Computational Mathematics for Learning and Data Analysis

Project for "Computational Mathematics for Learning and Data Analysis" course

Federico Finocchio: f.finocchio@studenti.unipi.it
Luca Santarella: l.santarella@studenti.unipi.it
Academic Year: 2021/2022

Abstract *Assigned project: ML project 3*
(M1) is a neural network with topology and activation function of your choice (differentiable or not), but mandatory L1 regularization.
(M2) is a standard L2 linear regression (least squares).
(A1) is a standard momentum descent (heavy ball) approach applied to **(M1)**.
(A2) is an algorithm of the class of accelerated gradient methods applied to **(M1)**.
(A3) is a basic version of the direct linear least squares solver of your choice (normal equations, QR, or SVD) applied to **(M2)**.

Project Structure

```
├──MLProject3.pdf
├──README.pdf
├──README.md
├──requirements.txt
├──testLS.sh
├──testNN.sh
├──data
│   ├──ML-CUP20-TR.csv
│   ├──monks-1.train
│   ├──monks-1.test
│   ├──...
│   ├──monks-3.train
│   └──monks-3.test
├──src
│   ├──LS
│   │   ├──LS.py
│   │   ├──testLS.py
│   │   ├──utils.py
│   │   └──plotsQR.ipynb
│   ├──NN
│   │   ├──ActivationFunctions.py
│   │   ├──metrics.py
```

```
│   │   ├──Network.py
│   │   ├──optimizers.py
│   │   ├──testNET.py
│   │   └──plotsNN.ipynb
│   │
│   └──utils.py
└──tests
    ├──LS
    │   └──plots
    └──NN
        ├──grids
        └──plots
```

Running the experiments

We provide bash scripts to test the implemented algorithms and get statistics on their execution.

Neural network

The network configurations used for the tests are those specified in `src/NN/testNET.py`. In this file two dictionaries are defined:

- *params*: defines the values for the configuration of neural network's parameters for a specific dataset and solver combination.
- *grids*: defines the ranges of parameters, for each solver/dataset pair, that will be tested in the gridsearch test.

The parameters naming convention follows the same signature of scikit-learn MLP models.

To run all the neural network related tests, you can use the `testNN.sh` bash script using three **mandatory** parameters:

```
./testNN.sh <data> <optimizer> <grid>
```

where the parameters are defined like:

- data: must be in `{'monk1', 'monk2', 'monk3', 'cup'}`, specifies the dataset to be used to run the test
- optimizer: must be in `{'CM', 'NAG', 'adam'}` (for `grid=true`) or `{'sgd', 'adam'}` (for `grid=false`)
- grid: either `true` of `false`

The execution via the script file provided will produce a bunch of files which can be used to check the network performances and statistics. The produced file are relative to:

- execution statistics: this is a dump of the console outputs produced by the execution of the `testNET.py` file. All the outputs are written in a `.txt` file saved in `tests/NN` main folder.
- plots: they comprise plots of the objective function, gradient's norm and gap term over the epochs of training. They are also produced by the execution of `testNET.py` and placed in `tests/NN/plots`.

Each of the produced files follows a naming convention which inserts in the file name the name of the dataset and solver used in the current execution. In the following, we show some straightforward examples to illustrate the effects produced by the provided script file.

Example Neural Network execution:

```
./testNN.sh 'monk2' 'NAG' false
```

Trains the neural network over the `monk2` dataset using the `sgd` optimizer with nesterov momentum. The third parameter set to `false` means that the model is only executed for testing. This execution will create the following files:

- `tests/NN/test_monk2_NAG.txt` : contains time statistics as well as the best achieved values over objective function and gradient's norm.
- `tests/NN/plots/[gap/grad/objectivefun]_monk2_NAG.png` : plots for the specific values over the epochs of training the model went through.

Example grid search:

```
./testNN.sh 'monk2' 'sgd' true
```

Runs a grid search (hence `true` as third parameter) over the `monk2` dataset by using the parameters for the `sgd` optimizer defined in `src/NN/testNET.py` in the `grids` dictionary. At the end of the grid search, the following file will be saved:

- `monk2_sgd.csv` : contains the results for all the tested configurations in terms of the scoring defined for the specific task.

Least Square solver and QR factorization

To run tests for the LS solver and QR factorization, we provide the `testLS.sh` bash script. Three kinds of tests can be run with this script file, namely:

- ***random test***: runs a test to solve `M @ x = b` over a random `m x n` matrix `M` and `m x 1` dependent variable vector `b`. Both LS solver and QR factorization are tested. For the computation of the LS solution, two vectors are used as dependent variables `b`, a random `m x`

`1` vector and a "hand-made" linear vector produced by generating a "toy" solution vector `sol` and multiplying the data matrix by the generated solution (`b = M @ sol`).

- ***scaling test***: allows to run a test over random `m x n` matrices with an increasing size in the row dimension `m`, in order to check that the linear scaling property for both LS solver and QR factorization are preserved. Allows to get time and performance statistics over increasing amount of data.
- ***cup test***: tests the implemented LS solver and QR factorization algorithm over the provided `data/ML-CUP20-TR.csv` dataset. Also for this dataset it creates a "toy" problem by generating a hand-made solution as in the random test.

To run the Least Square and QR factorization tests, you can use the `testLS.sh` bash script:

```
./testLS.sh <t> <m> <n> <step> <last>
```

where the parameters are defined like:

- t: must be in `{'RANDOM', 'SCALING', 'CUP'}`, specifies the type of test to run.
- m: ignored for `CUP`, specifies the `m` row dimension of the data matrix to use for the specific test.
- n: ignored for `CUP`, specifies the `n` column dimension of the data matrix to use for the specific test.
- step: increment over the `m` dimension, to be used only for the `SCALING` test.
- last: last `m` dimension, to be used only for the `SCALING` test.

The execution via the script file provided will produce a bunch of files which can be used to check the LS solver and QR factorization algorithm statistics. The produced file are relative to:

- execution statistics: this is a dump of the console outputs produced by the execution of the `testLS.py` file. All the outputs are written in a `.txt` file saved in `tests/LS` main folder.
- plots: they comprise plots of the tests performed, like scaling performances for both the implemented algorithms and 'out-of-the-box' solvers in the `numpy` library. The plots are save in `tests/LS/plots` folder.

Example Scaling test:

```
./testLS.sh 'SCALING' 10000 100 20000 50000
```

Performs execution of tests over matrices with increasing dimensions, starting from `10000 x 100` up to `50000 x 100` with a step of `10000` over the `m` dimension of the data matrix.

This execution will create the following files:

- `tests/LS/test_scaling_50000_100.txt` : contains, for each of the generated data, the time statistics as well as the achieved values in LS solution and QR reconstruction in terms of relative

errors over the solutions found with the 'out-of-the-box' algorithms in the `numpy` library.

- `tests/LS/plots/LSscaling_[a3/np/comparison]_n100m50000.png`: plots showing the time required to perform each of the generated LS problems in the scaling test. The files are, respectively, related to the implemented **A3** algorithm, the `numpy` LS solver and the comparison between the two.
- `tests/LS/plots/QRscaling_[a3/np/comparison]_n100m50000.png`: time scaling properties for the QR factorization algorithm for both **A3** algorithm and the `numpy` one, as well as the comparison between the two.

Example random test:

```
./testLS.sh 'RANDOM' 20000 500
```

Runs the LS and QR factorization algorithms over a randomly generated dataset with dimension `20000 x 500` using two vectors for the dependent variables, one random and one generated from a known solution vector. At the end of the grid search, the following file will be save:

- `monk2_sgd.csv`: contains the results for all the tested configurations in terms of the scoring defined for the specific task.

The execution will produce the following files:

- `tests/LS/test_random20000_500.txt`: contains the execution statistics for the solution genereated by the LS algorithm and the QR factorization.