Multistochgrad

This crate provides a Rust implementation of some stochastic gradient algorithms.

The algorithms implemented here are dedicated to the minimization of (convex) objective function represented by the mean of many functions as occurring in various statistical and learning contexts.

The implemented algorithms are:

1. The so-called SCSG algorithm described and analyzed in the two papers by L. Lei and M.I Jordan.
   - "On the adaptativity of stochastic gradient based optimization" (2019) SCSG-1
   - "Less than a single pass : stochastically controlled stochastic gradient" (2019) SCSG-2
2. The SVRG algorithm described in the paper by R. Johnson and T. Zhang "Accelerating Stochastic Gradient Descent using Predictive Variance Reduction" (2013).
   Advances in Neural Information Processing Systems, pages 315–323, 2013
3. The Stochastic Averaged Gradient Descent (SAG) as described in the paper: "Minimizing Finite Sums with the Stochastic Average Gradient" (2013, 2016) M.Schmidt, N.LeRoux, F.Bach

These algorithms minimize functions given by an expression:

```
    f(x) = 1/n ∑ fᵢ(x) where fᵢ is a convex function.
```

The algorithms alternate some form of large batch computation (computing gradient of many terms of the sum) and small or mini batches (computing a small number of terms, possibly just one, term of the gradient) and updating position by combining these global and local gradients.

Examples and tests

Small tests consist in a line fitting problem that is taken from the crate optimisation.

Examples are based on logisitc regression applied to digits MNIST database (as in the second paper on SCSG).
The data files can be downloaded from MNIST.

The logistic regression, with 10 classes, is tested with the 3 algorithms and some comments are provided, comparing the results. Times are obtained by launching twice the example to avoid the compilation time of the first pass.

Run times are those obtained on a 4 hyperthreaded i7-cores laptop at 2.7Ghz.

SCSG logistic regression

For the signification of the parameters B_0 , m_O, see documentation of SCSG. b_0 was set to 1 in all the runs.

Here we give some results:

- initialization position : 9 images with *constant pixel = 0.5*, error at initial position: 6.94

nb iter	B_0	m_0	step_0	y value	time(s)
70	0.02	0.002	0.15	0.295	6.6
50	0.015	0.004	0.1	0.29	7.0
50	0.015	0.006	0.1	0.279	9.1
100	0.02	0.004	0.1	0.266	12.5
50	0.02	0.004	0.1	0.288	6.5

- initialization position : 9 images with *constant pixel = 0.0*, error at initial position: 2.3

nb iter	B_0	m_0	step_0	y value	time(s)
50	0.015	0.006	0.1	0.27	8.5
50	0.015	0.004	0.1	0.274	6.8
50	0.02	0.004	0.1	0.275	6.4
50	0.02	0.006	0.1	0.268	8.0
100	0.02	0.006	0.1	0.258	15.5

It seems that convergence from the initialization from a null image is slightly easier than with a constant 0.5 pixel.

SVRG logistic regression

- initialization position : 9 images with *constant pixel = 0.5*, error at initial position: 6.94

nb iter	nb mini batch	step	y value	time(s)
100	1000	0.02	0.27	35
25	1000	0.05	0.288	9
50	1000	0.05	0.26	18
100	1000	0.05	0.25	36

- initialization position : 9 images with *constant pixel = 0.0*, error at initial position: 2.3

nb iter	nb mini batch	step	y value	time(s)
50	500	0.02	0.30	17
50	500	0.05	0.27	17
50	1000	0.05	0.26	18

nb iter	nb mini batch	step	y value	time(s)
50	2000	0.05	0.25	21
100	1000	0.05	0.246	35

SAG logisitc regression

- initialization position : 9 images with *constant pixel = 0.5*, error at initial position: 6.94

nb iter	batch size	step	y value	time(s)
1000	2000	0.1	0.86	45
1000	1000	0.2	0.47	40
2000	1000	0.2	0.37	80
1000	1000	0.3	0.40	38
1000	1000	0.4	0.37	38
1000	2000	0.4	0.37	41

Results

Tests show that the SCSG outperforms SVRG by a factor 1.5 or 2 at equivalent precision in both case with a correct initialization and one far from the solution. SVRG clearly outperforms SAG. SCSG is very fast at reaching a good approximation roughly 0.28 even though it never runs on the whole (one tenth) in this implementation.

Acknowledgement

This crate is indebted to the crate **optimisation** from which I kept the traits `Function`, `Summation` defining the user interface after various modifications which are detailed in the file `types.rs`

Julia implementation

There is also a Julia implementation of the SCSC and SVRG algorithm at MultiStochGrad

License