Journal of Machine Learning Research 18 (2018) 1-5 Submitted 10/17; Revised 4/18; Published 4/18

SGDLibrary: A MATLAB library for stochastic optimization algorithms

Hiroyuki Kasai KASAI@IS.UEC.AC.JP
Graduate School of Informatics and Engineering
The University of Electro-Communications
Tokyo, 182-8585, Japan

Editor: Geoff Holmes

Abstract

We consider the problem of finding the minimizer of a function $f : \mathbb{R}^d \to \mathbb{R}$ of the finite-sum form $\min f(w) = 1/n \sum_i^n f_i(w)$. This problem has been studied intensively in recent years in the field of machine learning (ML). One promising approach for large-scale data is to use a stochastic optimization algorithm to solve the problem. SGDLibrary is a readable, flexible and extensible pure-MATLAB library of a collection of stochastic optimization algorithms. The purpose of the library is to provide researchers and implementers a comprehensive evaluation environment for the use of these algorithms on various ML problems.

Keywords: Stochastic optimization, stochastic gradient, finite-sum minimization problem, large-scale optimization problem

1. Introduction

This work aims to facilitate research on stochastic optimization for large-scale data. We particularly address a regularized finite-sum minimization problem defined as

$$\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w) = \frac{1}{n} \sum_{i=1}^{n} L(w, x_i, y_i) + \lambda R(w), \tag{1}$$

where $w \in \mathbb{R}^d$ represents the model parameter and n denotes the number of samples (x_i, y_i). $L(w, x_i, y_i)$ is the loss function and $R(w)$ is the regularizer with the regularization parameter $\lambda \geq 0$. Widely diverse machine learning (ML) models fall into this problem. Considering $L(w, x_i, y_i) = (w^T x_i - y_i)^2$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$ and $R(w) = \|w\|_2^2$, this results in an ℓ_2-norm regularized linear regression problem (a.k.a. ridge regression) for n training samples $(x_1, y_1), \cdots, (x_n, y_n)$. In the case of binary classification with the desired class label $y_i \in \{+1, -1\}$ and $R(w) = \|w\|_1$, an ℓ_1-norm regularized logistic regression (LR) problem is obtained as $f_i(w) = \log(1 + \exp(-y_i w^T x_i)) + \lambda \|w\|_1$, which encourages the sparsity of the solution of w. Other problems covered are matrix completion, support vector machines (SVM), and sparse principal components analysis, to name but a few.

Full gradient descent (a.k.a. steepest descent) with a step-size η is the most straightforward approach for (1), which updates as $w_{k+1} \leftarrow w_k - \eta \nabla f(w_k)$ at the k-th iteration. However, this is expensive when n is extremely large. In fact, one needs a sum of n calculations of the inner product $w^T x_i$ in the regression problems above, leading to $\mathcal{O}(nd)$ cost overall per iteration. For this issue, a popular and effective alternative is *stochastic gradient* *descent* (SGD), which updates as $w_{k+1} \leftarrow w_k - \eta \nabla f_i(w_k)$ for the i-th sample uniformly at random (Robbins and Monro, 1951; Bottou, 1998). SGD assumes an *unbiased estimator* of the full gradient as $\mathbb{E}_i[\nabla f_i(w^k)] = \nabla f(w^k)$. As the update rule represents, the calculation cost is independent of n, resulting in $\mathcal{O}(d)$ per iteration. Furthermore, *mini-batch* SGD (Bottou, 1998) calculates $1/|\mathcal{S}_k| \sum_{i \in \mathcal{S}_k} \nabla f_i(w_k)$, where $\mathcal{S}_k$ is the set of samples of size $|\mathcal{S}_k|$. SGD needs a *diminishing* step-size algorithm to guarantee convergence, which causes a severe slow convergence rate (Bottou, 1998). To accelerate this rate, we have two active research directions in ML; *Variance reduction* (VR) techniques (Johnson and Zhang, 2013; Roux et al., 2012; Shalev-Shwartz and Zhang, 2013; Defazio et al., 2014; Nguyen et al., 2017) that explicitly or implicitly exploit a full gradient estimation to reduce the variance of the noisy stochastic gradient, leading to superior convergence properties. Another promising direction is to modify deterministic *second-order* algorithms into stochastic settings, and solve the potential problem of *first-order* algorithms for *ill-conditioned* problems. A direct extension of *quasi-Newton* (QN) is known as online BFGS (Schraudolph et al., 2007). Its variants include a regularized version (RES) (Mokhtari and Ribeiro, 2014), a limited memory version (oLBFGS) (Schraudolph et al., 2007; Mokhtari and Ribeiro, 2015), a stochastic QN (SQN) (Byrd et al., 2016), an incremental QN (Mokhtari et al., 2017), and a non-convex version. Lastly, hybrid algorithms of the SQN with VR are proposed (Moritz et al., 2016; Kolte et al., 2015). Others include (Duchi et al., 2011; Bordes et al., 2009).

The performance of stochastic optimization algorithms is strongly influenced not only by the distribution of data but also by the step-size algorithm (Bottou, 1998). Therefore, we often encounter results that are completely different from those in papers in every experiment. Consequently, an evaluation framework to test and compare the algorithms at hand is crucially important for fair and comprehensive experiments. One existing tool is *Lightning* (Blondel and Pedregosa, 2016), which is a Python library for large-scale ML problems. However, its supported algorithms are limited, and the solvers and the problems such as classifiers are mutually connected. Moreover, the implementations utilize Cython, which is a C-extension for Python, for efficiency. Subsequently, they decrease users' readability of code, and also make users' evaluations and extensions more complicated. SGDLibrary is a readable, flexible and extensible pure-MATLAB library of a collection of stochastic optimization algorithms. The library is also operable on GNU Octave. The purpose of the library is to provide researchers and implementers a collection of state-of-the-art stochastic optimization algorithms that solve a variety of large-scale optimization problems such as linear/non-linear regression problems and classification problems. This also allows researchers and implementers to easily extend or add solvers and problems for further evaluation. To the best of my knowledge, no report in the literature and no library describe a comprehensive experimental environment specialized for stochastic optimization algorithms. The code is available at `https://github.com/hiroyuki-kasai/SGDLibrary`.

2. Software architecture

The software architecture of SGDLibrary follows a typical *module-based* architecture, which separates *problem descriptor* and *optimization solver*. To use the library, the user selects one problem descriptor of interest and no less than one optimization solvers to be compared.

Problem descriptor: The problem descriptor, denoted as `problem`, specifies the problem of interest with respect to w, noted as `w` in the library. This is implemented by MATLAB `classdef`. The user does nothing other than calling a problem definition function, for instance, `logistic_regression()` for the ℓ_2-norm regularized LR problem. Each problem definition includes the functions necessary for solvers; (i) (full) cost function $f(w)$, (ii) mini-batch stochastic derivative v$=1/|\mathcal{S}|\nabla f_{i\in\mathcal{S}}(w)$ for the set of samples $\mathcal{S}$. (iii) stochastic Hessian (Bordes et al., 2009), and (iv) stochastic Hessian-vector product for a vector `v`. The built-in problems include, for example, ℓ_2-norm regularized multidimensional linear regression, ℓ_2-norm regularized linear SVM, ℓ_2-norm regularized LR, ℓ_2-norm regularized softmax classification (multinomial LR), ℓ_1-norm multidimensional linear regression, and ℓ_1-norm LR. The problem descriptor provides additional specific functions. For example, the LR problem includes the prediction and the classification accuracy calculation functions.

Optimization solver: The optimization solver implements the main routine of the stochastic optimization algorithm. Once a solver function is called with one selected problem descriptor `problem` as the first argument, it solves the optimization problem by calling some corresponding functions via `problem` such as the cost function and the stochastic gradient calculation function. Examples of the supported optimization solvers in the library are listed in categorized groups as; (i) **SGD methods:** Vanila SGD (Robbins and Monro, 1951), SGD with classical momentum, SGD with classical momentum with Nesterov's accelerated gradient (Sutskever et al., 2013), AdaGrad (Duchi et al., 2011), RMSProp, AdaDelta, Adam, and AdaMax, (ii) **Variance reduction (VR) methods:** SVRG (Johnson and Zhang, 2013), SAG (Roux et al., 2012), SAGA (Defazio et al., 2014), and SARAH (Nguyen et al., 2017), (iii) **Second-order methods**: SQN (Bordes et al., 2009), oBFGS-Inf (Schraudolph et al., 2007; Mokhtari and Ribeiro, 2015), oBFGS-Lim (oLBFGS) (Schraudolph et al., 2007; Mokhtari and Ribeiro, 2015), Reg-oBFGS-Inf (RES) (Mokhtari and Ribeiro, 2014), and Damp-oBFGS-Inf, (iv) **Second-order methods with VR**: SVRG-LBFGS (Kolte et al., 2015), SS-SVRG (Kolte et al., 2015), and SVRG-SQN (Moritz et al., 2016), and (v) **Else**: BB-SGD and SVRG-BB. The solver function also receives optional parameters as the second argument, which forms a *struct*, designated as `options` in the library. It contains elements such as the maximum number of epochs, the batch size, and the step-size algorithm with an initial step-size. Finally, the solver function returns to the caller the final solution `w` and rich statistical information, such as a record of the cost function values, the optimality gap, the processing time, and the number of gradient calculations.

Others: SGDLibrary accommodates a *user-defined* step-size algorithm. This accommodation is achieved by setting as `options.stepsizefun=@my_stepsize_alg`, which is delivered to solvers. Additionally, when the regularizer $R(w)$ in the minimization problem (1) is a non-smooth regularizer such as the ℓ_1-norm regularizer $\|w\|_1$, the solver calls the *proximal operator* as `problem.prox(w,stepsize)`, which is the wrapper function defined in each problem. The ℓ_1-norm regularized LR problem, for example, calls the *soft-threshold* function as `w = prox(w,stepsize)=soft_thresh(w,stepsize*lambda)`, where `stepsize` is the step-size η and `lambda` is the regularization parameter $\lambda > 0$ in (1).

3. Tour of the SGDLibrary

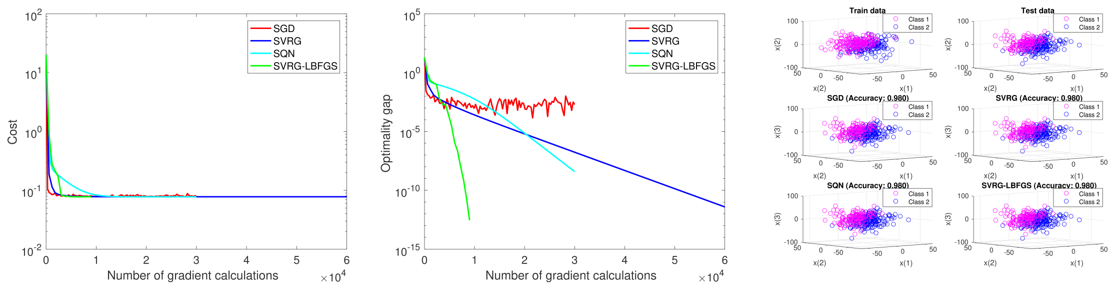
We embark on a tour of SGDLibrary exemplifying the ℓ_2-norm regularized LR problem. The LR model generates n pairs of (x_i, y_i) for an unknown model parameter w, where x_i is an input d-dimensional vector and $y_i \in \{-1, 1\}$ is the binary class label, as $P(y_i|x_i, w) = 1/(1 + \exp(-y_i w^T x_i))$. The problem seeks w that fits the regularized LR model to the generated data (x_i, y_i). This problem is cast as a minimization problem as $\min f(w) := 1/n \sum_{i=1}^{n} \log[1 + \exp(-y_i w^T x_i)] + \lambda/2\|w\|^2$. The code for this problem is in Listing 1.

```
% generate synthetic 300 samples of dimension 3 for logistic regression
d = logistic_regression_data_generator(300,3);
% define logistic regression problem
problem = logistic_regression(d.x_train,d.y_train,d.x_test,d.y_test);

options.w_init = d.w_init;                          % set initial value
options.step_init = 0.01;                           % set initial stepsize
options.verbose = 1;                                % set verbose mode
[w_sgd, info_sgd] = sgd(problem, options);          % perform SGD solver
[w_svrg, info_svrg] = svrg(problem, options);       % perform SVRG solver
[w_svrg, info_svrg] = sqn(problem, options);      % perform SQN solver
% display cost vs. number of gradient evaluations
display_graph('grad_calc_count','cost',{'SGD','SVRG'},...
                {w_sgd,w_svrg},{info_sgd,info_svrg});
```

Listing 1: Demonstration code for logistic regression problem.

First, we generate train/test datasets `d` using `logistic_regression_data_generator()`, where the input feature vector is with $n = 300$ and $d = 3$. $y_i \in \{-1, 1\}$ is its class label. The LR problem is defined by calling `logistic_regression()`, which internally contains the functions for cost value, the gradient and the Hessian. This is stored in `problem`. Then, we execute solvers, i.e., SGD and SVRG, by calling solver functions, i.e., `sgd()` and `svrg()` with `problem` and `options` after setting some options into the `options` struct. They return the final solutions of `{w_sgd,w_svrg}` and the statistical information `{info_sgd,info_svrg}`. Finally, `display_graph()` visualizes the behavior of the cost function values in terms of the number of gradient evaluations. It is noteworthy that each algorithm requires a different number of evaluations of samples in each epoch. Therefore, it is common to use this value to evaluate the algorithms instead of the number of iterations. Illustrative results additionally including SQN and SVRG-LBFGS are presented in Figure 1, which are generated by `display_graph()`, and `display_classification_result()` specialized for classification problems. Thus, SGDLibrary provides rich visualization tools as well.



(a) Cost function value (b) Optimality gap (c) Classification result

Figure 1: Results of ℓ_2-norm regularized logistic regression problem.

References

M. Blondel and F. Pedregosa. Lightning: large-scale linear classification, regression and ranking in Python, 2016. URL `https://doi.org/10.5281/zenodo.200504`.

A. Bordes, L. Bottou, and P. Callinari. SGD-QN: Careful quasi-Newton stochastic gradient descent. *JMLR*, 10:1737–1754, 2009.

L. Bottou. Online algorithm and stochastic approximations. In David Saad, editor, *On-Line Learning in Neural Networks*. Cambridge University Press, 1998.

R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. A stochastic quasi-Newton method for large-scale optimization. *SIAM J. Optim.*, 26(2), 2016.

A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, 2014.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159, 2011.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, 2013.

R. Kolte, M. Erdogdu, and A. Ozgur. Accelerating SVRG via second-order information,. In *OPT2015*, 2015.

A. Mokhtari and A. Ribeiro. RES: Regularized stochastic BFGS algorithm. *IEEE Trans. on Signal Process.*, 62(23):6089–6104, 2014.

A. Mokhtari and A. Ribeiro. Global convergence of online limited memory BFGS. *JMLR*, 16:3151–3181, 2015.

A. Mokhtari, M. Eizen, and A. Ribeiro. An incremental quasi-Newton method with a local superlinear convergence rate. In *ICASSP*, 2017.

P. Moritz, R. Nishihara, and M. I. Jordan. A linearly-convergent stochastic L-BFGS algorithm. In *AISTATS*, 2016.

L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takac. SARAH: A novel method for machine learning problems using stochastic recursive gradient. In *ICML*, 2017.

H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Statistics*, 22 (3):400–407, 1951.

N. L. Roux, M. Schmidt, and F. R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*, 2012.

N. N. Schraudolph, J. Yu, and S. Gunter. A stochastic quasi-Newton method for online convex optimization. In *AISTATS*, 2007.

S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *JMLR*, 14:567–599, 2013.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.