The problem

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Ref.</th>
<th>Name</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>[55]</td>
<td>RMSprop</td>
<td>[36]</td>
</tr>
<tr>
<td>Adamax</td>
<td>[54]</td>
<td>AdamP</td>
<td>[56]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[53]</td>
<td>Adadelta Plus</td>
<td>[57]</td>
</tr>
<tr>
<td>Adagrad</td>
<td>[52]</td>
<td>Adagrad</td>
<td>[58]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[51]</td>
<td>Adadelta SGI</td>
<td>[59]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[50]</td>
<td>AdaGrad</td>
<td>[60]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[49]</td>
<td>AdaGrad</td>
<td>[61]</td>
</tr>
<tr>
<td>Adam</td>
<td>[48]</td>
<td>Adam</td>
<td>[62]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[47]</td>
<td>Adam</td>
<td>[63]</td>
</tr>
<tr>
<td>Adadelta Plus</td>
<td>[46]</td>
<td>Adam</td>
<td>[64]</td>
</tr>
<tr>
<td>AdamP</td>
<td>[45]</td>
<td>Adam</td>
<td>[65]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[44]</td>
<td>Adam</td>
<td>[66]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[43]</td>
<td>Adam</td>
<td>[67]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[42]</td>
<td>Adam</td>
<td>[68]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[41]</td>
<td>Adam</td>
<td>[69]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[40]</td>
<td>Adam</td>
<td>[70]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[39]</td>
<td>Adam</td>
<td>[71]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[38]</td>
<td>Adam</td>
<td>[72]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[37]</td>
<td>Adam</td>
<td>[73]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[36]</td>
<td>Adam</td>
<td>[74]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[35]</td>
<td>Adam</td>
<td>[75]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[34]</td>
<td>Adam</td>
<td>[76]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[33]</td>
<td>Adam</td>
<td>[77]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[32]</td>
<td>Adam</td>
<td>[78]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[31]</td>
<td>Adam</td>
<td>[79]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[30]</td>
<td>Adam</td>
<td>[80]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[29]</td>
<td>Adam</td>
<td>[81]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[28]</td>
<td>Adam</td>
<td>[82]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[27]</td>
<td>Adam</td>
<td>[83]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[26]</td>
<td>Adam</td>
<td>[84]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[25]</td>
<td>Adam</td>
<td>[85]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[24]</td>
<td>Adam</td>
<td>[86]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[23]</td>
<td>Adam</td>
<td>[87]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[22]</td>
<td>Adam</td>
<td>[88]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[21]</td>
<td>Adam</td>
<td>[89]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[20]</td>
<td>Adam</td>
<td>[90]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[19]</td>
<td>Adam</td>
<td>[91]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[18]</td>
<td>Adam</td>
<td>[92]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[17]</td>
<td>Adam</td>
<td>[93]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[16]</td>
<td>Adam</td>
<td>[94]</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>[15]</td>
<td>Adam</td>
<td>[95]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[14]</td>
<td>Adam</td>
<td>[96]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[12]</td>
<td>Adam</td>
<td>[98]</td>
</tr>
<tr>
<td>Adadelta SGI</td>
<td>[10]</td>
<td>Adam</td>
<td>[100]</td>
</tr>
<tr>
<td>Adadelta</td>
<td>[8]</td>
<td>Adam</td>
<td>[102]</td>
</tr>
</tbody>
</table>

Frank Schneider
@frankstefansch1

Overwhelmed by the flood of optimizers for deep learning? We felt the same and performed an extensive benchmark. Joint work with @robinschmidt_ & @PhilippHennig5.

Results: github.com/SirRob1997/Cro...
Video: youtu.be/cz9RzlStFDE

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.
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.
Doing a benchmark for the $\ell_2$ 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 can also compare the same algo in different languages.

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

![Lasso Regression Example]

Lasso Regression[reg=0.5]
Data: Simulated[n_samples=100,n_features=5000]
benchopt also allow to publish easily benchmark results:

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

BenchOpt benchmark results

Last updated: 2021-06-08 15:02

8 benchmarks in total.

Available Benchmarks

- HUBER L2
- LASSO
- LOGREG L1
- LOGREG L2
- MCP
- NNLS
- OLS
- QUANTILE REGRESSION
Benchmark repository for Lasso

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:

$$\min_w \frac{1}{2} y - Xw^2 + \lambda \|w\|_1$$

where n (or n_samples) stands for the number of samples, p (or n_features) stands for the number of features and

```python
y \in \textbf{mathbb{R}}^n, X = [x_1|\ldots|x_n]\top \in \textbf{mathbb{R}}^n \times p
```
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

The `benchopt` client runs a cross product and generates a csv file + convergence plots like above.
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:...}
```python
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.
Benchmark repository for optimization

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:

```bash
$ 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:

![L2 Logistic Regression Plot](image-url)
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