benchopt:
Benchmarking optimization Algorithms

Thomas Moreau

http://tommoral.github.io

INRIA - Parietal

The problem

Frank Schneider
@frankstefansch1

📣 #ICML 2021 Paper 📣
Overwhelmed by the flood of optimizers for deep learning? We felt the same and performed an extensive benchmark. Joint work with @robinschmidt_ & @PhilippHennig5.
Paper: arxiv.org/abs/2007.01547
Results: github.com/SirRob1997/Cro…
Video: youtu.be/cz9RzlstFdE

Frank Schneider
@frankstefansch1

Our results? There is no winner consistently outperforming the competition. Instead, Adam remains a strong contender for many problems.
In some cases, just trying out a few optimizers with their default hyperparameters can work as well as tuning one specific method.

A. List of optimizers and schedules considered

Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

Name	Ref.	Name	Ref.
AcceleGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	L4Adam/L4Momentum	(Rolínek & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamP/SGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

Benchmarking algorithms in practice

Choosing the best algorithm to solve an optimization problem often depends on:

- The data **scale, conditionning**
- The objective parameters **regularisation**
- The implementation **complexity, language**

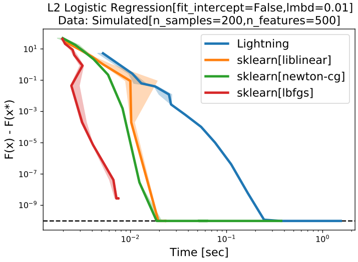
An impartial selection requires a time consuming **benchmark**!

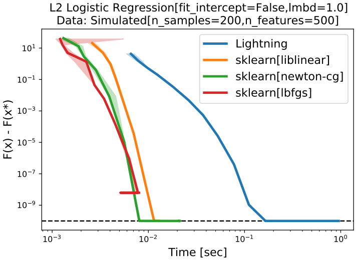
The goal of **`benchopt`** is to make this step as easy as possible.

benchopt

Doing a benchmark for the ℓ_2 regularized logistic regression with multiple solvers and datasets is now easy as calling:

```
git clone https://github.com/benchopt/benchmark_logreg_l2
benchopt run ./benchmark_logreg_l2
```

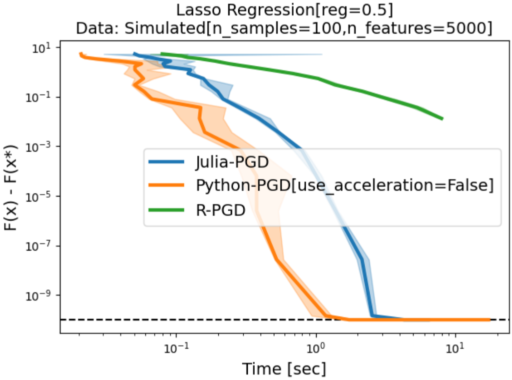




benchopt

benchopt can also compare the same algo in different languages.

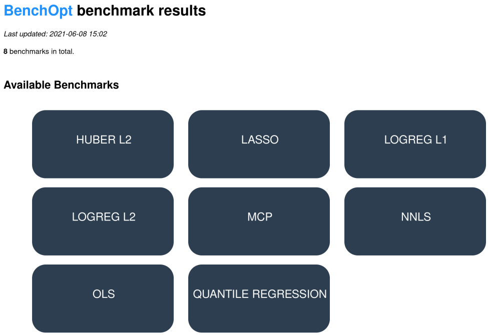
Here is an example comparing PGD in: Python; R; Julia.



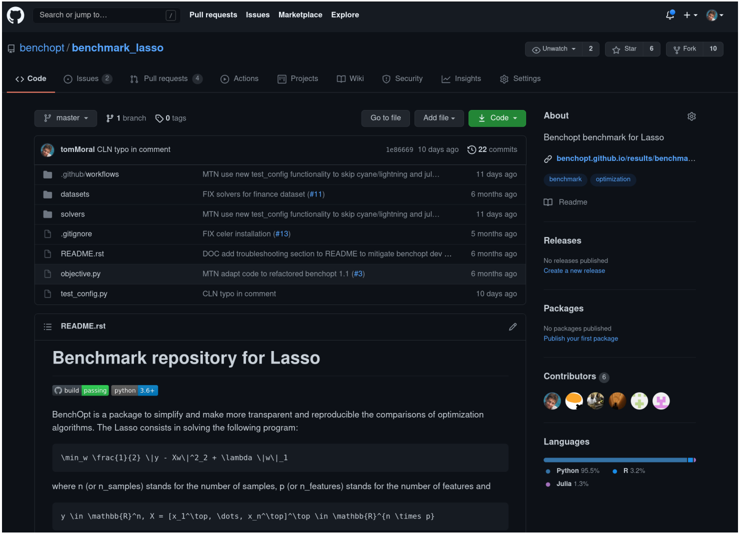
benchopt

benchopt also allow to publish easily benchmark results:

https://benchopt.github.io/results/



Benchmark


Search or jump to...
Pull requests
Issues
Marketplace
Explore
benchopt / benchmark_lasso
Unwatch 2
Star 6
Fork 10
Code
Issues 2
Pull requests 4
Actions
Projects
Wiki
Security
Insights
Settings
master
1 branch
0 tags
Go to file
Add file
Code
tomMoral CLN typo in comment
1e86669 10 days ago
22 commits
.github/workflows MTN use new test_config functionality to skip cyane/lightning and jul... 11 days ago
datasets FIX solvers for finance dataset (#11) 6 months ago
solvers MTN use new test_config functionality to skip cyane/lightning and jul... 11 days ago
.gitignore FIX celer installation (#13) 5 months ago
README.rst DOC add troubleshooting section to README to mitigate benchopt dev ... 6 months ago
objective.py MTN adapt code to refactored benchopt 1.1 (#3) 6 months ago
test_config.py CLN typo in comment 10 days ago
README.rst
Benchmark repository for Lasso
build passing
python 3.6+
BenchOpt is a package to simplify and make more transparent and reproducible the comparisons of optimization algorithms. The Lasso consists in solving the following program:
\min_w \frac{1}{2} \|y - Xw\|^2_2 + \lambda \|w\|_1
where n (or n_samples) stands for the number of samples, p (or n_features) stands for the number of features and
y \in \mathbb{R}^n, X = [x_1^\top, \dots, x_n^\top]^\top \in \mathbb{R}^{n \times p}
About
Benchopt benchmark for Lasso
benchopt.github.io/results/benchma...
benchmark
optimization
Readme
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Contributors 6
Languages
Python 95.5%
R 3.2%
Julia 1.3%

Benchmark: principle

A benchmark is a directory with:

- An `objective.py` file with an `Objective`
- A directory `solvers` with one file per `Solver`
- A directory `datasets` with `Dataset` generators/fetchers

```
my_benchmark/
├── README.rst
├── datasets
│   ├── simulated.py  # some dataset
│   └── real.py  # some dataset
├── objective.py  # contains the definition of the objective
└── solvers
    ├── solver1.py  # some solver
    └── solver2.py  # some solver
```

The `benchopt` client runs a cross product and generates a csv file + convergence plots like above.

Benchmark: Objective & Dataset

```
class Objective(BaseObjective):
    name = "Benchmark Name"

    def set_data(self, X, y):
        # Store data
    def compute(self, beta):
        return dict{obj1:.., obj2:..}
    def to_dict(self):
        return dict{X:.., y:.., reg:..}
```

```
class Dataset(BaseDataset):
    name = "Dataset Name"

    def get_data(self):
        return dict{X:.., y:..}
```

Benchmark: Solver

```
class Solver(BaseSolver):
    name = "Solver Name"

    def set_objective(self, X, y, reg):
        # Store objective info

    def run(self, n_iter):
        # Run computations for n_iter

    def get_result(self):
        return beta
```

Rem: **Flexible API**

- `get_data` and `set_objective` allow to compatibility between packages.
- `n_iter` can be replaced with a tolerance or a callback.

benchopt

benchopt 1.1.0.dev0 API Write a benchmark GitHub Site ▾ Search

Benchmark repository for optimization

Azure Pipelines succeeded | python 3.6+ | codecov 63%

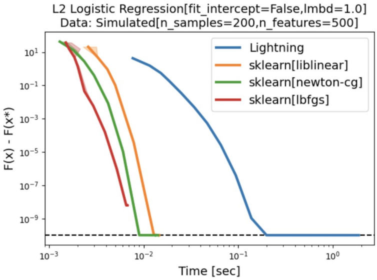
BenchOpt is a package to simplify, make more transparent and more reproducible the comparisons of optimization algorithms.

BenchOpt is written in Python but it is available with many programming languages. So far it has been tested with Python, R, Julia and compiled binaries written in C/C++ available via a terminal command. If it can be installed via conda it should just work!

BenchOpt is used through a command line as documented in API Documentation. Ultimately running and replicating an optimization benchmark should be **as simple as doing**:

```
$ git clone https://github.com/benchopt/benchmark_logreg_l2
$ benchopt run ./benchmark_logreg_l2
```

Running these commands will fetch the benchmark files and give you a benchmark plot on l2-regularized logistic regression:



`benchopt`: Making tedious tasks easy

Automatizing tasks:

- Automatic installation of competitors solvers.
- Parametrized datasets, objectives and solvers and run on cross products.
- Make sure to quantify the variance.
- Automatic caching.
- First visualization of the results.
- Automatic parallelization, ... ?

Credits

J. Salmon
INRIA Parietal

A. Gramfort
INRIA Parietal

T. Moreau
INRIA Parietal

N. Gazagnadou
Telecom Paris

T. Lefort
Univ. Montpellier

M. Massias
Univ. of Genova

T. Dupré la Tour
UC Berkeley