathbb{R}}$ be a 1-convex function with respect to the Euclidean norm, i.e., it is strongly convex with modulus 1, and denote its domain by $X := \operatorname{dom} \psi$. We assume that ψ is continuously differentiable on the interior of its domain $\operatorname{int}(X)$ and continuous on its closure $\operatorname{cl}(X)$.

Then, ψ generates a Bregman proximity function $D_\psi \colon X \times \operatorname{int}(X) \to \mathbb{R}$ by

$$D_\psi(x, \bar{x}) := \psi(x) - \psi(\bar{x}) - \langle \nabla \psi(\bar{x}), x - \bar{x} \rangle. \qquad (14)$$

For a sequence $(x^n)_{n \in \mathbb{N}}$ converging to $x \in X$, we require that $\lim_{n \to \infty} D_\psi(x, x^n) = 0$. The 1-convexity of ψ implies that the Bregman function satisfies the inequality

$$D_\psi(x, \bar{x}) \geq \frac{1}{2} \|x - \bar{x}\|^2, \quad \forall x \in X, \ \bar{x} \in \operatorname{int}(X).$$

These are the kind of Bregman proximity functions considered in [9]. Obviously, $\psi(x) = \frac{1}{2} \|x\|^2$ corresponds to $D_\psi(x, \bar{x}) = \frac{1}{2} \|x - \bar{x}\|^2$.

In iterative algorithms, the Bregman proximity function is used via the proximity operator for a proper, lower semi-continuous, convex function $g \colon X \to \overline{\mathbb{R}}$

$$\operatorname{prox}^\psi_{\alpha g}(\bar{x}) := \arg\min_{x \in X} \alpha g(x) + D_\psi(x, \bar{x}), \qquad (15)$$

where we define $\operatorname{prox}_{\alpha g} := \operatorname{prox}^{\frac{1}{2} \|\cdot\|^2}_{\alpha g}$.

There are two kinds of Bregman proximity functions: (i) the function $\nabla \psi$ can be continuously extended to X, i.e., D_ψ can be defined on $X \times X$, and (ii) ψ is differentiable on $\operatorname{int}(X)$ (i.e., $\nabla \psi$ cannot necessarily be extended to $\operatorname{cl}(X)$). In this case, $D_\psi(x, \bar{x})$ makes sense only on $X \times \operatorname{int}(X)$ and we must assure that $\operatorname{prox}^\psi_{\alpha g}(\bar{x}) \in \operatorname{int}(X)$ for any $\bar{x} \in \operatorname{int}(X)$. For this, we need to assume that $\|\nabla \psi(x)\| \to \infty$ whenever x approaches a boundary point $\operatorname{bdry}(X) := \operatorname{cl}(X) \smallsetminus \operatorname{int}(X)$ (which is sometimes referred to as ψ being essentially smooth [36]).

While solutions of the proximity operator for the first class can lie on the boundary $\operatorname{bdry}(X)$, this is not possible for the second class; boundary points can be reached only in the limit when the proximity operator is applied sequentially. Moreover, for $\bar{x} \in \operatorname{bdry}(X)$, (14) would imply that, unless $x = \bar{x}$, the Bregman distance is $+\infty$ for any x, which can

be represented by $\delta_{[x=\bar{x}]}(x)$. This means that $\bar{x} \in \operatorname{bdry}(X)$ is always a fixed point of this Bregman proximity operator. This precludes application of the fixed point approach from Sect. 4.3.

6.2 Examples of Bregman Functions

Since Bregman proximity functions play a key role in this paper, we consider a few examples.

Example 1 The Euclidean length $\psi(x) = \frac{1}{2} \|x\|^2_2$ is continuously differentiable on the whole space $\mathbb{R}^N$ and, therefore, belongs to class (i) of Bregman proximity functions.

Example 2 The Bregman proximity function generated by $\psi(x) = \frac{1}{2}((x+1)\log(x+1) + (1-x)\log(1-x))$ is defined on the interval $(-1, 1)$ and can be continuously extended to $[-1, 1]$, and is continuously differentiable on $(-1, 1)$ with $|\psi'(x)| \to \infty$ when $x \to \pm 1$. It is 1-strongly convex.

Example 3 The entropy function $\psi(x) = x \log(x)$, which can be continuously extended to $[x \geq 0]$, is continuously differentiable on $[x > 0]$ with derivative $\psi'(x) = \log(x) + 1$. The derivative cannot be continuously extended to $x = 0$. For $x \to 0$ we have $|\psi'(x)| \to +\infty$. Unfortunately, this function is not even 1-strongly convex on $[x \geq 0]$. However, the function $ax \log(x)$ is 1-strongly convex when restricted to a bounded subset $[0, 1/a]$, $a > 0$. For $a = 1$, the Bregman function $D_\psi(x, \bar{x}) = x(\log(x) - \log(\bar{x})) - (x - \bar{x})$ is generated.

Example 4 The entropy function can also be used in higher dimensions. Unfortunately, it is hard to assert a simple evaluation of an associated proximity mapping in this case. Consider a polyhedral set $\emptyset \neq X \in \mathbb{R}^N$ given by

$$X = \{x \in \mathbb{R}^N | \forall i = 1, \dots, M \colon \langle a_i, x \rangle \leq b_i\}$$
$$= \bigcap_{i=1}^{M} \{x \in \mathbb{R}^N | \langle a_i, x \rangle \leq b_i\}$$

for vectors $0 \neq a_i \in \mathbb{R}^N$, and $b_i \in \mathbb{R}^M$, $i = 1, \dots, M$. Then, the generating function

$$\psi(x) = \sum_{i=1}^{M} (b_i - \langle a_i, x \rangle) \log(b_i - \langle a_i, x \rangle)$$

is designed such that for any point $\bar{x} \in \operatorname{int}(X)$ any other point $x \notin X$ is "moved infinitely far away" with respect to the Bregman distance $D_\psi(x, \bar{x})$. Therefore, $\|\nabla \psi(x)\| \to \infty$ for x tends towards a point on the boundary $\operatorname{bdry}(X)$. Nevertheless, ψ is continuous on X and strongly convex, if X is bounded.

6.3 Examples of Smooth Bregman Proximity Operators

The Bregman proximity functions that we presented are particularly interesting if the evaluation of the proximal mapping (15) is a constrained minimization problem, i.e., the involved function g in prox_g^ψ is extended-valued and $+\infty$ outside the constraint (closed) convex set $X \subset \mathbb{R}^N$. The Bregman function can replace or simplify the constraint set. In the following, we consider a few examples of practical interest. The class of functions that are amenable to our approach can be broadened significantly thanks to the concept of (convex) conjugation.

We consider a basic class of functions $g(x) = \langle x, c \rangle + \delta_X(x)$ for some $c \in \mathbb{R}^N$. The associated (non-linear) proximity operator from (15) is given by

$$\mathrm{prox}_{\alpha g}^\psi(\bar{x}) = \arg\min_{x \in X} \alpha \langle x, c \rangle + D_\psi(x, \bar{x}).$$

The corresponding (necessary and sufficient) optimality condition, which has a unique solution, is

$$0 \in c + \nabla\psi(x) - \nabla\psi(\bar{x}) + \partial\delta_X(x)$$
$$\Leftrightarrow \nabla\psi(\bar{x}) - c \in \nabla\psi(x) + \mathrm{N}_X(x),$$

where $\mathrm{N}_X(x)$ denotes the normal cone at x of the set X. Suppose $\bar{x} \in \mathrm{int}(X)$. If ψ is chosen such that $\|\nabla\psi(x)\| \to +\infty$ for $x \to \tilde{x} \in \mathrm{bdry}(X)$, then the solution of the proximal mapping is in $\mathrm{int}(X)$. Since $\mathrm{N}_X(x) = 0$ for $x \in \mathrm{int}(X)$, the optimality condition simplifies to

$$\nabla\psi(\bar{x}) - c = \nabla\psi(x), \qquad (16)$$

i.e., the constraint is implicitly taken care of by the Bregman proximity function. Summarizing, the goal of our approach (for this basic function g) consists of determining ψ, respectively D_ψ, such that

- the constraint set can be handled implicitly,
- (16) can be solved efficiently (possibly in closed form),
- and the solution function of (16), which yields the solution of (16) for a given $\bar{x}$, is required to be differentiable w.r.t. x and ϑ, where possibly $c = c(\vartheta)$.

Example 5 For a linear function $g(x) = \langle c, x \rangle + \delta_{[x \geq 0]}(x)$, the entropy function from Example 3 can be summed up for each coordinate to remove the non-negativity constraint. The proximity operator reads

$$\left(\mathrm{prox}_{\alpha g}^{\sum_j x_j \log x_j}(\bar{x})\right)_i = \bar{x}_i \exp(-\alpha c_i).$$

A closer look at the iterations of the forward–backward splitting (FBS) algorithm (12) reveals that such a function g arises with $c = \nabla f(\bar{x})$, i.e., in the iterations of FBS for the minimization of

$$\min_{x \in \mathbb{R}^N} f(x) + \delta_{[x \geq 0]}(x).$$

A particular instance of this problem is the *non-negative least squares problem*, i.e., $f(x) = \frac{1}{2}\|Ax - b\|_2^2$ with a matrix A and a vector b.

Example 6 The most frequent application of the entropy-prox is for the minimization of a linear function $g(x) = \langle c, x \rangle$ over the unit simplex in $\mathbb{R}^N$. Since the entropy function restricts the solution of the proximity operator to the positive orthant, projecting a point $\bar{x} \in \mathbb{R}_+^N$ onto the unit simplex $\{x \in \mathbb{R}^N | \sum_{i=1}^N x_i = 1 \text{ and } x_i \geq 0\}$ reduces to the projection onto the affine subspace $\{x \in \mathbb{R}^N | \sum_{i=1}^N x_i = 1\}$, which can be given in closed form, i.e.,

$$\left(\mathrm{prox}_{\alpha g}^{\sum_j x_j \log x_j}(\bar{x})\right)_i = \frac{\bar{x}_i \exp(-\alpha c_i)}{\sum_{j=1}^N \bar{x}_j \exp(-\alpha c_j)}.$$

This proximal problem arises for example in the *multi-label segmentation problem* in Sect. 8.1 or in *Matrix games* (see [9, Sect. 7.1]).

Example 7 For the function $g(x) = \langle c, x \rangle + \delta_{[-1 \leq x \leq 1]}(x)$ the Bregman function from Example 2 reduces the minimization problem in the proximal mapping to an unconstrained problem. The proximal mapping with $\psi(x) = \sum_i \frac{1}{2}((x_i + 1)\log(x_i + 1) + (1 - x_i)\log(1 - x_i))$ reads:

$$\left(\mathrm{prox}_{\alpha g}^\psi(\bar{x})\right)_i = \frac{\exp(-2\alpha c_i) - \frac{1 - \bar{x}_i}{1 + \bar{x}_i}}{\exp(-2\alpha c_i) + \frac{1 - \bar{x}_i}{1 + \bar{x}_i}}.$$

Obviously, this example can be adjusted to any Cartesian product of interval constraints. The importance of this exemplary function g becomes clear in the following.

Functions that are linear on a constraint set also arise when conjugate functions are considered. For instance, the ℓ_1-norm can be represented as

$$\|x\|_1 = \max_y \langle x, y \rangle + \delta_{[-1 \leq y \leq 1]}(y).$$

In combination with the primal–dual (PD) algorithm (13), this representation results in subproblems of the type discussed in the preceding examples. From this perspective, optimization problems involving a linear operator $\mathcal{D}$ and the ℓ_1-norm $\|\mathcal{D}x\|_1$ are also easy to address.

This idea of conjugation can be put into a slightly larger framework, as the convex conjugate of any positively one-homogeneous proper, lsc, convex function is an indicator function of a closed convex set. Unfortunately, it is required

Fig. 2 Visualization of the loss function $\mathcal{L}(x(\vartheta), \vartheta)$ for the 1D example (17) on the *left side*. The optimum is marked with a *black star*. On the *right-hand side*, the solution map of the lower level problem is shown
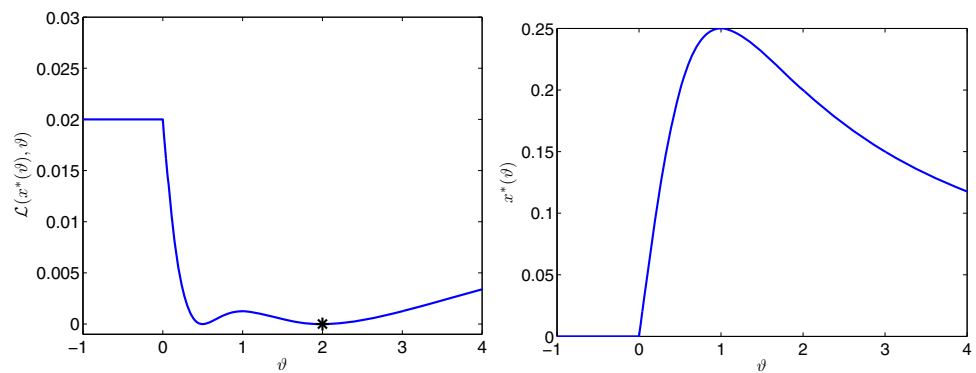
that projecting onto such a set is easy ("prox-friendliness"). Therefore, the following example is restricted to the (additively) separable case.

Example 8 Let g be an (additively) separable, positively one-homogeneous, proper, lsc, convex function $g(x) = \sum_{i=1}^{N} g_i(x_i)$. Thanks to its properties g coincides with its bi-conjugate function g^{**} and we can consider

$$g(x) = g^{**}(x) = \sum_{i=1}^{N} \max_{y_i} x_i y_i - \delta_{Y_i}(y_i),$$

where $Y_i = [a_i, b_i]$ is a closed interval in $\mathbb{R}$. Again, the dual update step of (13) involves problems such as in Example 7 with $h^*(y) = \sum_i \delta_{Y_i}(y_i)$.

7 Toy Example

The bilevel problem that we consider here is a parameter learning problem of a one-dimensional non-negative least squares problem:

$$\min_{\vartheta \in \mathbb{R}} \frac{1}{2}(x^*(\vartheta) - \mathfrak{g})^2$$
$$s.t. \ x^*(\vartheta) = \arg\min_{x \in \mathbb{R}} \frac{\lambda}{2}(\vartheta x - b)^2 + \frac{1}{2}x^2 + \delta_{[x \geq 0]}(x),$$
$$(17)$$

where ϑ is the optimization variable of the bilevel problem, $b \in \mathbb{R}$ is the input of the least squares problem, and λ is a positive weighting parameter. Given ϑ and b, the lower level problem solves the non-negative least squares problem. The squared Euclidean loss function in the upper level problem compares the output of the lower level problem for some ϑ and b to the ground truth $\mathfrak{g} := x^*(\vartheta^*)$, which is generated by solving the lower level problem with some predefined value ϑ^*. The goal of the bilevel optimization problem is to find ϑ^* given b and $\mathfrak{g}$.

The analytic solution of the lower level problem (the solution map) is

$$x^*(\vartheta) = \max\left(0, \frac{\lambda \vartheta b}{1 + \lambda \vartheta^2}\right)$$

and is shown on the right-hand side of Fig. 2. It is obviously a non-smooth function with a non-differentiable point at $\vartheta = 0$. Plugging the solution map into the upper level problem shows the actual objective to be minimized; see the left-hand side of Fig. 2.

7.1 Experimental Setup

In the following experiments, we numerically explore the gradients computed with the proposed techniques. We do not consider the actual minimization of the bilevel problem. The computed gradients could be used by any first-order gradient-based method.

7.1.1 Analytic Subdifferential

For $\vartheta \neq 0$, the standard chain rule from calculus can be applied and we can directly write down the derivative of the whole problem, namely

$$\frac{d\mathcal{L}}{d\vartheta}(x(\vartheta)) = \frac{\lambda b(1 - \lambda \vartheta^2)}{(1 + \lambda \vartheta^2)^2}(x(\vartheta) - \mathfrak{g}).$$

For $\vartheta = 0$, we consider the derivative

$$\frac{d\mathcal{L}}{d\vartheta}(x(\vartheta)) = [0, \lambda b(x(0) - \mathfrak{g})],$$

where $[0, \lambda b]$ is replaced by $[\lambda b, 0]$ if $\lambda b < 0$.

7.1.2 Implicit Differentiation Approach Sect. 4.1

In order to apply this technique, we must smooth the lower level problem. Since we want to avoid solutions $x^*(\vartheta) = 0$,

we introduce a log-barrier and replace the lower level problem by

$$f_\mu(x, \vartheta) := \frac{\lambda}{2}(\vartheta x - b)^2 + \frac{1}{2}x^2 - \mu \log(x)$$

for some small $\mu > 0$. Thus, we can drop the non-negativity constraint. To compute the gradient via the implicit differentiation formula (6), we minimize f_μ with respect to x and compute the second derivatives (we abbreviate the x-derivative with f_μ' and ϑ-derivative with $\partial_\vartheta f_\mu$)

$$\begin{aligned} f_\mu'(x, \vartheta) &= \lambda\vartheta(\vartheta x - b) + x - \frac{\mu}{x}; \\ f_\mu''(x, \vartheta) &= \lambda\vartheta^2 + 1 + \frac{\mu}{x^2}; \\ \partial_\vartheta f_\mu'(x, \vartheta) &= 2\lambda\vartheta x - \lambda b. \end{aligned} \quad (18)$$

Then, (6) yields

$$\frac{d\mathcal{L}}{d\vartheta}(x^*(\vartheta)) = -(x(\vartheta) - \mathfrak{g})(f_\mu''(x^*(\vartheta), \vartheta))^{-1}\partial_\vartheta f_\mu'(x^*(\vartheta), \vartheta).$$

This approach is denoted `Smoothed-impl`.

7.1.3 Algorithmic Differentiation Approach Sect. 4.2

We consider two algorithms: projected gradient descent and forward–backward splitting with Bregman proximity functions. Both algorithms are splitting methods that distribute the objective into a smooth function f and a non-smooth function g, for our example it reads

$$f(x, \vartheta) = \frac{\lambda}{2}(\vartheta x - b)^2 + \frac{1}{2}x^2 \quad \text{and} \quad g(x) = \delta_{[x \geq 0]}(x).$$

Projected gradient descent operates by a gradient descent step with respect to the smooth function f followed by a projection onto the (convex) set $[x \geq 0]$:

$$\begin{aligned} x^{(n+1)} &= \text{proj}_{[x \geq 0]}(x^{(n)} - \alpha f'(x^{(n)}, \vartheta)) \\ &= \max(0, x^{(n)} - \alpha f'(x^{(n)})). \end{aligned} \quad (19)$$

Note that the projection onto the convex set can also be interpreted as solving the proximity operator associated with the function g.

The second algorithm is obtained by changing the distance function for evaluating the proximity operator to the Bregman distance from Example 3. It results in

$$x^{(n+1)} = x^n \exp(-\alpha f'(x^{(n)}, \vartheta)). \quad (20)$$

As we assume that $x^0 \in [x > 0]$ the Bregman proximity function ensures that the solution stays in the feasible set. Thus, the back-projection can be dropped.

To apply Algorithm 1 or 2, we need the second derivatives of the update steps (19) and (20). The second derivatives of $f = f_\mu$ with $\mu = 0$ are given in (18). Although (19) is not differentiable, it is differentiable almost everywhere, and in the experiment, we formally applied the chain rule and assigned an arbitrary subgradient wherever it is not unique, i.e.,

$$\frac{\partial \text{proj}_{[x \geq 0]}}{\partial x}(x, \vartheta) = \begin{cases} 0, & \text{if } x < 0; \\ 1, & \text{if } x > 0; \\ [0, 1], & \text{if } x = 0; \end{cases}$$

and $\frac{\partial \text{proj}_{[x \geq 0]}}{\partial \vartheta} = 0$. This approach is denoted `Proj.GD`.

For (20), we use Algorithm 1 and obtain[5]

$$\begin{aligned} \frac{\partial \mathcal{A}}{\partial \vartheta}(x^{(n)}, \vartheta) &= -\alpha x \exp(-\alpha f'(x^{(n)}, \vartheta))\frac{\partial f'}{\partial \vartheta}(x^{(n)}, \vartheta) \\ \frac{\partial \mathcal{A}}{\partial x}(x^{(n)}.\vartheta) &= \exp(-\alpha f'(x^{(n)}, \vartheta)) \\ &\quad - \alpha x^{(n)} \exp(-\alpha f'(x^{(n)}, \vartheta))f''(x^{(n)}, \vartheta). \end{aligned}$$

This approach is denoted `Bregman-FB`.

7.1.4 Implicit Differentiation of the Fixed Point Equation Approach from Sect. 4.3

As explained above, direct differentiation of the fixed point equation of an algorithm implies two techniques. One is by applying Algorithm 1 to (20) but evaluating all derivatives at the optimum (denoted `Bregman-FB2`). The other is to do the numerical inversion as in (11) (denoted `Bregman-FB-impl`).

7.2 Analysis of the 1D Example

In the experiments, we focus on the estimation of the gradient (in Fig. 3). Therefore, the step size parameters of the individual algorithms are chosen such that a comparable convergence of the lower level energy is achieved, if possible.

For `Proj.GD`, `Bregman-FB`, and `Bregman-FB2`, the chain rule must be applied recursively. We plot the change of the gradient accumulation along these back-iterations (of 200 forward iterations) in bottom of Figure 4 and the energy evolution in the upper part of this figure.

In this example, we can observe a linear decrease in the contribution to the respective final gradient value, which shows that back-iterations can be stopped after a few iterations without making large errors.

---

[5] Note that we kept the order of the terms given by the chain rule, since for multi-dimensional problems the products are matrix products and are, in general, not commutative.

Fig. 3 Analytic tangents to the upper level objective function of (17) at $\vartheta = 0.3$ and $\vartheta = 0$. The function is non-smooth and, thus, at $\vartheta = 0$ there exists many tangent lines
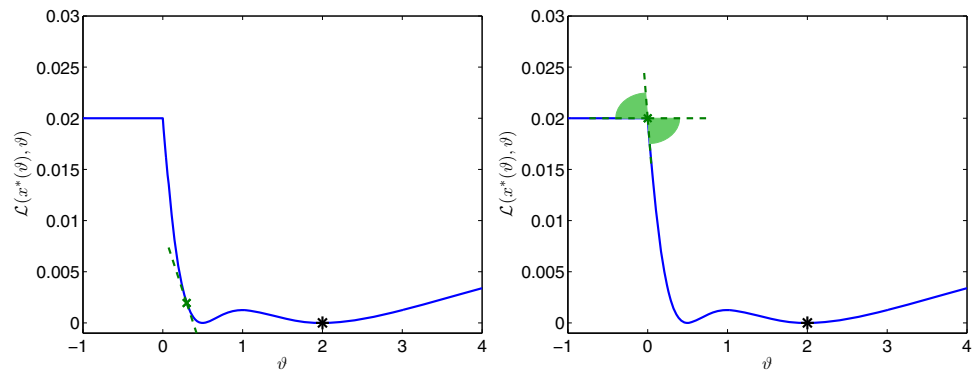

Fig. 4 The *upper row* shows the energy decrease along the forward iterations. The *lower row* shows the convergence to the respective gradient value along the back-iterations. On the *left-hand side* the plot is generated with $\vartheta = 0.3$ and on the *right-hand side* with $\vartheta = 0$. The "-impl" methods do not appear in the bottom row as no back-iterations are involved. For $\vartheta = 0$, due to the simple structure of the lower level problem, projected gradient descent converges exactly in one iteration, thus it is not shown. The gradient converges linearly to its final value, which means that often a few back-iterations are enough to achieve a gradient estimate of good quality
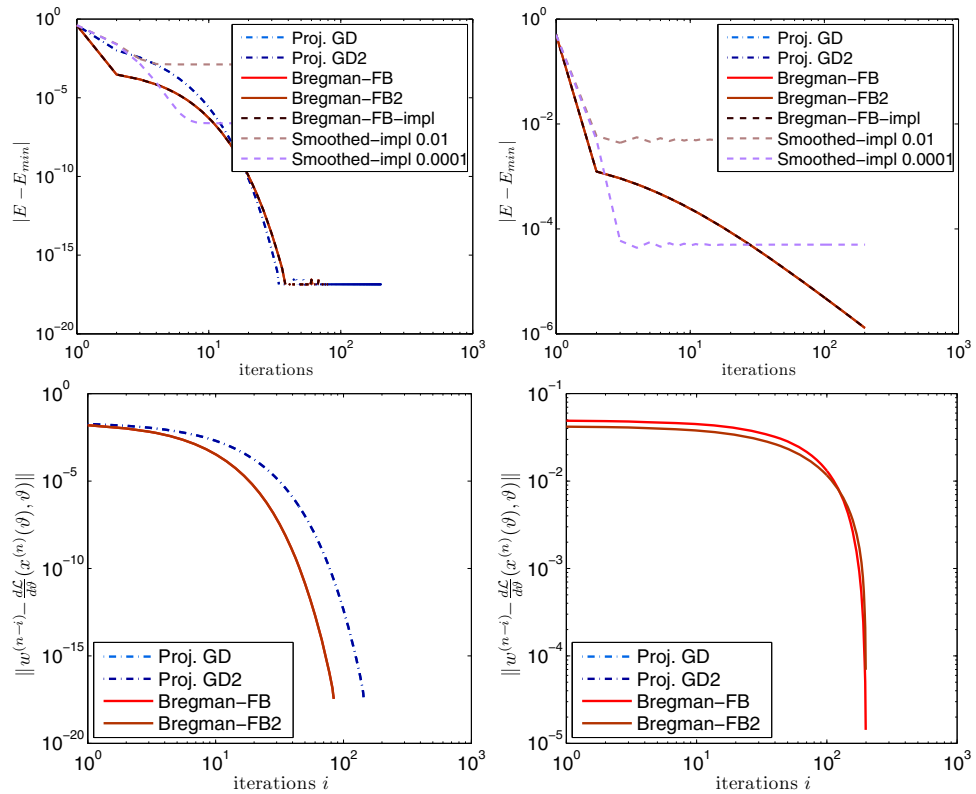
Interestingly, the approximations `Bregman-FB2` and `Proj.GD2` work well, as they show the same gradient accumulation as `Bregman-FB` and `Proj.GD`, respectively. This situation changes when the number of forward iterations is reduced. For about 15 forward iterations, a difference of order 10^{-4} becomes visible (case $\vartheta = 0.3$).

Figures 5 and 6 address the convergence of the gradient towards the analytic gradient, with respect to different approximation accuracies (varied by the number of back-iterations). Figure 5 shows the convergence for $\vartheta = 0.3$ and Fig. 6 for $\vartheta = 0$. Numerically, we observe convergence to the analytic gradients.

Surprisingly, all methods perform equally well in the case $\vartheta = 0$. The estimated gradient lies always in the subdifferential at this point. The range of the subdifferential is

indicated with bright green color in Fig. 6. While `Proj.GD` and `Proj.GD2` estimate a gradient from the boundary of the subdifferential, the other methods estimate a subgradient from the interior. However, all of these values are feasible and belong to the analytic subdifferential.

8 Application to Multi-Label Segmentation

In this section, we show how the idea can be applied in practice. To this end, we introduce a multi-label segmentation model. We use a convolutional neural network (CNN) to parametrize the segmentation model. Alternatively, this construction can be thought of as having a segmentation model as the final stage of a deep neural network. In this setting,
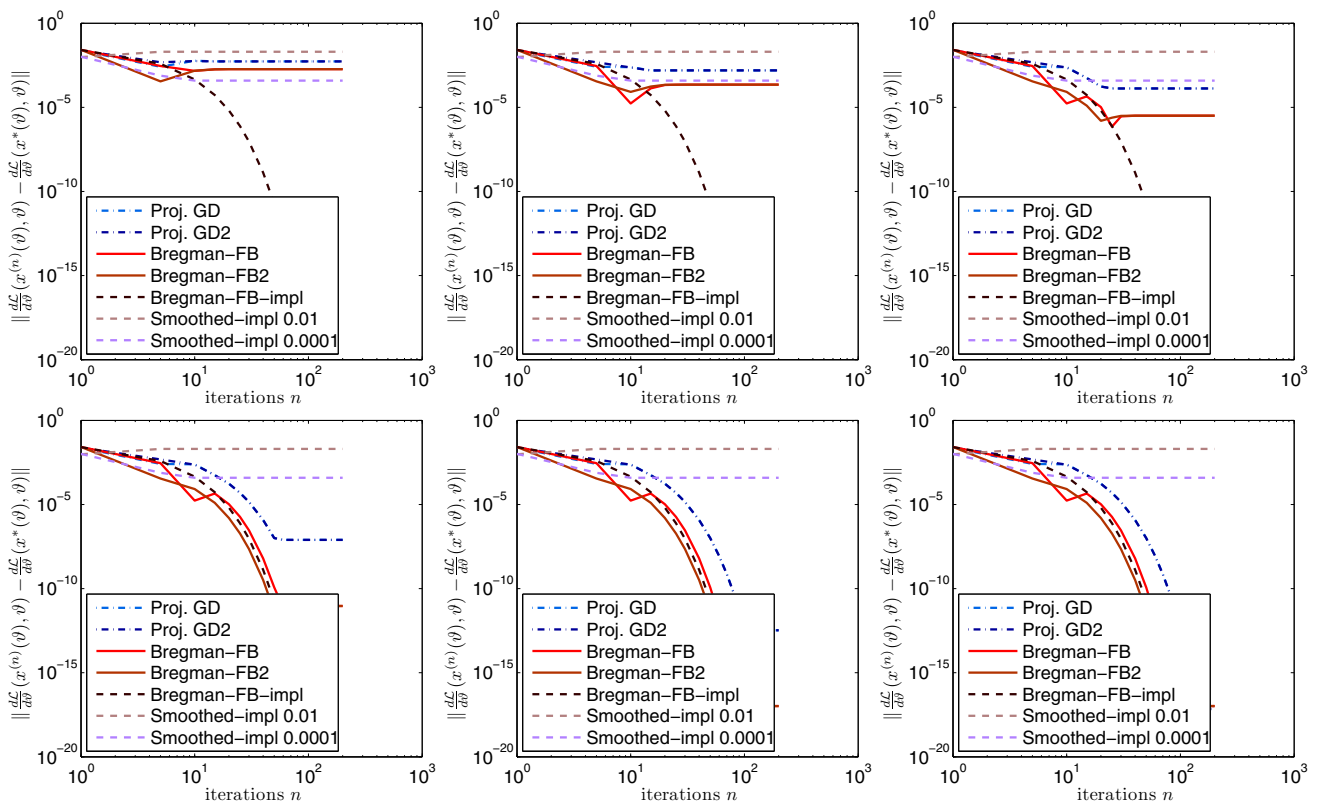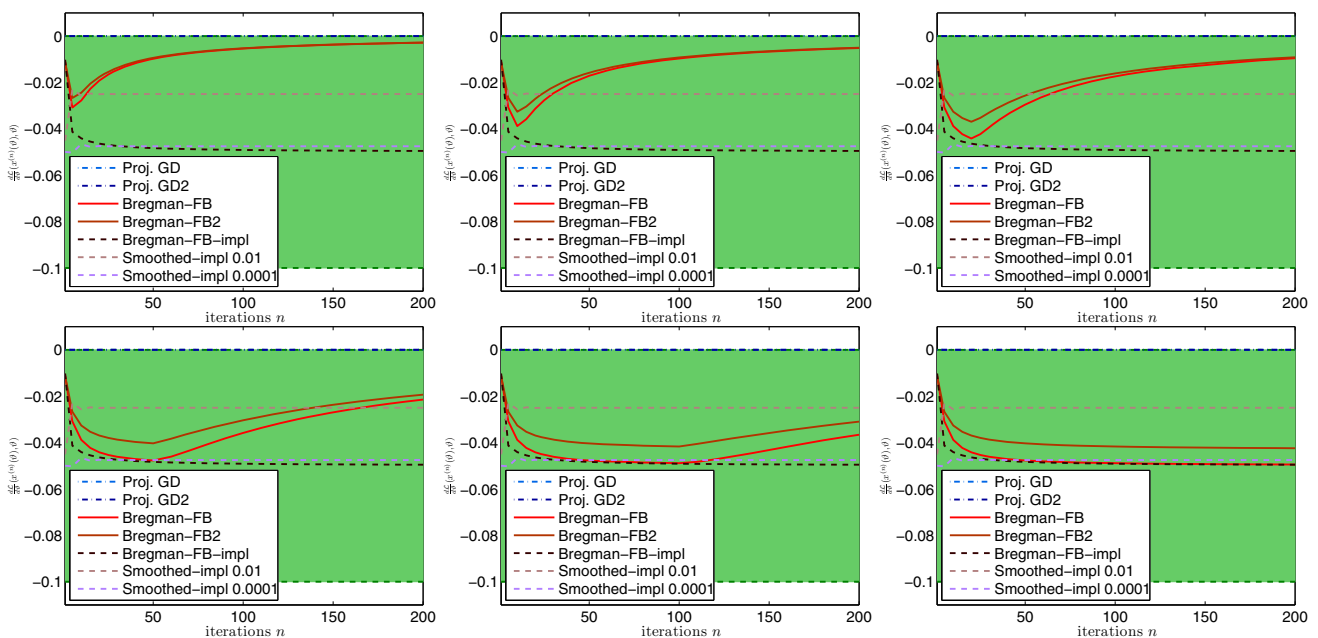
Fig. 5 Convergence of the numerical gradients towards the analytic gradient for $\vartheta = 0.3$. Row-wise, from *left* to *right*, the number of back-iterations is increased: 5, 10, 20, 50, 100, 200. More back-iterations lead to more accurate gradient estimates. The "-impl" methods always perform equally, as no back-iterations are required. `Smoothed-impl` performs worst due to the rough approximation. Our methods `Bregman-FB`, `Bregman-FB2`, and `Bregman-FB-impl` are the best and converge slightly better than `Proj.GD` and `Proj.GD2`

Fig. 6 Convergence of the numerical gradients towards the analytic gradient for $\vartheta = 0$. Row-wise, from *left* to *right*, the number of back-iterations is increased: 5, 10, 20, 50, 100, 200. All methods perform equally well, as they lie in the *bright green* area that indicates the range of the subdifferential (Color figure online)

the bilevel problem amounts to finding the parameters of the CNN such that the loss on training data is minimized. The presented approach provides a generic way to train such systems in an end-to-end fashion.

8.1 Model

Given a cost tensor $\mathfrak{c} \in X^{N_l}$, where $X = \mathbb{R}^{N_x N_y}$, that assigns to each pixel (i, j) and each label k, $i = 1, \ldots, N_x$, $j = 1, \ldots, N_y, k = 1, \ldots, N_l$, a cost $\mathfrak{c}_{i,j}^k$ for the pixel taking label k. We often identify $\mathbb{R}^{N_x \times N_y}$ with $\mathbb{R}^{N_x N_y}$ by $(i, j) \mapsto i + (j - 1)N_x$ to simplify the notation. The sought segmentation $u \in X_{[0,1]}^{N_l}$, where $X_{[0,1]} = [0, 1]^{N_x N_y} \subset X$, is represented by a binary vector for each label. As a regularizer for a segment's plausibility, we measure the boundary length using the total variation (TV). The discrete derivative operator $\nabla : X \rightarrow Y$, where we use the shorthand $Y := X \times X$ (elements from Y are considered as column vectors), is defined as

$$(\nabla u^k)_{i,j} := \begin{pmatrix} (\nabla u^k)_{i,j}^x \\ (\nabla u^k)_{i,j}^y \end{pmatrix} \in Y (= \mathbb{R}^{2N_x N_y}),$$

$$\mathcal{D}u := (\nabla u^1, \ldots, \nabla u^{N_l}),$$

$$(\nabla u^k)_{i,j}^x := \begin{cases} u_{i+1,j}^k - u_{i,j}^k, & \text{if } 1 \le i < N_x, 1 \le j \le N_y \\ 0, & \text{if } i = N_x, 1 \le j \le N_y. \end{cases}$$

$(\nabla u^k)_{i,j}^y$ is defined analogously. From now on, we work with the image as a vector indexed by $\mathbf{i} = 1, \ldots, N_x N_y$. Let elements in Y be indexed with $\mathbf{j} = 1, \ldots, 2N_x N_y$. Let the inner product in X and Y be given, for $u^k, v^k \in X$ and $p^k, q^k \in Y$, as

$$\left\langle u^k, v^k \right\rangle_X := \sum_{\mathbf{i}=1}^{N_x N_y} u_{\mathbf{i}}^k v_{\mathbf{i}}^k, \quad \left\langle p^k, q^k \right\rangle_Y := \sum_{\mathbf{j}=1}^{2N_x N_y} p_{\mathbf{j}}^k q_{\mathbf{j}}^k,$$

$$\langle u, v \rangle_{X^{N_l}} := \sum_{k=1}^{N_l} \left\langle u^k, v^k \right\rangle_X, \quad \langle p, q \rangle_{Y^{N_l}} := \sum_{k=1}^{N_l} \left\langle p^k, q^k \right\rangle_Y.$$

The (discrete, anisotropic) TV norm is given by

$$\|\mathcal{D}u\|_1 := \sum_{k=1}^{N_l} \sum_{\mathbf{j}=1}^{2N_x N_y} |(\nabla u^k)_{\mathbf{j}}|,$$

where $|\cdot|$ is the absolute value. In the following, the variables $\mathbf{i} = 1, \ldots, N_x N_y$ and $\mathbf{j} = 1, \ldots, 2N_x N_y$ always run over these index sets, thus we drop the specification; we adopt the same convention for $k = 1, \ldots, N_l$. We define the pixel-wise non-negative unit simplex

$$\Delta^{N_l} := \Big\{ \forall(\mathbf{i}, k) \colon 0 \le u_{\mathbf{i}}^k \le 1$$

$$\text{and } \forall \mathbf{i} \colon \sum_k u_{\mathbf{i}}^k = 1 \ u \in X^{N_l} \Big\}, \tag{21}$$

and the pixel-wise (closed) ℓ_∞-unit ball around the origin

$$B_1^{\ell_\infty}(0) := \Big\{ [\forall(\mathbf{j}, k) \colon |p_{\mathbf{j}}^k| \le 1] p \in Y^{N_l} \Big\}.$$

Finally, the segmentation model reads

$$\min_{u \in X^{N_l}} \ \langle \mathfrak{c}, u \rangle_{X^{N_l}} + \|\mathcal{W}\mathcal{D}u\|_1, \quad s.t. \ u \in \Delta^{N_l}, \tag{22}$$

where we use a diagonal matrix $\mathcal{W}$ to support contrast-sensitive penalization of the boundary length.

This model and the following reformulation as a saddle-point problem are well known (see, e.g., [8])

$$\min_{u \in X^{N_l}} \max_{p \in Y^{N_l}} \ \langle \mathcal{W}\mathcal{D}u, p \rangle_{Y^{N_l}} + \langle u, \mathfrak{c} \rangle_{X^{N_l}},$$

$$s.t. \ u \in \Delta^{N_l}, \ p \in B_1^{\ell_\infty}(0). \tag{23}$$

The saddle-point problem (23) can be solved using the ergodic primal–dual algorithm [9], which leads to an iterative algorithm with totally differentiable iterations. The primal update in (13) is discussed in Example 6 and the dual update of (13) is essentially Example 7. As a consequence, Algorithm 3 can be applied to estimate the derivatives. A detailed derivation of the individual steps of the algorithm can be found in [30].

8.2 Parameter Learning

We consider (22) where the cost $\mathfrak{c}$ is given by the output of a CNN which takes as input an image $\mathfrak{I} \in X^{N_c}$ to be segmented and is defined via a set of weights ϑ. Formally, we have $\mathfrak{c}_{\mathbf{i}}^k = \mathfrak{f}_{\mathbf{i}}^k(\vartheta, \mathfrak{I})$ with $\mathfrak{f} : \mathbb{R}^{N_\vartheta} \times X^{N_c} \rightarrow X^{N_l}$, where N_c denotes the number of channels of the input image and N_ϑ is the number of weights parametrizing the CNN.

The training set consists of N_T images $\mathfrak{I}^1, \ldots, \mathfrak{I}^{N_T} \in X^{N_c}$ and their corresponding ground truth segmentations $\mathfrak{g}^1, \ldots, \mathfrak{g}^{N_T} \in \{1, \ldots, N_l\}^{N_x N_y}$.

The parameters ϑ of the CNN are cast as an instance of the general bilevel optimization problem (3):

$$\min_{\vartheta \in \mathbb{R}^{N_\vartheta}} \sum_{t=1}^{N_T} \sum_{\mathbf{i}=1}^{N_x N_y} \log \Big(\sum_{k=1}^{N_l} \exp(u_{\mathbf{i}}^k(\vartheta, \mathfrak{I}^t)) \Big) - \mathfrak{g}_{\mathbf{i}}^t(\vartheta, \mathfrak{I}^t)$$

$$s.t. \ u(\vartheta, \mathfrak{I}^t) = \arg \min_{u \in X^{N_l}} E(u, \mathfrak{f}(\vartheta, \mathfrak{I}^t)), \tag{24}$$

where energy E in the lower level problem is (22) and the higher level problem is defined as the softmax loss.

Remark 3 We could equivalently use a multinomial logistic loss, since $u_{\mathbf{i}}(\vartheta, \mathfrak{I}^t)$ lies in the unit simplex by construction. We use this definition to allow for a simplified treatment of the case of training a CNN without the global segmentation model.
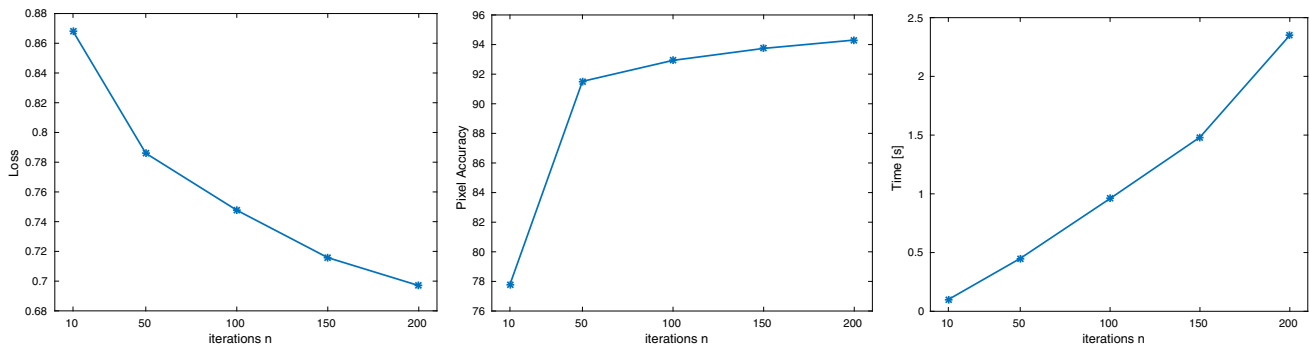
Fig. 7 Training error versus number of iterations of the algorithm solving the lower level problem. From *left* to *right*, average per-pixel loss, per-pixel accuracy, and time per outer iteration. The timing includes the forward pass as well as the gradient computations. Timings were taken on an NVIDIA Geforce Titan X GPU. A higher number of iterations clearly lead to lower error, but come at the cost of a higher computational complexity

8.3 Experiments

We implemented our approach as a custom layer in the Mat-ConvNet framework [39]. We used the Stanford Background dataset [20], which consists of 715 input images and pixel-accurate ground truth consisting of the geometric classes *sky*, *vertical*, and *horizontal*. We used ADAM [22] for the minimization of the higher level problem. We found that general plain stochastic gradient descent performs poorly in our setting, since the global segmentation model can lead to vanishing gradients.

In a first experiment, we used a small subset of 9 images from the dataset to show the influence of the number of iterations used to solve the lower level problem (22) on the training objective. We learned a small network consisting of four layers of alternating convolutions with a kernel width of 3 pixels and ReLU units followed by a fully connected layer. We added 3×3 max-pooling layers with a stride of two after the first and the second convolutional layers, which effectively downsamples the responses by a factor of 4. We added an up-convolutional layer to upsample the responses to the original image size. The penultimate layer of the CNN consists of a multiplicative scaling (analogous to a scalar smoothness parameter) of the CNN output followed by the global segmentation model (22). We ran ADAM with a learning rate of 10^{-3} for a total of 1000 iterations with a mini-batch size of one image to learn the parameters of this network.

Figure 7 shows the average per-pixel loss, the average pixel accuracy as well as the time per ADAM iteration vs. number of iterations used to solve the lower level problem (inner iterations). This experiment shows that by solving the lower level problem to higher accuracy, the overall capacity and thus the accuracy of the system can be enhanced. This comes at the price of a higher computational complexity, which increases linearly with the number of iterations.

Table 1 Accuracy on the Stanford Background dataset [20]

	Test		Train	
	Acc	IoU	Acc	IoU
FCN	92.40	82.65	97.54	92.21
FCN + Global	**93.00**	**84.01**	**97.90**	**93.53**

Bold values indicate that the CNN with global segmentation model performs better than the plain CNN model

Finally, we performed a large-scale experiment on this dataset. We partitioned the images into a training set of 572 images and used the remaining 143 images for testing. We used the pre-trained Fully Convolutional Network *FCN-32s* [26] as a basis for this experiment. We adapted the geometry of the last two layers to this dataset and retrained the network. We then added a multiplicative scaling layer followed by the global segmentation model and refined the parameters. The number of inner iterations was set to 100, which provides a good trade-off between accuracy and computational complexity. We use a mini-batch size of 5 images and a learning rate of 10^{-3}.

The average accuracy in terms of the average pixel accuracy (*Acc*) in percent and Intersection over Union (*IoU*) on both the test and the training sets is shown in Table 1. We compare the plain Fully Convolutional Network *FCN* to the network with the additional global segmentation model *FCN+Global*. We observed an increase of 1.4 % in terms of IoU on the test set when using the global model. This can be attributed to the fact that the CNN alone already provides good but coarse segmentations and the segmentation model uses only simple pairwise interactions. As such, it is unable to correct gross errors of the CNN.

Since the presented approach is applicable to a broad range of energies, training of more expressive energies which include more complex interactions (cf. [41]) is a promising
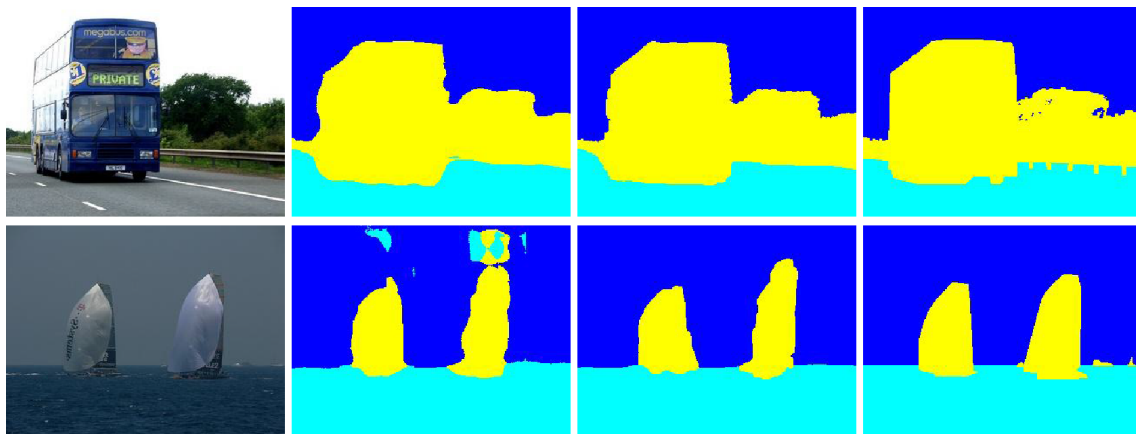
Fig. 8 Example results from the test set. Row-wise, from *left* to *right* input image, CNN, CNN+Global, ground truth. The global model is able to align results to edges and is able to correct spurious errors

direction for future research. Example segmentations from the test set are shown in Fig. 8.

Remark 4 For a comparison to the smoothing approach from Sect. 4.1, we refer to the conference version [30].

9 Conclusion

We considered a class of bilevel optimization problems with non-smooth lower level problem. By an appropriate approximation, we can formulate an algorithm with a smooth update mapping that solves a non-smooth optimization problem in the lower level. This allows us to apply gradient-based methods for solving the bilevel optimization problem. A second approach directly considers the fixed point equation of the algorithm as an optimality condition for the lower level problem. Key for both ideas are Bregman proximity functions.

The idea of estimating gradients for an abstract algorithm was exemplified for a forward–backward splitting method and a primal–dual algorithm with Bregman proximity functions. Several potential application examples were shown. A toy example confirmed our results and provided some more intuition. The contribution of our idea to practical applications was demonstrated by a multi-label segmentation model that was coupled with a convolutional neural network.

There are several open questions, for example convergence of the sequence of gradients or a full classification of optimization problems that allow for algorithms with smooth update mapping.

References

1. Al-Baali, M.: Descent property and global convergence of the Fletcher–Reeves method with inexact line search. IMA J. Numer. Anal. **5**(1), 121–124 (1985)
2. Attouch, H., Bolte, J., Svaiter, B.: Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward-backward splitting, and regularized Gauss-Seidel methods. Math. Program. **137**(1–2), 91–129 (2013)
3. Beck, A., Teboulle, M.: Mirror descent and nonlinear projected subgradient methods for convex optimization. Oper. Res. Lett. **31**(3), 167–175 (2003)
4. Bennett, K., Kunapuli, G., Hu, J., Pang, J.S.: Bilevel optimization and machine learning. In: Computational Intelligence: Research Frontiers. Lecture Notes in Computer Science, vol. 5050, pp. 25–47. Springer, Berlin (2008)
5. Bregman, L.M.: The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. USSR Comput. Math. Math. Phys. **7**(3), 200–217 (1967)
6. Calatroni, L., Reyes, J., Schönlieb, C.B.: Dynamic sampling schemes for optimal noise learning under multiple nonsmooth constraints. ArXiv e-prints (2014). arXiv:1403.1278
7. Calatroni, L., Reyes, J., Schönlieb, C.B., Valkonen, T.: Bilevel approaches for learning of variational imaging models. ArXiv e-prints (2015). arXiv:1505.02120
8. Chambolle, A., Pock, T.: A first-order primal-dual algorithm for convex problems with applications to imaging. J. Math. Imaging Vis. **40**(1), 120–145 (2011)
9. Chambolle, A., Pock, T.: On the ergodic convergence rates of a first-order primal–dual algorithm. Math. Program. (2015). doi:10.1007/s10107-015-0957-3
10. Chen, Y., Pock, T., Ranftl, R., Bischof, H.: Revisiting loss-specific training of filter-based MRFs for image restoration. In: German Conference on Pattern Recognition (GCPR). in Lecture Notes in Computer Science, vol. 8142, pp. 271–281. Springer, Berlin (2013)
11. Chen, Y., Ranftl, R., Pock, T.: Insights into analysis operator learning: from patch-based sparse models to higher order MRFs. IEEE Trans. Image Process. **23**(3), 1060–1072 (2014)
12. Deledalle, C.A., Vaiter, S., Fadili, J., Peyré, G.: Stein Unbiased GrAdient estimator of the Risk (SUGAR) for multiple parameter selection. SIAM J. Imaging Sci. **7**(4), 2448–2487 (2014)
13. Dempe, S.: Annotated Bibliography on bilevel programming and mathematical programs with equilibrium constraints. Optimization **52**(3), 333–359 (2003)

14. Dempe, S., Kalashnikov, V., Pérez-Valdés, G., Kalashnykova, N.: Bilevel Programming Problems. Energy Systems. Springer, Berlin (2015)
15. Dempe, S., Zemkoho, A.: The generalized Mangasarian–Fromowitz constraint qualification and optimality conditions for bilevel programs. J. Optim. Theory Appl. **148**(1), 46–68 (2010)
16. Domke, J.: Implicit differentiation by perturbation. In: Advances in Neural Information Processing Systems (NIPS), pp. 523–531 (2010)
17. Domke, J.: Generic methods for optimization-based modeling. In: International Workshop on Artificial Intelligence and Statistics, pp. 318–326 (2012)
18. Evans, L.C., Gariepy, R.F.: Measure Theory and Fine Properties of Functions. CRC Press, Boca Raton (1992)
19. Fletcher, R., Reeves, C.: Function minimization by conjugate gradients. Comput. J. **7**(2), 149–154 (1964)
20. Gould, S., Fulton, R., Koller, D.: Decomposing a scene into geometric and semantically consistent regions. In: International Conference on Computer Vision (ICCV) (2009)
21. Griewank, A., Walther, A.: Evaluating Derivatives, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2008)
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
23. Kunisch, K., Pock, T.: A bilevel optimization approach for parameter learning in variational models. SIAM J. Imaging Sci. **6**(2), 938–983 (2013)
24. Lions, P.L., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. SIAM J. Appl. Math. **16**(6), 964–979 (1979)
25. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Program. **45**(1), 503–528 (1989)
26. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
27. Moore, G.: Bilevel programming algorithms for machine learning model selection. Ph.D. thesis, Rensselaer Polytechnic Institute, Troy (2010)
28. Ochs, P.: Long term motion analysis for object level grouping and nonsmooth optimization methods. Ph.D. thesis, Albert–Ludwigs–Universität Freiburg, Freiburg im Breisgau (2015)
29. Ochs, P., Chen, Y., Brox, T., Pock, T.: ipiano: Inertial proximal algorithm for non-convex optimization. SIAM J. Imaging Sci. **7**(2), 1388–1419 (2014)
30. Ochs, P., Ranftl, R., Brox, T., Pock, T.: Bilevel optimization with nonsmooth lower level problems. In: International Conference on Scale Space and Variational Methods in Computer Vision (SSVM) (2015)
31. Passty, G.B.: Ergodic convergence to a zero of the sum of monotone operators in Hilbert space. J. Math. Anal. Appl. **72**(2), 383–390 (1979)
32. Peyré, G., Fadili, J.: Learning analysis sparsity priors. In: Proceedings of Sampta (2011)
33. Ranftl, R., Pock, T.: A deep variational model for image segmentation. In: German Conference on Pattern Recognition (GCPR), pp. 107–118 (2014)
34. Reyes, J., Schönlieb, C.B., Valkonen, T.: The structure of optimal parameters for image restoration problems. ArXiv e-prints (2015). arXiv:1505.01953
35. Reyes, J.C.D.L., Schönlieb, C.B.: Image denoising: learning noise distribution via pde-constrained optimisation. Inverse Probl. Imaging **7**, 1183–1214 (2013)
36. Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton (1970)
37. Tappen, M.: Utilizing variational optimization to learn MRFs. In: International Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–8 (2007)
38. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. J. Mach. Learn. Res. **6**, 1453–1484 (2005)
39. Vedaldi, A., Lenc, K.: MatConvNet—convolutional neural networks for MATLAB. In: Proceedings of the ACM International Conference on Multimedia (2015)
40. Zavriev, S., Kostyuk, F.: Heavy-ball method in nonconvex optimization problems. Comput. Math. Model. **4**(4), 336–341 (1993)
41. Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., Torr, P.: Conditional random fields as recurrent neural networks. In: International Conference on Computer Vision (ICCV) (2015)

Peter Ochs received his M.Sc. degree in mathematics from the Saarland University in Germany in 2010 and his Ph.D. degree in mathematics from the University of Freiburg in 2015. Since 2015, he is working as a postdoctoral researcher at the Saarland University in Germany. He spent three months as a visiting researcher at the TU-Graz in Austria. He received the best paper award at the Scale Space and Variational Methods Conference (SSVM) in 2015 for the work on "Bilevel Optimization with Nonsmooth Lower Level Problems." His research interests are in computer vision and optimization, in particular general non-smooth non-convex optimization and bilevel optimization.

René Ranftl received his M.Sc. and Ph.D. degrees in computer science (Telematik) from Graz University of Technology in 2010 and 2015, respectively. Since 2015, he a member of the Intel Visual Computing Lab located in Santa Clara, California, United States. His current research interests include the development of efficient and robust algorithms for 3D reconstruction, optical flow, and correspondence problems as well as the confluence between deep learning and energy minimization methods.

Thomas Brox received his Ph.D. degree in computer science from the Saarland University in Germany in 2005. He spent two years as a postdoctoral researcher at the University of Bonn and two years at the University of California at Berkeley. Since 2010, he is heading the Computer Vision Group at the University of Freiburg in Germany. His research interests are in computer vision, in particular video analysis and deep learning for computer vision. He

is an associate editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence and the International Journal of Computer Vision. He received the Longuet-Higgins Best Paper Award and the Koenderink Prize for Fundamental Contributions in Computer Vision.

Thomas Pock born 1978 in Graz, received his M.Sc. (1998-2004) and Ph.D. (2005-2008) degrees in Computer Engineering (Telematik) from Graz University of Technology. After a Post-doc position at the University of Bonn, he moved back to Graz University of Technology where he has been an Assistant Professor at the Institute for Computer Graphics and Vision. In 2013, he received the START price of the Austrian Science Fund (FWF) and the German Pattern recognition award of the German association for pattern recognition (DAGM) and in 2014 he received a starting grant from the European Research Council (ERC). Since June 2014, Thomas Pock is a Professor of Computer Science at Graz University of Technology (AIT Stiftungsprofessur "Mobile Computer Vision") and a principal scientist at the Department of Safety and Security at the Austrian Institute of Technology (AIT). The focus of his research is on the development of mathematical models for computer vision and image processing in mobile scenarios as well as the development of efficient algorithms to compute these models.