Abstract

SGDLibrary is an open source MATLAB library of stochastic optimization algorithms, which finds the minimizer of a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ of the finite-sum form $\min f(w) = 1/n \sum_{i=1}^{n} f_i(w)$. 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 Introduction

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

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

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 $\ell_2$-norm regularized linear regression problem (a.k.a. ridge regression) and an $\ell_1$-norm regularized logistic regression (LR) problem. Full gradient descent (a.k.a. steepest descent) with a step-size $\eta$ is the most straightforward approach for (1), which updates as $w_{k+1} = w_k - \eta \nabla f(w_k)$ 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 $w_{k+1} = w_k - \eta \nabla f_i(w_k)$ for the $i$-th sample uniformly at random [2, 3]. SGD assumes an unbiased estimator of the full gradient as $\E_i[\nabla f_i(w_k)] = \nabla f(w_k)$. As the update rule clearly represents, the calculation cost is independent of $n$. Furthermore, mini-batch SGD [3] calculates $1/|S_k| \sum_{i \in S_k} \nabla f_i(w_k)$, where $S_k$ is the set of samples of size $|S_k|$. 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 speeds. 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].

---

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 \([\ell] \). 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, \( \ell_2 \)-norm regularized multidimensional linear regression, \( \ell_2 \)-norm regularized linear SVM, \( \ell_2 \)-norm regularized linear LR, \( \ell_2 \)-norm regularized softmax classification (multinomial LR), \( \ell_1 \)-norm multidimensional linear regression, and \( \ell_1 \)-norm LR. The problem descriptor provides additional specific functions. For example, the LR problem includes the prediction and the classification accuracy calculation functions.

<table>
<thead>
<tr>
<th>No.</th>
<th>Class functions (methods).</th>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>calculate (full) cost function ( f(w) ).</td>
<td>✓</td>
</tr>
<tr>
<td>(ii)</td>
<td>calculate mini-batch stochastic derivative ( v=1/</td>
<td>S</td>
</tr>
<tr>
<td>(iii)</td>
<td>calculate stochastic Hessian.</td>
<td>✓</td>
</tr>
<tr>
<td>(iv)</td>
<td>calculate stochastic Hessian-vector product for a vector ( v ).</td>
<td>✓</td>
</tr>
<tr>
<td>(v)</td>
<td>problem-specific functions. (e.g., classification accuracy calculation in LR problem.)</td>
<td>✓</td>
</tr>
</tbody>
</table>

Optimization solver: Once a solver function is called with one selected problem descriptor \( \text{problem} \) as the first argument, an optimization solver solves the optimization problem by calling corresponding functions via \( \text{problem} \) 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 \( \text{struct} \), designated as \( \text{options} \) 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 \([\ell] \) is a non-smooth regularizer such as the \( \ell_1 \)-norm regularizer \( \|w\|_1 \), the solver calls the proximal operator as \( \text{problem.prox}(w,\text{stepsize}) \), which is the wrapper function defined in each problem. The \( \ell_1 \)-norm regularized LR problem, for example, calls the soft-threshold function as \( w = \text{prox}(w,\text{stepsize})=\text{soft.thresh}(w,\text{stepsize}*\lambda) \), where \( \text{stepsize} \) is the step-size \( \eta \) and \( \lambda \) is the regularization parameter \( \lambda > 0 \) in \([\ell] \).
Table 2: Supported stochastic optimization algorithms.

<table>
<thead>
<tr>
<th>Category</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD method</td>
<td>Vanilla SGD [4], SGD-CM (classical momentum), SGD-CM-NAG (Nesterov’s</td>
</tr>
<tr>
<td></td>
<td>accelerated gradient) [5], AdaGrad [6], RMSProp [7], AdaDelta [8],</td>
</tr>
<tr>
<td></td>
<td>Adam [9], AdaMax [9]</td>
</tr>
<tr>
<td>Variance reduction (VR)</td>
<td>SVRG [10], SAG [11], SAGA [12], SARAH [13], SARAH-Plus [13]</td>
</tr>
<tr>
<td>Second-order (SO)</td>
<td>SQN [14], oBFGS-Inf [15], oBFGS-Lim [15, 16], Reg-oBFGS-Inf [17],</td>
</tr>
<tr>
<td>method</td>
<td>Damp-oBFGS-Inf [18], IQN [19]</td>
</tr>
<tr>
<td>SO with VR method</td>
<td>SVRG-SQN [20], SVRG-LBFGS [21], SS-SVRG [24]</td>
</tr>
<tr>
<td>Sub-sampled Hessian</td>
<td>SCR (Sub-sampled cubic regularization) [24], Sub-sampled TR (trust region)</td>
</tr>
<tr>
<td>method</td>
<td>[23, 24]</td>
</tr>
<tr>
<td>Other method</td>
<td>BB-SGD [25], SVRG-BB [26]</td>
</tr>
</tbody>
</table>

3 Tour of the SGDLibrary: Softmax classification problem

We embark on a tour of SGDLibrary exemplifying the $\ell_2$-norm regularized softmax classification problem. The code for this problem is in Listing 1.

Listing 1: Demonstration code for $\ell_2$-norm regularized softmax classification problem.

```matlab
% 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});
```

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.

Figure 1: Results of $\ell_2$-norm regularized softmax classification problem.
References


