Stochastic Gradient Descent with Momentum and Line Searches

David Nardi
MSc student in Artificial Intelligence, Univeristy of Florence

30th March 2024

Abstract

In recent years, tailored line search approaches have proposed to define the step-size, or learning rate, in SGD-type algorithms for finite-sum problems. In particular, a stochastic extension of standard Armijo line search has been proposed in Vaswani, Mishkin, Laradji *et al.* [1]. The development of this kind of techniques is relevant, because it shall allow to enforce a stronger converging behaviour (due to the Armijo condition), similar to that of standard GD, within SGD methods that are commonly employed with large scale training problems.

However, the stochastic line search is not immediately employable when the momentum term is part of the update equation, as the search direction might not be a descent direction (which is a necessary condition for the Armijo condition). This problem is addressed in Fan, Vaswani, Thrampoulidis *et al.* [2], where a strategy is proposed to guarantee the descent property with momentum.

Contents

1 Introduction

Different SGD-type algorithms proposed by the literature were implemented and tested on different datasets for solving the ℓ_2-regularized Logistic Regression training problem.

For the purpose of this work, those algorithms were grouped into one, see algorithm 5 on page 9, follows a list of the variants

- SGD with fixed or decreasing step-size, and line search, see section 2.1 on page 5;
- SGD with momentum term and momentum correction or restart, see section 2.2 on page 6.

This section describes the Machine Learning (ML) problem and the related optimization problem, then section 2 on page 4 summarizes the approaches proposed from the retrieved papers. Section 3 on page 8 describes the experiments performed for showing the behaviour of the algorithms on different datasets.

1.1 Classification task

Given a dataset as follows

$$\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in \mathcal{X},\, y^{(i)} \in \mathcal{Y},\, i = 1, 2, \ldots, N\}$$

the general machine learning optimization problem in the context of *supervised learning* is

$$\min_{w} f(w) = L(w) + \lambda\Omega(w) \longrightarrow \begin{cases} L(w) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(w) \\ \Omega_{\ell_2} = \frac{1}{2}\|w\|_2^2 \end{cases}$$

where $L(w)$ is the *loss function* which is dived by the total number of samples in the dataset and $\Omega(w)$ is the *regularization term* with its coefficient λ. There are three regularization possible choices, the ℓ_2 regularization was chosen for the problem that we want to address. The vector w contains the model weights associated to the dataset features.

The task performed is the *binary classification* (so the allowed values for the response variable are $\mathcal{Y} = \{-1, 1\}$), using the Logistic Regression model. The selected loss function is the *log-loss*, for one dataset sample is

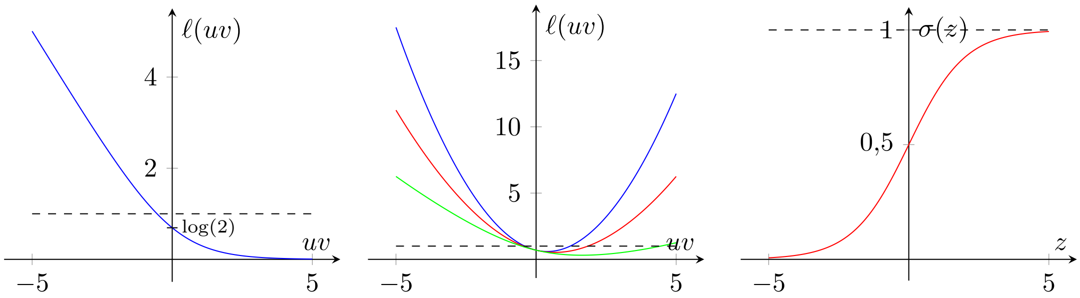
$$\ell_i(w) = \log\bigl(1 + \exp(-y^{(i)} w^T x^{(i)})\bigr) \tag{1}$$

figure 1a on the next page shows a plot of the loss function $\ell(uv) = \log\bigl(1 + \exp(-uv)\bigr)$ where $u = y^{(i)}$ and $v = w^T x^{(i)}$.

Prediction

Once the model is trained, we use the sigmoid function, see figure 1c on the following page, to classify (as positive or negative class) unseen data as follows

$$y^{(i)} = \begin{cases} 1 & \text{if } w^T x^{(i)} > 0.5 \\ -1 & \text{if } w^T x^{(i)} \leq 0.5 \end{cases}$$



(a) *Log-loss, equation (1). if $uv \gg 0$ then the example is labelled correctly; if $uv \ll 0$ then the label is the wrong one; if $uv \approx 0$ then w is the null model.*

(b) *Influence of the regularization term on the loss function, equation (3a), $\lambda = 1, 0.5, 0.1$*

(c) *Sigmoid function. Used for prediction with encoding: if $v > 0.5 \Rightarrow \hat{u} = 1$ and if $v \leq 0.5 \Rightarrow \hat{u} = -1$.*

1.2 Optimization problem

Putting together the loss function and the regularization term, we can obtain the optimization problem that we want to solve using Stochastic Gradient Descent (SGD) algorithm variants

$$\min_{w \in \mathbb{R}^{(p+1)}} f(w) = \frac{1}{N} \sum_{i=1}^{N} \log\bigl(1 + \exp(-y^{(i)} w^T x^{(i)})\bigr) + \lambda \frac{1}{2} \|w\|^2 \tag{2}$$

where $i = 1, \dots, N$ are the dataset samples, $\mathcal{X} \subseteq \mathbb{R}^{(p+1)}$ where $p+1$ means that there are p features and the intercept. We define the matrix associated to the dataset and the model weights as follows

$$X^T = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_p^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_p^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_p^{(N)} \end{pmatrix} \in \mathbb{R}^{N \times (p+1)} \qquad x^{(i)} = \begin{pmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_p^{(i)} \end{pmatrix} \qquad w = \begin{pmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{pmatrix}$$

the constant column is meant for the intercept, also known as *bias*, the b weight in vector w. A compact definition for the dataset matrix is $X = (x^{(1)}, x^{(2)}, \dots, x^{(N)})$.

The objective function $f \colon \mathbb{R}^{(p+1)} \to \mathbb{R}$ is of class $f \in C^2(\mathbb{R}^{(p+1)})$, we compute the first and second order derivatives

$$f(w) = \frac{1}{N} \sum_{i=1}^{N} \log\bigl(1 + \exp(-y^{(i)} w^T x^{(i)})\bigr) + \lambda \frac{1}{2} \|w\|^2 \tag{3a}$$

$$\nabla f(w) = \frac{1}{N} X r + \lambda w \tag{3b}$$

$$\nabla^2 f(w) = \frac{1}{N} X D X^T + \lambda I_{(p+1)} \tag{3c}$$

where $r \in \mathbb{R}^N$ is a vector of the same length as the total number of samples, whose elements are $r_i = -y^{(i)} \sigma(-y^{(i)} w^T x^{(i)})$, note that $\sigma(z)$ is the sigmoid function, $D \in \mathbb{R}^{N \times N}$ is a diagonal matrix whose elements are $d_{ii} = \sigma(y^{(i)} w^T x^{(i)}) \sigma(-y^{(i)} w^T x^{(i)})$ which implies $d_{ii} \in (0, 1)$, and $I_{(p+1)}$ is the identity matrix with size $p+1$. Dividing by N means dividing by the total number of samples involved.

The next proposition allows to solve the optimization problem.

Proposition 1. *Problem* (2) *admits a unique optimal solution.*

Proof. We need to prove the existence and the uniqueness of the global minimum.
(i) *Existence* of a optimal solution. The problem is quadratic and the objective function is coercive, in fact $\forall \{w^k\}$ s.t. $\lim_{k\to\infty} \|w^k\| = \infty$ holds

$$\lim_{k\to\infty} f(w^k) \geq \lim_{k\to\infty} \lambda \frac{1}{2} \|w^k\|^2 = \infty \Rightarrow \lim_{k\to\infty} f(w^k) = \infty$$

hence by a corollary of the Weirstrass theorem, the problem admits global minimum in $\mathbb{R}^{(p+1)}$.
(ii) *Unicity* of the optimal solution. We now prove that the hessian matrix (3c) is positive definite

$$w^T \nabla^2 f(w) w = w^T X D X^T w + \lambda w^T I w = \underbrace{y^T D y}_{\geq 0} + \lambda \|w\|^2 \geq \lambda \|w\|^2 > 0 \quad \forall w$$

the hessian matrix positive definite implies that the objective function is strictly convex and that implies that the global minimum, if exists, is unique. Being in the convex case, the global minimum is a $w^* \in \mathbb{R}^{(p+1)}$ s.t. $\nabla f(w^*) = 0$ for first-order optimality conditions. ■

Remark 1. Since the log-loss is convex, the regularization term makes the objective function also *strongly convex*, this should speed up the optimization process.

2 Mini-batch gradient descent variants

In this section we tackle the algorithmic part, specifically the SGD-type is the Mini-batch Gradient Descent where the mini-batch size M is greater than 1 and much less than the dataset size, i.e. $1 < |B| = M \ll N$, however, we will call it SGD.

In order to use the algorithm, it is necessary to make further assumptions on the objective function and the gradients (how far the gradient samples are from the *true gradients*)

- the objective function in problem 2 is a loss function plus a quadratic regularization term, f is bounded below by some value f^* as we can also see in figure 1a;
- for some constant $G > 0$ the magnitude of all gradients samples is bounded $\forall w \in \mathbb{R}^{(p+1)}$, by $\|\nabla f_i(w)\| \leq G$;
- other than twice continuously differentiable, we assume that f has Lipschitz-continuous gradients with constant $L > 0$, one can also say that f is L-smooth.

The algorithm is globally convergent, so the starting point will be an arbitrary $w^0 \in \mathbb{R}^{(p+1)}$.

Stopping criterion and failures

Regarding the implementation of the algorithm, it is essential to define a stopping criterion. Given a small $\varepsilon > 0$ the chosen criterion is as follows

$$\|\nabla f(w^k)\| \leq \varepsilon \tag{4}$$

unlike the first one, the second is independent from the scale of the objective function. Note that the criterion uses the full gradient.

Other than the stopping criterion, we can add conditions of premature termination like

- exceeding a threshold for the epochs number k^* or function and gradient evaluations;
- internal failures when computing w^{k+1}, for example exceeding q^* iterations during the line search (as you will se later, for the step-size α as well as the momentum term β).

Mini-batch gradient

Now we spend few words about the notation and the computation of the mini-batch gradient. Being on epoch k at iteration t, a model update starting from a w^k has the following form

$$z^{t+1} = z^t + \alpha_t d_t \tag{5}$$

the update uses information from the mini-batch B_t in the direction d_t and the step-size α_t follows a certain rule.*

The direction is an expression involving the gradient, so we want to compute the gradient w.r.t. z^t on the mini-batch B_t whose indices are randomly chosen $i_t \subset \{1, \dots, N\}$. Knowing that $\nabla f_i(w^k) = x^{(i)} r_i + \lambda w^k$

$$\begin{aligned}\nabla f_{i_t}(z^t) &= \frac{1}{M} \sum_{i \in B_t} \nabla \ell_i(z^t) + \lambda \nabla \Omega(z^t) \\ &= \frac{1}{M} \underbrace{Xr}_{i \in B_t} + \lambda z^t\end{aligned} \tag{6}$$

the expression is the same as the full gradient (3b) except that the dataset matrix contains just the mini-batch samples, so the r vector.

2.1 Stochastic gradient descent

The basic SGD version has the following update rule

$$z^{t+1} = z^t - \alpha_t \nabla f_{i_t}(z^t) \tag{7}$$

so the direction is defined as $d_t = -\nabla f_{i_t}(z^t)$ that is the *anti-gradient* evaluated on the considered mini-batch, we know that on average is a *descent direction* so the objective function doesn't decrease necessarily at each step.

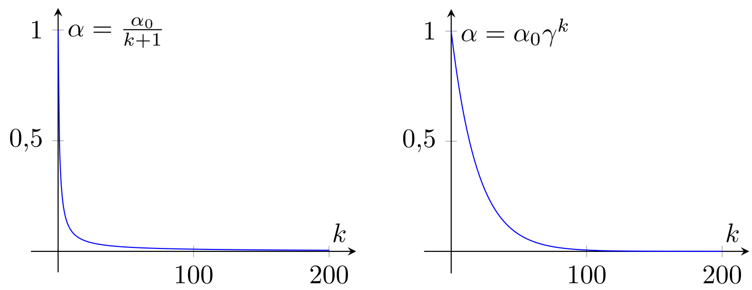
Given an initial step-size $\alpha_0 \in \mathbb{R}^+$, the first two basic version are

- `SGD-Fixed`: constant step-size $\alpha_t = \alpha_0$;
- `SGD-Decreasing`: decreasing step-size $\alpha_t = \frac{\alpha_0}{k+1}$.

The first choice sees the same step-size between the epochs and so the iterations. The second choice changes the step-size at every epoch, while being constant between iterations, that particular form ensures the convergence. For this two algorithms the momentum term in algorithm 5 is set to $\beta_0 = 0$.

---

* *Iterations* is defined as the total number of mini-batches extracted from the dataset, while one *epoch* is when the entire dataset is passed forward. The counter for the mini-batch currently processed is t while k is for the epoch.



(a) *Inverse decay, starting with* $\alpha_0 = 1$

(b) *Exponential decay, starting with* $\alpha_0 = 1$ *and decay rate* $\gamma = 0.95$

Figure 1: Step-size schedules for the SGD-Decreasing algorithm

2.1.1 Stochastic line search

Now we move forward to the approach by Vaswani, Mishkin, Laradji *et al.* [1]. For using the algorithm proposed by the paper, one more assumption is needed, that is, the model is able to *interpolate* the data, this property requires that the gradient w.r.t. each samples converges to zero at the optimal solution

$$\text{if } w^* \mid \nabla f(w^*) = 0 \Rightarrow \nabla f_i(w^*) = 0 \;\; \forall i = 1, \dots, N$$

The proposed approach applies the Armijo line search to the SGD algorithm at every iteration, specializing the sufficient reduction condition in the context of finite-sum problems. Hence the *Armijo condition* has the following form

$$f_{i_t}\big(z^t - \alpha_t \nabla f_{i_t}(z^t)\big) \le f_{i_t}(z^t) - \gamma \alpha_t \|\nabla f_{i_t}(z^t)\|^2 \tag{8}$$

the coefficient γ is an hyper-parameter that will be set to $1/2$ for convergence properties stated by the paper.

As the standard Armijo method, the proposed line search uses a *backtracking* technique that iteratively decreases the initial step-size $\alpha_0 \in \mathbb{R}^+$ by a constant factor δ usually set to $1/2$ until the condition is satisfied.

The authors also gave heuristics in order to avoid unnecessary function evaluations by *restarting* at each iteration the step-size, to the previous multiplied by the factor $a^{M/N}/\delta$, see algorithm 1 on page 8.

For this algorithm the momentum term is set to $\beta_0 = 0$.

2.2 Adding momentum term

The iteration performed over the mini-batches is (5) what differs from the previous versions is the direction that is

$$d_t = -\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big)$$

in a finite-sum problem the momentum term lies in a specific range $\beta_0 \in (0,1)$ and is a constant value, the algorithm that uses this direction is the SGDM, the resulting iteration

$$z^{t+1} = z^t - \alpha_t\big((1-\beta)\nabla f_{i_t}(z^t) + \beta d_{t-1}\big) \tag{9}$$

which is applied as the general update rule in algorithm 5, in this case the momentum term is set to a constant value $\beta = \beta_0$. To be clear we have the following cases

$$z^{t+1} = z^t - \alpha_t\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big) \begin{cases} \xrightarrow{\beta_0 = 0} & (7)\ \texttt{SGD-Fixed, SGD-Decreasing} \\ \xrightarrow[\beta_0 \in (0,1)]{} & (9)\ \texttt{SGDM} \end{cases}$$

2.2.1 Stochastic line search

As the paper by Fan, Vaswani, Thrampoulidis *et al.* [2] says, when using the momentum term together with a line search, β_0 complicates the selection of a suitable step-size. The Armijo line search applied to the SGDM algorithm has the following condition

$$f_{i_t}(z^{t+1}) \leq f_{i_t}(z^t) - \gamma\alpha_t \nabla f_{i_t}(z^t)^T\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big) \tag{10}$$

but this approach is not robust to the choice of the momentum term as the paper says.

The problem is that $\nabla f_{i_t}(z^t)^T d_t < 0$ isn't always guaranteed, i.e. the direction is not descent, therefore the line search doesn't converge. Starting from an initial $\beta_0 \in (0, 1)$, there are two situations that can be resolved as follows

$$\begin{tikzcd} & \nabla f(z^t)^T d_t \arrow[dl, "< 0"'] \arrow[dr, "\geq 0"] & \\ \texttt{evaluate } (10) & & \beta = \delta\beta \arrow[ul, bend right, "\texttt{check}"'] \end{tikzcd}$$

in algorithmic terms, until the direction is descent, damp the momentum term by a factor δ, which is usually set to 0.5 like in the line search. Using this procedure, a descent direction d_t is guaranteed and it is possible to apply the algorithm 2, the procedure is called *momentum correction*, see algorithm 3 on the following page. The resulting algorithm is MSL-SGDM-C.

This procedure can be expensive, so the paper suggests another approach called *momentum restart*, when the descent direction condition for d_t isn't satisfied, the procedure restarts that direction by setting $d_{t-1} = d_0$, the paper suggests $d_0 = 0$, in general

$$\begin{tikzcd} & \nabla f(z^t)^T d_t \arrow[dl, "< 0"'] \arrow[dr, "\geq 0"] & \\ \texttt{evaluate } (10) & & d_{t-1} = d_0 \arrow[d, "d_t = -\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_0\big)"] \\ & & \texttt{evaluate } (10) \end{tikzcd}$$

so if $d_0 = 0$ the direction will be $d_t = -(1-\beta_0)\nabla f_{i_t}(z^t)$ that is a descent direction, see algorithm 4 on the next page. The resulting algorithm is MSL-SGDM-R.

The authors suggest to set the momentum term to $\beta_0 = 0.9$.

Algorithm 1: reset

Input: α, α_0, M, N, t, $a \in \mathbb{R}^+$, $\texttt{opt} \in \{0, 1, 2\}$

if $t = 0$ **or** $\texttt{opt} = 1$ **then**
  return α_0
else if $\texttt{opt} = 0$ **then**
  $\alpha \leftarrow \alpha$
else if $\texttt{opt} = 2$ **then**
  $\alpha \leftarrow \alpha a^{M/N}$
end

Output: α

Algorithm 2: armijo-method

Data: $\gamma \in (0, 1)$, $\delta \in (0, 1)$, q^*

Input: z^t, d_t, α

$\alpha \leftarrow \alpha/\delta$;
$q \leftarrow 0$;
repeat
  $\alpha \leftarrow \delta\alpha$;
  $z^{t+1} \leftarrow z^t + \alpha d_t$;
  $q \leftarrow q + 1$;
until $f_{i_t}(z^{t+1}) \leq f_{i_t}(z^t) + \gamma\alpha\nabla f_{i_t}(z^t)^T d_t$ **or** $q \geq q^*$;

Output: α

Algorithm 3: momentum-correction

Data: $\delta \in (0, 1)$, q^*

Input: β_0, $\nabla f_{i_t}(z^t)$, d_{t-1}

$\beta \leftarrow \beta_0$;
$q \leftarrow 0$;
repeat
  $\beta \leftarrow \delta\beta$;
  $d_t \leftarrow -\big((1 - \beta)\nabla f_{i_t}(z^t) + \beta d_{t-1}\big)$;
  $q \leftarrow q + 1$;
until $\nabla f_{i_t}(z^t)^T d_t < 0$ **or** $q \geq q^*$;

Output: d_t

Algorithm 4: momentum-restart

Data: d_0

Input: β_0, $\nabla f_{i_t}(z^t)$, d_{t-1}

$q \leftarrow 0$;
$d_t \leftarrow -\big((1 - \beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big)$;
if not $\nabla f_{i_t}(z^t)^T d_t < 0$ **then**
  $d_{t-1} \leftarrow d_0$;
end

Output: d_t

3 Experiments and results discussion

To test the efficiency the algorithms, a benchmark of six datasets retrieved from LIBSVM is used, see table 1 on the following page for details.

First the six algorithms are tested on a fixed number of epochs, Fan, Vaswani, Thrampoulidis *et al.* [2] set the value to 200, so we do the same. We keep track of the loss function value for every epoch and the running time that every epoch took; our aim is to show how the value decreases on every epoch and the running time that takes, see figures on pages 12, 13 and 14.

Once we have the algorithms performance at different step-size values, a fine-tuning of the hyper-parameter is done in order to obtain the best solver for every dataset based on the accuracy score and loss function values. For a better comparison, the L-BFGS, Conjugate Gradient and Newton-CG, and the full-batch gradient descent algorithms are also tested.

The only hyper-parameter that varies is the step-size α, the momentum term is set to 0.9, the mini-batch size is a power of 2 and is set according to perform at least 100 *iterations* depending on the considered dataset and the ε tolerance from (4) is set to 10^{-3}.

Algorithm 5: SGD variants

Data: $w^0 \in \mathbb{R}^{(p+1)}$, $M > 1$, k^*, $\varepsilon > 0$, $\alpha_0 \in \mathbb{R}^+$, $\beta_0 \in (0,1)$

$k \leftarrow 0$;

while $\|\nabla f(w^k)\| > \varepsilon$ **and** $k < k^*$ **do**

  create mini-batches $B_0, \ldots, B_{N/M-1}$;

  $z^0 \leftarrow w^k$;

  $d_{-1} \leftarrow 0$;

  $\alpha_{-1} \leftarrow \begin{cases} \frac{\alpha_0}{k+1} & \text{if } \texttt{SGD-Decreasing} \\ \alpha_0 & \text{otherwise} \end{cases}$;

  for $t = 0$ **to** $N/M - 1$ **do**

    get indices i_t from B_t then get the samples;

    $\nabla f_{i_t}(z^t) \leftarrow \sum_{j \in B_t} \nabla f_j(z^t)$;

    $d_t \leftarrow \begin{cases} -\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big) & \text{if } \texttt{SGD, SGDM} \\ \texttt{momentum-correction}\big(\beta_0, \nabla f_{i_t}(z^t), d_{t-1}\big) & \text{if } \texttt{MSL-SGDM-C} \\ \texttt{momentum-restart}\big(\beta_0, \nabla f_{i_t}(z^t), d_{t-1}\big) & \text{if } \texttt{MSL-SGDM-R} \end{cases}$;

    if *SGD-Armijo, MSL-SGDM-C/R* **then**

      $\alpha \leftarrow \texttt{reset}(\alpha_{t-1}, \alpha_0, M, N, t, a, \texttt{opt})$;

      $\alpha_t \leftarrow \texttt{armijo-method}(z^t, d_t, \alpha)$;

    end

    $z^{t+1} \leftarrow z^t + \alpha_t d_t$;

  end

  $w^{k+1} \leftarrow z^{N/M}$;

  $k \leftarrow k + 1$;

end

Table 1: Benchmark datasets

Name	Train	Test	Features	Distribution
w1a	2477	47 272	300	-1:0.97 1:0.03
w3a	4912	44 837	300	-1:0.97 1:0.03
Phishing	8844	2211	68	-1:0.45 1:0.55
a2a	2265	30 296	119	-1:0.75 1:0.25
Mushrooms	6499	1625	112	-1:0.48 1:0.52
German	800	200	24	-1:0.70 1:0.30

Table 2: w1a dataset

Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	6	NaN	0.464 614	4.60×10^{-5}	0.970 236
CG	NaN	7	NaN	0.464 614	9.00×10^{-6}	0.970 236
L-BFGS-B	NaN	7	NaN	0.464 614	2.30×10^{-5}	0.970 236
BatchGD-Fixed	1.000	12	0.0100	0.464 614	5.64×10^{-4}	0.970 236
SGD-Decreasing	0.500	27	0.2149	0.464 614	7.92×10^{-4}	0.970 236
SGD-Fixed	0.010	27	0.1865	0.464 615	8.52×10^{-4}	0.970 236
SGDM	0.010	386	5.9150	0.464 615	9.78×10^{-4}	0.970 236
MSL-SGDM-R	0.005	600	6.4977	0.464 693	9.14×10^{-3}	0.970 236
MSL-SGDM-C	0.005	600	6.3884	0.464 693	9.15×10^{-3}	0.970 236
SGD-Armijo	0.100	600	7.6648	0.536 467	3.64×10^{-1}	0.971 400

Table 3: w3a dataset

Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	6	NaN	0.462 742	1.10×10^{-5}	0.970 203
CG	NaN	7	NaN	0.462 742	2.20×10^{-5}	0.970 203
L-BFGS-B	NaN	7	NaN	0.462 742	3.30×10^{-5}	0.970 203
BatchGD-Fixed	1.000	12	0.0165	0.462 742	5.64×10^{-4}	0.970 203
SGD-Decreasing	0.500	19	0.0818	0.462 743	8.76×10^{-4}	0.970 203
SGD-Fixed	0.010	23	0.1622	0.462 743	9.49×10^{-4}	0.970 203
SGDM	0.100	45	0.3513	0.462 743	8.95×10^{-4}	0.970 203
MSL-SGDM-C	0.005	600	4.0216	0.462 787	6.92×10^{-3}	0.970 203
MSL-SGDM-R	0.005	600	4.1246	0.462 787	6.92×10^{-3}	0.970 203
SGD-Armijo	0.010	600	6.8512	0.500 431	2.68×10^{-1}	0.971 006

Table 4: Phishing dataset

Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	5	NaN	0.685 065	0.00	0.567 616
L-BFGS-B	NaN	5	NaN	0.685 065	8.00×10^{-6}	0.567 616
CG	NaN	6	NaN	0.685 065	2.30×10^{-5}	0.567 616
SGD-Decreasing	0.100	6	0.0403	0.685 065	5.08×10^{-4}	0.567 616
SGDM	0.100	22	0.2711	0.685 065	5.75×10^{-4}	0.567 616
BatchGD-Fixed	1.000	11	0.0551	0.685 065	5.34×10^{-4}	0.567 616
SGD-Fixed	0.010	13	0.1737	0.685 065	9.27×10^{-4}	0.567 616
MSL-SGDM-R	0.100	600	17.4982	0.685 660	3.26×10^{-2}	0.568 521
MSL-SGDM-C	1.000	600	9.5985	0.685 705	3.27×10^{-2}	0.568 973
SGD-Armijo	0.005	600	7.5786	0.687 736	6.65×10^{-2}	0.865 219

Table 5: a2a dataset

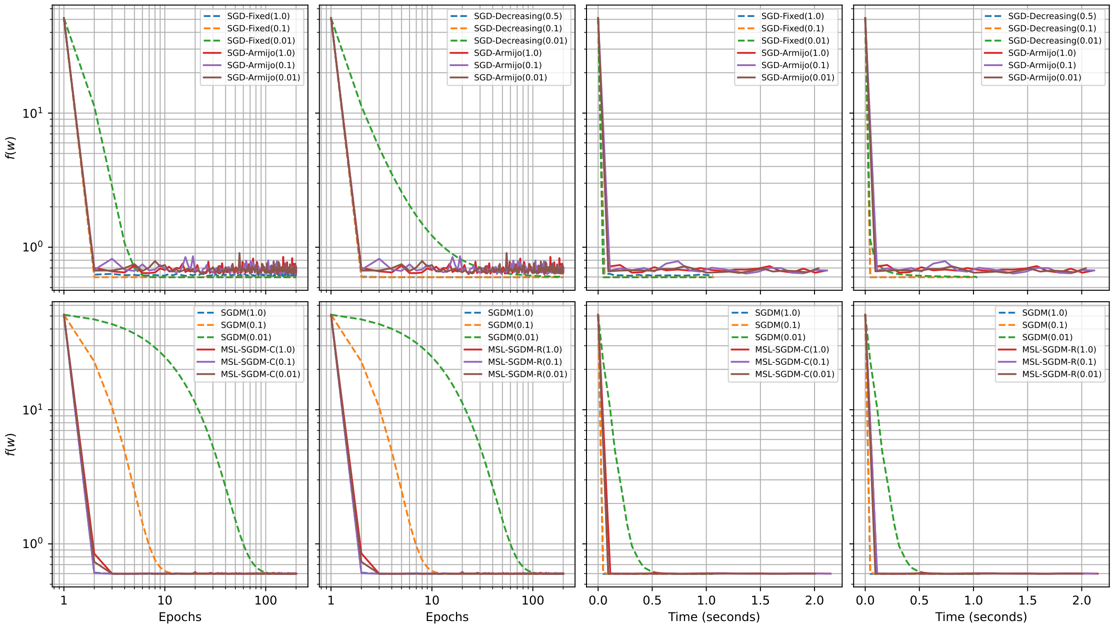
Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	5	NaN	0.564 027	4.00×10^{-6}	0.760 265
CG	NaN	12	NaN	0.564 027	1.50×10^{-5}	0.760 265
L-BFGS-B	NaN	8	NaN	0.564 027	1.20×10^{-5}	0.760 265
SGD-Decreasing	0.800	59	0.1832	0.564 028	7.26×10^{-4}	0.760 265
SGDM	0.100	600	1.8731	0.564 030	2.63×10^{-3}	0.760 298
MSL-SGDM-R	0.010	600	7.8895	0.577 575	2.28×10^{-1}	0.790 236
MSL-SGDM-C	0.010	600	7.4829	0.579 879	2.29×10^{-1}	0.789 345
BatchGD-Fixed	1.000	600	0.2600	0.594 416	3.64×10^{-1}	0.822 386
SGD-Fixed	1.000	600	4.3316	0.602 741	3.56×10^{-1}	0.807 136
SGD-Armijo	1.000	600	6.0688	0.617 908	4.33×10^{-1}	0.798 917

Table 6: Mushrooms dataset

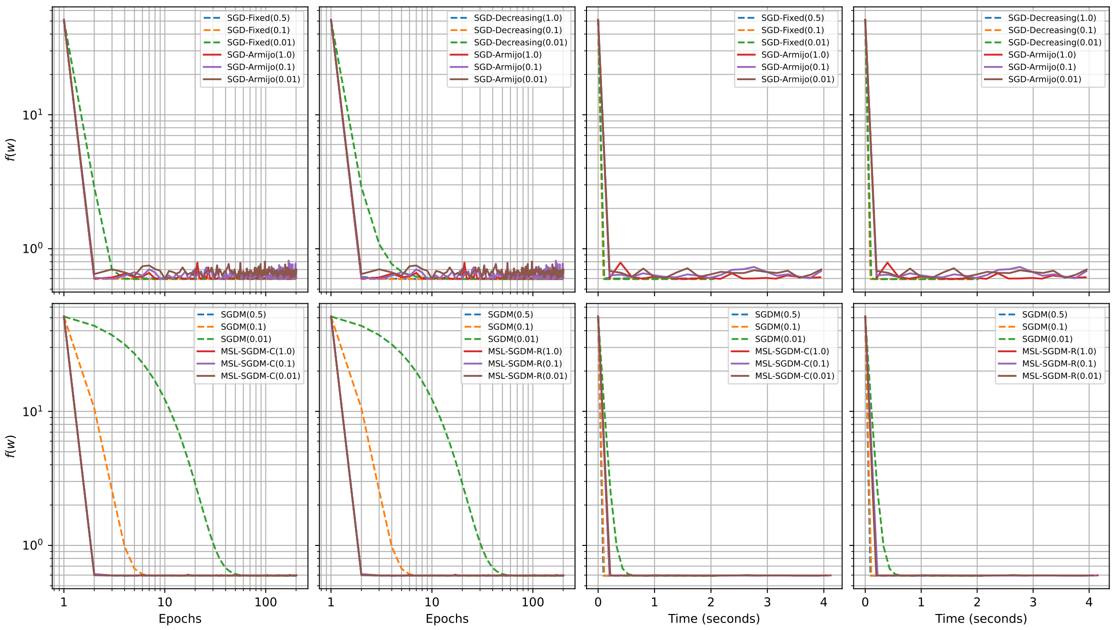
Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	7	NaN	0.517 726	3.00×10^{-6}	0.892 923
CG	NaN	11	NaN	0.517 726	2.40×10^{-5}	0.892 923
L-BFGS-B	NaN	10	NaN	0.517 726	1.70×10^{-5}	0.892 923
SGD-Decreasing	0.100	26	0.5330	0.517 727	7.79×10^{-4}	0.893 538
BatchGD-Fixed	0.500	26	0.0599	0.517 727	7.57×10^{-4}	0.892 923
SGD-Fixed	0.500	600	2.9938	0.525 499	2.00×10^{-1}	0.926 154
MSL-SGDM-R	0.100	600	7.8855	0.527 069	2.32×10^{-1}	0.940 308
MSL-SGDM-C	0.100	600	13.4117	0.527 262	2.24×10^{-1}	0.939 692
SGD-Armijo	0.100	600	11.3674	0.535 765	2.34×10^{-1}	0.953 231
SGDM	1.000	600	4.3694	0.557 069	4.79×10^{-1}	0.924 308

Table 7: German dataset

Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	5	NaN	0.597 303	1.00×10^{-5}	0.710 000
CG	NaN	12	NaN	0.597 303	4.00×10^{-6}	0.710 000
L-BFGS-B	NaN	7	NaN	0.597 303	1.40×10^{-5}	0.710 000
SGD-Fixed	0.010	58	0.1293	0.597 303	7.75×10^{-4}	0.710 000
BatchGD-Fixed	0.500	20	0.0119	0.597 303	8.82×10^{-4}	0.710 000
MSL-SGDM-R	0.005	600	2.2218	0.597 456	2.32×10^{-2}	0.710 000
MSL-SGDM-C	0.005	600	2.8619	0.607 466	1.40×10^{-1}	0.735 000
SGD-Decreasing	0.010	600	2.8675	0.607 993	1.14×10^{-1}	0.720 000
SGD-Armijo	0.100	600	2.4842	0.614 589	2.30×10^{-1}	0.740 000
SGDM	1.000	600	2.5225	0.616 375	3.14×10^{-1}	0.745 000

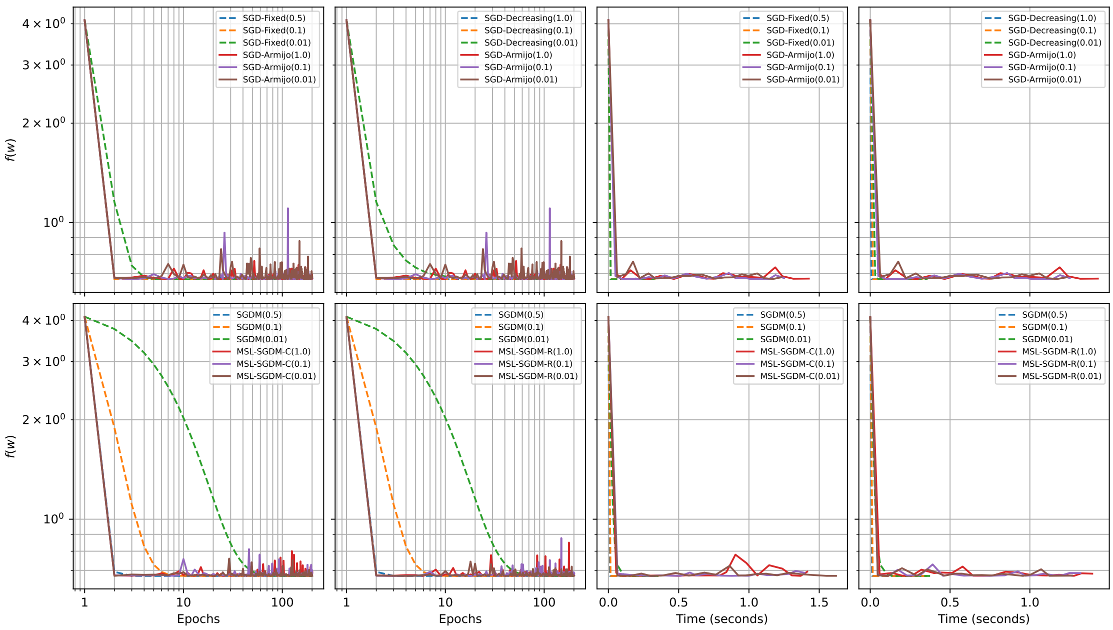


(a) *w1a dataset*

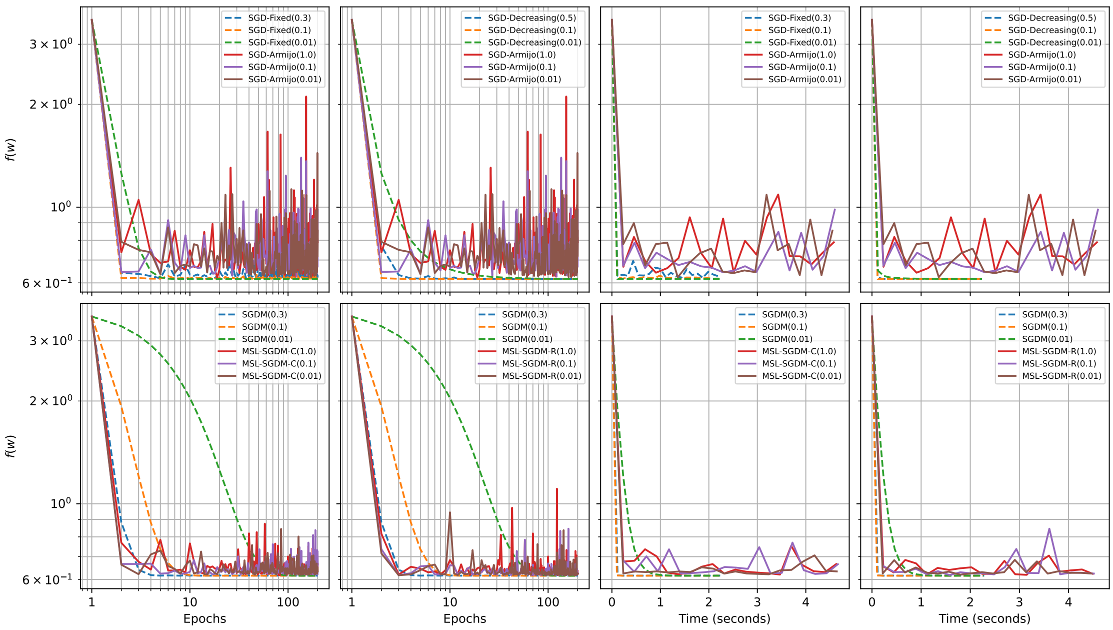


(b) *w3a dataset*

Figure 2: w1a and w3ar

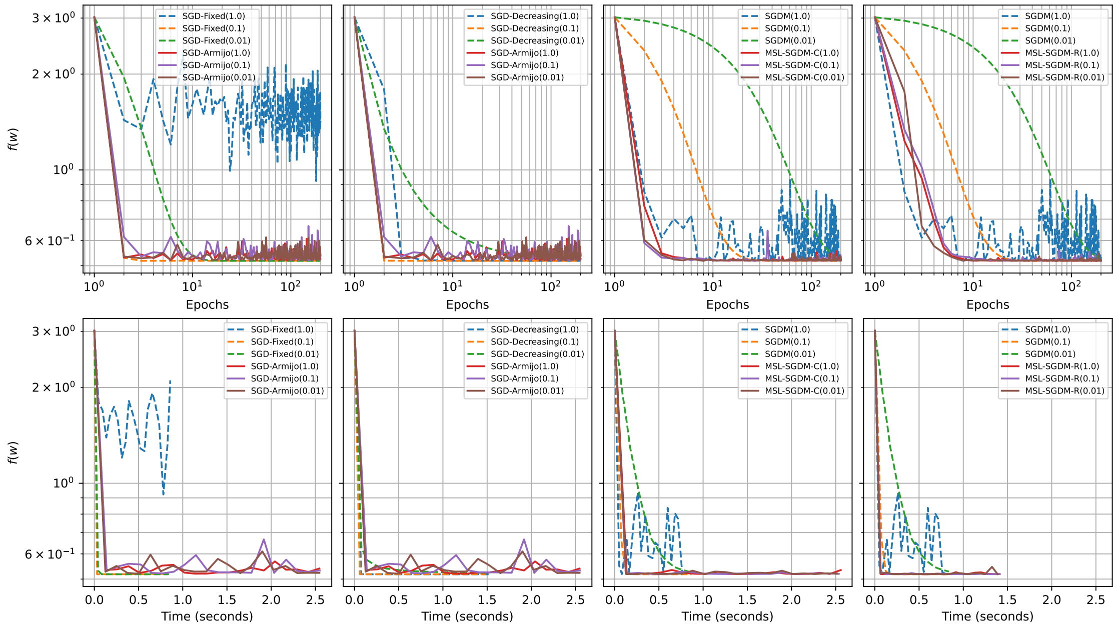


(a) *Phishing dataset*

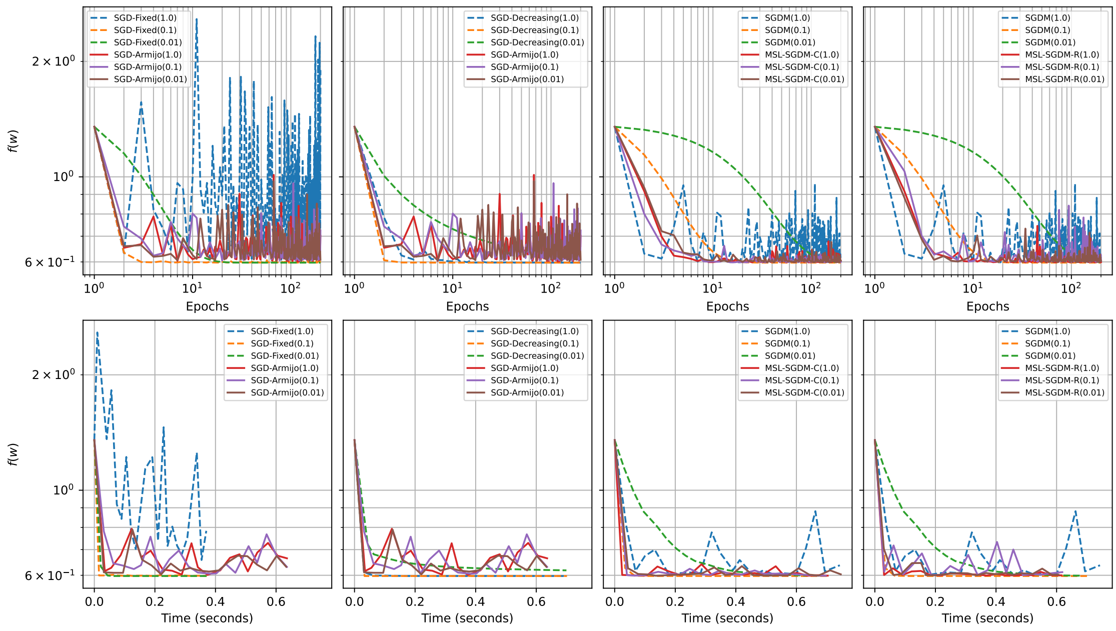


(b) *a2a dataset*

Figure 3: Phishing and a2a datasets



(a) *Mushrooms dataset*



(b) *German dataset*

Figure 4: Mushrooms and German datasets

References

[1] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel and S. Lacoste-Julien, 'Painless stochastic gradient: Interpolation, line-search, and convergence rates,' presented at the Advances in Neural Information Processing Systems, ISSN: 1049-5258, vol. 32, 2019 (cit. on pp. 1, 6).

[2] C. Fan, S. Vaswani, C. Thrampoulidis and M. Schmidt, 'MSL: An adaptive momentem-based stochastic line-search framework,' presented at the OPT 2023: Optimization for Machine Learning, 2023 (cit. on pp. 1, 7, 8).