Benchopt: Reproducible, efficient and collaborative optimization benchmarks

Thomas Moreau[1,*], **Mathurin Massias**[2,*], **Alexandre Gramfort**[1,*], **Pierre Ablin**[3], **Pierre-Antoine Bannier**, **Benjamin Charlier**[4], **Mathieu Dagréou**[1], **Tom Dupré la Tour**[6], **Ghislain Durif**[4], **Cassio F. Dantas**[7], **Quentin Klopfenstein**[8], **Johan Larsson**[9], **En Lai**[1], **Tanguy Lefort**[4], **Benoit Malézieux**[1], **Badr Moufad**[2], **Binh T. Nguyen**[10], **Alain Rakotomamonjy**[11], **Zaccharie Ramzi**[12], **Joseph Salmon**[4,5], **Samuel Vaiter**[13]

[1] Université Paris-Saclay, Inria, CEA, 91120 Palaiseau, France
[2] Univ Lyon, Inria, CNRS, ENS de Lyon, UCB Lyon 1, LIP UMR 5668, F-69342, Lyon, France
[3] Université Paris-Dauphine, PSL University, CNRS, 75016, Paris, France
[4] IMAG, Univ Montpellier, CNRS, Montpellier, France [5] Institut Universitaire de France (IUF)
[6] University of California, Berkeley, CA 94720, USA [7] TETIS, Univ Montpellier, INRAE, Montpellier, France
[8] University of Luxembourg, LCSB, Esch-sur-Alzette, Luxembourg
[9] The Department of Statistics, Lund University [10] LTCI, Télécom Paris, 91120 Palaiseau, France
[11] Criteo AI Lab, Paris, France [12] ENS Ulm, CNRS, UMR 8553, Paris, France
[13] CNRS & Université Côte d'Azur, Laboratoire J.A. Dieudonné, CNRS, Nice, France

Abstract

Numerical validation is at the core of machine learning research as it allows to assess the actual impact of new methods, and to confirm the agreement between theory and practice. Yet, the rapid development of the field poses several challenges: researchers are confronted with a profusion of methods to compare, limited transparency and consensus on best practices, as well as tedious re-implementation work. As a result, validation is often very partial, which can lead to wrong conclusions that slow down the progress of research. We propose `Benchopt`, a collaborative framework to automate, reproduce and publish optimization benchmarks in machine learning across programming languages and hardware architectures. `Benchopt` simplifies benchmarking for the community by providing an off-the-shelf tool for running, sharing and extending experiments. To demonstrate its broad usability, we showcase benchmarks on three standard learning tasks: ℓ_2-regularized logistic regression, Lasso, and ResNet18 training for image classification. These benchmarks highlight key practical findings that give a more nuanced view of the state-of-the-art for these problems, showing that for practical evaluation, the devil is in the details. We hope that `Benchopt` will foster collaborative work in the community hence improving the reproducibility of research findings.

1 Introduction

Numerical experiments have become an essential part of statistics and machine learning (ML). It is now commonly accepted that every new method needs to be validated through comparisons with existing approaches on standard problems. Such validation provides insight into the method's benefits and limitations and thus adds depth to the results. While research aims at advancing knowledge and not just improving the state of the art, experiments ensure that results are reliable and support theoretical claims (Sculley et al., 2018). Practical validation also helps the ever-increasing number of ML users in applied sciences to choose the right method for their task. Performing rigorous and extensive experiments is, however, time-consuming (Raff, 2019), particularly because comparisons

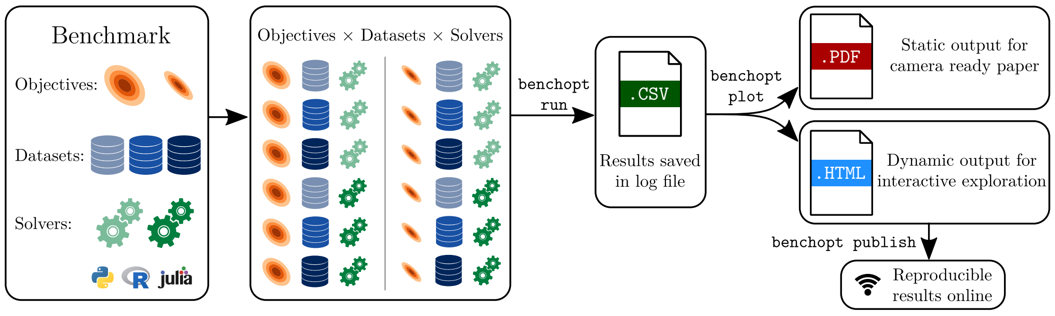
Benchmark
Objectives:
Datasets:
Solvers:
julia
Objectives × Datasets × Solvers
benchopt run
.CSV
Results saved in log file
benchopt plot
.PDF
Static output for camera ready paper
.HTML
Dynamic output for interactive exploration
benchopt publish
Reproducible results online

Figure 1: A visual summary of `Benchopt`. Each `Solver` is run (in parallel) on each `Dataset` and each variant of the `Objective`. Results are exported as a `CSV` file that is easily shared and can be automatically plotted as interactive `HTML` visualizations or `PDF` figures.

against existing methods in new settings often requires reimplementing baseline methods from the literature. In addition, ingredients necessary for a proper reimplementation may be missing, such as important algorithmic details, hyperparameter choices, and preprocessing steps (Pineau et al., 2019).

In the past years, the ML community has actively sought to overcome this "reproducibility crisis" (Hutson, 2018) through collaborative initiatives such as open datasets (`OpenML`, Vanschoren et al. 2013), standardized code sharing (Forde et al., 2018), benchmarks (`MLPerf`, Mattson et al. 2020), the NeurIPS and ICLR reproducibility challenges (Pineau et al., 2019; Pineau et al., 2021) and new journals (*e.g.,* Rougier and Hinsen 2018). As useful as these endeavors may be, they do not, however, fully address the problems in optimization for ML since, in this area, there are no clear community guidelines on how to perform, share, and publish benchmarks.

Optimization algorithms pervade almost every area of ML, from empirical risk minimization, variational inference to reinforcement learning (Sra et al., 2012). It is thus crucial to know which methods to use depending on the task and setting (Bartz-Beielstein et al., 2020). While some papers in optimization for ML provide extensive validations (Lueckmann et al., 2021), many others fall short in this regard due to lack of time and resources, and in turn feature results that are hard to reproduce by other researchers. In addition, both performance and hardware evolve over time, which eventually makes static benchmarks obsolete. An illustration of this is the recent work by Schmidt et al. (2021), which extensively evaluates the performances of 15 optimizers across 8 deep-learning tasks. While their benchmark gives an overall assessment of the considered solvers, this assessment is bound to become out-of-date if it is not updated with new solvers and new architectures. Moreover, the benchmark does not reproduce state-of-the-art results on the different datasets, potentially indicating that the considered architectures and optimizers could be improved.

> We firmly believe that this critical task of **maintaining an up-to-date benchmark in a field cannot be solved without a collective effort**. We want to empower the community to take up this challenge and **build a living, reproducible and standardized state of the art that can serve as a foundation for future research.**

`Benchopt` provides the tools to structure the optimization for machine learning (Opt-ML) community around standardized benchmarks, and to aggregate individual efforts for reproducibility and results sharing. `Benchopt` can handle algorithms written in `Python`, `R`, `Julia` or `C/C++` via binaries. It offers built-in functionalities to ease the execution of benchmarks: parallel runs, caching, and automatic results archiving. Benchmarks are meant to evolve over time, which is why `Benchopt` offers a modular structure through which a benchmark can be easily extended with new objective functions, datasets, and solvers by the addition of a single file of code.

The paper is organized as follows. We first detail the design and usage of `Benchopt`, before presenting results on three canonical problems:

- ℓ_2-regularized logistic regression: a convex and smooth problem which is central to the evaluation of many algorithms in the Opt-ML community, and remains of high relevance for practitioners;
- the Lasso: the prototypical example of non-smooth convex problem in ML;
- training of ResNet18 architecture for image classification: a large scale non-convex deep learning problem central in the field of computer vision.

The reported benchmarks, involving dozens of implementations and datasets, shed light on the current state-of-the-art solvers for each problem, across various settings, highlighting that the best algorithm largely depends on the dataset properties (*e.g.,* size, sparsity), the hyperparameters, as well as hardware. A variety of other benchmarks (*e.g.,* MCP, TV1D, etc.) are also presented in Appendix, with the goal to facilitate contributions from the community.

By the open source and collaborative design of Benchopt (BSD 3-clause license), we aim to open the way towards community-endorsed and peer-reviewed benchmarks that will improve the tracking of progress in optimization for ML.

2 The Benchopt library

The Benchopt library aims to provide a standard toolset and structure to implement benchmarks for optimization in ML, where the problems depend on some input dataset $\mathcal{D}$. The considered problems are of the form

$$\theta^* \in \operatorname*{arg\,min}_{\theta \in \Theta} f(\theta; \mathcal{D}, \Lambda) \ , \tag{1}$$

where f is the objective function, Λ are its hyperparameters, and Θ is the feasible set for θ. The **scope** of the library is to evaluate optimization methods in their wide sense by considering the sequence $\{\theta^t\}_t$ produced to approximate θ^*. We emphasize than Benchopt does not provide a fixed set of benchmarks, but a framework to create, extend and share benchmarks on any problem of the form (1). To provide a flexible and extendable coding standard, benchmarks are defined as the association of three types of object classes:

Objective: It defines the function f to be minimized as well as the hyperparameters Λ or the set Θ, and the metrics to track along the iterations (*e.g.,* objective value, gradient norm for smooth problems, or validation loss). Multiple metrics can be registered for each θ^t.

Datasets: The Dataset objects provide the data $\mathcal{D}$ to be passed to the Objective class. They control how data is loaded and preprocessed. Datasets are separated from the Objective, making it easy to add new ones, provided they are coherent with the Objective.

Solvers: The Solver objects define how to run the algorithm. They are provided with the Objective and Dataset objects and output a sequence $\{\theta^t\}_t$. This sequence can be obtained using a single run of the method, or with multiple runs in case the method only returns its final iterate.

```
benchmark/
├── objective.py
├── datasets/
│   ├── dataset1.py
│   └── dataset2.py
└── solvers/
    ├── solver1.py
    └── solver2.py
```

Figure 2: Standard benchmark structure

Each of these objects can have parameters that change their behavior, *e.g.,* the regularization parameters for the Objective, the choice of preprocessing for the Datasets, or the step size for the Solvers. By exposing these parameters in the different objects, Benchopt can evaluate their influence on the benchmark results. The Benchopt library defines an application programming interface (API) for each of these concepts and provides a command line interface (CLI) to make them work together. A benchmark is defined as a folder that contains an Objective as well as subfolders containing the Solvers and Datasets. Appendix B presents a concrete example on Ridge regression of how to construct a Benchopt benchmark while additional design design choices of Benchopt are discussed in Appendix C.

For each Dataset and Solver, and for each set of parameters, Benchopt retrieves a sequence $\{\theta^t\}_t$ and evaluates the metrics defined in the Objective for each θ^t. To ensure fair evaluation, the computation of these metrics is done off-line. The metrics are gathered in a CSV file that can be used to display the benchmark results, either locally or as HTML files published on a website that reference the benchmarks run with Benchopt. This workflow is described in Figure 1.

This modular and standardized organization for benchmarks empowers the optimization community by making numerical experiments easily reproducible, shareable, flexible and extendable. The benchmark can be shared as a git repository or a folder containing the different definitions for the Objective, Datasets and Solvers and it can be run with the Benchopt CLI, hence becoming a convenient reference for future comparisons. This ensures fair evaluation of baselines in follow-up experiments, as implementations validated by the community are available. Moreover, benchmarks can be extended easily as one can add a Dataset or a Solver to the comparison by adding a single file. Finally, by supporting multiple metrics – *e.g.,* training and testing losses, error on parameter estimates, sparsity of the estimate – the `Objective` class offers the flexibility to define the concurrent evaluation, which can be extended to track extra metrics on a per benchmark basis, depending on the problem at hand.

As one of the goal of `Benchopt` is to make benchmarks as simple as possible, it also provides a set of features to make them easy to develop and run. `Benchopt` is written in `Python`, but `Solvers` run with implementations in different languages (*e.g.,* `R` and `Julia`, as in Section 4) and frameworks (*e.g.,* `PyTorch` and `TensorFlow`, as in Section 5). Moreover, benchmarks can be run in parallel with checkpointing of the results, enabling large scale evaluations on many CPU or GPU nodes. `Benchopt` also makes it possible to run solvers with many different hyperparameters' values , allowing to assess their sensitivity on the method performance. Benchmark results are also automatically exported as interactive visualizations, helping with the exploration of the many different settings.

Benchmarks All presented benchmarks are run on 10 cores of an Intel Xeon Gold 6248 CPUs @ 2.50GHz and NVIDIA V100 GPUs (16GB). The results' interactive plots and data are available at `https://benchopt.github.io/results/preprint_results.html`.

3 First example: ℓ_2-regularized logistic regression

Logistic regression is a very popular method for binary classification. From a design matrix $X \in \mathbb{R}^{n \times p}$ with rows X_i and a vector of labels $y \in \{-1, 1\}^n$ with corresponding element y_i, ℓ_2-regularized logistic regression provides a generalized linear model indexed by $\theta^* \in \mathbb{R}^p$ to discriminate the classes by solving

$$\theta^* = \underset{\theta \in \mathbb{R}^p}{\arg\min} \sum_{i=1}^{n} \log\left(1 + \exp(-y_i X_i^\top \theta)\right) + \frac{\lambda}{2}\|\theta\|_2^2 \ , \tag{2}$$

where $\lambda > 0$ is the regularization hyperparameter. Thanks to the regularization part, Problem (2) is strongly convex with a Lipschitz gradient, and thus its solution can be estimated efficiently using many iterative optimization schemes.

The most classical methods to solve this problem take inspiration from Newton's method (Wright and Nocedal, 1999). On the one hand, quasi-Newton methods aim at approximating the Hessian of the cost function with cheap to compute operators. Among these methods, L-BFGS (Liu and Nocedal, 1989) stands out for its small memory footprint, its robustness and fast convergence in a variety of settings. On the other hand, truncated Newton methods (Dembo et al., 1982) try to directly approximate Newton's direction by using *e.g.,* the conjugate gradient method (Fletcher and Reeves, 1964) and Hessian-vector products to solve the associated linear system. Yet, these methods suffer when n is large: each iteration requires a pass on the whole dataset.

In this context, methods based on stochastic estimates of the gradient have become standard (Bottou, 2010), with Stochastic Gradient Descent (SGD) as a main instance. The core idea is to use cheap and noisy estimates of the gradient (Robbins and Monro, 1951; Kiefer and Wolfowitz, 1952). While SGD generally converges either slowly due to decreasing step sizes, or to a neighborhood of the solution for constant step sizes, variance-reduced adaptations such as SAG (Schmidt et al., 2017), SAGA (Defazio et al., 2014) and SVRG (Johnson and Zhang, 2013) make it possible to solve the problem more efficiently and are often considered to be state-of-the-art for large scale problems.

Finally, methods based on coordinate descent (Bertsekas, 1999) have also been proposed to solve Problem (2). While these methods are usually less popular, they can be efficient in the context of sparse datasets, where only few samples have non-zero values for a given feature, or when accelerated on distributed systems or GPU (Dünner et al., 2018).

The code for the benchmark is available at `https://github.com/benchopt/benchmark_logreg_l2/`. To reflect the diversity of solvers available, we showcase a `Benchopt` benchmark with 3 datasets, 10 optimization strategies implemented in 5 packages, leveraging GPU hardware when possible. We also consider different scenarios for the objective function: (i) **scaling** (or not) the features, a recommended data preprocessing step, crucial in practice to have comparable regularization strength on all variables; (ii) fitting (or not) an unregularized **intercept** term, important in practice and making optimization harder when omitted from the regularization term (Koh et al., 2007); (iii) working (or

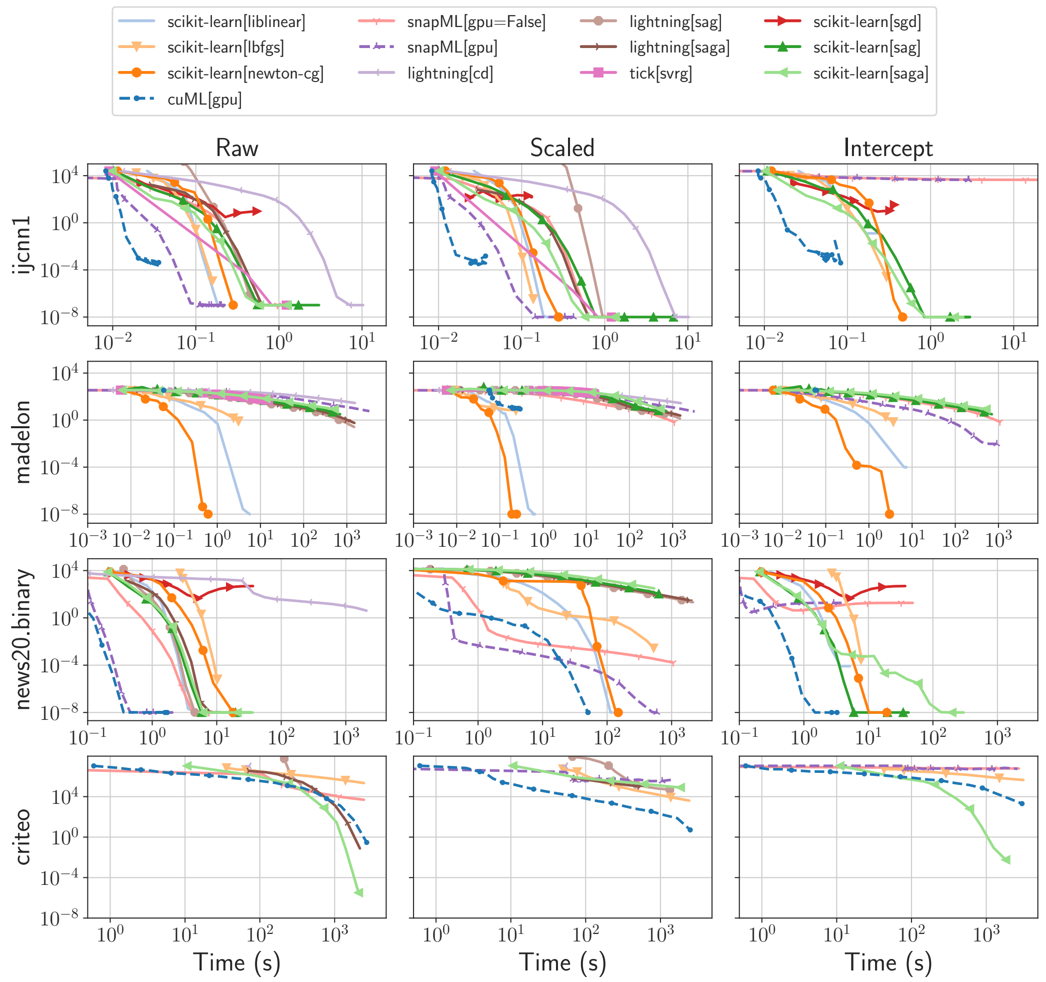

Figure 3: Benchmark for the ℓ_2-regularized logistic regression, on 13 solvers, 4 datasets (*rows*), and 3 variants of the `Objective` (*columns*) with $\lambda = 1$. The curves display the suboptimality of the iterates, $f(\theta^t) - f(\theta^*)$, as a function of time. The first column corresponds to the objective function detailed in Problem (2). In the second column, datasets were preprocessed by normalizing each feature to unit standard deviation. The third column is for an objective function which includes an unregularized intercept.

not) with **sparse** features, which prevent explicit centering during preprocessing to keep memory usage limited. Details on packages, datasets and additional scenarios are available in Appendix D.

Results Figure D.1 presents the results of the benchmarks, in terms of suboptimality of the iterates, $f(\theta^t) - f(\theta^*)$, for three datasets and three scenarios. Here, because the problem is convex, θ^* is approximated by the best iterate across all runs (see Section C.1). Overall, the benchmark shows the benefit of using GPU solvers (`cuML` and `snapML`), even for small scale tasks such as *ijcnn1*. Note that these two accelerated solvers converge to a higher suboptimality level compared to other solvers, due to operating with 32-bit float precision. Another observation is that data scaling can drastically change the picture. In the case of *madelon*, most solvers have a hard time converging for the scaled data. For the solvers that converge, we note that the convergence time is one order of magnitude smaller with the scaled dataset compared to the raw one. This stems from the fact that in this case, the scaling improves the conditioning of the dataset.[1] For *news20.binary*, the stochastic solvers such as SAG and SAGA have degraded performances on scaled data. Here, the scaling makes the problem harder.[2]

[1]The condition number of the dataset is divided by 5.9 after scaling.

[2]The condition number is multiplied by 407 after scaling.

On CPU, quasi-Newton solvers are often the most efficient ones, and provide a reasonable choice in most situations. For large scale *news20.binary*, stochastic solvers such as SAG, SAGA or SVRG –that are often considered as state of the art for such problem– have worst performances for the presented datasets. While this dataset is often used as a test bed for benchmarking new stochastic solvers, we fail to see an improvement over non-stochastic ones for this experimental setup. In contrast, the last row in Figure D.1 displays an experiment with the larger scale *criteo* dataset, which demonstrates a regime where variance-reduced stochastic gradient methods outperform quasi-Newton methods. For future benchmarking of stochastic solvers, we therefore recommend using such a large dataset.

Finally, the third column in Figure D.1 illustrates a classical problem when benchmarking different solvers: their specific (and incompatible) definition and resolution of the corresponding optimization problem. Here, the objective function is modified to account for an intercept (bias) in the linear model. In most situations, this intercept is not regularized when it is fitted. However, `snapML` and `liblinear` solvers do regularize it, leading to incomparable losses.

4 Second example: The Lasso

The Lasso, (Tibshirani, 1996; Chen et al., 1998), is an archetype of non-smooth ML problems, whose impact on ML, statistics and signal processing in the last three decades has been considerable (Bühlmann and van de Geer, 2011; Hastie et al., 2015). It consists of solving

$$\theta^* \in \mathop{\arg\min}_{\theta \in \mathbb{R}^p} \tfrac{1}{2} \left\| y - X\theta \right\|^2 + \lambda \left\| \theta \right\|_1 \quad , \tag{3}$$

where $X \in \mathbb{R}^{n \times p}$ is a design matrix containing p features as columns, $y \in \mathbb{R}^n$ is the target vector, and $\lambda > 0$ is a regularization hyperparameter. The Lasso estimator was popularized for variable selection: when λ is high enough, many entries in θ^* are exactly equal to 0. This leads to more interpretable models and reduces overfitting compared to the least-squares estimator.

Solvers for Problem (3) have evolved since its introduction by Tibshirani (1996). After generic quadratic program solvers, new dedicated solvers were proposed based on iterative reweighted least-squares (IRLS) (Grandvalet, 1998), followed by LARS (Efron et al., 2004), a homotopy method computing the full Lasso path[3]. The LARS solver helped popularize the Lasso, yet the algorithm suffers from stability issues and can be very slow for worst case situations (Mairal and Yu, 2012). General purpose solvers became popular for Lasso-type problems with the introduction of the iterative soft thresholding algorithm (ISTA, Daubechies et al. 2004), an instance of forward-backward splitting (Combettes and Wajs, 2005). These algorithms became standard in signal and image processing, especially when accelerated (FISTA, Beck and Teboulle 2009).

In parallel, proximal coordinate descent has proven particularly relevant for the Lasso in statistics. Early theoretical results were proved by Tseng (1993) and Sardy et al. (2000), before it became the standard solver of the widely distributed packages `glmnet` in `R` and `scikit-learn` in Python. For further improvements, some solvers exploit the sparsity of θ^*, trying to identify its support to reduce the problem size. Best performing variants of this scheme are screening rules (*e.g.,* El Ghaoui et al., 2012; Bonnefoy et al., 2015; Ndiaye et al., 2017) and working/active sets (*e.g.,* Johnson and Guestrin 2015; Massias et al. 2018), including strong rules (Tibshirani et al., 2012).

While reviews of Lasso solvers have already been performed (Bach et al., 2012, Sec. 8.1), they are limited to certain implementation and design choices, but also naturally lack comparisons with more recent solvers and modern hardware, hence drawing biased conclusions.

The code for the benchmark is available at `https://github.com/benchopt/benchmark_lasso/`. Results obtained on 4 datasets, with 9 standard packages and some custom reimplementations, possibly leveraging GPU hardware, and 17 different solvers written in `Python/numba/Cython`, `R`, `Julia` or `C++` (Table E.1) are presented in Figure 4. All solvers use efficient numerical implementations, possibly leveraging calls to `BLAS`, precompiled code in `Cython` or just-in-time compilation with `numba`.

The different parameters influencing the setup are

- the regularization strength λ, controlling the sparsity of the solution, parameterized as a fraction of $\lambda_{\max} = \left\| X^\top y \right\|_\infty$ (the minimal hyperparameter such that $\theta^* = 0$),

[3]The Lasso path is the set of solutions of Problem (3) as λ varies in $(0, \infty)$.

- the dataset dimensions: *MEG* has small n and medium p; *rcv1.binary* has medium n and p; *news20.binary* has medium n and very large p while *MillionSong* has very large n and small p (Table E.2).

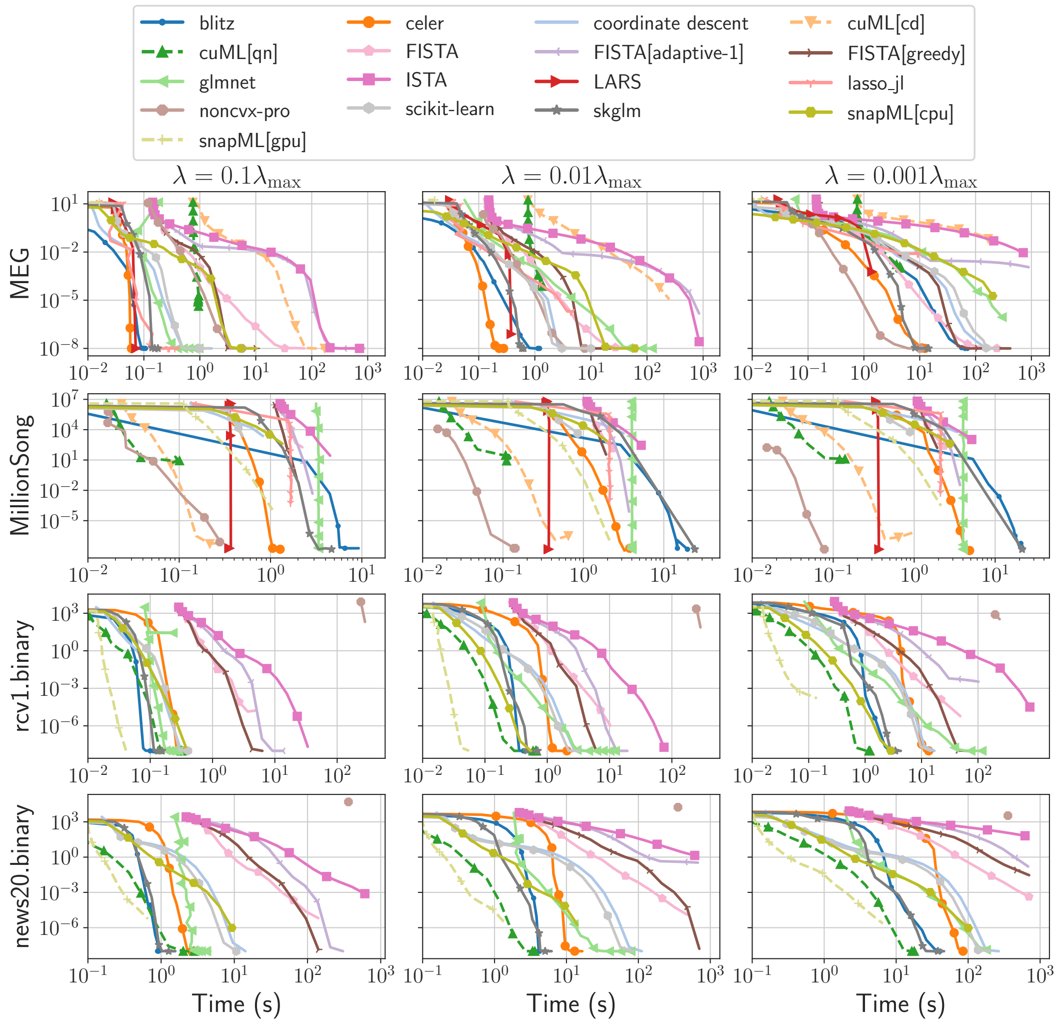

Figure 4: Benchmark for the Lasso, on 17 solvers, 4 datasets (*rows*), and 3 variants of the `Objective` (*columns*) with decreasing regularization λ. The curves display the suboptimality of the objective function, $f(\theta^t) - f(\theta^*)$, as a function of time.

Results Figure 4 presents the result of the benchmark on the Lasso, in terms of objective suboptimality $f(\theta^t) - f(\theta^*)$ as a function of time.

Similarly to Section 3, the GPU solvers obtain good performances in most settings, but their advantage is less clear. A consistent finding across all settings is that coordinate descent-based methods outperform full gradient ones (ISTA and FISTA, even restarted), and are improved by the use of working set strategies (`blitz`, `celer`, `skglm`, `glmnet`). This observation is even more pronounced when the regularization parameter is large, as the solution is sparser.

When observing the influence of the dataset dimensions, we observe 3 regimes. When n is small (*MEG*), the support of the solution is small and coordinate descent, LARS and `noncvx-pro` perform the best. When n is much larger than p (*MillionSong*), `noncvx-pro` clearly outperforms other solvers, and working set methods prove useless. Finally, when n and p are large (*rcv1.binary*, *news20.binary*), CD and working sets vastly outperforms the rest while `noncvx-pro` fails, as it requires solving a linear system of size $\min(n, p)$. We note that this setting was not tested in the original experiment of Poon and Peyré (2021), which highlights the need for extensive, standard experimental setups.

When the support of the solution is small (either small λ, either small n since the Lasso solution has at most n nonzero coefficients), LARS is a competitive algorithm. We expect this to degrade when n increases, but as the LARS solver in `scikit-learn` does not support sparse design matrices we could not include it for *news20.binary* and *rcv1.binary*.

This benchmark is the first to evaluate solvers across languages, showing the competitive behavior of `lasso.jl` and `glmnet` compared to `Python` solvers. Both solvers have a large initialization time, and then converge very fast. To ensure that the benchmark is fair, even though the `Benchopt` library is implemented in `Python`, we made sure to ignore conversion overhead, as well as just-in-time compilation cost. We also checked the timing's consistency with native calls to the libraries.

Since the Lasso is massively used for it feature selection properties, the speed at which the solvers identify the support of the solution is also an important performance measure. Monitoring this with `Benchopt` is straightforward, and a figure reporting this benchmark is in Appendix E.

5 Third example: How standard is a benchmark on ResNet18?

As early successes of deep learning have been focused on computer vision tasks (Krizhevsky et al., 2012), image classification has become a *de facto* standard to validate novel methods in the field. Among the different network architectures, ResNets (He et al., 2016) are extensively used in the community as they provide strong and versatile baselines (Xie et al., 2017; Tan and Le, 2019; Dosovitskiy et al., 2021; Brock et al., 2021; Liu et al., 2022). While many papers present results with such model on classical datasets, with sometimes extensive ablation studies (He et al., 2019; Wightman et al., 2021; Bello et al., 2021; Schmidt et al., 2021), the lack of standardized codebase and missing implementation details makes it hard to replicate their results.

The code for the benchmark is available at `https://github.com/benchopt/benchmark_resnet_classif/`. We provide a cross-dataset –*SVHN*, Netzer et al. (2011); *MNIST*, LeCun et al. (2010) and *CIFAR-10*, Krizhevsky (2009)– and cross-framework –`TensorFlow/Keras`, Abadi et al. (2015) and Chollet et al. (2015); `PyTorch`, Paszke et al. (2019)– evaluation of the training strategies for image classification with ResNet18 (see Appendix F for details on architecture and datasets). We train the network by minimizing the cross entropy loss relatively to the weights θ of the model. Contrary to logistic regression and the Lasso, this problem is non-convex due to the non-linearity of the model f_θ. Another notable difference is that we report the evolution of the test error rather than the training loss.

Because we chose to monitor the test loss, the `Solvers` are defined as the combination of an optimization algorithm, its hyperparameters, the learning rate (LR) and weight decay schedules, and the data augmentation strategy. This is in contrast to a case where we would monitor the train loss, and therefore make the LR and weight decay schedules, as well as the data augmentation policy, part of the objective. We focus on 2 standard methods: stochastic gradient descent (SGD) with momentum and Adam (Kingma and Ba, 2015), as well as a more recently published one: Lookahead (Zhang et al., 2019). The LR schedules are chosen among fixed LR, step LR[4], and cosine annealing (Loshchilov and Hutter, 2017). We also consider decoupled weight decay for Adam (Loshchilov and Hutter, 2019), and coupled weight decay (*i.e.,* ℓ_2-regularization) for SGD. Regarding data augmentation, we use random cropping for all datasets and add horizontal flipping only for *CIFAR-10*, as the digits datasets do not exhibit a mirror symmetry. We detail the remaining hyperparameters in Table F.2, and discuss their selection as well as their sensitivity in Appendix F.

Aligning cross-framework implementations Due to some design choices, components with the same name in the different frameworks do not have the same behavior. For instance, when it comes to applying weight decay, `PyTorch`'s SGD uses coupled weight decay, while in `TensorFlow/Keras` weight decay always refers to decoupled weight decay. These two methods lead to significantly different performance and it is not straightforward to apply coupled weight decay in a post-hoc manner in `TensorFlow/Keras` (see further details in Section F.3). We conducted an extensive effort to align the networks implementation in different frameworks using unit testing to make the conclusions of our benchmarks independent of the chosen framework. We found additional significant differences (reported in Table F.3) in the initialization, the batch normalization, the convolutional layers and the weight decay scaling.

[4]decreasing the learning rate by a factor 10 at mid-training, and again at 3/4 of the training

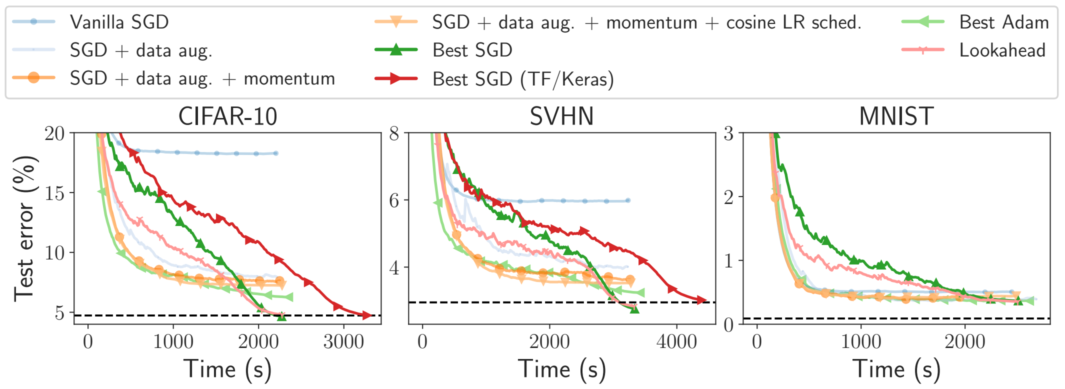

Figure 5: **ResNet18 image classification benchmark with `PyTorch` Solvers.** The best SGD configuration features data augmentation, momentum, cosine learning rate schedule and weight decay. In dashed black is the state of the art for the corresponding datasets with a ResNet18 measured by Zhang et al. (2019) for *CIFAR-10*, by Zheng et al. (2021) for *SVHN* with a PreAct ResNet18, by PapersWithCode for *MNIST* with all networks considered. Off-the-shelf ResNet implementations in `TensorFlow/Keras` do not support images smaller than 32×32 and is hence not shown for *MNIST*. Curves are exponentially smoothed.

Results The results of the benchmark are reported in Figure 5. Each graph reports the test error relative to time, with an ablation study on the solvers' parameters. Note that we only report selected settings for clarity but that we run every possible combination.[5]

Firstly, reaching the state of the art for a vanilla ResNet18 is not straightforward. On the popular website Papers with code it has been so far underestimated. It can achieve 4.45% and 2.65% test error rates on *CIFAR-10* and *SVHN* respectively (compared to 4.73% and 2.95% – for a PreAct ResNet18 – before that). Our ablation study shows that a variety of techniques is required to reach it. The most significant one is an appropriate data augmentation strategy, which lowers the error rate on *CIFAR-10* from about 18% to about 8%. The second most important one is weight decay, but it has to be used in combination with a proper LR schedule, as well as momentum. While these techniques are not novel, they are regularly overlooked in baselines, resulting in underestimation of their performance level.

This reproducible benchmark not only allows a researcher to get a clear understanding of how to achieve the best performances for this model and datasets, but also provides a way to reproduce and extend these performances. In particular, we also include in this benchmark the original implementation of Lookahead (Zhang et al., 2019). We confirm that it slightly accelerates the convergence of the Best SGD, even with a cosine LR schedule – a setting that had not been studied in the original paper.

Our benchmark also evaluates the relative computational performances of the different frameworks. We observe that `PyTorch-Lightning` is significantly slower than the other frameworks we tested, in large part due to their callbacks API. We also notice that our `TensorFlow/Keras` implementation is significantly slower ($\approx$ 28%) than the `PyTorch` ones, despite following the best practices and our profiling efforts. Note that we do not imply that `TensorFlow` is intrinsically slower than `PyTorch`, but a community effort is needed to ensure that the benchmark performances are framework-agnostic.

A recurrent criticism of such benchmarks is that only the best test error is reported. In Figure 6, we measure the effect of using a train-validation-test split, by keeping a fraction of the training set as a validation set. The splits we use are detailed in Table F.1. Our finding is that the results of the ablation study do not change significantly when using such procedure, even though their validity is reinforced by the use of multiple trainings. Yet, a possible limitation of our findings is that some of the hyperparameters we used for our study, coming from the PyTorch-CIFAR GitHub repository, may have been tuned while looking at the test set.

[5]The results are available online as a user-friendly interactive `HTML` file.

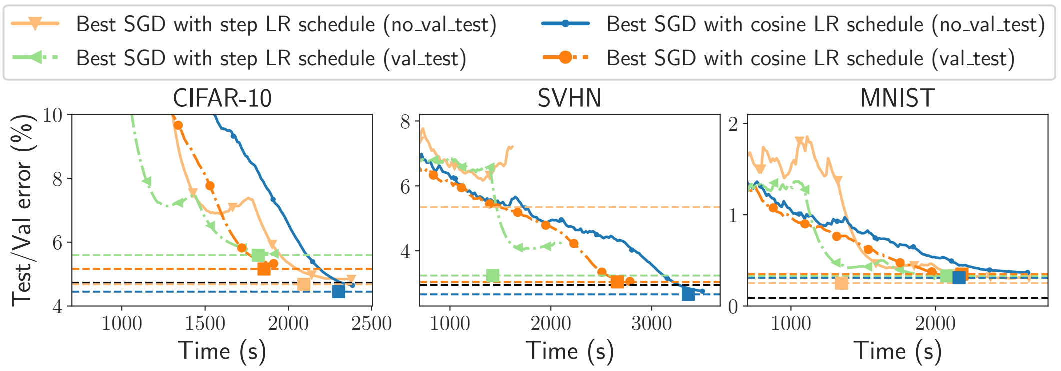

Figure 6: **ResNet18 image classification benchmark with a validation split.** In dashed black is the state of the art (see caption of Figure 5 for more details). In addition, we show in colored horizontal dashed lines, the test results for early stopping on the validation and on the test set for the different solvers, the square mark indicating the moment this stopping would happen. The curves for the train-val splits show the exponentially smoothed median results for five different random seeds.

6 Conclusion and future work

We have introduced Benchopt, a library that makes it easy to collaboratively develop fair and extensive benchmarks of optimization algorithms, which can then be seamlessly published, reproduced, and extended. In the future, we plan on supporting the creation of new benchmarks, that could become the standards the community builds on. This work is part of a wider effort to improve reproducibility of machine learning results. It aims to contribute to raising the standard of numerical validation for optimization, which is pervasive in the statistics and ML community as well as for the experimental sciences that rely more and more on these tools for research.

7 Acknowledgements

It can not be stressed enough how much the Benchopt library relies on contributions from the community and in particular the Python open source ecosystem. In particular, it could not exist without the libraries mentioned in Appendix A.

This work was granted access to the HPC resources of IDRIS under the allocation 2022-AD011011172R2 and 2022-AD011013570 made by GENCI, which was used to run all the benchmarks. MM also gratefully acknowledges the support of the Centre Blaise Pascal's IT test platform at ENS de Lyon (Lyon, France) for Machine Learning facilities. The platform operates the SIDUS solution (Quemener and Corvellec, 2013).

TL, CFD and JS contributions were supported by the Chaire IA CaMeLOt (ANR-20-CHIA-0001-01). AG, EL and TM contributions were supported by the Chaire IA ANR BrAIN (ANR-20-CHIA-0016). BMa contributions were supported by a grant from Digiteo France. MD contributions were supported by a public grant overseen by the French National Research Agency (ANR) through the program UDOPIA, project funded by the ANR-20-THIA-0013-01 and DATAIA convergence institute (ANR-17-CONV-0003). BN work was supported by the Télécom Paris's Chaire DSAIDIS (Data Science & Artificial Intelligence for Digitalized Industry Services). BMo contributions were supported by a grant from the Labex MILYON.

References

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Akiba, T., S. Sano, T. Yanase, T. Ohta, and M. Koyama (2019). *Optuna: A Next-generation Hyperparameter Optimization Framework*.

Bach, F., R. Jenatton, J. Mairal, and G. Obozinski (2012). "Optimization with sparsity-inducing penalties". In: *Foundations and Trends in Machine Learning* 4.1, pp. 1–106.

Bacry, E., M. Bompaire, S. Gaïffas, and S. Poulsen (2017). "Tick: A Python Library for Statistical Learning, with a Particular Emphasis on Time-Dependent Modeling". In: *ArXiv e-prints*.

Barbero, A. and S. Sra (2018). "Modular Proximal Optimization for Multidimensional Total-Variation Regularization". In: *Journal of Machine Learning Research* 19.56, pp. 1–82.

Barlow, R. E. and H. D. Brunk (1972). "The Isotonic Regression Problem and Its Dual". In: *Journal of the American Statistical Association* 67.337, pp. 140–147.

Bartz-Beielstein, T., C. Doerr, D. van den Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. La Cava, M. Lopez-Ibanez, et al. (2020). "Benchmarking in optimization: Best practice and open issues". In: *arXiv preprint arXiv:2007.03488*.

Beck, A. and M. Teboulle (2009). "A fast iterative shrinkage-thresholding algorithm for linear inverse problems". In: *SIAM J. Imaging Sci.* 2.1, pp. 183–202.

Bello, I., W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T.-Y. Lin, J. Shlens, and B. Zoph (2021). "Revisiting ResNets: Improved Training and Scaling Strategies". In: *Advances in Neural Information Processing Systems*.

Bergstra, J. and Y. Bengio (2012). "Random search for hyper-parameter optimization". In: *J. Mach. Learn. Res.* 13.2.

Bergstra, J., D. Yamins, and D. Cox (2013). "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *ICML*. Vol. 28. 1, pp. 115–123.

Bertin-Mahieux, T., D. P. Ellis, B. Whitman, and P. Lamere (2011). "The Million Song Dataset". In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

Bertrand, Q., Q. Klopfenstein, P.-A. Bannier, G. Gidel, and M. Massias (2022). "Beyond L1: Faster and Better Sparse Models with skglm". In: *arXiv preprint arXiv:2204.07826*.

Bertsekas, D. P. (1999). *Nonlinear programming*. Athena Scientific.

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah (2017). "Julia: A fresh approach to numerical computing". In: *SIAM Review* 59.1, pp. 65–98.

Bleakley, K. and J.-P. Vert (2011). "The group fused Lasso for multiple change-point detection". In.

Blondel, M. and F. Pedregosa (2016). *Lightning: Large-Scale Linear Classification, Regression and Ranking in Python*.

Boisbunon, A., R. Flamary, and A. Rakotomamonjy (2014). "Active set strategy for high-dimensional non-convex sparse optimization problems". In: *ICASSP*. IEEE, pp. 1517–1521.

Bolte, J., S. Sabach, and M. Teboulle (2014). "Proximal alternating linearized minimization for nonconvex and nonsmooth problems". In: *Mathematical Programming* 146.1, pp. 459–494.

Bonnefoy, A., V. Emiya, L. Ralaivola, and R. Gribonval (2015). "Dynamic screening: accelerating first-order algorithms for the Lasso and Group-Lasso". In: *IEEE Trans. Signal Process.* 63.19, p. 20.

Bottou, L. (2010). "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *COMPSTAT*. Physica-Verlag, pp. 177–186.

Boyd, S., N. Parikh, E. Chu, B. Peleato, and J. Eckstein (2011). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3.1.

Boyd, S. P. and L. Vandenberghe (2004). *Convex Optimization*. Cambridge, UK ; New York: Cambridge University Press.

Boykov, Y., O. Veksler, and R. Zabih (2001). “Fast approximate energy minimization via graph cuts”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.11, pp. 1222–1239.

Brandl, G. (2010). “Sphinx documentation”. In: *URL http://sphinx-doc. org/sphinx. pdf.*

Bredies, K., K. Kunisch, and T. Pock (2010). “Total generalized variation”. In: *SIAM J. Imaging Sci.* 3.3, pp. 492–526.

Breheny, P. and J. Huang (2011). “Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection”. In: *Ann. Appl. Stat.* 5.1, p. 232.

Brock, A., S. De, S. L. Smith, and K. Simonyan (2021). “High-performance large-scale image recognition without normalization”. In: *ICML*. PMLR, pp. 1059–1071.

Bühlmann, P. and S. van de Geer (2011). *Statistics for high-dimensional data*. Springer Series in Statistics. Methods, theory and applications. Heidelberg: Springer.

Candès, E. J., M. B. Wakin, and S. P. Boyd (2008). “Enhancing Sparsity by Reweighted l_1 Minimization”. In: *J. Fourier Anal. Applicat.* 14.5-6, pp. 877–905.

Chambolle, A. and P.-L. Lions (1997). “Image recovery via total variation minimization and related problems”. In: *Numerische Mathematik* 76.2, pp. 167–188.

Chambolle, A. and T. Pock (2011). “A first-order primal-dual algorithm for convex problems with applications to imaging”. In: *Journal of Mathematical Imaging and Vision* 40.1.

Chen, S. S., D. L. Donoho, and M. A. Saunders (1998). “Atomic decomposition by basis pursuit”. In: *SIAM J. Sci. Comput.* 20.1, pp. 33–61.

Cherkaoui, H., T. Moreau, A. Halimi, and P. Ciuciu (2019). “Sparsity-Based Semi-Blind Deconvolution of Neural Activation Signal in fMRI”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Brighton, UK.

Chollet, F. et al. (2015). *Keras*.

Clark, A. (2015). *Pillow (PIL Fork) Documentation*.

Combettes, P. L. and V. Wajs (2005). “Signal recovery by proximal forward-backward splitting”. In: *Multiscale modeling & simulation* 4.4, pp. 1168–1200.

Combettes, P. L. and L. E. Glaudin (2021). “Solving Composite Fixed Point Problems with Block Updates”. In: *Advances in Nonlinear Analysis*.

Condat, L. (2013a). “A Direct Algorithm for 1-D Total Variation Denoising”. In: *IEEE SIGNAL PROCESSING LETTERS* 20.12.

Condat, L. (2013b). “A primal-dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms”. In: *Journal of Optimization Theory and Applications, Springer Verlag*.

Criteo-Labs (2015). “Criteo releases industry’s largest-ever dataset for machine learning to academic community”. In.

Daubechies, I., M. Defrise, and C. De Mol (2004). “An iterative thresholding algorithm for linear inverse problems with a sparsity constraint”. In: *Commun. Pure Appl. Math.* 57.11, pp. 1413–1457.

Defazio, A., F. Bach, and S. Lacoste-Julien (2014). “SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives”. In: *Advances in Neural Information Processing Systems*. Vol. 28, pp. 1646–1654.

Dembo, R. S., S. C. Eisenstat, and T. Steihaug (1982). “Inexact Newton Methods”. In: *SIAM J. Numer. Anal.* 19.2, pp. 400–408.

Dosovitskiy, A., L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby (2021). “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *ICLR*.

Dünner, C., T. Parnell, D. Sarigiannis, N. Ioannou, A. Anghel, G. Ravi, M. Kandasamy, and H. Pozidis (2018). "Snap ML: A hierarchical framework for machine learning". In: *Advances in Neural Information Processing Systems*. Vol. 31.

Efron, B., T. J. Hastie, I. M. Johnstone, and R. Tibshirani (2004). "Least angle regression". In: *Ann. Statist.* 32.2. With discussion, and a rejoinder by the authors, pp. 407–499.

El Ghaoui, L., V. Viallon, and T. Rabbani (2012). "Safe feature elimination in sparse supervised learning". In: *J. Pacific Optim.* 8.4, pp. 667–698.

Elad, M., P. Milanfar, and R. Rubinstein (2006). "Analysis versus synthesis in signal priors". In: *2006 14th European Signal Processing Conference*.

Falcon, W., J. Borovec, A. Wälchli, N. Eggert, J. Schock, J. Jordan, N. Skafte, Ir1dXD, V. Bereznyuk, E. Harris, T. Murrell, P. Yu, S. Præsius, T. Addair, J. Zhong, D. Lipin, S. Uchida, S. Bapat, H. Schröter, B. Dayma, A. Karnachev, A. Kulkarni, S. Komatsu, Martin.B, J.-B. SCHIRATTI, H. Mary, D. Byrne, C. Eyzaguirre, cinjon, and A. Bakhtin (May 2020). *PyTorchLightning/pytorch-lightning: 0.7.6 release*. Version 0.7.6.

Fan, J. and R. Li (2001). "Variable selection via nonconcave penalized likelihood and its oracle properties". In: *J. Amer. Statist. Assoc.* 96.456, pp. 1348–1360.

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin (2008). "LIBLINEAR: A Library for Large Linear Classification". In: *J. Mach. Learn. Res.* 9.

Farrens, S., A. Grigis, L. El Gueddari, Z. Ramzi, G. Chaithya, S. Starck, B. Sarthou, H. Cherkaoui, P. Ciuciu, and J.-L. Starck (2020). "PySAP: Python Sparse Data Analysis Package for multidisciplinary image processing". In: *Astronomy and Computing* 32, p. 100402.

Fletcher, R. and C. M. Reeves (Jan. 1964). "Function Minimization by Conjugate Gradients". In: *The Computer Journal* 7.2, pp. 149–154.

Forde, J., T. Head, C. Holdgraf, Y. Panda, G. Nalvarete, B. Ragan-Kelley, and E. Sundell (2018). *Reproducible research environments with repo2docker*. Tech. rep.

Friedman, J., T. Hastie, and R. Tibshirani (2010). "Regularization paths for generalized linear models via coordinate descent". In: *J. Stat. Softw.* 33.1, pp. 1–22.

Gao, F. and L. Han (2012). "Implementing the Nelder-Mead simplex algorithm with adaptive parameters". In: *Computational Optimization and Applications* 51.1, pp. 259–277.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *AISTATS*. Vol. 9, pp. 249–256.

Golub, T. R., D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, et al. (1999). "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring". In: *science* 286.5439, pp. 531–537.

Gong, P., C. Zhang, Z. Lu, J. Huang, and J. Ye (2013). "A general iterative shrinkage and thresholding algorithm for non-convex regularized optimization problems". In: *ICML*, pp. 37–45.

Gramfort, A., M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, L. Parkkonen, and M. S. Hämäläinen (2014). "MNE software for processing MEG and EEG data". In: *NeuroImage* 86, pp. 446–460.

Grandvalet, Y. (1998). "Least absolute shrinkage is equivalent to quadratic penalization". In: *International Conference on Artificial Neural Networks*. Springer, pp. 201–206.

Guyon, I., S. Gunn, A. Ben-Hur, and G. Dror (2004). "Result analysis of the nips 2003 feature selection challenge". In: *Advances in neural information processing systems* 17.

Hansen, N., A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff (2021). "COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting". In: *Optimization Methods and Software* 36.1. ArXiv e-prints, arXiv:1603.08785, pp. 114–144.

Hansen, N. and A. Ostermeier (2001). "Completely derandomized self-adaptation in evolution strategies". In: *Evolutionary computation* 9.2, pp. 159–195.

Harris, C. R., K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. (2020). “Array programming with NumPy”. In: *Nature* 585.7825, pp. 357–362.

Hastie, T. J., R. Tibshirani, and M. Wainwright (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. CRC Press.

He, K., X. Zhang, S. Ren, and J. Sun (2015). “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *CVPR*, pp. 1026–1034.

He, K., X. Zhang, S. Ren, and J. Sun (2016). “Deep residual learning for image recognition”. In: *CVPR*.

He, T., Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li (2019). “Bag of tricks for image classification with convolutional neural networks”. In: *CVPR*. Vol. 2019-June, pp. 558–567.

Hunter, J. D. (2007). “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3, pp. 90–95.

Hutson, M. (2018). “Artificial intelligence faces reproducibility crisis”. In: *Science* 359.6377, pp. 725–726.

Inc., P. T. (2015). *Collaborative data science*. URL: https://plot.ly.

Ioffe, S. and C. Szegedy (2015). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *ICML*, pp. 448–456.

Johnson, R. and T. Zhang (2013). “Accelerating Stochastic Gradient Descent Using Predictive Variance Reduction”. In: *Advances in Neural Information Processing Systems*. Vol. 26.

Johnson, T. B. and C. Guestrin (2015). “Blitz: A Principled Meta-Algorithm for Scaling Sparse Optimization”. In: *ICML*. Vol. 37, pp. 1171–1179.

Karahanoğlu, F. I., C. Caballero-Gaudes, F. Lazeyras, and D. Van De Ville (June 2013). “Total Activation: fMRI Deconvolution through Spatio-Temporal Regularization”. In: *NeuroImage* 73, pp. 121–134.

Keerthi, S. S., D. DeCoste, and T. Joachims (2005). “A modified finite Newton method for fast solution of large scale linear SVMs.” In: *Journal of Machine Learning Research* 6.3.

Kiefer, J. and J. Wolfowitz (1952). “Stochastic estimation of the maximum of a regression function”. In: *Ann. Math. Stat.*, pp. 462–466.

Kingma, D. P. and J. Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *ICLR*, pp. 1–10.

Koh, K., J. Kim, and S. Boyd (2007). “An interior-point method for large-scale l1-regularized logistic regression.” In: *J. Mach. Learn. Res.* 8.8, pp. 1519–1555.

Kolmogorov, V. and R. Zabin (2004). “What energy functions can be minimized via graph cuts?” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.2, pp. 147–159.

Komodakis, N. and J.-C. Pesquet (2015). “Playing with Duality: An overview of recent primal-dual approaches for solving large-scale optimization problems”. In: *IEEE Signal Processing Magazine, Institute of Electrical and Electronics Engineers*.

Kornblith, S. (Oct. 28, 2021). *Lasso.jl*. Version 0.6.3. JuliaStats.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images*. Tech. rep.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. South Lake Tahoe, NV, USA, pp. 1097–1105.

Lalanne, C., M. Rateaux, L. Oudre, M. P. Robert, and T. Moreau (July 2020). “Extraction of Nystagmus Patterns from Eye-Tracker Data with Convolutional Sparse Coding”. In: *Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. Montreal, QC, Canada: IEEE, pp. 928–931.

LeCun, Y., C. Cortes, and C. Burges (2010). “MNIST handwritten digit database”. In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2.

Lewis, D. D., Y. Yang, T. Russell-Rose, and F. Li (2004). “Rcv1: A new benchmark collection for text categorization research”. In: *Journal of machine learning research* 5.Apr, pp. 361–397.

Liang, J., T. Luo, and C.-B. Schöenlieb (2022). “Improving “Fast Iterative Shrinkage-Thresholding Algorithm”: Faster, Smarter, and Greedier”. In: *SIAM J. Sci. Comput.* 44.3, A1069–A1091.

Liu, D. C. and J. Nocedal (Aug. 1989). “On the Limited Memory BFGS Method for Large Scale Optimization”. In: *Math. Program.* 45.1-3, pp. 503–528.

Liu, Z., H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie (2022). “A ConvNet for the 2020s”. In: *CVPR*.

Loshchilov, I. and F. Hutter (2017). “SGDR: Stochastic Gradient Descent with Warm Restarts”. In: *ICLR*.

Loshchilov, I. and F. Hutter (2019). “Decoupled Weight Decay Regularization”. In: *ICLR*.

Lueckmann, J.-M., J. Boelts, D. S. Greenberg, P. J. Gonçalves, and J. H. Macke (2021). “Benchmarking Simulation-Based Inference”. In: *AISTATS*. Vol. 130. PMLR, pp. 343–351.

Mairal, J. and B. Yu (2012). “Complexity analysis of the Lasso regularization path”. In: *ICML*, pp. 353–360.

Mairal, J. (2019). “Cyanure: An open-source toolbox for empirical risk minimization for python, c++, and soon more”. In: *arXiv preprint arXiv:1912.08165*.

Massias, M., A. Gramfort, and J. Salmon (2018). “Celer: a fast solver for the lasso with dual extrapolation”. In: *ICML*, pp. 3315–3324.

Mattson, P., V. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, et al. (2020). “MLPerf: An industry standard benchmark suite for machine learning performance”. In: *IEEE Micro* 40.2, pp. 8–16.

Mazumder, R., J. H. Friedman, and T. Hastie (2011). “Sparsenet: Coordinate descent with nonconvex penalties”. In: *J. Amer. Statist. Assoc.* 106.495, pp. 1125–1138.

McKinney, W. et al. (2010). “Data structures for statistical computing in python”. In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX, pp. 51–56.

Moreau, T., P. Glaser, R. Yurchak, and O. Grisel (June 2017). *Loky*. Version 3.0. URL: https://github.com/joblib/loky.

Nájera, Ó., E. Larson, L. Estève, G. Varoquaux, L. Liu, J. Grobler, E. S. de Andrade, C. Holdgraf, A. Gramfort, M. Jas, J. Nothman, O. Grisel, N. Varoquaux, E. Gouillart, M. Luessi, A. Lee, J. Vanderplas, T. Hoffmann, T. A. Caswell, B. Sullivan, A. Batula, jaeilepp, T. Robitaille, S. Appelhoff, P. Kunzmann, M. Geier, Lars, K. Sunden, D. Stańczak, and A. Y. Shih (May 2020). *sphinx-gallery/sphinx-gallery: Release v0.7.0*. Version v0.7.0.

Ndiaye, E., O. Fercoq, A. Gramfort, and J. Salmon (2017). “Gap Safe screening rules for sparsity enforcing penalties”. In: *J. Mach. Learn. Res.* 18.128, pp. 1–33.

Nesterov, Y. E. (1983). “A method for solving the convex programming problem with convergence rate $O(1/k^2)$”. In: *Dokl. akad. nauk Sssr*. Vol. 269, pp. 543–547.

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng (2011). “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: *Advances in Neural Information Processing Systems*.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems*, pp. 8024–8035.

Pedregosa, F., G. Negiar, and G. Dresdner (2020). “copt: composite optimization in Python”. In.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. (2011). “Scikit-learn: Machine learning in Python”. In: *J. Mach. Learn. Res.* 12, pp. 2825–2830.

Pineau, J., P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and H. Larochelle (2021). “Improving reproducibility in machine learning research: a report from the NeurIPS 2019 reproducibility program”. In: *J. Mach. Learn. Res.* 22.

Pineau, J., K. Sinha, G. Fried, R. N. Ke, and H. Larochelle (2019). “ICLR reproducibility challenge 2019”. In: *ReScience C* 5.2, p. 5.

Poon, C. and G. Peyré (2021). “Smooth Bilevel Programming for Sparse Regularization”. In: *Advances in Neural Information Processing Systems*. Vol. 34, pp. 1543–1555.

Powell, M. J. (1964). “An efficient method for finding the minimum of a function of several variables without calculating derivatives”. In: *The computer journal* 7.2, pp. 155–162.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

Prokhorov, D. (2001). “IJCNN 2001 neural network competition”. In: *Slide presentation in IJCNN* 1.97, p. 38.

Quemener, E. and M. Corvellec (2013). “SIDUS—the solution for extreme deduplication of an operating system”. In: *Linux Journal* 2013.235, p. 3.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

Raff, E. (2019). “A step toward quantifying independently reproducible machine learning research”. In: *Advances in Neural Information Processing Systems*. Vol. 32, pp. 5486–5496.

Rapin, J. and O. Teytaud (2018). *Nevergrad - A gradient-free optimization platform*. https://GitHub.com/FacebookResearch/Nevergrad.

Raschka, S., J. Patterson, and C. Nolet (2020). “Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence”. In: *Information-an International Interdisciplinary Journal* 11.4, p. 193.

Robbins, H. and S. Monro (1951). “A stochastic approximation method”. In: *Ann. Math. Stat.*, pp. 400–407.

Rodola, G. (2016). “Psutil package: a cross-platform library for retrieving information on running processes and system utilization”. In: *Google Scholar*.

Rougier, N. P. and K. Hinsen (2018). “ReScience C: a journal for reproducible replications in computational science”. In: *International Workshop on Reproducible Research in Pattern Recognition*. Springer, pp. 150–156.

Rudin, L. I., S. Osher, and E. Fatemi (1992). “Nonlinear total variation based noise removal algorithms”. In: *Physica D: Nonlinear Phenomena* 60.1-4, pp. 259–268.

Sardy, S., A. G. Bruce, and P. Tseng (2000). “Block coordinate relaxation methods for nonparametric wavelet denoising”. In: *J. Comput. Graph. Stat.* 9.2, pp. 361–379.

Schmidt, M., N. Le Roux, and F. Bach (2017). “Minimizing Finite Sums with the Stochastic Average Gradient”. In: *Math. Program.* 162.arXiv:1309.2388, pp. 83–112.

Schmidt, R. M., F. Schneider, and P. Hennig (2021). “Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers”. In: *ICML*. Vol. 139. PMLR, pp. 9367–9376.

Sculley, D., J. Snoek, A. Wiltschko, and A. Rahimi (2018). “Winner’s curse? On pace, progress, and empirical rigor”. In.

Silva, T. S. (2019). “How to Add Regularization to Keras Pre-trained Models the Right Way”. In: *https://sthalles.github.io*.

Simonyan, K. and A. Zisserman (2015). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *ICLR*.

Sra, S., S. Nowozin, and S. J. Wright (2012). *Optimization for machine learning*. MIT Press.

Tan, M. and Q. Le (2019). “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *ICML*, pp. 6105–6114.

Tibshirani, R., J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani (2012). “Strong rules for discarding predictors in lasso-type problems”. In: *J. R. Stat. Soc. Ser. B Stat. Methodol.* 74.2, pp. 245–266.

Tibshirani, R. (1996). “Regression shrinkage and selection via the lasso”. In: *J. R. Stat. Soc. Ser. B Stat. Methodol.* 58.1, pp. 267–288.

Tibshirani, R. J. (Feb. 2014). “Adaptive Piecewise Polynomial Estimation via Trend Filtering”. In: *The Annals of Statistics* 42.1.

Tibshirani, R. J. and J. Taylor (2011). “The solution path of the generalized lasso”. In: *Ann. Statist.* 39.3, pp. 1335–1371.

Tseng, P. (1993). “Dual coordinate ascent methods for non-strictly convex minimization”. In: *Math. Program.* 59.1, pp. 231–247.

Vanschoren, J., J. van Rijn, B. Bischl, and L. Torgo (2013). “OpenML: networked science in machine learning”. In: *SIGKDD Explorations* 15.2, pp. 49–60.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors (2020). “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17, pp. 261–272.

Wales, D. J. and J. P. Doye (1997). “Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms”. In: *The Journal of Physical Chemistry A* 101.28, pp. 5111–5116.

Wightman, R., H. Touvron, and H. Jégou (2021). *ResNet strikes back: An improved training procedure in timm*. Tech. rep., pp. 1–22.

Wright, S. and J. Nocedal (1999). *Numerical Optimization*. Science Springer.

Xie, S., R. Girshick, P. Dollár, Z. Tu, and K. He (2017). “Aggregated residual transformations for deep neural networks”. In: *CVPR*, pp. 1492–1500.

Zhang, . (2010a). “Nearly unbiased variable selection under minimax concave penalty”. In: *Ann. Statist.* 38.2, pp. 894–942.

Zhang, M., J. Lucas, J. Ba, and G. E. Hinton (2019). “Lookahead optimizer: k steps forward, 1 step back”. In: *Advances in Neural Information Processing Systems*. Vol. 32.

Zhang, T. (2010b). “Analysis of multi-stage convex relaxation for sparse regularization”. In: *J. Mach. Learn. Res.* 11.Mar, pp. 1081–1107.

Zheng, Y., R. Zhang, and Y. Mao (2021). “Regularizing neural networks via adversarial model perturbation”. In: *CVPR*, pp. 8156–8165.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] These are specified in Appendix, on a per-benchmark basis.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Error bars are not reported for clarity, but `Benchopt` allows this in particular in html versions of the plots that can be found in `https://benchopt.github.io/results/preprint_results.html`.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] In the introduction.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]