Computational Optimization and Applications
https://doi.org/10.1007/s10589-025-00723-7

prunAdag: an adaptive pruning-aware gradient method

Margherita Porcelli[1,2] · Giovanni Seraghiti[1,3] · Philippe L. Toint[4]

Received: 11 February 2025 / Accepted: 29 July 2025

Abstract
A pruning-aware adaptive gradient method is proposed which classifies the variables in two sets before updating them using different strategies. This technique extends the "relevant/irrelevant" approach of Ding et al. (Adv Neural Inf Process Syst 32, 2019) and Zimmer et al. (Mathematical optimization for machine learning: proceedings of the MATH+ thematic Einstein semester 2023, 2025) and allows a posteriori sparsification of the solution of model parameter fitting problems. The new method is proved to be convergent with a global rate of decrease of the averaged gradient's norm of the form $\mathcal{O}(\log(k)/\sqrt{k+1})$. Numerical experiments on several applications show that it is competitive with existing pruning-aware Frank-Wolfe algorithms, see e.g. Zimmer et al. (Mathematical optimization for machine learning: proceedings of the MATH+ thematic Einstein semester 2023, 2025).

Keywords Model pruning · Adaptive first-order methods · Objective-function-free optimisation (OFFO) · Global convergence rate

✉ Giovanni Seraghiti
giovanni.seraghiti@umons.ac.be

Margherita Porcelli
margherita.porcelli@unifi.it

Philippe L. Toint
philippe.toint@unamur.be

[1] Dipartimento di Ingegneria Industriale, Università degli Studi di Firenze, Viale Morgagni 40/44, 50134 Florence, Italy

[2] ISTI–CNR, Via Moruzzi 1, Pisa, Italy

[3] University of Mons, Rue de Houdain 9, 7000 Mons, Belgium

[4] NAXYS, University of Namur, Namur, Belgium

Published online: 19 September 2025

1 Introduction

This paper deals with first-order objective-function-free optimization (OFFO) and parameter pruning for optimisation problems of the form

$$\min_{x\in\mathbb{R}^n} f(x) \tag{1}$$

where f is a smooth function from $\mathbb{R}^n$ to $\mathbb{R}$. In particular, our framework can be applied to problems where the objective function f is a loss function depending on the variable $x=(x_1,\dots,x_n)$ defining a model's parameters. For example, in several machine learning applications, the training of the neural network model consists in solving a finite sum minimization problem of the form

$$f(x)=\frac{1}{m}\sum_{i=1}^{m}\ell_i(x),$$

where the components of x correspond to parameters of the network and where both n and m are large, the latter giving the number of samples in the training set and ℓ_i being per-sample loss functions. Therefore, we will often refer to the components of the problem variable x as the model/problem parameters.

OFFO algorithms are methods where the objective function is never computed; instead, they rely only on derivative information, that is on the gradient in the first-order case. A class of OFFO methods, known as adaptive gradient algorithms, gained popularity in the machine learning community, emerging as state-of-the-art techniques to train neural networks. Some examples include Adagrad [3, 4], Adam [5], RMSprop [6], ADADELTA [7]. All of these methods share the common characteristic of only relying on current and past gradient information to adaptively determine the step or the step size or both at each iteration. As Gratton et al. suggested in [8–10], Adagrad can be interpreted as *trust-region* method (see [11, 12] for a comprehensive coverage) in which the radius of the trust-region is computed without evaluating the objective function, which makes it significantly more resistant to noise. Specifically, we propose a new OFFO method based on a modified version of Adagrad in the context of parameter pruning. As the name implies, pruning a model refers to the process of reducing its size and complexity, typically by removing or zeroing certain parameters that are considered unnecessary for its performance. In particular, in the neural network context, the goal of pruning is to improve efficiency, reduce overfitting, and speed up inference or training, without sacrificing much predictive accuracy [13–15]. We refer to our algorithm as pruning Adagrad (prunAdag).

Pruning emerges as a compression technique for neural networks alternative to matrix or tensor factorization [16–18] or quantization [19–21]. Pruning can be performed after training at the cost of re-training the model solely on the remaining parameters [22], but this procedure can sometimes be computationally impractical. Alternatively, one can induce sparsity during training adding sparsity-inducing regularizer to the objective function [23–25]; however, this strategy is significantly influenced by the choice of the regularisation parameter, and the level of sparsity of the solution cannot be altered without re-training the model. To address these limitations, *pruning-aware* methods have been developed. They require just one training task and they typically do not involve sparsity-inducing regularisation but they aim at finding a possibly dense solution, which is robust to pruning, in the sense that the performance of the model is not significantly affected when individual parameters (unstructured pruning) [22] or groups of parameters (structured pruning) [26] are pruned after training. In contrast to regularised methods, the level of sparsity of the solution of a pruning-aware method is chosen by the user after training. Consequently, a key concept in pruning-aware methods consists in the choice of the criteria to determine which parameters or group of parameters can be removed with less impact on the model's performance [27]. This paper is mainly concerned with unstructured pruning, but our approach can be extended to structured pruning as well. We also mention the connection of pruning-aware methods with implicit regularisation schemes for gradient descent methods, where sparsity is implicitly induced by prescribed choices of initialization, step size, and stopping time, rather than through regularisation. Implicit gradient methods have been proven to be effective in recovering sparse solutions of unpenalised least squares regression problems [28, 29].

The majority of the pruning-aware schemes share two common aspects. The first consists in classifying all the parameters, at each iteration, into *relevant* and *irrelevant* according to specific criteria. Secondly, the method promotes the relevant parameters by update rules which usually involve derivative information, and penalizes the irrelevant ones by diminishing their magnitudes or setting them to zero. This latter strategy may be suboptimal in the context of neural network training, as it has been shown that the importance of network weights can change dynamically during the training process [30–32], meaning that zeroing parameters might decrease the ability to capture these changes. Therefore, a controlled decrease of irrelevant parameters is preferable. At the end of the training phase, a model trained by a pruning-aware method has relevant components with larger magnitudes than the irrelevant ones. Finally, irrelevant components are pruned by removing/zeroing those parameters that are below some threshold such that the model matches any desired level of sparsity. We now give an overview of existing pruning-aware schemes and then present our contributions in this context.

1.1 Related works

We distinguish two classes of pruning-aware methods. The first divides parameters into relevant and irrelevant and then updates them following various rules. This approach is referred to as *activation selection* in [1]. The second updates parameters all at once and forces the magnitude of irrelevant parameters to decrease by adding specific sparsity-inducing constraints to the problem, see [2, 33].

The first pruning-aware approach is presented in [1, 34, 35] and employs Taylor series to measure the importance of a parameter by estimating the impact of its removal/zeroing on the value of the objective function in (1). More specifically, let $x_k = (x_{1,k}, \ldots, x_{n,k}) \in \mathbb{R}^n$ be the *k*-th iterate of the method, then in the first-order case, the relevance of the parameter $x_{i,k}$ at iteration *k* is measured by

$$\frac{\partial f}{\partial x_i}(x_k)(0 - x_{i,k}) = f(x_{1,k}, \dots, \underbrace{0}_{x_{i,k}}, \dots, x_{n,k}) - f(x_{1,k}, \dots, x_{i,k}, \dots, x_{n,k}) - o(x_{i,k}^2).$$

Then, the T_k parameters with largest values of $\left|\frac{\partial f}{\partial x_i}(x_k)(0 - x_{i,k})\right|$ are classified as relevant, as they are the ones that most significantly affect the objective function's value. Using this criterion, Ding et al. [1] propose to optimize relevant components via momentum SGD, while gradually decreasing the magnitudes of all others classified as irrelevant. One of the advantages of this approach is that the number of relevant components T_k can be chosen at each iteration, in order to match a prescribed level of sparsity. An adaptive choice to select T_k, using the ℓ_0-norm of the parameters, is proposed in [1]. Although very efficient in practice, the convergence of gradient descent methods using Taylor series approach to classify relevant and irrelevant parameters is not analyzed in [1]. Furthermore, it is not clear how much the irrelevant components can be reduced at each iteration without affecting the convergence of the method towards a stationary point.

A second class of pruning-aware methods consists in adding sparsity-inducing constraints to the formulation in (1) and solving the deriving constrained optimisation problem using the (stochastic) Frank-Wolfe (SFW) algorithm [2, 36]. Specifically, the constraints considered are *T*-sparse polytope and *T*-support-norm-ball for unstructured pruning and group-*T*-support-norm-ball for structured pruning, see [33, 37, 38]. Since we are primarily interested in unstructured pruning in a deterministic setting, we briefly describe the FW two-step framework [39, 40] for solving a *T*-support-norm-ball constrained problem, which is used for comparison in Sect. 3. Let $T > 0$ and $\tau \in \mathbb{R}$, then the *T*-support-norm-ball $\mathcal{C}_T(\tau)$ is defined as

$$\mathcal{C}_T(\tau) = \text{conv}\{x \in \mathbb{R}^n \mid \|x\|_0 \leq T, \quad \|x\|_2 \leq \tau\},$$

where $\text{conv}(\cdot)$ denotes the convex hull. The FW algorithm is applied to the sparse-constrained model

$$\min_{x \in \mathcal{C}_T(\tau)} f(x)$$

and consists of two main steps. First, a descent direction is computed, solving the linear minimization oracle (LMO)

$$v_k = \ \text{argmin}_{v \in \mathcal{C}_T(\tau)} \langle v, g_k \rangle,$$

where g_k is the gradient of *f* at the *k*-th iterate x_k. The optimal solution of this problem is given by

$$v_{i,k} = \begin{cases} -\tau \ g_{i,k}/\|g_k\|_{\mathcal{R}_k} & \text{if } i \in \mathcal{R}_k, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

for $i = 1, \dots, n$, where $\| \cdot \|_{\mathcal{R}_k}$ is the Euclidean norm of the subvector containing the indices in the set $\mathcal{R}_k$ of the first T among the largest components of the absolute value of g_k. Next, the new feasible iterate is obtained as a convex combination of the past iterate and the descent direction, that is

$$x_{k+1} = x_k + \eta_k (v_k - x_k), \tag{3}$$

for some learning rate $\eta_k \in [0, 1]$. In machine learning or neural networks applications, SFW is often employed and utilizes the stochastic gradient instead of the actual gradient g_k. The convergence of SFW has been established in [41] for a finite sum minimization problem in the nonconvex setting and it has been extended to the gradient rescaling version by Zimmer et al. in [2]. The SFW algorithm has demonstrated performance comparable to state-of-the-art pruning methods [2]; however, like the deterministic version, it has certain drawbacks. Specifically, the stepsize τ is related to the radius of the constrained region, which must be predetermined and cannot vary throughout the iterations; as well as the number of components T that are updated using gradient information. Moreover, the method is extremely sensitive to the tuning of its parameters such as the scalar τ, the learning rate η_k, and the number of gradient components T in the search direction.

Our prunAdag method is a new pruning-aware scheme. At each iteration, we classify as relevant those parameters corresponding to the T largest directional derivative in magnitude, similarly to Zimmer et al. in [2], but the overall classification and updating strategy are significantly different.

1. We develop a new adaptive strategy to separately update parameters that extends the distinction between relevant and irrelevant ones. Specifically, we introduce the concepts of *optimisable* and *decreasable* parameters. We consider all parameters that benefit from being updated using derivative information as optimisable, including all relevant parameters and those irrelevant parameters that can be both optimized and penalized simultaneously, using gradient information. We define as decreasable those irrelevant parameters that are not penalized by a gradient update and therefore require a specific penalization strategy to decrease their magnitudes.
2. We propose to use the Adagrad step for updating the optimisable parameters, while we develop an Adagrad-like trust-region framework to gradually decrease the magnitude of the decreasable ones without relying on gradient information or any function evaluations.
3. We prove the convergence of prunAdag with global convergence rate of $O(\log(k)/\sqrt{k+1})$ in the case when $g_k = \nabla f(x_k)$ for all k (deterministic setting).
4. We validate our method on several preliminary examples obtained from signal processing, dictionary learning and binary classification applications.

1.2 Organization of the paper

Our presentation is structured as follows. Section 2 introduces our prunAdag method in Algorithm 2.1 and discusses both the criterion used for classifying parameters at each iteration into optimisable and decreasable and the different update rules for the two classes of parameters. Convergence of the method is analyzed in Sect. 2.1. Section 3 presents a set of illustrative examples of the behaviour of prunAdag when applied to random least-squares, sparse signal recovery, sparse coding step in dictionary learning, and binary classification with logistic loss function. A brief conclusion is provided in Sect. 4.

1.3 Notations

Throughout the paper we adopt the following notations. At the *k*-th iteration, the gradient *g* of the function *f* evaluated at the current iterate x_k is denoted as $g_k = g(x_k)$ and its *i*-th component by $g_{i,k}$. Moreover, the superscript *T* denotes the transpose and $v_{i,k}$ denotes the *i*-th component of a vector $v_k \in \mathbb{R}^n$. Unless specified otherwise, $\| \cdot \|$ is the Euclidean norm on $\mathbb{R}^n$ and $\|x\|_{\mathcal{I}} = \|x_{i\in\mathcal{I}}\|$ is the Euclidean norm when we consider only the indices in $\mathcal{I}$. We denote as A^C the complementary set of *A* in $\{1, \ldots, n\}$. We use the notation $\lceil x \rceil$ for the minimum integer greater than *x*. Given two sequences $\{\alpha_k\}$ and $\{\beta_k\}$ of non-negative reals, we also say that α_k is $\mathcal{O}(\beta_k)$ is there exists a finite constant κ such that $\lim_{k\to\infty}(\alpha_k/\beta_k) \leq \kappa$.

2 A first-order sparsity inducing adaptive gradient method without regularisation

In the following discussion, we make the standard assumptions on problem (1) for first-order convergence rate analysis.

AS.1: the objective function *f* is continuously differentiable;

AS.2: its gradient *g* is Lipschitz continuous with Lipschitz constant $L \geq 0$, that is

$$\|g(x) - g(y)\| \leq L\|x - y\|$$

for all $x, y \in \mathbb{R}^n$;

AS.3: there exists a constant f_{low} such that, for all *x*, $f(x) \geq f_{\text{low}}$.

We now motivate and describe the prunAdag method. It falls in the class of pruning-aware methods that, at a given iteration, first classifies the variables/parameters in relevant and irrelevant before updating them.

Following Zimmer et al. [2, 33] and given a target *T* for the cardinality of the relevant parameters,[1] we first define the set $\mathcal{R}_k$ of $\mathcal{R}$elevant parameters as the set of cardinality *T* of indices corresponding to the first *T* among the largest directional derivatives in magnitude at the *k*th iterate:

[1] The integer parameter *T* is fixed during the iterations.

$$\mathcal{R}_k \overset{\text{def}}{=} \{i \in \{1,\dots,n\} \mid |g_{i,k}| \text{ is one of the T largest components of } |g_k|\}. \tag{4}$$

The parameters indexed by $\mathcal{R}_k^C$ are thus considered irrelevant. The idea is then to optimize on the relevant parameters and to reduce the magnitude of the others. In our approach, optimisation is performed by applying the component-wise version of Adagrad, in which the step $s_{i,k}$ in the *i*-th variable is given by

$$s_{i,k} = -\frac{g_{i,k}}{w_{i,k}^{\mathcal{O}}}, \tag{5}$$

for some suitably chosen weight $w_{i,k}^{\mathcal{O}}$ derived from the history of past gradients. However, it may happen that the step (5) that would be taken by the optimizer for some of the irrelevant parameters does actually reduce their magnitudes: this happens when the signs of $x_{i,k}$ and $g_{i,k}$ coincide. Thus it makes sense to take the optimisation step on these parameters as well, provided it remains within reasonable bounds compared to $|x_{i,k}|$. More formally, we deem the *i*-th component ($i \in \mathcal{R}_k^C$) to be "$\mathcal{O}$ ptimisable" (in the sense that its update can be performed by (5)) if it belongs to the set $\mathcal{A}_k$ of $\mathcal{A}$cceptable indices

$$\mathcal{A}_k \overset{\text{def}}{=} \left\{ i \in \mathcal{R}_k^C \mid \text{sign}(g_{i,k}) = \text{sign}(x_{i,k}) \quad \text{and} \quad a_{i,k} \le \left| \frac{g_{i,k}}{w_{i,k}^{\mathcal{O}}} \right| \le b_{i,k} \right\}, \tag{6}$$

where $a_{i,k}$ and $b_{i,k}$ are suitable bounds. As a consequence, the index set of the optimisable parameters at iteration *k* is given by

$$\mathcal{O}_k \overset{\text{def}}{=} \mathcal{R}_k \cup \mathcal{A}_k. \tag{7}$$

We also see that the "decreasable" parameters, whose magnitude we wish to $\mathcal{D}$ecrease by other means than (5), have indices in the set $\mathcal{D}_k = \mathcal{R}_k^C \setminus \mathcal{A}_k$. We intentionally do not impose any restrictions on the bounds $a_{i,k}$ and $b_{i,k}$ in (6) as the general analysis, including the convergence analysis, of prunAdag does not depend on them. In practice, one can choose $a_{i,k}$ as a fraction of the absolute value of $x_{i,k}$. Further practical examples are given in Sect. 3 (see Table 1).

While the course of action for the "optimisable" parameters is clear (apply (5)), what to do with the other parameters is, at this stage, less obvious. Our proposal is

Table 1 Four versions of prunAdag, depending on the choice of the bounding sequences $a_{i,k}$ and $b_{i,k}$ used in Step 3 of Algorithm 2.1

	$a_{i,k}$	$b_{i,k}$
prunAdag-V1	$\left\lvert \frac{x_{i,k}}{k+1} \right\rvert \cdot \frac{\lVert g_k \rVert_{\mathcal{R}_k}}{\lVert x_k \rVert_{\mathcal{S}_k}}$	$+\infty$
prunAdag-V2	$\left\lvert \frac{x_{i,k}}{k+1} \right\rvert$	$+\infty$
prunAdag-V3	$\left\lvert \frac{x_{i,k}}{k+1} \right\rvert \cdot \frac{\lVert g_k \rVert_{\mathcal{R}_k}}{\lVert x_k \rVert_{\mathcal{S}_k}}$	$\lvert x_{i,k} \rvert$
prunAdag-V4	$\left\lvert \frac{x_{i,k}}{k+1} \right\rvert$	$\lvert x_{i,k} \rvert$

to extend our Adagrad-based approach by defining a step for the parameters in $\mathcal{D}_k$ which is also of the form (optimality measure/weight), where now the optimality measure is no longer given by gradient values, but by the magnitude of the irrelevant parameters themselves (as we wish to drive them to zero, if possible), that is a step of the form

$$s_{i,k}^{\mathcal{D}} = -\frac{x_{i,k}}{w_{i,k}^{\mathcal{D}}}, \tag{8}$$

where the weight $w_{i,k}^{\mathcal{D}}$ is now derived from the history of past irrelevant values of the i-th parameter.

We are now ready to state the prunAdag algorithm more formally in Algorithm 2.1.

Algorithm 2.1 prunAdag

Step 0: Initialization. A starting point x_0, a target number of relevant parameters $T \in \{1, \dots, n\}$, a constant $\varsigma \in (0,1)$, and two initial weight vectors $w_{i,-1}^{\mathcal{O}} = \varsigma$ and $w_{i,-1}^{\mathcal{D}} = \varsigma$, for $i = 1, \dots, n$ are given. Set $k = 0$.

Step 1: Select relevant parameters. Compute the gradient g_k and define

$$\mathcal{R}_k = \{i \in \{1, \dots, n\} \mid |g_{i,k}| \text{ is one of T largest components of } |g_k|\}$$

Step 2: Optimisation weights. Compute

$$w_{i,k}^{\mathcal{O}} = \sqrt{(w_{i,k-1}^{\mathcal{O}})^2 + g_{i,k}^2} \quad (i \in \{1, \dots, n\}) \tag{9}$$

Step 3: Classify components. Define $a_{i,k}, b_{i,k} \geq 0$ for $i \in \mathcal{R}_k^C$, and

$$\mathcal{A}_k = \left\{ i \in \mathcal{R}_k^C \mid \text{sign}(x_{i,k}) = \text{sign}(g_{i,k}) \quad \text{and} \quad a_{i,k} \leq \left| \frac{g_{i,k}}{w_{i,k}^{\mathcal{O}}} \right| \leq b_{i,k} \right\}.$$

Then set $\mathcal{O}_k = \mathcal{R}_k \cup \mathcal{A}_k$ and $\mathcal{D}_k = \mathcal{O}_k^C$.

Step 4: Compute a step for the optimisable components. Compute

$$s_{i,k} = -\frac{g_{i,k}}{w_{i,k}^{\mathcal{O}}} \quad (i \in \mathcal{O}_k). \tag{10}$$

Step 5: Compute a step for the decreasable components. Compute

$$w_{i,k}^{\mathcal{O}} = w_{i,k-1}^{\mathcal{O}}, \quad w_{i,k}^{\mathcal{D}} = \sqrt{(w_{i,k-1}^{\mathcal{D}})^2 + x_{i,k}^2}, \quad s_{i,k}^{L} = -\frac{x_{i,k}}{w_{i,k}^{\mathcal{D}}} \quad (i \in \mathcal{D}_k) \tag{11}$$

$$w^{\mathcal{D}}_{i,k} = w^{\mathcal{D}}_{i,k-1} \quad (i \in \mathcal{O}_k). \tag{12}$$

Compute a step $s_{i,k}$ for all the decreasable components in $\mathcal{D}_k$ such that

$$|s_{i,k}| \le |s^L_{i,k}| \qquad (i \in \mathcal{D}_k) \tag{13}$$

$$\sum_{i \in \mathcal{D}_k} g_{i,k} s_{i,k} \le 0. \tag{14}$$

Step 6: New iterate. Define

$$x_{k+1} = x_k + s_k,$$

increment *k* by one and return to Step 1.

A few comments are useful after this formal description.

1. The weights defined in (9) and (11) superficially look identical to weights used by Adagrad. There is however a crucial difference for our purpose: each of these updating formula only selects in the past those iterations for which the considered component (the *i*-th) is either optimisable (for the former) or decreasable (for the latter). More specifically, if we set

$$g^{\mathcal{O}}_{i,k} = \begin{cases} g_{i,k} & i \in \mathcal{O}_k \\ 0 & \text{otherwise,} \end{cases} \qquad x^{\mathcal{D}}_{i,k} = \begin{cases} x_{i,k} & i \in \mathcal{D}_k \\ 0 & \text{otherwise,} \end{cases} \tag{15}$$

then one verifies, using the first part of (11) and (12), that

$$w^{\mathcal{O}}_{i,k} = \sqrt{\varsigma + \sum_{j=0}^{k} (g^{\mathcal{O}}_{i,j})^2} \quad \text{and} \quad w^{\mathcal{D}}_{i,k} = \sqrt{\varsigma + \sum_{j=0}^{k} (x^{\mathcal{D}}_{i,j})^2}. \tag{16}$$

2. We have left the choice of the exact technique to define $s^{\mathcal{D}}_k$ very general, as long as (13–14) hold (these conditions can be interpreted as “trust-region" conditions with radius $|s^L_{i,k}|$). One simple technique is, for example, to set

$$s^{\mathcal{D}}_{i,k} = \begin{cases} -\text{sign}(x_{i,k}) \cdot \min[a_{i,k}, |s^L_{i,k}|] & i \in \{i \in \mathcal{D}_k \mid \text{sign}(x_{i,k}) = \text{sign}(g_{i,k})\} \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

3. The specification of the set $\mathcal{A}_k$ is not strictly necessary in our convergence theory below, but our experience indicates that extending the set of optimisable parameters from $\mathcal{R}_k$ to $\mathcal{R}_k \cup \mathcal{A}_k$ is beneficial in practice, as illustrated in Sect. 3.1.1.

2.1 Convergence analysis

First, we assume that the sequence produced by prunAdag is bounded, that is, AS.4: The sequence produced by Algorithm 2.1 is bounded, i.e. for some $\kappa_x > 0$, $|x_{i,k}| \leq \kappa_x$ for every $i = \{1, \dots, n\}$ and all $k \geq 0$.

AS.4 is needed to provide an upper bound for the contribution given by the update of the decreasable variables/parameters which cannot be related to the gradient norm, since the update does not involve any gradient information. Moreover, our framework reduces the magnitudes of the decreasable parameters at every iteration, thus it is reasonable to assume that the magnitudes of the parameters remain finite. We also assume, without loss of generality, that $\varsigma \leq \left(\frac{8nL}{3}\right)^2$.

We start the convergence analysis of the prunAdag algorithm by stating a lemma characterizing the descent properties of the method.

Lemma 2.1 *Suppose that AS.1 and AS.2 hold. Then we have that, for all $j \geq 0$,*

$$f(x_{j+1}) \leq f(x_j) - \sum_{i=1}^{n} \frac{(g_{i,j}^{\mathcal{O}})^2}{w_{i,j}^{\mathcal{O}}} + \frac{L}{2} \sum_{i=1}^{n} \frac{(g_{i,j}^{\mathcal{O}})^2}{(w_{i,j}^{\mathcal{O}})^2} + \frac{L}{2} \sum_{i=1}^{n} \frac{(x_{i,j}^{\mathcal{D}})^2}{(w_{i,j}^{\mathcal{D}})^2} \tag{18}$$

and for all $k \geq 0$

$$f(x_0) - f(x_{k+1}) \geq \sum_{j=0}^{k} \sum_{i=1}^{n} \frac{(g_{i,j}^{\mathcal{O}})^2}{w_{i,j}^{\mathcal{O}}} - \frac{L}{2} \sum_{j=0}^{k} \sum_{i=1}^{n} \frac{(g_{i,j}^{\mathcal{O}})^2}{(w_{i,j}^{\mathcal{O}})^2} - \frac{L}{2} \sum_{j=0}^{k} \sum_{i=1}^{n} \frac{(x_{i,j}^{\mathcal{D}})^2}{(w_{i,j}^{\mathcal{D}})^2}. \tag{19}$$

Proof Using (14), we have that, at each iteration j of Algorithm 2.1,

$$g_j^T s_j = \sum_{i \in \mathcal{O}_j} g_{i,j} s_{i,j} + \sum_{i \in \mathcal{D}_j} g_{i,j} s_{i,j} \leq \sum_{i \in \mathcal{O}_j} g_{i,j} s_{i,j} < 0. \tag{20}$$

Therefore, using Assumptions AS.1 and AS.2, (20), (10), (11), and (14) we derive that

$$\begin{aligned} f(x_{j+1}) &\leq f(x_j) + g_j^T s_j + \frac{L}{2} ||s_j||^2 \\ &\leq f(x_j) + \sum_{i \in \mathcal{O}_j} g_{i,j} s_{i,j} + \frac{L}{2} \sum_{i \in \mathcal{O}_j} s_{i,j}^2 + \frac{L}{2} \sum_{i \in \mathcal{D}_j} s_{i,j}^2 \\ &\leq f(x_j) - \sum_{i \in \mathcal{O}_j} \frac{g_{i,j}^2}{w_{i,j}^{\mathcal{O}}} + \frac{L}{2} \sum_{i \in \mathcal{O}_j} \frac{g_{i,j}^2}{(w_{i,j}^{\mathcal{O}})^2} + \frac{L}{2} \sum_{i \in \mathcal{D}_j} \frac{x_{i,j}^2}{(w_{i,j}^{\mathcal{D}})^2} \\ &= f(x_j) - \sum_{i=1}^{n} \frac{(g_{i,j}^{\mathcal{O}})^2}{w_{i,j}^{\mathcal{O}}} + \frac{L}{2} \sum_{i=1}^{n} \frac{(g_{i,j}^{\mathcal{O}})^2}{(w_{i,j}^{\mathcal{O}})^2} + \frac{L}{2} \sum_{i=1}^{n} \frac{(x_{i,j}^{\mathcal{D}})^2}{(w_{i,j}^{\mathcal{D}})^2} \end{aligned} \tag{21}$$

and (18) therefore holds. Finally, summing for $j = 0, \dots, k$ gives (19). □

We next proceed by recalling the following lemma due to [42–44], which is crucial in the derivation of upper bounds for the second and third terms in equation (19). The proof of the lemma can be found in [10, Lemma 3.1].

Lemma 2.2 *Let $\{c_k\}_{k\geq 0}$ be a non-negative sequence and $\xi > 0$. Then*

$$\sum_{j=0}^{k} \frac{c_j}{(\xi + \sum_{\ell=0}^{j} c_\ell)} \leq \log\left(\frac{\xi + \sum_{j=0}^{k} c_j}{\xi}\right). \tag{22}$$

We now state our main result on the convergence rate of prunAdag, partly inspired by [10, Theorem 3.2].

Theorem 2.3 Suppose that AS.1–AS.4 hold. Assume that $\mathcal{R}_k$ defined in (4), has cardinality T for every k and that the prunAdag algorithm is applied to problem (1). If we define $\Gamma_0 \overset{\text{def}}{=} f(x_0) - f_{\text{low}}$, then,

$$\underset{j\in\{0,\dots,k\}}{\text{average}} \|g_j\|^2 \leq \lceil n/T \rceil \frac{\theta(k)}{k+1} \tag{23}$$

with

$$\theta(k) \overset{\text{def}}{=} \max\left\{\varsigma, \frac{\varsigma}{2} e^{\frac{\Gamma_0}{nL}}, 32n^2L^2 \left|W_{-1}\left(-\frac{\sqrt{\varsigma}}{8nL}\right)\right|^2, \; 2\left(\Gamma_0 + nL\log\left(1 + \frac{(k+1)\kappa_x^2}{\varsigma}\right)\right)^2\right\}, \tag{24}$$

where W_{-1} is the second branch of the Lambert function [45].

Proof Let us first observe that bounding the average of $\|g_j^{\mathcal{O}}\|^2$ allows us to derive a bound for the average of the norm of the actual gradient $\|g_j\|^2$. Indeed, since $\mathcal{R}_j \subseteq \mathcal{O}_j$ contains the largest components of the gradient at iteration j and its cardinality is always equal to T, we have that

$$\|g_j\|^2 \leq \lceil n/T \rceil \sum_{i\in\mathcal{R}_j} g_{i,j}^2 \leq \lceil n/T \rceil \sum_{i\in\mathcal{O}_j} g_{i,j}^2 = \lceil n/T \rceil \|g_j^{\mathcal{O}}\|^2$$

and therefore

$$\underset{j\in\{0,\dots,k\}}{\text{average}} \|g_j\|^2 \leq \lceil n/T \rceil \underset{j\in\{0,\dots,k\}}{\text{average}} \|g_j^{\mathcal{O}}\|^2. \tag{25}$$

Now, using (19) and the fact that the sequence $w_{i,k}^{\mathcal{O}}$ in (10) is increasing in k for every i, we have that

$$f(x_{k+1}) \leq f(x_0) - \sum_{j=0}^{k} \frac{\|g_j^{\mathcal{O}}\|^2}{\max_{i\in\{1,\dots,n\}} w_{i,k}^{\mathcal{O}}} + \frac{L}{2}\sum_{j=0}^{k}\sum_{i=1}^{n} \frac{(g_{i,j}^{\mathcal{O}})^2}{(w_{i,j}^{\mathcal{O}})^2} + \frac{L}{2}\sum_{j=0}^{k}\sum_{i=1}^{n} \frac{(x_{i,j}^{\mathcal{D}})^2}{(w_{i,j}^{\mathcal{D}})^2},$$

from which we obtain that

$$\sum_{j=0}^{k} \frac{\|g_j^{\mathcal{O}}\|^2}{\max_{i\in\{1,\ldots,n\}} w_{i,k}^{\mathcal{O}}} \le \Gamma_0 + \frac{L}{2}\sum_{j=0}^{k}\sum_{i=1}^{n} \frac{(x_{i,j}^{\mathcal{D}})^2}{(w_{i,j}^{\mathcal{D}})^2} + \frac{L}{2}\sum_{j=0}^{k}\sum_{i=1}^{n}\frac{(g_{i,j}^{\mathcal{O}})^2}{(w_{i,j}^{\mathcal{O}})^2}. \tag{26}$$

We then use Lemma 2.2 with $c_j = (g_{i,j}^{\mathcal{O}})^2$ and $\xi = \zeta$, and the first part of (16), yielding

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=0}^{k}\frac{(g_{i,j}^{\mathcal{O}})^2}{(w_{i,j}^{\mathcal{O}})^2} &= \sum_{i=1}^{n}\sum_{j=0}^{k}\frac{(g_{i,j}^{\mathcal{O}})^2}{\varsigma + \sum_{\ell=0}^{j}(g_{i,\ell}^{\mathcal{O}})^2} \le \sum_{i=1}^{n}\log\left(\frac{1}{\varsigma}\left(\varsigma + \sum_{\ell=0}^{k}(g_{i,\ell}^{\mathcal{O}})^2\right)\right)\\ &\le n\log\left(1+\frac{1}{\varsigma}\sum_{\ell=0}^{k}\|g_\ell^{\mathcal{O}}\|^2\right).\end{aligned} \tag{27}$$

Using again Lemma 2.2, this time with $c_j = (x_{i,j}^{\mathcal{D}})^2$, the second part of (16) and Assumption AS.4, we deduce that

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=0}^{k}\frac{(x_{i,j}^{\mathcal{D}})^2}{(w_{i,j}^{\mathcal{D}})^2} &= \sum_{i=1}^{n}\sum_{j=0}^{k}\frac{(x_{i,j}^{\mathcal{D}})^2}{\varsigma + \sum_{\ell=0}^{j}(x_{i,\ell}^{\mathcal{D}})^2} \le \sum_{i=1}^{n}\log\left(\frac{1}{\varsigma}\left(\varsigma + \sum_{\ell=0}^{k}(x_{i,\ell}^{\mathcal{D}})^2\right)\right)\\ &\le n\log\left(1+\frac{(k+1)\kappa_x^2}{\varsigma}\right).\end{aligned} \tag{28}$$

Combining (26), (27), (28) therefore gives that

$$\sum_{j=0}^{k}\frac{\|g_j^{\mathcal{O}}\|^2}{\max_{i\in\{1,\ldots,n\}} w_{i,k}^{\mathcal{O}}} \le \Gamma_0 + \frac{nL}{2}\log\left(1+\frac{(k+1)\kappa_x^2}{\varsigma}\right) + \frac{nL}{2}\log\left(1+\frac{1}{\varsigma}\sum_{j=0}^{k}\|g_j^{\mathcal{O}}\|^2\right), \tag{29}$$

where the first logarithmic term depends on the upper bound on the entries of the iterate in Assumption AS.4, while the second depends on the sum of the norms of the past optimisable gradients.

We now proceed by analyzing two separate cases.

Case 1. Assume first that the contribution given by the optimisable gradients exceeds that of the first logarithmic term in *k*, that is

$$\log\left(1+\frac{1}{\varsigma}\sum_{j=0}^{k}\|g_j^{\mathcal{O}}\|^2\right) \ge \log\left(1+\frac{(k+1)\kappa_x^2}{\varsigma}\right).$$

Then, the inequality in (29) becomes

$$\sum_{j=0}^{k}\frac{\|g_j^{\mathcal{O}}\|^2}{\max_{i\in\{1,\ldots,n\}} w_{i,k}^{\mathcal{O}}} \le \Gamma_0 + nL\log\left(1+\frac{1}{\varsigma}\sum_{j=0}^{k}\|g_j^{\mathcal{O}}\|^2\right). \tag{30}$$

Assume now that

$$\sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2 \geq \max\left[\varsigma, \frac{\varsigma}{2} e^{\frac{\Gamma_0}{nL}}\right], \tag{31}$$

which implies

$$1 + \frac{1}{\varsigma} \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2 \leq \frac{2}{\varsigma} \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2 \quad \text{and} \quad \Gamma_0 \leq nL \log\left(\frac{2}{\varsigma} \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2\right),$$

and observe from the first part of (16) that, for all $i \in \{1, \ldots, n\}$ and all k,

$$w_{i,k}^{\mathcal{O}} \leq \sqrt{\varsigma + \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2}. \tag{32}$$

Thus, from (30), (31), and (32) we obtain that

$$\frac{\sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2}{\sqrt{2 \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2}} \leq 2nL \log\left(\frac{2}{\varsigma} \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2\right),$$

that is

$$\frac{\sqrt{\varsigma}}{2} \sqrt{\frac{2}{\varsigma} \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2} \leq 4nL \log\left(\sqrt{\frac{2}{\varsigma} \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2}\right). \tag{33}$$

If we now define

$$\gamma_1 = \frac{\sqrt{\varsigma}}{2}, \qquad \gamma_2 = 4nL, \qquad u = \sqrt{\frac{2}{\varsigma} \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2}, \tag{34}$$

we first note that our assumption that $\varsigma \leq (\frac{8nL}{3})^2$ ensures that $\gamma_2 > 3\gamma_1$. Furthermore, the inequality (33) can then be rewritten as

$$\gamma_1 u \leq \gamma_2 \log(u). \tag{35}$$

Let us denote by $\psi(u) \overset{\text{def}}{=} \gamma_1 u - \gamma_2 \log(u)$. Since $\gamma_2 > 3\gamma_1$, the equation $\psi(u) = 0$ admits two roots $u_1 \leq u_2$ and (35) holds for $u \in [u_1, u_2]$. The definition of u_2 then gives that

$$\log(u_2) - \frac{\gamma_1}{\gamma_2} u_2 = 0,$$

which is

$$u_2 e^{-\frac{\gamma_1}{\gamma_2} u_2} = 1.$$

Setting $z = -\frac{\gamma_1}{\gamma_2} u_2$, we obtain that

$$z e^z = -\frac{\gamma_1}{\gamma_2}.$$

Thus $z = W_{-1}(-\frac{\gamma_1}{\gamma_2}) < 0$, where W_{-1} is the second branch of the Lambert function defined over $[-\frac{1}{e}, 0)$. As $-\frac{\gamma_1}{\gamma_2} \geq -\frac{1}{3}$, z is well defined and thus

$$u_2 = -\frac{\gamma_2}{\gamma_1} z = -\frac{\gamma_2}{\gamma_1} W_{-1}\left(-\frac{\gamma_1}{\gamma_2}\right) > 0.$$

Therefore, using (34),

$$\sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2 = \frac{\varsigma}{2} u_2^2 = 32 n^2 L^2 \left| W_{-1}\left(-\frac{\sqrt{\varsigma}}{8nL}\right)\right|^2.$$

Hence, taking the average gives that

$$\underset{j \in \{0,\ldots,k\}}{\text{average}} \|g_j^{\mathcal{O}}\|^2 \leq 32 n^2 L^2 \left| W_{-1}\left(-\frac{\sqrt{\varsigma}}{8nL}\right)\right|^2 \cdot \frac{1}{k+1}. \tag{36}$$

Suppose now that (31) fails. Then, obviously,

$$\underset{j \in \{0,\ldots,k\}}{\text{average}} \|g_j^{\mathcal{O}}\|^2 \leq \max\left[\varsigma, \frac{\varsigma}{2} e^{\frac{\Gamma_0}{nL}}\right] \cdot \frac{1}{k+1}. \tag{37}$$

Case 2. Consider now the case where

$$\log\left(1 + \frac{1}{\varsigma} \sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2\right) \leq \log\left(1 + \frac{(k+1)\kappa_x^2}{\varsigma}\right).$$

Then inequality in (29) becomes

$$\sum_{j=0}^{k} \frac{\|g_j^{\mathcal{O}}\|^2}{\max_{i \in \{1,\ldots,n\}} w_{i,k}^{\mathcal{O}}} \leq \Gamma_0 + nL \log\left(1 + \frac{(k+1)\kappa_x^2}{\varsigma}\right). \tag{38}$$

If, on one hand,

$$\sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2 \geq \varsigma, \tag{39}$$

then we have that

$$\frac{1}{\sqrt{2}} \frac{\sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2}{\sqrt{\sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2}} \leq \Gamma_0 + nL \log\left(1 + \frac{(k+1)\kappa_x^2}{\varsigma}\right),$$

that is

$$\sum_{j=0}^{k} \|g_j^{\mathcal{O}}\|^2 \leq 2\left(\Gamma_0 + nL \log\left(1 + \frac{(k+1)\kappa_x^2}{\varsigma}\right)\right)^2. \tag{40}$$

Taking the average then gives that

$$\underset{j\in\{0,\ldots,k\}}{\text{average}} \|g_j^{\mathcal{O}}\|^2 \leq 2\left(\Gamma_0 + nL \log\left(1 + \frac{(k+1)\kappa^2}{\varsigma}\right)\right)^2 \cdot \frac{1}{k+1}. \tag{41}$$

If, on the other hand, (39) does not hold, then

$$\underset{j\in\{0,\ldots,k\}}{\text{average}} \|g_j^{\mathcal{O}}\|^2 \leq \frac{\varsigma}{k+1}. \tag{42}$$

We finally deduce (23) by considering the largest upper among (36), (37), (41), (42) and using (25). □

Theorem 2.3 demonstrates that prunAdag has a global rate of convergence in $O(\log(k)/\sqrt{k+1})$, in contrast with the original Adagrad algorithm for which the average gradient norm decreases like $O(1/\sqrt{k+1})$. Therefore, prunAdag can in general be (marginally) slower than the original Adagrad algorithm. This is due to the fact that deviating the gradient flow to reduce the magnitude of decreasable parameters comes at a convergence rate cost. Indeed, Lemma 2.1 shows that the decrease of the function at each iteration, expressed by equation (19) is governed by two positive sums in the right-hand side, depending on the gradient of optimisable components $g_{i,k}^{\mathcal{O}}$ and on the magnitudes of decreasable components $x_{i,k}^{\mathcal{D}}$. By contrast, the descent lemma (Lemma 2.1 in [8]) of first-order OFFO methods, such as Adagrad, only contains a single positive sum depending on the gradient entries. It is easy to see from equation (19) that the function value eventually decreases monotonically if the two positive sums in the right-hand side are dominated by the first, negative sum. However, we might expect a monotonic decrease of the function for large enough weights for the Adagrad algorithm, while the non-monotonic behaviour may persist until convergence for prunAdag. This is because the magnitude of the decreasable components in $\mathcal{D}_k$ is (hopefully) small at convergence but often non zero for all of them. Therefore, the second positive sum in (18), depending on the decreasable components in $\mathcal{D}_k$, might remain significant, even for large k, but its growth is fortunately bounded by a term in $\mathcal{O}(\log(k))$.

- If we start from an initial point that is far from any stationary point of the problem, we might expect the largest components of the gradient to be large in magnitude for several iterations, potentially exceeding the magnitudes of the decreasable components of the iterate. This is exactly the scenario described in Case 1 of the proof of Theorem 2.3. Consequently, we observe an empirical Adagrad-like almost linear decrease, as suggested by Eqs. (36) and (37), until the contribution of the decreasable components $x_{i,k}^{\mathcal{D}}$ exceeds that of the optimisable gradient components $g_{i,k}^{\mathcal{O}}$.
- Theorem 2.3 proves that the average norm of the gradient converges to zero. However, we cannot expect the same behaviour for the decreasable components of the iterates, which are only likely to be small in magnitude at convergence. This means that after a certain iteration, the contribution of the optimisable gradient will be smaller than that of the decreasable components in Eq. (26). Thus, Case 2 in the proof typically occurs for large k. As a consequence, we may then expect, for a sufficiently large k, a decrease of the order of $\log(k)/\sqrt{k+1}$ as suggested by Eq. (41).

Both these observations suggest that we might expect a faster decrease during the first iterations, followed by a potential slowdown when $\|g_k^{\mathcal{O}}\|$ becomes small, as can be observed for the non-rescaled prunAdag-V2 and prunAdag-V4 in Figs. 4 and 5 in the next section. In general, the rescaling of the step in the decreasable components promotes a faster convergence and this behaviour is not observed. Nevertheless, a fast convergence is not the only purpose of a pruning-aware method since robustness to pruning also needs to be considered.

3 Numerical experiments

We now present numerical tests on a variety of problems from different applications originating in

- a standard class of randomly generated under-determined linear least-squares problems,
- the SPARCO library for sparse signal recovery [46] (as supplied by S2MPJ [47]), which contains test cases from signal processing applications specifically designed for sparse optimisation,
- the “sparse coding step" in dictionary learning problem,
- minimizing the logistic function in binary classification problems on several well-know data sets.

These test problems were chosen to test if prunAdag is able to enhance convergence to a solution which is robust to pruning. (It is not our purpose to compare prunAdag to state-of-the-art techniques in each of the applications considered.)

We implemented four versions of prunAdag with different choices of $a_{i,k}$ and $b_{i,k}$ at Step 3 of Algorithm 2.1. Defining

$$\mathcal{S}_k \overset{\text{def}}{=} \{i \in \mathcal{R}_k^C \mid \text{sign}(x_{i,k}) = \text{sign}(g_{i,k})\},$$

we considered the rules described in Table 1 where $\mathcal{R}_k$ is given in (4), yielding the corresponding implementations prunAdag-V1, prunAdag-V2, prunAdag-V3 and prunAdag-V4.

In all four versions, the lower bounding sequence $a_{i,k}$ represents a fraction of the absolute value of *i*-th component of the iterate. In prunAdag-V2 and prunAdag-V4, the sequence linearly decreases with the iteration number, while in prunAdag-V1 and prunAdag-V3, in addition to the linearly decreasing factor, we rescale the sequence to match the magnitude of the gradient g_k in $\mathcal{R}_k$. A similar rescaling of the learning rate has proven effective in SFW [2, 33]. In our experience, choosing a lower bound sequence $a_{i,k}$ that decreases as $1/k$ helps to make the convergence of the method faster and reduces the oscillations due to the non-monotone behaviour of the algorithm. On the other hand, the upper bounding sequence $b_{i,k}$ in prunAdag-V3 and prunAdag-V4 is set to be equal to the iterate magnitude $|x_{i,k}|$. This choice prevents the Adagrad-like step from exceeding $|x_{i,k}|$, thus avoiding sign changes for the components in $\mathcal{A}_k$. Allowing potential sign changes in some components can be obtained by choosing any $b_{i,k} > |x_{i,k}|$, e.g. as in prunAdag-V1 and prunAdag-V2 where we do not consider any upper bound on $b_{i,k}$ ($b_{i,k} = \infty$), resulting in a larger set $\mathcal{A}_k$, as Fig. 2 shows. This implies that the number of optimisable components is then larger and the algorithm's speed is potentially enhanced. Conversely, the number of indices in $\mathcal{D}_k$ is smaller, potentially leading to less effective pruning. We illustrate the evaluation of the sets $\mathcal{O}_k$, $\mathcal{A}_k$ and $\mathcal{D}_k$ for the four versions of prunAdag in Sect. 3.1.2.

In addition, we consider a fifth version of prunAdag denoted Relevant only, which is identical to prunAdag, except that Step 3 is reduced to the definitions $\mathcal{O}_k = \mathcal{R}_k$ and $\mathcal{D}_k = \mathcal{R}_k^C$. Thus no "acceptable" parameter is added to the list of the optimisable ones in Relevant only.

We also implemented the deterministic version of the Frank-Wolfe method of [2] for unstructured pruning, considering two learning rates η_k in (3): a linearly decreasing rate $\eta_k = 1/(k+1)$ (FW1) and the adaptively rescaled rate [33] $\eta_k = \min\left[\frac{\beta\|g_k\|_{\mathcal{R}_k}}{\|v_k - x_k\|}, 1\right]$, where v_k is given in (2) and $\beta \in (0,1)$ (FW2). Finally, prunAdag reduces to the standard Adagrad algorithm if one chooses $T = n$ and avoids performing any classification of the parameters. We set $\varsigma = 1/100$ as for prunAdag and the Adagrad implementations.

The algorithms are implemented in MATLAB R2021b on a 64-bit Samsung/Galaxy with 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz and 8 GB of RAM, under Windows 11 version 23H2.

All experiments are randomly initialized with a normalized starting point with exactly T nonzero entries, representing a feasible point for FW1 and FW2 whenever the T-support-norm-ball has radius greater than 1. We performed 20 runs per test problem and report the complete averaged results in Tables 2, 3 and 4. The best results achieved among all the methods are highlighted in bold in the Tables. In what follows, we also show plots for a single run which are selected to visualize the typical behaviour of the methods. We set $T = n/10$ in both our method and the FW variants (but we propose an analysis on the impact of the parameter T in Sect. 3.1.3). We also tuned, for each set of experiments, the stepsize τ in the FW implementations (named τ_1 and τ_2 in FW1 and FW2, respectively) and the learning parameter β of FW2 by trial and error. In the comparison of prunAdag with FW, each algorithm is terminated when either $\|g_k\| \leq 10^{-9}$ or when 10^4 iterations have been performed, unless otherwise specified. When analyzing the behaviour of prunAdag and the comparison among its variants, we report plots where fewer iterations were run for better plot readability (see Figs. 1, 2, 3).

Throughout this section we perform pruning on the solution *after optimisation* using two different strategies, depending on the analysis considered. Therefore, either we fix a scalar threshold δ and we remove all the components below this threshold, or we choose a sparsity level of the solution σ (in percentage) and we set the threshold δ to achieve a σ-sparse solution after pruning. (Note that the sparsity σ of the solution after pruning and the percentage of parameters removed are equivalent quantities; therefore, we will use the two terms interchangeably.)

We now describe two performance measures that will be used below to discuss the numerical results. Given a fixed $\delta > 0$, let x be the approximated solution returned by some pruning-aware algorithm and denote by $\bar{x}$ the pruned version of x, that is the vector x whose components with magnitude less than δ have been zeroed out. We then define a quality measure to evaluate the robustness of the pruning approach considering

$$\rho \overset{\text{def}}{=} \|g(\bar{x})\| \quad \text{and} \quad \omega \overset{\text{def}}{=} \sqrt{|f(\bar{x}) - f(x)|}. \tag{43}$$

Both these quantities provide estimates of how much pruning perturbs the solution from x. A small value for ρ means that the pruned solution $\bar{x}$ is close to stationarity, while a small ω means that the objective function's value at the pruned solution does not differ much from that at x.

3.1 Random linear least-squares

In the first set of experiments, we consider a class of randomly generated linear least-squares of the form

$$f(x) = \frac{1}{2}\|Ax - b\|^2, \tag{44}$$

with five different matrices $A \in \mathbb{R}^{m\times n}$ as in [48] and [49]. Specifically, we choose six different classes ($A1$,...,$A6$) of matrix A such that

$A1$) A is randomly generated from a normal Gaussian distribution,

$A2$) A is a random orthogonal matrix,

$A3$) A is random and has orthogonal columns,
$A4$) A is random and has orthogonal rows,
$A5$) $A \in (0, 1)$ generated from a Bernoulli distribution,
$A6$) A is obtained from the discrete cosine transform matrix of dimension n.

We set $m = 100$ and $n = 1000$, making the problem under-determined with a relatively large-dimensional subspace of solutions. We then generate a random solution x^* from a standard normal distribution ($x^* = \mathrm{randn}(n, 1)$ in MATLAB notation) and compute the right-hand side as $b = Ax^*$.

3.1.1 Optimisable versus relevant

We first use the random least-squares problem A3 to show the advantage of extending the parameter classification from relevant/irrelevant to optimisable/decreasable. Figure 1 shows the results of running prunAdag-V1, prunAdag-V2, prunAdag-V3 and prunAdag-V4 (which differ by the choice of the bounding sequences $a_{i,k}$ and $b_{i,k}$ as defined in Table 1) in comparison with the Relevant only version.

The contrast between Relevant only and the other version is striking in terms of achieved number of components below the sparsity threshold and, consequently, of robustness to pruning. Not only Relevant only is much less efficient in this respect, but it does not converge faster to a stationary point than the other versions (except when compared to prunAdag-V3, which achieves the best sparsity). In our experience, this behaviour is quite general and, in our view, fully justifies the introduction of $\mathcal{A}_k$ in Step 3.

3.1.2 Classification and convergence for the four prunAdag versions

We next illustrate the evolution of the cardinality of the index sets $\mathcal{O}_k$, $\mathcal{A}_k$ and $\mathcal{D}_k$ along iterations for the four versions of prunAdag.

Figure 2 shows the cardinality of the sets $\mathcal{O}_k$, $\mathcal{A}_k$, and $\mathcal{D}_k$ for all the versions of prunAdag when solving the random least-squares problem A1. As shown by this example, the cardinality does vary significantly from version to version, and we may expect these variations to affect performance. We observe that robustness to pruning is unsurprisingly better when the size of $\mathcal{D}_k$ is larger, favoring prunAdag-V3 and prunAdag-V4 on average. If we now turn to speed of convergence to stationary points, the conclusion is less clear-cut. While, for prunAdag-V1 and prunAdag-V3 (that are the methods using gradient rescaling in $a_{i,k}$), the speed seems to improve with the size of $\mathcal{A}_k$, the effect is more problem-dependent for the (unscaled) prunAdag-V2 and prunAdag-V4.

3.1.3 Influence of *T*

As can be expected, the choice of the target number of relevant parameters T does influence the behaviour of the four versions of prunAdag. As we now show for problem A1 in Fig. 3, this effect may vary from version to version. Indeed, asking for a small T does not necessarily result in a large final number of parameters with small magnitude, as is clear for the (unscaled) prunAdag-V2. Fortunately, the behaviour

Table 2 Random least-squares: the error measure ρ for different percentages of pruned components σ

Case	σ	V1	V2	V3	V4	FW1	FW2	Adagrad
A1	10%	$2.2 \cdot 10^{-5}$	1.39	$9.4 \cdot 10^{-10}$	0.88	6.34	8.92	670
	20%	0.01	1.39	$9.7 \cdot 10^{-10}$	0.88	6.34	25.67	$1.7 \cdot 10^{3}$
	30%	0.21	1.38	$5.2 \cdot 10^{-4}$	0.88	6.34	47.2	$3.0 \cdot 10^{3}$
	40%	1.69	1.36	0.17	0.88	6.43	79.1	$4.4 \cdot 10^{3}$
	50%	12.9	1.35	9.73	**0.89**	7.52	109	$6.0 \cdot 10^{3}$
A2	20%	$8.7 \cdot 10^{-4}$	$8.2 \cdot 10^{-4}$	$5.9 \cdot 10^{-6}$	$1.1 \cdot 10^{-3}$	$4.5 \cdot 10^{-3}$	0.39	0.61
	40%	0.02	$8.2 \cdot 10^{-4}$	$6.0 \cdot 10^{-6}$	$1.1 \cdot 10^{-3}$	$5.4 \cdot 10^{-3}$	0.40	1.72
	60%	0.20	0.12	$5.3 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$	0.03	0.42	3.57
	70%	0.63	0.57	0.15	0.21	**0.14**	0.44	4.90
	80%	1.62	1.67	0.67	1.06	0.60	**0.49**	6.43
A3	20%	$4.1 \cdot 10^{-3}$	0.02	$9.7 \cdot 10^{-10}$	$9.7 \cdot 10^{-3}$	0.06	0.09	6.29
	30%	0.12	0.02	$9.7 \cdot 10^{-10}$	$9.7 \cdot 10^{-3}$	0.06	0.44	29.7
	40%	0.38	0.02	$3.4 \cdot 10^{-5}$	$9.7 \cdot 10^{-3}$	0.06	0.70	44.1
	50%	1.19	0.01	0.03	$9.8 \cdot 10^{-3}$	0.08	1.00	60.1
	60%	3.57	0.33	0.50	0.23	**0.22**	1.38	76.7
A4	20%	$6.1 \cdot 10^{-4}$	$8.8 \cdot 10^{-4}$	$4.2 \cdot 10^{-8}$	$9.7 \cdot 10^{-4}$	$4.5 \cdot 10^{-3}$	0.38	0.65
	30%	$4.1 \cdot 10^{-3}$	$8.8 \cdot 10^{-4}$	$4.2 \cdot 10^{-8}$	$9.7 \cdot 10^{-4}$	$4.6 \cdot 10^{-3}$	0.38	1.13
	50%	0.06	$6.9 \cdot 10^{-3}$	$7.5 \cdot 10^{-6}$	$9.7 \cdot 10^{-4}$	0.01	0.39	2.65
	70%	0.65	0.57	**0.13**	0.22	0.14	0.42	5.12
	90%	3.98	4.10	2.51	3.34	2.53	**1.91**	8.43
A5	10%	$3.7 \cdot 10^{-4}$	1.58	$9.5 \cdot 10^{-10}$	0.88	6.34	8.96	648
	20%	0.03	1.58	$9.5 \cdot 10^{-10}$	0.88	6.35	25.4	$1.7 \cdot 10^{3}$
	30%	0.34	1.57	$6.5 \cdot 10^{-4}$	0.88	6.35	47.6	$3.0 \cdot 10^{3}$
	40%	2.41	1.54	**0.12**	0.88	6.47	77.0	$4.4 \cdot 10^{3}$
	50%	14.6	1.49	7.64	**0.89**	8.21	113	$6.0 \cdot 10^{3}$

Table 2 (continued)

Case	σ	V1	V2	V3	V4	FW1	FW2	Adagrad
A6	10%	$2.2 \cdot 10^{-3}$	$4.3 \cdot 10^{-4}$	$3.0 \cdot 10^{-3}$	$3.6 \cdot 10^{-4}$	$6.4 \cdot 10^{-3}$	1.40	0.20
	30%	0.11	0.13	0.29	0.10	**0.02**	1.40	1.34
	50%	0.64	0.80	1.01	**0.73**	0.35	1.41	3.15
	60%	1.23	1.47	1.60	1.40	**0.91**	1.42	4.2
	70%	2.2	2.5	2.4	2.4	1.89	**1.44**	5.33

Table 3 SPARCO problems: the error measure ρ and different percentages of pruned components σ

Problem	σ	V1	V2	V3	V4	FW1	FW2	Adagrad
S3	10%	$4.2 \cdot 10^{-5}$	$3 \cdot 10^{-3}$	$9.6 \cdot 10^{-10}$	$1.6 \cdot 10^{-3}$	0.02	0.41	1.52
	30%	$8.1 \cdot 10^{-3}$	$3.1 \cdot 10^{-3}$	$1.2 \cdot 10^{-4}$	$1.6 \cdot 10^{-3}$	0.02	0.41	7.02
	50%	0.08	0.05	0.12	$2.7 \cdot 10^{-3}$	0.12	0.41	13.5
	60%	0.25	0.08	0.55	$6.1 \cdot 10^{-3}$	0.12	0.41	17.2
	70%	6.21	6.69	6.29	5.56	**0.31**	0.41	21.9
S5	20%	0.02	0.03	$3.6 \cdot 10^{-5}$	0.02	0.11	0.01	7.68
	30%	0.06	0.03	$1.5 \cdot 10^{-3}$	0.02	0.11	0.22	57.0
	40%	0.23	**0.02**	0.03	**0.02**	0.14	0.42	88.3
	50%	0.86	0.03	0.34	**0.02**	0.44	0.63	125
	60%	4.49	2.21	1.78	1.78	1.83	**0.87**	164
S7	10%	$2.5 \cdot 10^{-4}$	$4.6 \cdot 10^{-6}$	$1.7 \cdot 10^{-5}$	$6.0 \cdot 10^{-4}$	0.01	0.05	0.08
	30%	$2.3 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$	$1.7 \cdot 10^{-5}$	$6.0 \cdot 10^{-4}$	0.01	0.07	0.43
	50%	$7.0 \cdot 10^{-3}$	$7.0 \cdot 10^{-3}$	$1.7 \cdot 10^{-5}$	$6.0 \cdot 10^{-4}$	0.01	0.14	0.95
	70%	0.02	0.02	$2.0 \cdot 10^{-4}$	$6.0 \cdot 10^{-4}$	0.01	0.27	2.0
	90%	0.03	0.05	$3.3 \cdot 10^{-3}$	$3.3 \cdot 10^{-3}$	0.01	0.50	2.35
S9	10%	**1.23**	1.62	5.52	5.51	44.9	2.4	2.5
	20%	**1.23**	1.62	5.53	5.51	44.9	9.56	9.46
	30%	**1.23**	1.62	5.52	5.51	44.9	16.2	31.8
	40%	**1.23**	1.67	5.52	5.51	45.9	45.4	40.8
	50%	**1.23**	2.05	5.46	5.54	33.6	75.7	60.6
S11	10%	$3.3 \cdot 10^{-6}$	0.08	$9.7 \cdot 10^{-10}$	0.50	19.3	13.1	165
	20%	$4.2 \cdot 10^{-3}$	0.08	$9.7 \cdot 10^{-10}$	0.50	19.3	47.3	489
	30%	0.13	0.07	$2.5 \cdot 10^{-9}$	0.50	19.3	84.3	975
	40%	1.19	1.01	$3.9 \cdot 10^{-4}$	0.50	19.3	152	$1.5 \cdot 10^{3}$
	50%	5.6	12.1	**0.37**	0.50	31.6	332	$3.7 \cdot 10^{3}$

of the (scaled) prunAdag-V3 is typically more consistent. For problem A1 and in general, choosing a smaller T then results in a larger number of solution components whose absolute value is below the sparsity threshold, but, as can be expected, at the price of slower convergence (see the two bottom panels of Fig. 3).

3.1.4 prunAdag, FW and Adagrad

We now turn to results obtained when running the four versions of prunAdag along FW and the standard Adagrad, using $\tau_1 = 50$ for FW1, $\tau_2 = 100$, and $\beta = 0.001$ for FW2. Table 2 reports a comparison between the four versions of prunAdag, FW1, FW2, and Adagrad, and displays, for each random matrix and each algorithm, the pruning quality measure ρ defined in (43), averaged over 20 runs, for 5 different percentages σ of pruned components. In summary, these results show that, when the percentage of pruned components σ is below 50%, prunAdag-V3 is the most reliable method in five of the six problems considered, followed by prunAdag-V4. On the contrary, prunAdag-V1 is extremely reliable when the percentage of pruned parameters is below 30%, but its performance degrades rapidly as this percentage increases. For very aggressive pruning, that is for σ around 70%, FW1, FW2 and prunAdag-V4 exhibit the best results, making those methods particularly suitable for applications where sparsity is to be preferred to high accuracy. We illustrate these results by graphically detailing, in Fig. 4, the (typical) results obtained for problem A2.

This figure shows that version prunAdag-V1 is the fastest algorithm, performing comparably to the original Adagrad algorithm. However, it is the less robust to pruning among all prunAdag versions. prunAdag-V3 satisfies the stopping criterion on the gradient norm while being by far the best choice in terms of robustness to pruning up to 50% of sparsity. Despite their poor performance for lower percentages of pruned components, FW1 and FW2 remain a valid alternative for very aggressive pruning, even though their convergence is the slowest among the algorithms considered. However, one should remember that FW is quite sensitive to the choice of its parameters and those used here have been tuned once for all the least-square problems considered (faster convergence can sometimes be achieved by further problem-by-problem tuning).

3.2 SPARCO problems

The aim of sparse signal recovery is finding a sparse representation of an observed noisy signal b as a linear combination of some redundant dictionary A. Typically, the dictionary is a wide matrix with more columns than rows, consisting of various bases such as wavelet, discrete cosine, and Fourier. The SPARCO library [46] as supplied by S2MPJ [47] includes examples of these problems for different dictionaries. Given a sparse vector x^*, the observation is generated as $b = Ax^* + r$, where r is additive noise vector of appropriate dimension and A is a fixed dictionary. The aim is to recover a robust solution by solving the related under-determined least-squares and using the prunAdag algorithm instead of enhancing sparsity using a regularizing

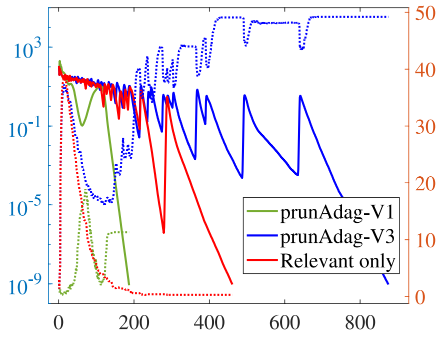
Table 4 Binary classification. Percentage of correctly classified samples in the testing set

Problem	σ (%)	V1 (%)	V2 (%)	V3 (%)	V4 (%)	FW1 (%)	FW2 (%)	Adagrad (%)
GISETTE	75	93.75	94.50	93.68	94.51	94.66	94.30	94.13
	80	93.75	94.42	93.55	94.33	94.66	94.26	93.61
	85	93.40	94.03	93.10	94.05	94.68	94.35	92.45
	90	92.63	93.55	92.20	93.70	94.31	93.85	89.76
	95	87.13	90.86	83.73	89.53	91.81	90.20	83.70
MNIST	75	81.90	81.91	80.60	80.46	83.65	81.85	80.15
	80	81.48	81.53	80.48	80.41	83.33	81.21	79.45
	85	80.43	80.53	80.35	80.12	82.80	80.83	77.32
	90	78.02	78.15	78.83	78.85	81.51	79.02	75.68
	95	72.26	72.38	74.11	73.92	76.60	72.48	66.06
REGEDO	70	96.36	95.93	96.36	96.33	97.33	97.33	96.20
	75	96.33	95.93	96.50	96.67	97.33	97.33	95.86
	80	96.20	95.83	96.23	96.76	97.33	97.33	96.00
	85	96.76	96.10	96.96	96.60	97.33	97.46	93.06
	90	95.90	95.67	84.50	96.40	97.33	98.60	65.20
A9A	65	81.65	82.16	81.87	81.97	83.07	83.33	76.93
	70	81.33	81.34	81.53	81.67	83.37	83.10	72.78
	75	80.68	80.86	80.77	80.22	83.40	83.03	71.25
	80	79.53	78.63	79.55	79.01	82.70	82.85	71.25
	85	75.83	75.95	76.20	76.23	82.36	81.60	72.76
MOLECULE	65	79.30	78.77	78.70	78.39	78.53	79.54	65.00
	70	77.51	78.11	78.43	78.18	78.56	78.95	62.76
	75	76.11	76.22	78.04	78.28%	78.32	79.12	62.86
	80	74.65	74.90	77.80	78.11	76.74	75.17	62.51
	85	71.01	70.83	75.69	76.53	76.50	73.00	60.73

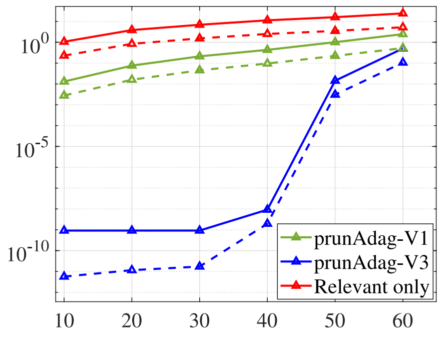
term[2] For these tests, we set the FW parameter to $\tau_1 = 100$, $\tau_2 = 100$ and $\beta = 0.001$. The complete results for different percentages σ of pruned components are given in Table 3. We observe that prunAdag-V3 is the most robust algorithm in four out of five considered examples when the percentage of pruned components is below 50%. As for random-least squares, FW1 and prunAdag-V3 are the most reliable for a percentage of pruned components σ exceeding 50%, thus they represent a better choice for very aggressive pruning.

Figure 5 shows that FW2 is the fastest algorithm to converge; however, it does not exhibit strong robustness properties, showing a similar behaviour to Adagrad algorithm. By contrast, prunAdag-V3 reaches the tolerance set for the norm of the gradient and it has the lowest value of the error measure ρ up to 50% of pruned components and the lowest value for error measure ω up to 40% of pruned components. All versions of prunAdag exhibit more robust performance than Adagrad algorithm.

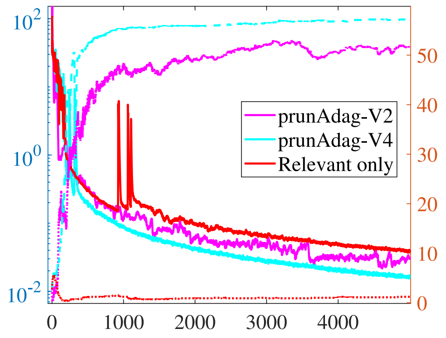
[2] Since our focus is robustness to pruning, we deliberately ignore the robustness-to-noise issue that might occur if we solve the non-regularised least-squares. However, as long as the assumptions AS.1, AS.2, and AS.3 are satisfied, any regularisation term can be added to the objective function.



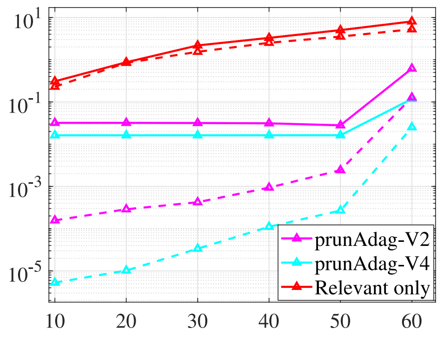
(a) gradient norm (left), sparsity (right)



(b) measures ρ (solid) and ω (dashed)



(a) gradient norm (left), sparsity (right)



(b) measures ρ (solid) and ω (dashed)

Fig. 1 Effect of introducing the class of optimisable parameters within the minimization framework of prunAdag. **a** Gradient norm (solid line) on the left y-axis and percentage of parameters below $\delta = 10^{-3}$ (dotted line) on the right y-axes along the iterations. **b** Error measures ρ (solid) and ω (dashed) for different percentages of pruned components after the optimisation. Each algorithm runs for 5000 iterations or until convergence (Random least-squares A3)

3.3 Sparse coding in dictionary learning

Let Y be a given dataset, the aim of dictionary learning is to find a dictionary D and a sparse coefficient matrix X, such that $Y \approx DX$. This problem is frequently solved by alternating optimisation and we focus on the so-called sparse coding step, that is, given D we aim at finding a sparse X such that $Y \approx DX$. Given $k > 0$, for each element of the dataset y (column of Y) and positive integer m, the standard formulation of the sparse coding step is the following

$$\min_x \|y - Dx\|^2 \qquad \text{such that } \|x\|_0 \leq m, \tag{45}$$

where ℓ_0 denotes the zero-norm of a vector, defined as the number of its nonzero entries. We test our framework by addressing problem (45) removing the explicit constraint and using prunAdag to find a possibly dense solution that is robust to pruning. This approach has the advantage of allowing *a posteriori* pruning with different sparsity levels.

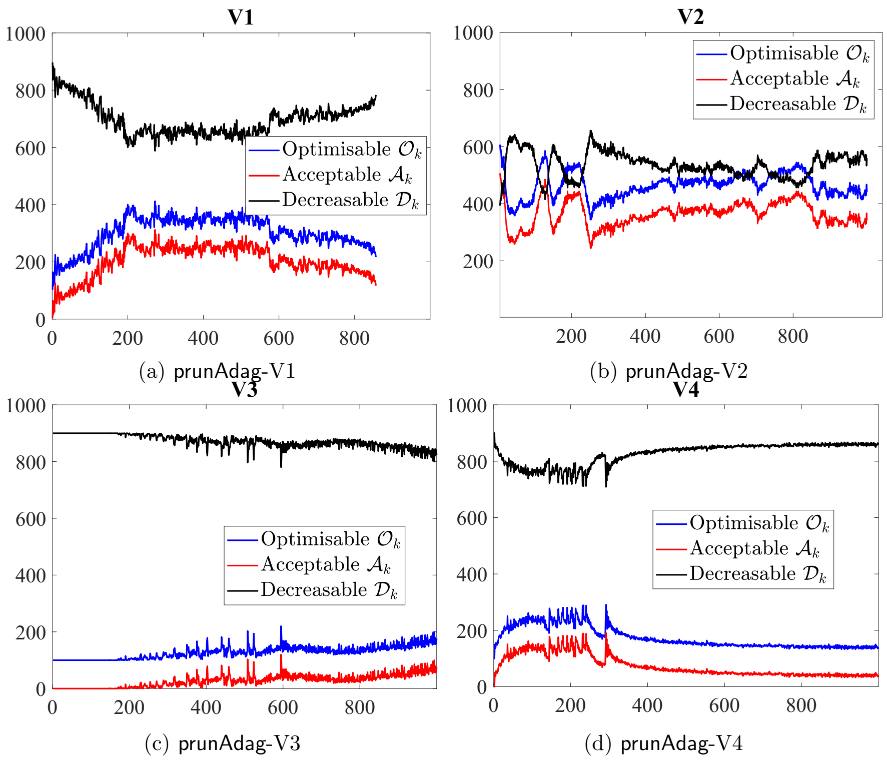
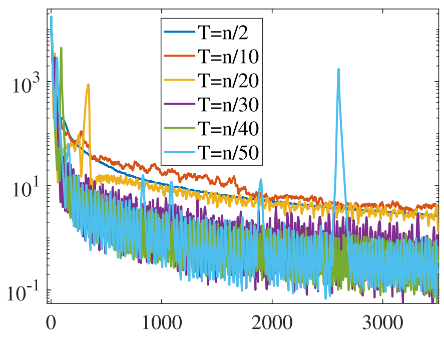


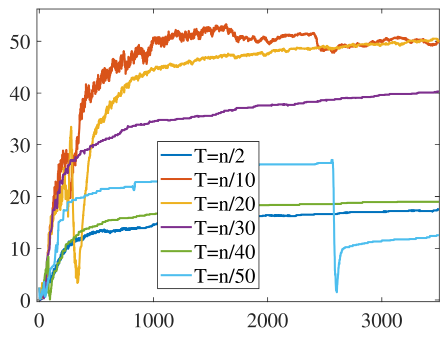
Fig. 2 Dynamic of parameters' classification that corresponds to the cardinality of the sets $\mathcal{O}_k$, $\mathcal{A}_k$, and $\mathcal{D}_k$ within 1000 iteration or until convergence (Random least-squares A1)

In our tests, we consider a subset of the MNIST data set [14] and we generated D in problem (45) by using KSVD [50], the state-of-the-art solver for solving the dictionary learning problems.[3] The data set Y has dimension 784×4000, the dictionary D has 784 rows and 1000 columns and c in (45) is chosen equal to 100. We set the FW parameters to $\tau_1 = 10$, $\tau_2 = 20$, and $\beta = 0.001$. In Fig. 6 we illustrate the reconstruction of an instance $y \approx D\bar{x}$, where $\bar{x}$ is the pruned solution obtained by prunAdag-V1 algorithm for increasing percentages of pruned parameters. The results show that the solution begins to degrade when more than 40% of parameters are pruned. Figure 7 confirms the visual intuition since the error measure ρ remains below 10^{-1} for prunAdag-V1 algorithm. Moreover, Fig. 7 on the left highlights that prunAdag-V1 is the most robust algorithm up to 70% of pruned components, while prunAdag-V3 and prunAdag-V4, despite their poor global accuracy, remain the best choice for aggressive pruning with more than 80% of pruned parameters.

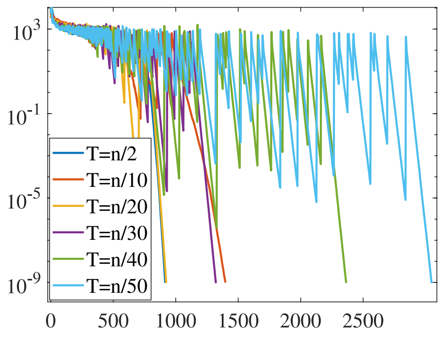
[3] We used the Matlab implementation KSVD-Box v13 of K-SVD available at http://www.cs.technion.ac.il/~ronrubin/software.html with default parameters.



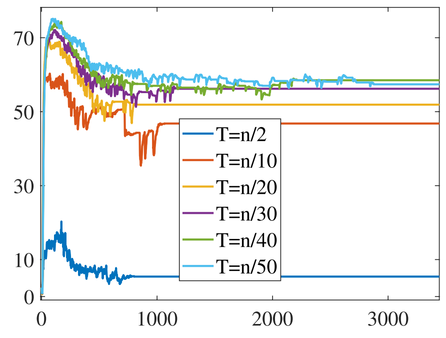
(a) prunAdag-V2 - gradient norm



(b) prunAdag-V2 - sparsity



(a) prunAdag-V3 - gradient norm



(b) prunAdag-V3 - sparsity

Fig. 3 Norm of the gradient on the left and percentage of parameters below $\delta = 10^{-3}$ on the right along 3000 iterations or until convergence for prunAdag-V2 and prunAdag-V3 and different target numbers of relevant parameters T (Random least-squares A1)

3.4 Binary classification

Finally, we test our method on the averaged logistic loss for binary classification. We assume that a labeled training set $\{y_i, z_i\}$ with $y_i \in \mathbb{R}^n$ and $z_i \in \{0, 1\}$ for $i = 1, \ldots, N$ is available, where z_i classifies each sample into two distinct classes. The averaged logistic loss over all samples is neither linear nor convex, and it is defined as

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} \log(1 + e^{-z_i y_i^T x}). \tag{46}$$

If the number of features in the data set is large, we expect that some may be redundant or irrelevant in the classification process; thus, pruning can be used to achieve a sparse solution that does not consistently degrade the classification performance. We use prunAdag to minimize the function in (46) on the training set and to promote convergence towards a solution x in which the largest components correspond to relevant features. Then, we prune the parameters to achieve different levels of sparsity σ and evaluate the prediction on the testing set using the pruned solution. For

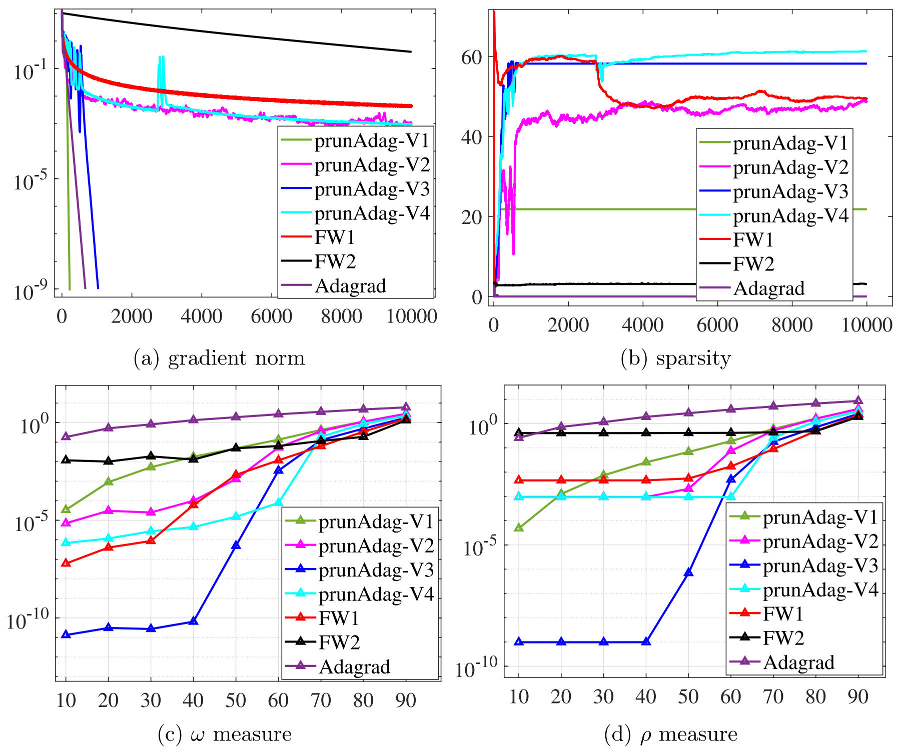


Fig. 4 On top, **a** gradient norm and **b** percentage of components below a fixed threshold $\delta = 10^{-3}$ along iterations; at the bottom, **c** error measure ω and **d** error measure ρ for different percentages of pruned components after the optimisation (Random least-squares A2)

this experiment, we select small-size data sets[4] that have at least 100 features, that are MNIST[5] [14], GISETTE [51], REGEDO [52], A9A [51], and MOLECULE [51]. We split the training and testing set following a ratio 70:30. The data is normalized using min-max normalization and each algorithm is randomly initialized and stopped after 2000 iterations. We set the FW parameters to $\tau_1 = 10$, $\tau_2 = 100$, and $\beta = 0.5$. The results are collected in Table 4, where we show the average test accuracy and the percentage of components pruned for each algorithm. In Fig. 8 we analyze the performance of the algorithms on the GISETTE data set.

Table 4 shows that all four versions of prunAdag yield a consistent reduction in the number of parameters of the model over 80% for all problems, without affecting the classification performance, and are significantly more robust to pruning than Adagrad. We are not able to identify one of the four versions that outperforms the others; however, we can observe that prunAdag-V4 tolerates the largest number of pruned components. In Fig. 8 (c) we compare Adagrad, FW and prunAdag in terms of average robustness to pruning parameters after training on the GISETTE data set

[4] We randomly selected 1000 samples for MNIST and A9A.

[5] Classification between even and odd numbers.

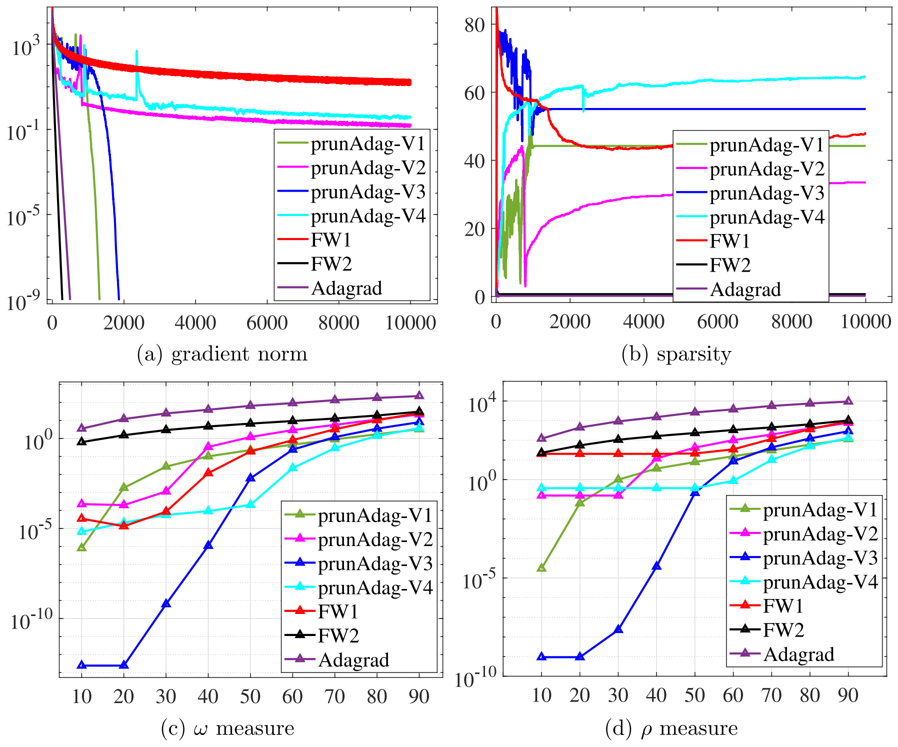


Fig. 5 On top, **a** gradient norm and **b** percentage of components below the threshold $\delta = 10^{-3}$ along iterations; at the bottom, **c** error measure ω and **d** error measure ρ for different percentages of pruned components σ after the optimisation (Sparco 11)

from 20 independent starting points. The figure clearly shows that all four versions of prunAdag are more robust to pruning than Adagrad, as small components correspond to irrelevant components of the model. Indeed, all the versions show a stable accuracy for a 70% sparse solution and that of prunAdag-V2 and prunAdag-V4 are not affected significantly by pruning 90% of solution's components. One also notes the excellent performance of FW on this example and level of sparsity. It seems that, even if the norm of the gradient at the final iterate is still larger after the training phase than is the case for prunAdag (see Fig. 8a)), it is sufficient to produce good model predictions.

4 Conclusions

We have proposed a new first-order OFFO method, named prunAdag, intended for applications where pruning of the variables/parameters is desirable. The new "pruning-aware" algorithm uses a new strategy to classify parameters at each iteration of prunAdag algorithm into "optimisable" and "decreasable", instead of "relevant" and "irrelevant" as suggested in [1], and extending the concept introduced in [2], where the optimisation is performed on the components related to the largest par-

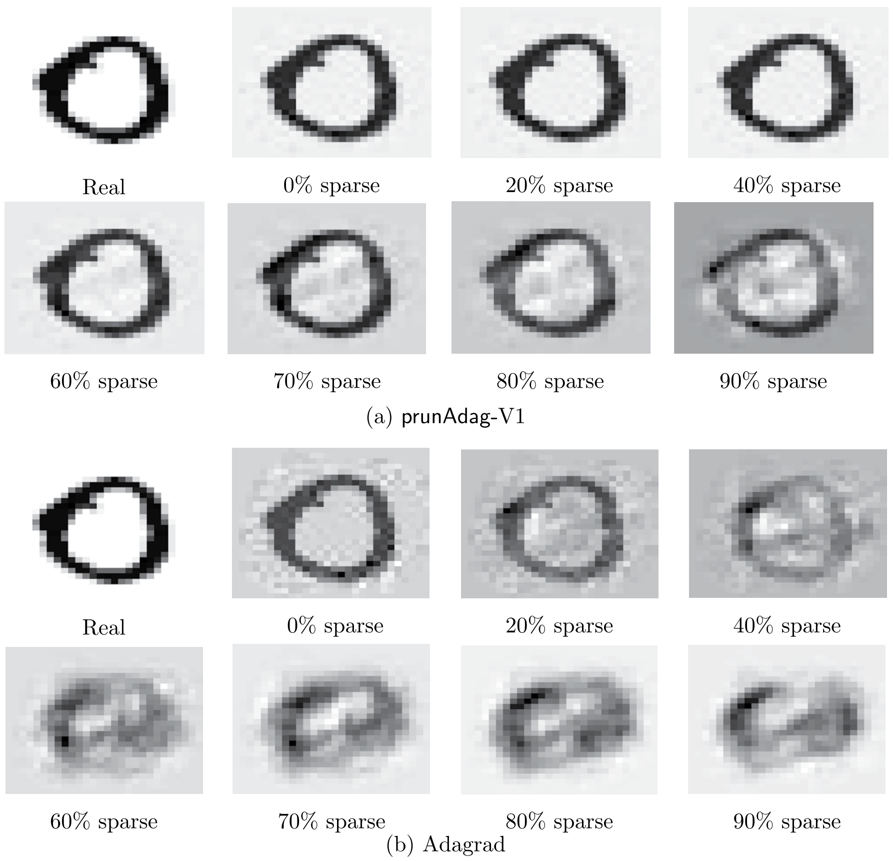


Fig. 6 MNIST sparse coding step in dictionary learning. Visual representations of pruned solutions of our best algorithm prunAdag-V1 in dictionary learning application, for different percentage σ of pruned parameters in the first two rows (**a**). Adagrad solution for different levels of pruned components in the last two rows (**b**)

tial derivatives. It also features a new framework to update parameters in these two classes separately, based on an Adagrad-like step for the first and on an adaptive trust-region approach to decrease the magnitude of the variables in the second. We proved the convergence of the method to first-order stationary points with global rate $O(\log(k)/\sqrt{k+1})$. Finally, we conducted numerical experiments on several real-world applications, such as sparse signal recovery, dictionary learning, and binary classification. These experiments suggest that the new approach (and its prunAdag-V3 version in particular) has a clear practical potential.

While we have, in this paper, focused on the “deterministic case" where the gradient values are computed exactly, the “stochastic case" where the gradient may be contaminated by random noise (such as sampling) is also clearly of interest, and the object of current research. Also of interest is the inclusion of momentum in prunAdag or a similar algorithm.

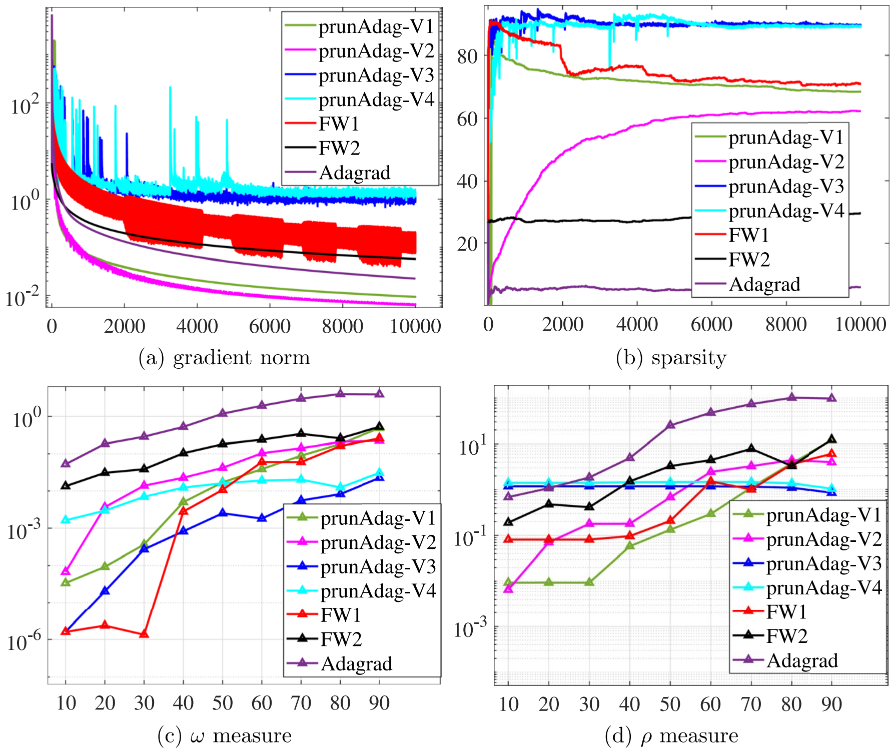


Fig. 7 On top, **a** gradient norm decrease and **b** percentage of parameters below the threshold $\delta = 10^{-2}$ along iterations; at the bottom, **c** error measure ω and **d** error measure ρ for pruned components percentage σ from 10% to 90% on the *x*-axis (MNIST)

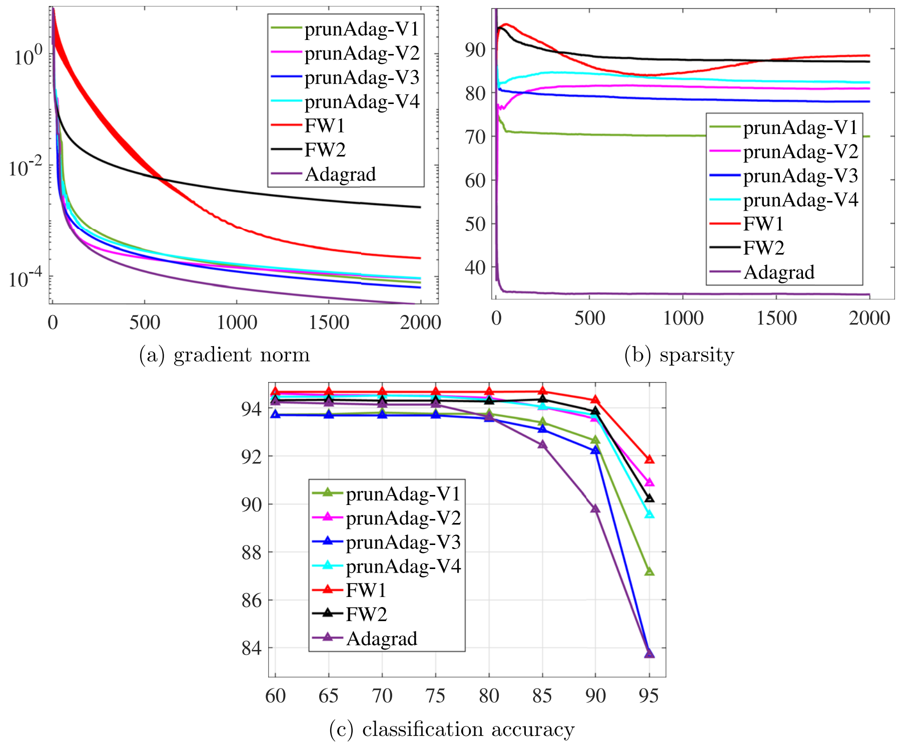


Fig. 8 On top, **a** gradient norm decrease and **b** percentage of parameters below the threshold $\delta = 10^{-1}$ along iterations; at the bottom, **c** average percentage of correctly classified samples as a function of sparsity σ from 60% and 95% (GISETTE)

Acknowledgements Philippe Toint gratefully acknowledges the friendly support of DIEF (University of Florence) during his visit in the fall of 2024.

Author contributions All authors contributed equally to the writing of this article. All authors reviewed the manuscript.

Funding Open access funding provided by Università degli Studi di Firenze within the CRUI-CARE Agreement. The work of M.P. and G.S. was partially supported by INdAM-GNCS under the project CUP E53C23001670001. The research of M.P. was partially granted by the Italian Ministry of University and Research (MUR) through the PRIN 2022 “MOLE: Manifold constrained Optimization and LEarning,” code: 2022ZK5ME7 MUR D.D. financing decree n. 20428 of November 6th, 2024 (CUP B53C24006410006), and by PNRR - Missione 4 Istruzione e Ricerca - Componente C2 Investimento 1.1, Fondo per il Programma Nazionale di Ricerca e Progetti di Rilevante Interesse Nazionale (PRIN) funded by the European Commission under the NextGeneration EU programme, project “Advanced optimization METhods for automated central veIn Sign detection in multiple sclerosis from magneTic resonAnce imaging (AMETISTA),” code: P2022J9SNP, MUR D.D. financing decree n. 1379 of 1st September 2023 (CUP E53D23017980001). G.S. acknowledges the support by the European Union (ERC consolidator, eLinoR, no 101085607).

Data availability The data that support the findings of this study are available from the corresponding author upon request.

Declarations

Ethical approval and consent to participate Not applicable.

Consent for publication Not applicable.

Conflict of interest The authors declare that they have no Conflict of interest.

References

1. Ding, X., Zhou, X., Guo, Y., Han, J., Liu, J. et al.: Global sparse momentum SGD for pruning very deep neural networks. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
2. Zimmer, M., Spiegel, C., Pokutta, S.: Compression aware training of neural networks using frank-wolfe. In: Mathematical Optimization for Machine Learning: Proceedings of the MATH+ Thematic Einstein Semester 2023, p. 137 (2025)
3. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**(7), 2121–2159 (2011)
4. McMahan, H.B., Streeter, M.: Adaptive bound optimization for online convex optimization. arXiv preprint arXiv:1002.4908 (2010)
5. Kingma, D.P.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
6. Tieleman, T., Hinton, G.: Lecture 6.5-Rmsprop, coursera: Neural networks for machine learning. University of Toronto, Technical Report, vol. 6 (2012)
7. Zeiler, M.D.: ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
8. Gratton, S., Jerad, S., Toint, P.L.: First-order objective-function-free optimization algorithms and their complexity. arXiv preprint arXiv:2203.01757 (2022)
9. Gratton, S., Kopaničáková, A., Toint, P.L.: Multilevel objective-function-free optimization with an application to neural networks training. SIAM J. Optim. **33**(4), 2772–2800 (2023)
10. Gratton, S., Jerad, S., Toint, P.L.: Complexity of a class of first-order objective-function-free optimization algorithms. Optim. Methods Softw. pp. 1–31 (2024)
11. Conn, A.R., Gould, N.I., Toint, P.L.: Trust Region Methods. SIAM, Philadelphia (2000)
12. Yuan, Y.-X.: Recent advances in trust region algorithms. Math. Program. **151**, 249–281 (2015)
13. Reed, R.: Pruning algorithms-a survey. IEEE Trans. Neural Netw. **4**(5), 740–747 (1993)
14. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. In: Advances in Neural Information Processing Systems, vol. 2 (1989)
15. Zhu, M., Gupta, S.: To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878 (2017)
16. Zhang, X., Zou, J., He, K., Sun, J.: Accelerating very deep convolutional networks for classification and detection. IEEE Trans. Pattern Anal. Mach. Intell. **38**(10), 1943–1955 (2015)
17. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in neural information processing systems, vol. 27 (2014)
18. Yu, X., Liu, T., Wang, X., Tao, D.: On compressing deep models by low rank and sparse decomposition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7370–7379 (2017)

19. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1. arXiv preprint arXiv:1602.02830 (2016)
20. Wang, P., Hu, Q., Zhang, Y., Zhang, C., Liu, Y., Cheng, J.: Two-step quantization for low-bit neural networks. In: Proceedings of the IEEE Conference on computer vision and pattern recognition, pp. 4376–4384 (2018)
21. Kim, J., Yoo, K., Kwak, N.: Position-based scaled gradient for model quantization and pruning. Adv. Neural. Inf. Process. Syst. **33**, 20415–20426 (2020)
22. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems, vol. 28 (2015)
23. Yu, D., Seide, F., Li, G., Deng, L.: Exploiting sparseness in deep neural networks for large vocabulary speech recognition. In: 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4409–4412, IEEE (2012)
24. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through ℓ_0 regularization. arXiv preprint arXiv:1712.01312 (2017)
25. Alvarez, J.M., Salzmann, M.: Compression-aware training of deep networks. In: Advances in neural information processing systems, vol. 30 (2017)
26. Hu, H.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016)
27. Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., Peste, A.: Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. J. Mach. Learn. Res. **22**(241), 1–124 (2021)
28. Vaskevicius, T., Kanade, V., Rebeschini, P.: Implicit regularization for optimal sparse recovery. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
29. Li, J., Nguyen, T.V., Hegde, C., Wong, R.K.: Implicit regularization for group sparsity. arXiv preprint arXiv:2301.12540 (2023)
30. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient DNNs. In: Advances in neural information processing systems, vol. 29 (2016)
31. Mocanu, D.C., Mocanu, E., Stone, P., Nguyen, P.H., Gibescu, M., Liotta, A.: Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nat. Commun. **9**(1), 2383 (2018)
32. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI'18, p. 2234–2240, AAAI Press (2018)
33. Pokutta, S., Spiegel, C., Zimmer, M.: Deep neural network training with Frank-Wolfe. arXiv preprint arXiv:2010.07243 (2020)
34. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440 (2016)
35. Theis, L., Korshunova, I., Tejani, A., Huszár, F.: Faster gaze prediction with dense networks and Fisher pruning. arXiv preprint arXiv:1801.05787 (2018)
36. Lu, M., Luo, X., Chen, T., Chen, W., Liu, D., Wang, Z.: Learning pruning-friendly networks via Frank-Wolfe: One-shot, any-sparsity, and no retraining. In: International Conference on Learning Representations (2022)
37. Argyriou, A., Foygel, R., Srebro, N.: Sparse prediction with the k-support norm. In: Advances in Neural Information Processing Systems, vol. 25 (2012)
38. Rao, N., Dudík, M., Harchaoui, Z.: The group k-support norm for learning with structured sparsity. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2402–2406, IEEE (2017)
39. Frank, M., Wolfe, P., et al.: An algorithm for quadratic programming. Naval Res Logistics Q. **3**(1–2), 95–110 (1956)
40. Levitin, E.S., Polyak, B.T.: Constrained minimization methods. USSR Comput. Math. Math. Phys. **6**(5), 1–50 (1966)
41. Reddi, S.J., Sra, S., Póczos, B., Smola, A.: Stochastic Frank-Wolfe methods for nonconvex optimization. In: 2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 1244–1251, IEEE (2016)

42. Ward, R., Wu, X., Bottou, L.: AdaGrad stepsizes: Sharp convergence over nonconvex landscapes. In: Proceedings of the 36th International Conference on Machine Learning (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of Proceedings of Machine Learning Research, (Long Beach, California, USA), pp. 6677–6686, PMLR (2019)
43. Défossez, A., Bottou, L., Bach, F., Usunier, N.: A simple convergence proof of Adam and Adagrad. Trans. Mach. Learn. Res. (2022)
44. Wu, X., Ward, R., Bottou, L.: WNGrad: Learn the learning rate in gradient descent. arXiv preprint arXiv:1803.02865 (2018)
45. Corless, R.M., Gonnet, G.H., Hare, D.E., Jeffrey, D.J., Knuth, D.E.: On the Lambert W function. Adv. Comput. Math. **5**, 329–359 (1996)
46. van den Berg, E., Friedlander, M., Hennenfent, G., Herrmann, F., Saab, R., Yılmaz, O.: Sparco: A testing framework for sparse reconstruction, Dept. Comput. Sci., Univ. British Columbia, Vancouver, Tech. Rep. TR-2007-20. http://www.cs.ubc.ca/labs/scl/sparco (2007)
47. Gratton, S., Toint, P.L.: S2MPJ and CUTEst optimization problems for Matlab, Python and Julia. arXiv:2407.07812 (2024)
48. Wen, Z., Yin, W., Goldfarb, D., Zhang, Y.: A fast algorithm for sparse reconstruction based on shrinkage, subspace optimization, and continuation. SIAM J. Sci. Comput. **32**(4), 1832–1857 (2010)
49. Porcelli, M., Rinaldi, F.: A variable fixing version of the two-block nonlinear constrained Gauss–Seidel algorithm for ℓ 1-regularized least-squares. Comput. Optim. Appl. **59**(3), 565–589 (2014)
50. Aharon, M., Elad, M., Bruckstein, A.: K-SVD: an algorithm for designing overcomplete dictionaries for sparse representation. IEEE Trans. Signal Process. **54**(11), 4311–4322 (2006)
51. Aharon, M., Elad, M., Bruckstein, A.: UCI machine learning repository (2013)
52. Chang, C.-C., Lin, C.-J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 1–27 (2011)