Stochastic optimization library: SGDLibrary

Hiroyuki Kasai
The University of Electro-Communications
Tokyo, 182-8585, Japan
kasai@is.uec.ac.jp

Abstract

SGDLibrary is an open source MATLAB library of stochastic optimization algorithms, which finds the minimizer of a function $f : \mathbb{R}^d \to \mathbb{R}$ of the finite-sum form $\min f(w) = 1/n \sum_i f_i(w)$. This problem has been studied intensively in recent years in the field of machine learning. One typical but 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 machine learning problems. The code is available at `https://github.com/hiroyuki-kasai/SGDLibrary` [1].

1 Introduction

SGDLibrary 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. This type of problems include, for example, an ℓ_2-norm regularized linear regression problem (a.k.a. ridge regression) and an ℓ_1-norm regularized logistic regression (LR) problem. *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. 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 [2, 3]. 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 clearly represents, the calculation cost is independent of n. Furthermore, *mini-batch* SGD [3] 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 [3]. To accelerate this rate, we have two active research directions in ML; *Variance reduction* (VR) techniques that explicitly or implicitly exploit a full gradient estimation to reduce the variance of the noisy stochastic gradient, leading to superior convergence spped. Another direction is to modify deterministic *second-order* (SO) algorithms into stochastic settings, and solve the potential problem of *first-order* algorithms for *ill-conditioned* problems. In a different direction, as a result of the recent success of deep learning, non-convex algorithms have been studied extensively. Among them, some *sub-sampled Hessian* algorithms achieve the second-order optimality condition [4].

Machine Learning Open Source Software 2018: Sustainable communities, Montréal, Canada.

[1]This paper is the updated version of the author's paper [1].

Why is SGDLibrary needed?: The performance of stochastic optimization algorithms is strongly influenced not only by the distribution of data but also by the step-size algorithm [3]. 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. 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.

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. Each problem definition includes the functions necessary for solvers as summarized in Table 1. 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.

Table 1: Supported class functions (methods) of problem descriptor.

No.	Class functions (methods).	Mandatory
(i)	calculate (full) cost function $f(w)$.	✓
(ii)	calculate mini-batch stochastic derivative `v`$=1/\lvert\mathcal{S}\rvert\nabla f_{i\in\mathcal{S}}(w)$ for the set of samples $\mathcal{S}$.	✓
(iii)	calculate stochastic Hessian.	✓
(iv)	calculate stochastic Hessian-vector product for a vector `v`.	✓
(v)	problem-specific functions. (e.g., classification accuracy calculation in LR problem.)	

Optimization solver: Once a solver function is called with one selected problem descriptor `problem` as the first argument, an optimization solver solves the optimization problem by calling corresponding functions via `problem` such as the stochastic gradient calculation function. Examples of the supported optimization solvers in the library are listed in categorized groups as in Table 2. 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).

Table 2: Supported stochastic optimization algorithms.

Category	Algorithm
SGD method	Vanila SGD [2], SGD-CM (classical momentum), SGD-CM-NAG (Nesterov's accelerated gradient) [5], AdaGrad [6], RMSProp [7], AdaDelta [8], Adam [9], AdaMax [9]
Variance reduction (VR)	SVRG [10], SAG [11], SAGA [12], SARAH [13], SARAH-Plus [13]
Second-order (SO) method	SQN [14], oBFGS-Inf [15], oLBFGS-Lim [15, 16], Reg-oBFGS-Inf [17], Damp-oBFGS-Inf [18], IQN [19]
SO with VR method	SVRG-SQN [20], SVRG-LBFGS [21], SS-SVRG [21]
Sub-sampled Hessian method	SCR (Sub-sampled cubic regularization) [22], Sub-sampled TR (trust region) [23, 24]
Other method	BB-SGD [25], SVRG-BB [26]

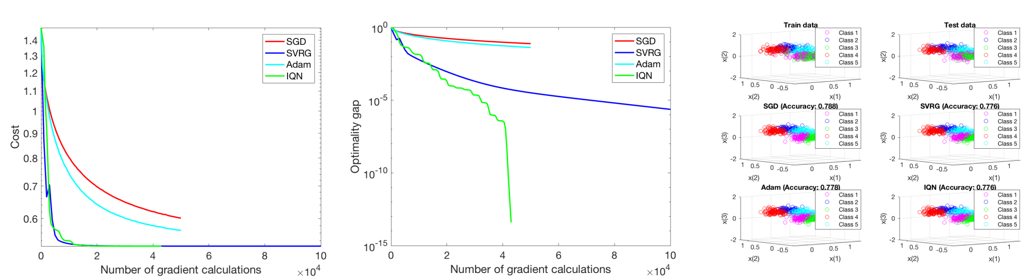
3 Tour of the SGDLibrary: Softmax classification problem

We embark on a tour of SGDLibrary exemplifying the ℓ_2-norm regularized softmax classification problem. The code for this problem is in Listing 1.

```
% generate 100 samples of dimension 3, class 5 and std 0.15 for
    softmax regression
d = multiclass_data_generator(100,3,5,0.15);
% define softmax regression problem
problem = softmax_regression(d.x_tr,d.y_tr,d.x_te,d.y_te,5,0.001);
% execute solvers
options.w_init = d.w_init;                           % set initial value
options.step_init = 0.0001;                          % 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
% 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 ℓ_2-norm regularized softmax classification problem.

First, we generate train/test datasets `d` using `multiclass_data_generator()`, where the input feature vector is with 100 samples and 3 dimension. The number of classes is 5. The softmax classification problem is defined by calling `softmax_regression()`, which internally contains the functions for cost value and the gradient. 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. Illustrative results additionally including Adam and IQN 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 (b) Classification result

Figure 1: Results of ℓ_2-norm regularized softmax classification problem.

References

[1] H. Kasai. SGDLibrary: A MATLAB library for stochastic gradient optimization algorithms. *J. Mach. Learn. Res.*, 18(215):1–5, 2018.

[2] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

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

[4] J. Nocedal and Wright S.J. *Numerical Optimization*. Springer, New York, USA, 2006.

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

[6] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[7] G. Tieleman, T. Hinton. Lecture 6.5 - RMSProp. Technical report, COURSERA: Neural Networks for Machine Learning, 2012.

[8] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

[11] 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.

[12] 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.

[13] 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.

[14] A. Bordes, L. Bottou, and P. Callinari. SGD-QN: Careful quasi-Newton stochastic gradient descent. *J. Mach. Learn. Res.*, 10:1737–1754, 2009.

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

[16] A. Mokhtari and A. Ribeiro. Global convergence of online limited memory BFGS. *J. Mach. Learn. Res.*, 16:3151–3181, 2015.

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

[18] X. Wang, S. Ma, D. Goldfarb, and W. Liu. Stochastic quasi-Newton methods for nonconvex stochastic optimization. *SIAM J. Optim.*, 27(2):927–956, 2017.

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

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

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

[22] J. M. Kohler and A. Lucchi. Sub-sampled cubic regularization for non-convex optimization. In *ICML*, 2017.

[23] P. Xu, F. Roosta-Khorasani, and M. W. Mahoney. Newton-type methods for non-convex optimization under inexact hessian information. *arXiv preprint arXiv:1708.07164*, 2017.

[24] Z. Yao, P. Xu, F. Roosta-Khorasani, and M. W. Mahoney. Inexact non-convex Newton-type methods. *arXiv preprint arXiv:1802.06925*, 2018.

[25] S. De, A. Yadav, D. Jacobs, and T. Goldstein. Automated inference with adaptive batches. In *AISTATS*, 2017.

[26] C. Tan, S. Ma, Y. Dai, and Y. Qian. Barzilai-Borwein step size for stochastic gradient descent. In *NIPS*, 2016.