
neurtu Documentation

Release 0.3.0

Roman Yurchak

Sep 25, 2019

Contents:

1	Installation	3
2	Quickstart	5
3	Examples	7
3.1	Time complexity of numpy.sort	7
3.2	LogisticRegression scaling in scikit-learn	9
4	API Reference	17
4.1	neurtu.timeit	17
4.2	neurtu.memit	18
4.3	neurtu.Benchmark	18
4.4	neurtu.delayed	19
5	Release notes	21
5.1	Version 0.3	21
5.2	Version 0.2	21
5.3	Version 0.1	22
	Index	23

Simple performance measurement tool

neurtu is a Python package providing a common interface for multi-metric benchmarks (including time and memory measurements). It can be used to estimate time and space complexity of algorithms, while pandas integration allows quick analysis and visualization of the results.

Setting the number of threads at runtime in OpenBlas, and MKL is also supported on Linux and MacOS.

neurtu means “to measure / evaluate” in Basque language.

CHAPTER 1

Installation

neurtu requires Python 2.7 or 3.4+, it can be installed with,

```
pip install neurtu
```

`pandas` is an optional (but highly recommended) dependency.

Note: the above command will install `memory_profiler`, `shutil` (to measure memory use) and `tqdm` (to make progress bars) mostly for convinience. However, neurtu does not have any hard depedencies, if you don't need these function-alites, you can install it with `pip install --no-deps neurtu`

CHAPTER 2

Quickstart

To illustrate neurtu usage, we will benchmark array sorting in numpy. First, we will generate a generator of cases,

```
import numpy as np
import neurtu

def cases():
    rng = np.random.RandomState(42)

    for N in [1000, 10000, 100000]:
        X = rng.rand(N)
        tags = {'N' : N}
        yield neurtu.delayed(X, tags=tags).sort()
```

that yields a sequence of delayed calculations, each tagged with the parameters defining individual runs.

We can evaluate the run time with,

```
>>> df = neurtu.timeit(cases())
>>> print(df)
      wall_time
N
1000      0.000014
10000     0.000134
100000    0.001474
```

which will internally use `timeit` module with a sufficient number of evaluation to work around the timer precision limitations (similarly to IPython's `%timeit`). It will also display a progress bar for long running benchmarks, and return the results as a `pandas.DataFrame` (if `pandas` is installed).

By default, all evaluations are run with `repeat=1`. If more statistical confidence is required, this value can be increased,

```
>>> neurtu.timeit(cases(), repeat=3)
      wall_time
      mean      max      std
```

(continues on next page)

(continued from previous page)

N			
1000	0.000012	0.000014	0.000002
10000	0.000116	0.000149	0.000029
100000	0.001323	0.001714	0.000339

In this case we will get a frame with a `pandas.MultiIndex` for columns, where the first level represents the metric name (`wall_time`) and the second – the aggregation method. By default `neurtu.timeit` is called with `aggregate=['mean', 'max', 'std']` methods, as supported by the [pandas aggregation API](#). To disable aggregation and obtains timings for individual runs, use `aggregate=False`. See [neurtu.timeit documentation](#) for more details.

To evaluate the peak memory usage, one can use the `neurtu.memit` function with the same API,

```
>>> neurtu.memit(cases(), repeat=3)
      peak_memory
      mean  max  std
N
10000      0.0  0.0  0.0
100000     0.0  0.0  0.0
1000000    0.0  0.0  0.0
```

More generally `neurtu.Benchmark` supports a wide number of evaluation metrics,

```
>>> bench = neurtu.Benchmark(wall_time=True, cpu_time=True, peak_memory=True)
>>> bench(cases())
      cpu_time  peak_memory  wall_time
N
10000    0.000100          0.0   0.000142
100000   0.001149          0.0   0.001680
1000000  0.013677          0.0   0.018347
```

including `[psutil process metrics]`(<https://psutil.readthedocs.io/en/latest/#psutil.Process>).

For more information see the [Examples](#).

The following examples illustrate neurtu usage

Note: Click [here](#) to download the full example code

3.1 Time complexity of numpy.sort

In this example we will look into the time complexity of `numpy.sort()`

```
import numpy as np
from neurtu import timeit, delayed

rng = np.random.RandomState(42)

df = timeit(delayed(np.sort, tags={'N': N, 'kind': kind})(rng.rand(N), kind=kind)
            for N in np.logspace(2, 5, num=5).astype('int')
            for kind in ["quicksort", "mergesort", "heapsort"])

print(df.to_string())
```

Out:

		wall_time
N	kind	
100	quicksort	0.000005
	mergesort	0.000006
	heapsort	0.000006
562	quicksort	0.000011
	mergesort	0.000017

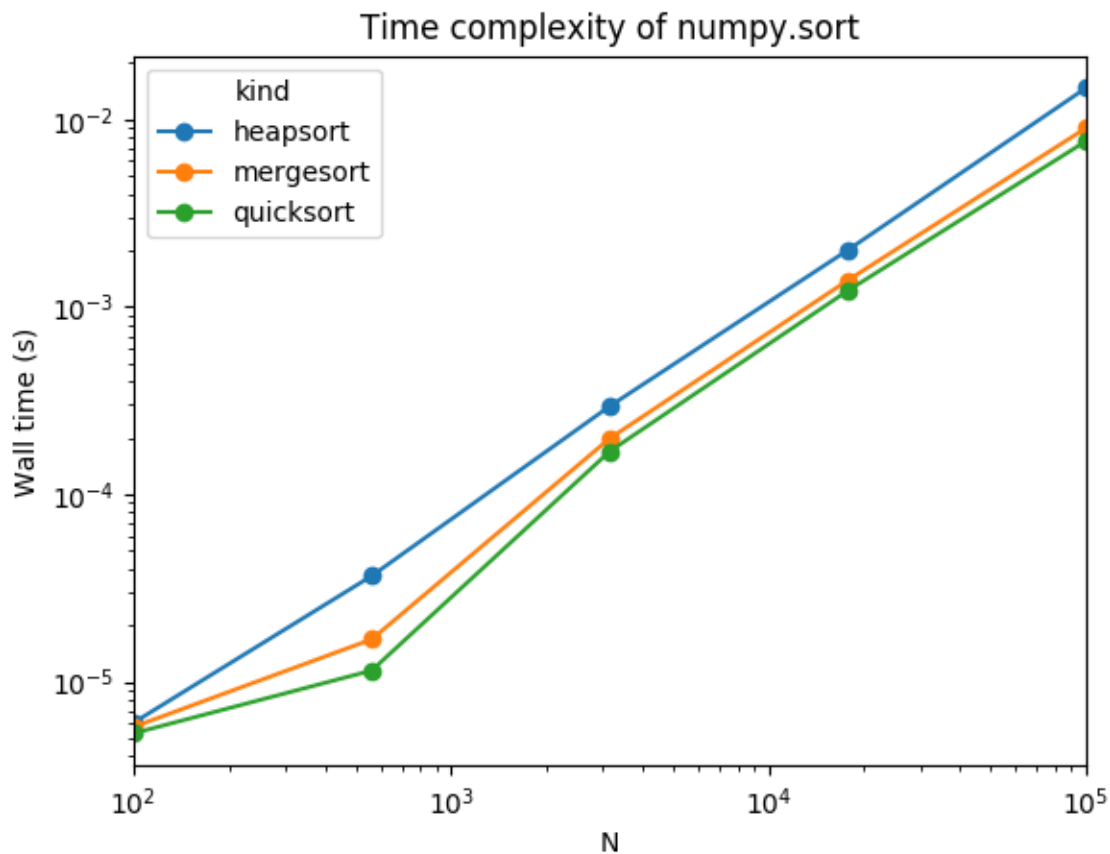
(continues on next page)

(continued from previous page)

	heapsort	0.000037
3162	quicksort	0.000170
	mergesort	0.000199
	heapsort	0.000296
17782	quicksort	0.001223
	mergesort	0.001389
	heapsort	0.002012
100000	quicksort	0.007667
	mergesort	0.009005
	heapsort	0.014728

we can use the pandas plotting API (that requires matplotlib)

```
ax = df.wall_time.unstack().plot(marker='o')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_ylabel('Wall time (s)')
ax.set_title('Time complexity of numpy.sort')
```



Total running time of the script: (0 minutes 3.379 seconds)

Note: Click [here](#) to download the full example code

3.2 LogisticRegression scaling in scikit-learn

In this example we will look into the time and space complexity of `sklearn.linear_model.LogisticRegression`

```
from collections import OrderedDict

import numpy as np
from sklearn.linear_model import LogisticRegression
from neurtu import Benchmark, delayed

rng = np.random.RandomState(42)

n_samples, n_features = 50000, 100

X = rng.rand(n_samples, n_features)
y = rng.randint(2, size=(n_samples))

def benchmark_cases():
    for N in np.logspace(np.log10(100), np.log10(n_samples), 5).astype('int'):
        for solver in ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']:
            tags = OrderedDict(N=N, solver=solver)
            model = delayed(LogisticRegression, tags=tags)(
                solver=solver, random_state=rng)

            yield model.fit(X[:N], y[:N])

bench = Benchmark(wall_time=True, peak_memory=True)
df = bench(benchmark_cases())

print(df.tail())
```

Out:

```
0%|          | 0/50 [00:00<?, ?it/s]
4%|4         | 2/50 [00:00<00:04, 11.74it/s]
6%|6         | 3/50 [00:00<00:07, 6.39it/s]
8%|8         | 4/50 [00:00<00:08, 5.56it/s]
10%|#        | 5/50 [00:00<00:08, 5.11it/s]/home/docs/checkouts/readthedocs.org/
↳user_builds/neurtu/envs/stable/lib/python3.7/site-packages/sklearn/linear_model/sag.
↳py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not
↳converge
   "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↳site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↳reached which means the coef_ did not converge
   "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↳site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↳reached which means the coef_ did not converge
   "the coef_ did not converge", ConvergenceWarning)

14%|#4       | 7/50 [00:01<00:08, 5.08it/s]
```

(continues on next page)

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)

18%|#8      | 9/50 [00:01<00:08, 4.85it/s]/home/docs/checkouts/readthedocs.org/
↪user_builds/neurtu/envs/stable/lib/python3.7/site-packages/sklearn/linear_model/sag.
↪py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not
↪converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)

20%|##      | 10/50 [00:01<00:07, 5.54it/s]
22%|##2     | 11/50 [00:02<00:08, 4.77it/s]
24%|##4     | 12/50 [00:02<00:06, 5.54it/s]
26%|##6     | 13/50 [00:02<00:07, 4.97it/s]
28%|##8     | 14/50 [00:02<00:06, 5.64it/s]
30%|###     | 15/50 [00:02<00:07, 4.95it/s]/home/docs/checkouts/readthedocs.org/
↪user_builds/neurtu/envs/stable/lib/python3.7/site-packages/sklearn/linear_model/sag.
↪py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not
↪converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)

```

(continues on next page)

(continued from previous page)

```

/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)

34%|####         | 17/50 [00:03<00:06,  5.06it/s]/home/docs/checkouts/readthedocs.org/
↪user_builds/neurtu/envs/stable/lib/python3.7/site-packages/sklearn/linear_model/sag.
↪py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not
↪converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)

36%|####         | 18/50 [00:03<00:05,  5.39it/s]/home/docs/checkouts/readthedocs.org/
↪user_builds/neurtu/envs/stable/lib/python3.7/site-packages/sklearn/linear_model/sag.
↪py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not
↪converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)

38%|####         | 19/50 [00:03<00:06,  4.67it/s]/home/docs/checkouts/readthedocs.org/
↪user_builds/neurtu/envs/stable/lib/python3.7/site-packages/sklearn/linear_model/sag.
↪py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not
↪converge
    "the coef_ did not converge", ConvergenceWarning)
/home/docs/checkouts/readthedocs.org/user_builds/neurtu/envs/stable/lib/python3.7/
↪site-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning: The max_iter was
↪reached which means the coef_ did not converge
    "the coef_ did not converge", ConvergenceWarning)

40%|#####        | 20/50 [00:03<00:06,  4.94it/s]
42%|#####2       | 21/50 [00:04<00:06,  4.21it/s]
44%|#####4       | 22/50 [00:04<00:05,  4.70it/s]
46%|#####6       | 23/50 [00:04<00:05,  4.63it/s]
48%|#####8       | 24/50 [00:04<00:05,  5.03it/s]
50%|#####        | 25/50 [00:05<00:05,  4.60it/s]
52%|#####2       | 26/50 [00:05<00:04,  5.16it/s]
54%|#####4       | 27/50 [00:05<00:05,  4.02it/s]
56%|#####6       | 28/50 [00:05<00:05,  3.93it/s]
58%|#####8       | 29/50 [00:06<00:06,  3.48it/s]
60%|#####        | 30/50 [00:06<00:05,  3.38it/s]
62%|#####2       | 31/50 [00:06<00:05,  3.55it/s]
64%|#####4       | 32/50 [00:07<00:06,  2.76it/s]
66%|#####6       | 33/50 [00:07<00:05,  2.88it/s]

```

(continues on next page)

(continued from previous page)

```

68%|#####8 | 34/50 [00:07<00:05, 3.05it/s]
70%|##### | 35/50 [00:08<00:04, 3.02it/s]
72%|#####2 | 36/50 [00:08<00:04, 3.41it/s]
74%|#####4 | 37/50 [00:08<00:04, 2.66it/s]
76%|#####6 | 38/50 [00:09<00:05, 2.35it/s]
78%|#####8 | 39/50 [00:09<00:04, 2.46it/s]
80%|##### | 40/50 [00:10<00:04, 2.47it/s]
82%|#####2 | 41/50 [00:12<00:08, 1.03it/s]
84%|#####4 | 42/50 [00:14<00:10, 1.34s/it]
86%|#####6 | 43/50 [00:15<00:07, 1.07s/it]
88%|#####8 | 44/50 [00:15<00:04, 1.21it/s]
90%|##### | 45/50 [00:16<00:03, 1.29it/s]
92%|#####2 | 46/50 [00:16<00:02, 1.36it/s]
94%|#####3 | 47/50 [00:19<00:04, 1.38s/it]
96%|#####6 | 48/50 [00:22<00:03, 1.90s/it]
98%|#####8 | 49/50 [00:24<00:01, 1.92s/it]
100%|##### | 50/50 [00:26<00:00, 1.89s/it]

```

		wall_time	peak_memory
N	solver		
49999	newton-cg	2.245351	75.835938
	lbfgs	0.189091	0.003906
	liblinear	0.596465	79.875000
	sag	2.844747	0.007812
	saga	1.905433	0.000000

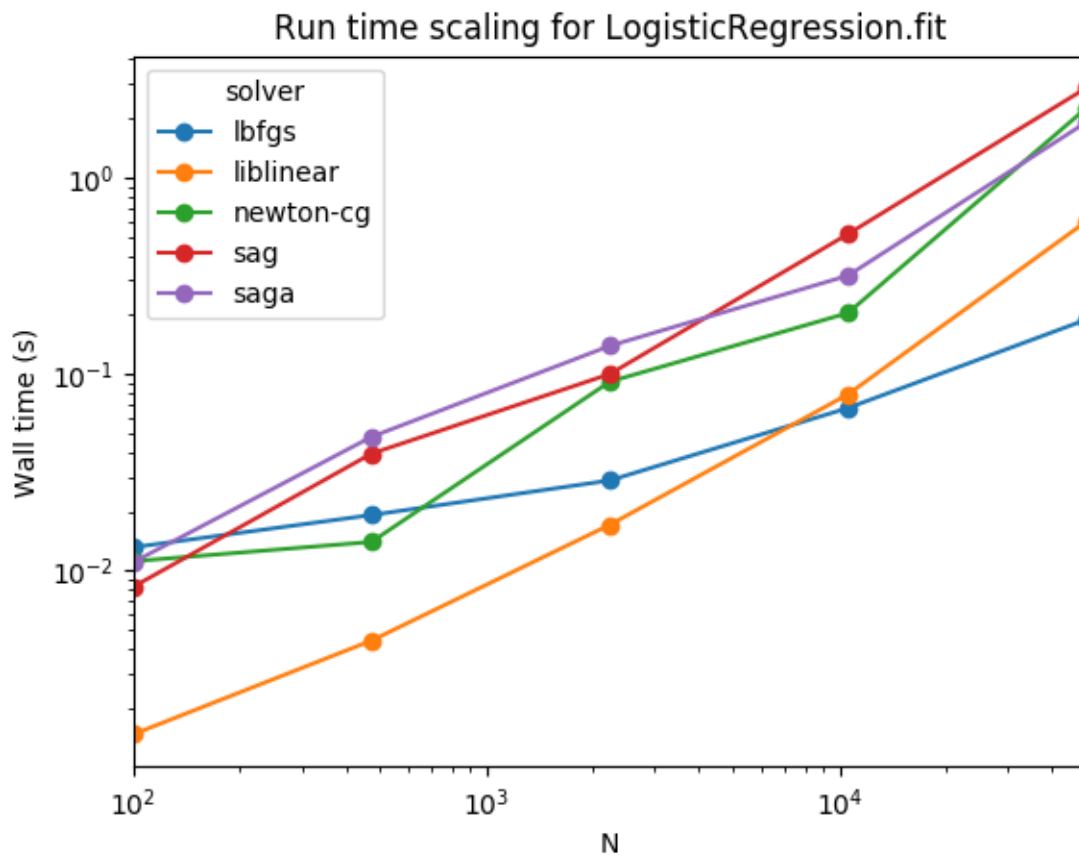
The above section will run in approximately 1min, a progress bar will be displayed.

We can use the pandas plotting API (that requires matplotlib) to visualize the results,

```

ax = df.wall_time.unstack().plot(marker='o')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_ylabel('Wall time (s)')
ax.set_title('Run time scaling for LogisticRegression.fit')

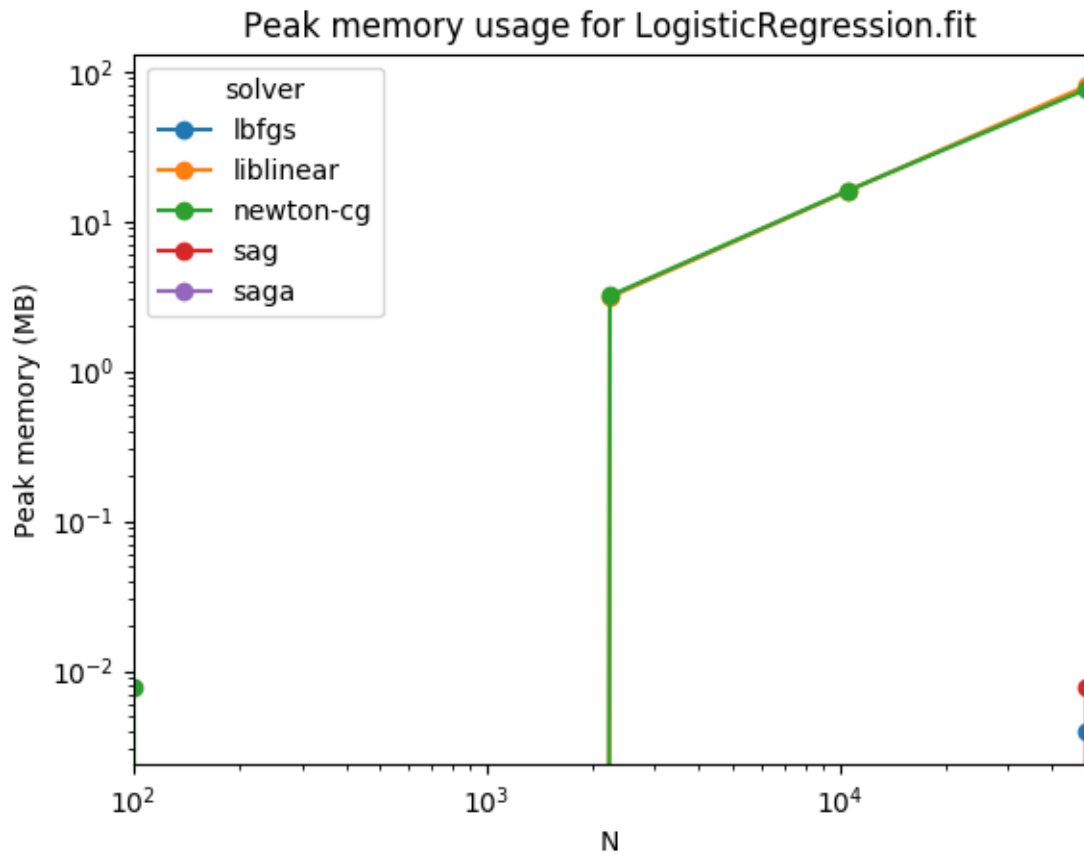
```



The solver with the best scalability in this example is “lbfgs”.

Similarly the memory scaling is represented below,

```
ax = df.peak_memory.unstack().plot(marker='o')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_ylabel('Peak memory (MB)')
ax.set_title('Peak memory usage for LogisticRegression.fit')
```



Peak memory usage for “liblinear” and “newton-cg” appear to be significant above 10000 samples, while the other solvers use less memory than the detection threshold. Note that these benchmarks do not account for the memory used by X and y arrays.

Total running time of the script: (0 minutes 28.220 seconds)

<code>neurtu.timeit(obj[, timer, number, repeat, ...])</code>	A benchmark decorator
<code>neurtu.memit(obj[, repeat, aggregate, ...])</code>	Measure the memory use.
<code>neurtu.Benchmark([wall_time, cpu_time, ...])</code>	Benchmark calculations
<code>neurtu.delayed(obj[, tags, env])</code>	Delayed object evaluation

4.1 neurtu.timeit

`neurtu.timeit(obj, timer='wall_time', number=1, repeat=1, aggregate=('mean', 'max', 'std'), to_dataframe=None, progress_bar=5.0)`

A benchmark decorator

This is an alias for `Benchmark` with `wall_time=True`.

Parameters

- **obj** (`{Delayed, iterable of Delayed}`) – delayed object to compute, or an iterable of Delayed objects
- **number** (`int, default=1`) – number of runs to pass to `timeit.Timer`
- **repeat** (`int, default=1`) – number of repeated measurements
- **aggregate** (`{collection, False}, default=('mean', 'max', 'std')`) – when `repeat > 1`, different runs are indexed by the `runid` key. If `pandas` is installed and `aggregate` is a collection, `aggregate` repeated runs with the provided methods.
- **to_dataframe** (`bool, default=None`) – whether to convert parametric results to a dataframe. By default convert to dataframe is `pandas` is installed.
- **progress_bar** (`{bool, float}, default=5.0`) – if a number, and `tqdm` is installed, display the progress bar when the estimated benchmark time is larger than the given number of seconds. If `False`, the progress bar will not be displayed.

Returns `res` – computed timing

Return type `dict`, `list` or `pandas.DataFrame`

4.2 neurtu.memit

```
neurtu.memit(obj, repeat=1, aggregate=('mean', 'max', 'std'), interval=0.01, to_dataframe=None,
             progress_bar=5.0)
```

Measure the memory use.

This is an alias for `Benchmark` with `peak_memory=True`).

Parameters

- **repeat** (`int`, `default=1`) – number of repeated measurements
- **aggregate** (`{collection, False}`, `default=('mean', 'max', 'std')`) – when `repeat > 1`, different runs are indexed by the `runid` key. If `pandas` is installed and `aggregate` is a collection, aggregate repeated runs with the provided methods.
- **to_dataframe** (`bool`, `default=None`) – whether to convert parametric results to a dataframe. By default convert to dataframe is `pandas` is installed.
- **progress_bar** (`{bool, float}`, `default=5.0`) – if a number, and `tqdm` is installed, display the progress bar when the estimated benchmark time is larger than the given number of seconds. If `False`, the progress bar will not be displayed.

Returns `res` – computed memory usage

Return type `dict`, `list` or `pandas.DataFrame`

4.3 neurtu.Benchmark

```
class neurtu.Benchmark(wall_time=None, cpu_time=False, peak_memory=False, repeat=1,
                       aggregate=('mean', 'max', 'std'), to_dataframe=None, progress_bar=5.0,
                       **kwargs)
```

Benchmark calculations

Parameters

- **wall_time** (`{bool, dict}`, `default=None`) – measure wall time. When a dictionary, it is passed as parameters to the `func:measure_wall_time` function. Will default to `True`, unless some other metric is enabled.
- **cpu_time** (`{bool, dict}`, `default=False`) – measure CPU time. When a dictionary, it is passed as parameters to the `measure_cpu_time()` function.
- **peak_memory** (`{bool, dict}`, `default=False`) – measure peak memory usage. When a dictionary, it is passed as parameters to the `measure_peak_memory()` function.
- **repeat** (`int`, `default=1`) – number of repeated measurements
- **aggregate** (`{collection, False}`, `default=('mean', 'max', 'std')`) – when `repeat > 1`, different runs are indexed by the `runid` key. If `pandas` is installed and `aggregate` is a collection, aggregate repeated runs with the provided methods.
- **to_dataframe** (`bool`, `default=None`) – whether to convert parametric results to a dataframe. By default convert to dataframe is `pandas` is installed.

- **progress_bar** (*{bool, float}*, *default=5.0*) – if a number, and tqdm is installed, display the progress bar when the estimated benchmark time is larger than the given number of seconds. If False, the progress bar will not be displayed.
- ****kwargs** (*dict*) – custom evaluation metrics of the form `key=func`, where `key` is the metric name, and the `func` is the evaluation metric that accepts a `Delayed` object: `func(obj)`.

__init__ (*wall_time=None, cpu_time=False, peak_memory=False, repeat=1, aggregate=('mean', 'max', 'std'), to_dataframe=None, progress_bar=5.0, **kwargs*)
Initialize self. See `help(type(self))` for accurate signature.

Methods

__init__ (*wall_time, cpu_time, peak_memory, Initialize self. ...*)

4.4 neurtu.delayed

neurtu.delayed (*obj, tags=None, env=None*)
Delayed object evaluation

Parameters

- **obj** (*object*) – object or function to wrap
- **tags** (*dict*) – optional tags for the produced delayed object
- **env** (*dict*) – optional environment variables to set when evaluating the delayed object

Returns **result** – a delayed object

Return type *class:neurtu.Delayed*

Example

```
>>> x = delayed('some string').split(' ')[::-1]
>>> x
<Delayed('some string').split(' ')[slice(None, None, -1)]>
>>> x.compute()
['string', 'some']
```

Using tags

```
>>> x = delayed([2, 3], tags={'a': 0}).sum()
>>> x.get_tags()
{'a': 0}
```


5.1 Version 0.3

July 21, 2019

5.1.1 API changes

- Functions to set the number of BLAS threads at runtime were removed in favour of using `threadpoolctl`.

5.1.2 Enhancements

- Add `get_args` and `get_kwargs` to `Delayed` object.
- Better progress bars in Jupyter notebooks with the `tqdm.auto` backend.

5.1.3 Bug fixes

- Fix progress bar rendering when `repeat>1`.
- Fix warnings due to `collection.abc`.

5.2 Version 0.2

August 28, 2018

5.2.1 New features

- Runtime detection of the BLAS used by numpy #14
- Ability to set the number of threads in OpenBlas and MKL BLAS at runtime on Linux. #15.

5.2.2 Enhancements

- Better test coverage
- Documentation improvements
- In depth refactoring of the benchmarking code

5.2.3 API changes

- The API of `timeit`, `memit`, `Benchmark` changed significantly with respect to v0.1

5.3 Version 0.1

March 4, 2018

First release, with support for,

- wall time, cpu time and peak memory measurements
- parametric benchmarks using delayed objects

Symbols

`__init__()` (*neurtu.Benchmark method*), [19](#)

B

`Benchmark` (*class in neurtu*), [18](#)

D

`delayed()` (*in module neurtu*), [19](#)

M

`memit()` (*in module neurtu*), [18](#)

T

`timeit()` (*in module neurtu*), [17](#)