Stochastic Gradient Descent with Momentum and Line Searches

David Nardi
MSc in Artificial Intelligence
Univeristy of Florence

28th April 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 Binary classification

Given a dataset as follows

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

in binary classification the response variable is in the set $\mathcal{Y} = \{-1, 1\}$. In statistical terms we want to model the probability of being in the positive class

$$\mathbb{E}\big(Y | X = x^{(i)}\big) = \mathbb{P}\big(y^{(i)} = 1 | X = x^{(i)}\big) = m(x^{(i)}) \in (0, 1)$$

so we must model the probability using a function that gives outputs in $(0, 1)$ for all values of X.

Once we have the model, we need a decision function $h\colon \mathcal{X} \to \{-1, 1\}$ to classify between the two classes, the one that minimizes the classification risk is the Bayes classifier

$$h(x) = \begin{cases} 1 & \text{if } m(x) > 1/2 \\ -1 & \text{otherwise} \end{cases}$$

Recall the optimization problem where L is the loss function, in the classification setting we call it the classification risk

$$\min_{w,b} L(w, b; X, y) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(f(w, b; x^{(i)}), y^{(i)}\big)$$

Calling $u = y$ and $v = m(x)$ possible loss functions are

- 0-1 loss $\ell(u, v) = \mathbb{I}\{u \neq v\}$, applying first the decision function
- Log-loss $\ell(u, v) = -\big(u \log(v) - (1 - u) \log(1 - v)\big)$
- Hinge loss $\ell(u, v) = \max\{0, 1 - uv\}$

see figure 1.

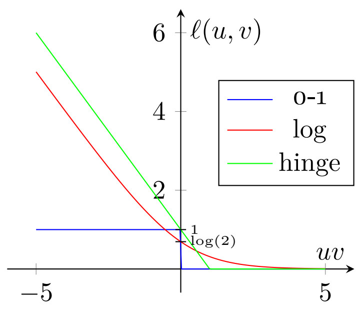


Figure 1: Loss functions

1.1 Logistic regression

Logistic regression uses the logistic function to model the probability as follows

$$m(x) = \sigma(x) = \frac{1}{1 + \exp\big(-(w^T x + b)\big)} \to \log\left(\frac{m(x)}{1 - m(x)}\right) = w^T x + b$$

The empirical risk minimization

$$\min_{w,b} \frac{1}{N} \sum_{i=1}^{N} \log\Big(1 + \exp\Big(-y^{(i)}\big(w^T x^{(i)} + b\big)\Big)\Big)$$

2 Multi-class classification

2.1 Multinomial logistic regression

3 Introduction

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

For the purpose of this work, those algorithms were grouped into one, see algorithm 5 on page 10, follows a list of the variants

- SGD with fixed or decreasing step-size, and line search, see section 4.1 on page 7;
- SGD with momentum term and line search, see section 4.2 on page 8.

This section describes the Machine Learning (ML) problem and the related optimization problem, then section 4 on page 6 summarizes the approaches proposed from the retrieved papers. Section 5 on page 11 describes the experiments performed for showing the performance of the algorithms on different datasets.

3.1 Classification task

Given a dataset as follows

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

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 for scaling issues 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

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

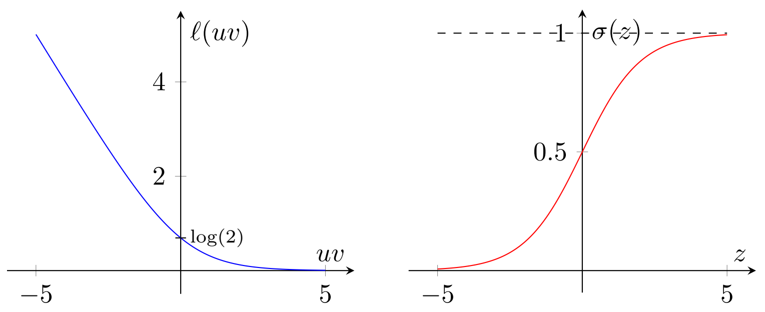
figure 2a on the following page shows a plot of the loss function $\ell(uv) = \log\big(1 + \exp(-uv)\big)$ where $u = y^{(i)}$ and $v = w^T x^{(i)}$.

Prediction

The sigmoid function, see figure 2b on the next page, is used for predicting the labels (positive or negative class) of unseen samples as follows

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

the threshold is set according to the Bayes classifier.



(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.*

(b) *Sigmoid function. Decision function:* if $\sigma(z) > 0.5 \Rightarrow \hat{u} = 1$ *and* if $\sigma(z) \leq 0.5 \Rightarrow \hat{u} = -1$.

Figure 2: Loss function and sigmoid

3.2 Optimization problem

Putting together the loss function and the regularization term, we can obtain the optimization problem that we want to solve using the 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 from the dataset 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, known as *bias* the b weight in vector w. The dataset matrix can be written as $X = (x^{(1)}, x^{(2)}, \dots, x^{(N)})$.

The objective function $f \colon \mathbb{R}^{(p+1)} \to \mathbb{R}$ is at least of class $f \in C^2(\mathbb{R}^{(p+1)})$, we can 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 (3a) is *coercive*, that is $\forall\{w^k\}$ 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 $1/N$ is omitted. 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.

4 Stochastic 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 anyway.

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

- the objective function from problem (2) is a convex loss function plus a quadratic regularization term, since f admits global minimum in $\mathbb{R}^{(p+1)}$ the function is bounded below by some value f^*;
- 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 solution will be an arbitrary $w^0 \in \mathbb{R}^{(p+1)}$, that choice will is discussed in the experiments section.

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

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

note that the criterion uses the full gradient. In machine learning, one does not seek a very low precision. Other than the stopping criterion, one can add conditions of premature termination like

- exceeding a threshold for the epochs number k^*;
- 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 in the Armijo method as well as the momentum term β in the momentum correction).

Mini-batch gradient

Now we spend few words about the notation and the computation of the gradient evaluated on certain mini-batch. Being on *epoch* k with weights w^k, for every t *iteration* a (internal) model update has the following form

$$z^{t+1} = z^t + \alpha_t d_t, \qquad z^0 = w^k \tag{5}$$

until all the mini-batch are processed. The update uses information from the mini-batch B_t when calculating the direction d_t, the step-size α_t follows a certain rule that may use mini-batch information.[*]

The direction makes use of the gradient, so we want to compute the gradient w.r.t. z^t using information from the mini-batch B_t whose indices are randomly chosen from the full dataset $i_t \subset \{1, \ldots, N\}$ using a uniform distribution

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

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.

4.1 Basic stochastic gradient descent

The basic SGD version has the following iteration update rule

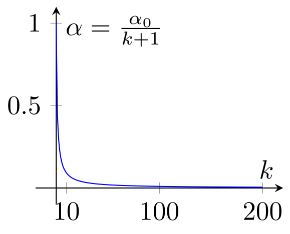
$$z^{t+1} = z^t - \alpha_t \nabla f_{i_t}(z^t) \tag{7}$$

so the direction is defined as $d_t = -\nabla f_{i_t}(z^t)$ that is the negative 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.

[*] *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.

Given an initial $\alpha_0 \in \mathbb{R}^+$, the first two basic versions are

- SGD-Fixed: step-size s.t. $\alpha_t = \alpha_0$;
- SGD-Decreasing: step-size s.t. $\alpha_t = \frac{\alpha_0}{k+1}$, see on the right.



The first choice sees the same step-size between epochs and so iterations. The second choice changes the step-size on 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 $\beta_0 = 0$.

4.1.1 Stochastic line search

Now we move 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 evaluated on each sample 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

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

the coefficient γ is the hyper-parameter controlling the aggressiveness of the condition, the paper suggests to set $1/2$ as its maximum value.

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 one multiplied by the factor $a^{M/N}/\delta$, see algorithm 1 on page 10. Like the basic version, the momentum term is set to $\beta_0 = 0$.

4.2 Adding momentum term

The iteration performed over the mini-batches is still (5) what 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 will be

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

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

$$\begin{tikzcd}
 & (7)\ \text{SGD-} \\
z^{t+1} = z^t - \alpha_t\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big) \arrow[ur, "\beta_0 = 0"] \arrow[dr, "\beta_0 \in (0,1)"'] & \\
 & (9)\ \text{SGDM-}
\end{tikzcd}$$

4.2.1 Momentum and 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 the line search is not robust to the choice of the momentum term as the paper stated.

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. When 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 next page. The resulting algorithm is MSL-SGDM-C.

The momentum correction 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

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

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 on the considered mini-batch, see algorithm 4 on the following page. The algorithm that uses this procedure is MSL-SGDM-R.

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

$$\begin{aligned}
f(w,b) &= \frac{1}{N}\sum_{i=1}^{N} \log\Big(1 + \exp\big(-y^{(i)}\big(w^T x^{(i)} + b\big)\big)\Big) + \lambda\frac{1}{2}\|w\|^2 \\
\nabla f(w,b) &= \frac{1}{N} X r + \lambda w \\
\nabla^2 f(w,b) &= \frac{1}{N} X D X^T + \lambda I_{(p+1)}
\end{aligned}$$

Algorithm 1: reset

```
Input: $\alpha$, $\alpha_0$, $M$, $N$, $t$, $a \in \mathbb{R}^+$,
       $\mathsf{opt} \in \{0, 1, 2\}$
if $t = 0$ or $\mathsf{opt} = 1$ then
    return $\alpha_0$
else if $\mathsf{opt} = 0$ then
    $\alpha \leftarrow \alpha$
else if $\mathsf{opt} = 2$ then
    $\alpha \leftarrow \alpha a^{M/N}$
end
Output: $\alpha$
```

Algorithm 2: armijo-method

```
Data: $\gamma \in (0, 1)$, $\delta \in (0, 1)$, $q^*$
Input: $z^t$, $d_t$, $\alpha$
$\alpha \leftarrow \alpha/\delta$;
$q \leftarrow 0$;
repeat
    $\alpha \leftarrow \delta\alpha$;
    $z^{t+1} \leftarrow z^t + \alpha d_t$;
    $q \leftarrow q + 1$;
until
  $f_{i_t}(z^{t+1}) \le f_{i_t}(z^t) + \gamma\alpha\nabla f_{i_t}(z^t)^T d_t$ or
  $q \ge q^*$;
Output: $\alpha$
```

Algorithm 3: momentum-correction

```
Data: $\delta \in (0, 1)$, $q^*$
Input: $\beta_0$, $\nabla f_{i_t}(z^t)$, $d_{t-1}$
$\beta \leftarrow \beta_0$;
$q \leftarrow 0$;
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 $q \ge q^*$;
Output: $d_t$
```

Algorithm 4: momentum-restart

```
Data: $d_0$
Input: $\beta_0$, $\nabla f_{i_t}(z^t)$, $d_{t-1}$
$q \leftarrow 0$;
$d_t \leftarrow -\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big)$;
if not $\nabla f_{i_t}(z^t)^T d_t < 0$ then
    $d_{t-1} \leftarrow d_0$;
    compute $d_t$;
end
Output: $d_t$
```

Algorithm 5: SGD variants

```
Input: $w^0 \in \mathbb{R}^{(p+1)}$, $M > 1$, $k^*$, $\varepsilon > 0$, $\alpha_0 \in \mathbb{R}^+$, $\beta_0 \in (0, 1)$
$k \leftarrow 0$;
while $\|\nabla f(w^k)\| > \varepsilon$ and $k < k^*$ do
    create mini-batches $B_0, \dots, B_{N/M-1}$;
    $z^0 \leftarrow w^k$;
    $d_{-1} \leftarrow 0$;
    $\alpha_{-1} \leftarrow \begin{cases} \frac{\alpha_0}{k+1} & \text{if SGD-Decreasing} \\ \alpha_0 & \text{otherwise} \end{cases}$;
    for $t = 0$ to $N/M - 1$ do
        get indices $i_t$ from $B_t$ then get the samples;
        $\nabla f_{i_t}(z^t) \leftarrow \sum_{j \in B_t} \nabla f_j(z^t)$;
        $d_t \leftarrow \begin{cases} -\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big) & \text{if SGD-, SGDM} \\ \text{momentum-correction}\big(\beta_0, \nabla f_{i_t}(z^t), d_{t-1}\big) & \text{if MSL-SGDM-C} \\ \text{momentum-restart}\big(\beta_0, \nabla f_{i_t}(z^t), d_{t-1}\big) & \text{if MSL-SGDM-R} \end{cases}$;
        if SGD-Armijo, MSL-SGDM-C/R then
            $\alpha \leftarrow \mathtt{reset}(\alpha_{t-1}, \alpha_0, M, N, t, a, \mathsf{opt})$;
            $\alpha_t \leftarrow \text{armijo-method}(z^t, d_t, \alpha)$;
        else
            $\alpha_t \leftarrow \alpha_{-1}$;
        end
        $z^{t+1} \leftarrow z^t + \alpha_t d_t$;
    end
    $w^{k+1} \leftarrow z^{N/M}$; $k \leftarrow k + 1$;
end
Output: $w^*$
```

5 Experiments and results discussion

To test the efficiency the algorithms, a benchmark of six datasets retrieved from LIBSVM is used, see table 1 on page 13 for details. Every dataset comes already pre-processed, with every sample scaled in range $[-1, 1]$ and the response variable in $\{-1, 1\}$; many features are categorical with values $0, 1, 2 \dots$, this implies that the dataset matrix can be stored in sparse format, so the SciPy CSR matrix format was used.

Compared to the available benchmark dataset, those chosen are not that large, the choice is due to the hardware available (Intel® Core™ i7, memory 16GB). As can be seen few dataset are unbalanced, this will affect the accuracy.

5.1 Solving the optimization problem

In order to solve the optimization problem, an initial guess w^0 for the model weights is given: we set a null *bias* and the other features are drawn from an uniform distribution in range $[-0.5, 0.5]$.

Then a *hyper-parameters tuning* is performed. We set fixed values for the λ regularization coefficient from (2), the ε tolerance from the stopping criterion (4), the initial momentum term β_0 and the aggressiveness of the Armijo condition γ to a small value. Regarding failures for exceeding epochs and iterations, k^* and q^* for both Armijo method and momentum correction were set. Follow the values

$\lambda = 0.5$	$\varepsilon = 10^{-3}$	$\beta_0 = 0.9$
$\gamma = 10^{-3}$	$k^* = 600$	$q^* = 100$

Moving to the other hyper-parameters, a *grid search*, without cross-validation but just using test data for comparison, is applied to find the best combination for each algorithm. The procedure confronts different combinations of the mini-batch size, the learning rate in the basic SGD version and the ones with line search, and in the latter are confronted also different values for the damping both in the Armijo method and momentum correction.

The mini-batch size grid depends on the dataset being considered. As said, the retrieved datasets vary in size, but the general rule is to stay around 100 iterations using values that are powers of 2, the grid is composed by at least two values. For the first five datasets, the size starts at 32. Follows the grids used

α_0	SGD-Fixed, SGDM	$1,\ 0.5,\ 0.1,\ 0.05,\ 0.01,\ 0.005$ and 0.001
α_0	SGD-Decreasing	$1,\ 0.8,\ 0.5,\ 0.1,\ 0.05,\ 0.01$ and 0.005
α_0	SGD-Armijo, MSL-SGDM-C/R	$1,\ 0.5,\ 0.1,\ 0.05,\ 0.01$ and 0.005
M	SGD-, SGDM-	depends on dataset
δ_a	SGD-Armijo, MSL-SGDM-C/R	$0.3,\ 0.5$ and 0.7
δ_m	MSL-SGDM-C	$0.3,\ 0.5$ and 0.7

where δ_a is the damping for the Armijo line search and δ_m for the momentum correction, the combinations for the MSL-SGDM-C are three times those of SGD-Armijo and MSL-SGDM-R.

As you can see, the upper-bound for the step-size is limited to 1, this because, as the grid search pointed out then, the algorithms perform better with a smaller step-size. Therefore, on each epoch the line search will spend time reducing the initial learning rate, while the basic versions, SGD-Fixed and SGDM, will end up far from the solution.

The grid search chooses the best combination for a certain solver based on the greatest *test accuracy* and lowest *objective function* value. The results can be seen in tables from 2 on page 14 to 7 ordered descending by accuracy on the test dataset and ascending by objective function on reached solution. The other displayed values are the number of epochs and the run-time, the solution norm and the gradient norm.

For benchmarking purposes the optimization problem is solved also using the `Full-batch Gradient Descent` and three solvers from `SciPy` which are `L-BFGS`, `Conjugate Gradient` and `Newton-CG`.

Regarding the grid search implementation, the `Joblib` module was used to parallelize on multiple cores the `for loop` needed to evaluate the algorithm with various combinations, see algorithm 6 on the next page.

First thing to say, the regularization term has a great influence on the final model, as you can see in the solution norm column the values are not that high, lowering the λ coefficient would lead to a lower objective function value, should be seen the change in the solution norm.

Speaking of the algorithms that use the Armijo line search, first of all one notices that their execution time is obviously longer than the others. More important, in each dataset you can see that the ε tolerance value is never reached though the limit of 600 epochs, especially the `SGD-Armijo` algorithm. However, in machine learning a very low tolerance is not necessarily required. Nonetheless the SLS solvers prove to achieve a higher accuracy, particularly the `SGD-Armijo`.

The full-batch method for these dataset sizes is still a valid choice and could reach also a lower tolerance value, most of all has the lower run-time among the SGD solvers, followed by the `SGD-Decreasing`. Same for the `scipy.optimize` solvers that can reach the smallest gradient norm. The `Mushrooms` dataset is an exception, where these solvers converge to the lowest solution norm with the lowest accuracy.

As expected the accuracy is greatly influenced by the class distribution of each datasets, for which different metrics that deal with unbalanced datasets could be used. Between al solvers the test score is very similar, as the $f(w^*)$ except in some cases for the `SGD-Armijo` solver that ended on a different solution with even higher accuracy.

5.2 Performance of the objective function

Now, as done by the authors of both articles, we want to show how the value of the objective function decreases with each epoch and also during time. For this purpose, another *grid search* is performed, again the fixed hyper-parameters are

$\lambda = 0.5$	$\varepsilon = 10^{-3}$	$\beta_0 = 0.9$
$\gamma = 10^{-3}$	$k^* = 200$	$q^* = 100$

What differs now is that the grid search finds the best hyper-parameters for each learning rate in 1, 0.1 and 0.01, hence the grids are

α_0	`SGD-`, `SGDM-`	1 or 0.1 or 0.01
M	`SGD-`, `SGDM-`	depends on dataset
δ_a	`SGD-Armijo`, `MSL-SGDM-C/R`	0.3, 0.5 and 0.7
δ_m	`MSL-SGDM-C`	0.3, 0.5 and 0.7

where the mini-batch size M as in the previous grid search is composed by at least two values.

The algorithm finds the best values for each α_0, then the objective function for every epoch and for the time took by each epoch is plotted in figures from 3 on page 16 to 5; for each plot, both axis are set to logarithmic scale with base 10.

Regarding the $f(w)$ against run-time plot, we saw that between certain epochs, the run-time can be the same or slightly different, so the $f(w)$ values were taken every 4 epochs, also to avoid problems due to the run-time sequence beginning at 0.0, the starting value was set to 10^{-3}.

Although the first two datasets are unbalanced, the performance of the objective function tends to the minimum without significant fluctuations. In figure 5b the `SGD-Armijo` shows important oscillations, that may also be due to small dataset size.

Generally seems that the line search methods can reach a neighbourhood of the solution after few epochs, while the others' performance strongly depends on α_0. Regarding the run-time there are slightly more evident differences, one could then be more concerned with this, but priority is on accuracy.

In figure 4 we note that the `SGD-Armijo` reaches a low objective function value quickly but struggles to stay around and terminate, unlike the momentum versions that may take more epochs but oscillate less. Perhaps a different line search like a Wolfe-type may reduce this behaviour.

As pointed out by the authors, an important conclusion is that the methods with line search have a low sensitivity to the initial learning rate, thus having a similar performance. In contrast to the basic versions where significant differences in the trend can be seen, for example in figure 5a the `SGD-Fixed` and `SGDM` have very different behaviours for the each value in the α_0 grid, note that the momentum term tries to correct this abnormal behaviour.

Among the basic versions, the `SGD-Decreasing` seems to be the least affected by different initial learning rates. However if α_0 is too small, the algorithm takes smaller steps and requires more epochs to reach a solution. Same thing with `SGD-Fixed`, seems that in range $\alpha_0 \in [0.01, 0.1]$ the time required to get in a neighbourhood of w^* increases significantly, so adapting the step-size is a good solution.

Algorithm 6: Grid search for hyper-parameters tuning

```
   Input: $\alpha_0$, $M$, $\delta_a$, $\delta_m$
 1 prepare the combinations using the given grids, store in params list;
 2 store evaluations in performance list, whose elements are (param, metrics);
 3 for param in params do
 4     model training using param (training data);
 5     model testing and metrics computation (test data);
 6     add current param and metrics to performance;
 7 end
 8 order performance list by metrics;
   Output: best-model, best-param
```

Table 1: Benchmark datasets

Name	Train	Test	Features	Distribution	M grid
w1a	2477	47 272	300	-1:0.97 1:0.03	32, 64
w3a	4912	44 837	300	-1:0.97 1:0.03	64, 128
Phishing	8844	2211	68	-1:0.45 1:0.55	64, 128
a2a	2265	30 296	120	-1:0.75 1:0.25	32, 64
Mushrooms	6499	1625	112	-1:0.48 1:0.52	64, 128
a4a	4781	27 780	120	-1:0.75 1:0.25	64, 128

Table 2: w1a dataset

Solver	Epochs	Run-time	$\|w^*\|$	$f(w^*)$	$\|\nabla f(w^*)\|$	Test score
Newton-CG	6	NaN	0.667 394	0.464 614	4.60×10^{-5}	0.970 236
CG	7	NaN	0.667 395	0.464 614	9.00×10^{-6}	0.970 236
L-BFGS-B	7	NaN	0.667 406	0.464 614	2.30×10^{-5}	0.970 236
BatchGD-Fixed	12	0.0000	0.667 389	0.464 614	5.64×10^{-4}	0.970 236
SGD-Decreasing	14	0.2471	0.667 605	0.464 614	8.90×10^{-4}	0.970 236
MSL-SGDM-C	44	1.3954	0.667 133	0.464 615	8.52×10^{-4}	0.970 236
SGD-Armijo	42	0.6816	0.667 250	0.464 615	8.57×10^{-4}	0.970 236
SGD-Fixed	27	0.2742	0.667 293	0.464 615	8.79×10^{-4}	0.970 236
SGDM	108	1.8930	0.666 975	0.464 615	9.73×10^{-4}	0.970 236
MSL-SGDM-R	183	2.5609	0.667 560	0.464 615	9.61×10^{-4}	0.970 236

Table 3: w3a dataset

Solver	Epochs	Run-time	$\|w^*\|$	$f(w^*)$	$\|\nabla f(w^*)\|$	Test score
Newton-CG	6	NaN	0.666 640	0.462 742	1.10×10^{-5}	0.970 203
CG	7	NaN	0.666 648	0.462 742	2.20×10^{-5}	0.970 203
L-BFGS-B	7	NaN	0.666 658	0.462 742	3.30×10^{-5}	0.970 203
BatchGD-Fixed	12	0.0102	0.666 635	0.462 742	5.64×10^{-4}	0.970 203
SGD-Armijo	21	0.5118	0.666 633	0.462 743	7.76×10^{-4}	0.970 203
SGD-Fixed	27	0.2009	0.666 559	0.462 743	8.26×10^{-4}	0.970 203
MSL-SGDM-C	26	1.1672	0.666 495	0.462 743	8.36×10^{-4}	0.970 203
SGDM	61	1.0608	0.666 868	0.462 743	8.61×10^{-4}	0.970 203
SGD-Decreasing	15	0.2744	0.666 746	0.462 743	8.58×10^{-4}	0.970 203
MSL-SGDM-R	172	3.3423	0.666 902	0.462 743	9.83×10^{-4}	0.970 203

Table 4: Phishing dataset

Solver	Epochs	Run-time	$\|w^*\|$	$f(w^*)$	$\|\nabla f(w^*)\|$	Test score
SGD-Fixed	600	12.3473	0.149 831	0.687 962	7.06×10^{-2}	0.903 664
SGD-Armijo	600	33.7333	0.149 831	0.687 962	7.06×10^{-2}	0.903 664
MSL-SGDM-C	600	17.2734	0.145 904	0.686 425	4.91×10^{-2}	0.699 683
Newton-CG	5	NaN	0.164 188	0.685 065	0.00	0.567 616
L-BFGS-B	5	NaN	0.164 196	0.685 065	8.00×10^{-6}	0.567 616
CG	6	NaN	0.164 214	0.685 065	2.30×10^{-5}	0.567 616
BatchGD-Fixed	11	0.0105	0.164 001	0.685 065	5.34×10^{-4}	0.567 616
SGD-Decreasing	39	1.3267	0.163 747	0.685 065	5.65×10^{-4}	0.567 616
MSL-SGDM-R	48	2.3211	0.164 120	0.685 065	5.78×10^{-4}	0.567 616
SGDM	49	1.6438	0.163 887	0.685 065	6.49×10^{-4}	0.567 616

Table 5: a2a dataset

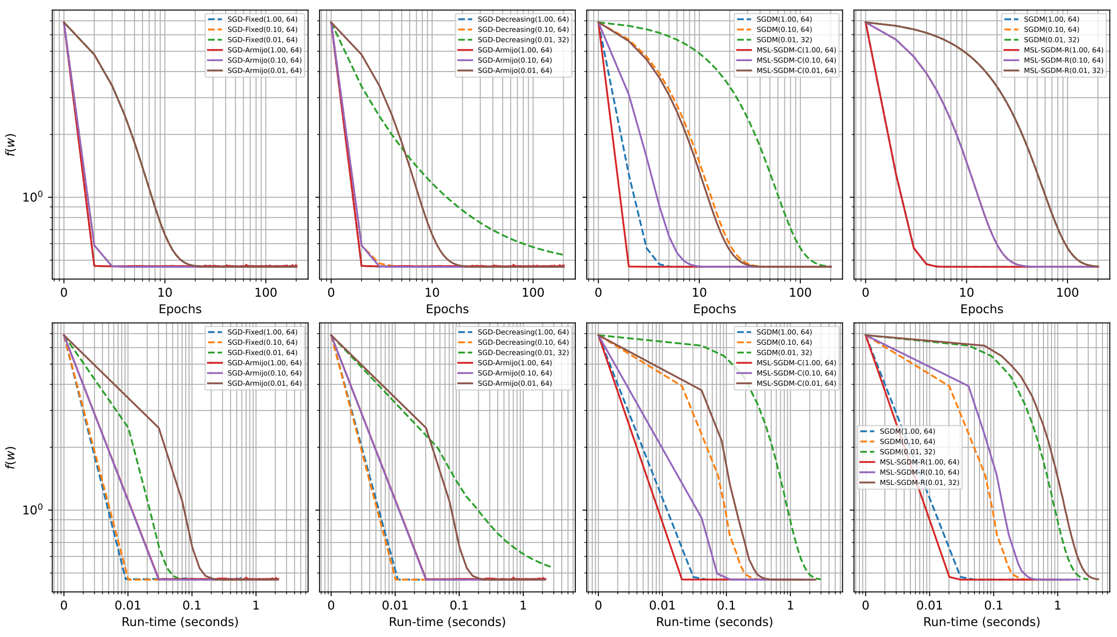
Solver	Epochs	Run-time	$\|w^*\|$	$f(w^*)$	$\|\nabla f(w^*)\|$	Test score
BatchGD-Fixed	600	0.1239	0.355 752	0.594 416	3.64×10^{-1}	0.822 386
MSL-SGDM-C	600	7.8410	0.395 984	0.577 631	2.25×10^{-1}	0.781 291
SGD-Armijo	600	7.4286	0.416 947	0.570 790	1.43×10^{-1}	0.766 075
SGD-Fixed	600	4.3739	0.413 794	0.569 481	1.34×10^{-1}	0.765 184
MSL-SGDM-R	600	16.4529	0.410 532	0.566 502	9.42×10^{-2}	0.762 015
SGDM	600	4.7068	0.428 972	0.564 436	3.86×10^{-2}	0.760 430
Newton-CG	5	NaN	0.438 972	0.564 027	4.00×10^{-6}	0.760 265
CG	12	NaN	0.438 961	0.564 027	1.50×10^{-5}	0.760 265
L-BFGS-B	8	NaN	0.438 969	0.564 027	1.20×10^{-5}	0.760 265
SGD-Decreasing	31	0.2414	0.439 216	0.564 027	9.46×10^{-4}	0.760 265

Table 6: Mushrooms dataset

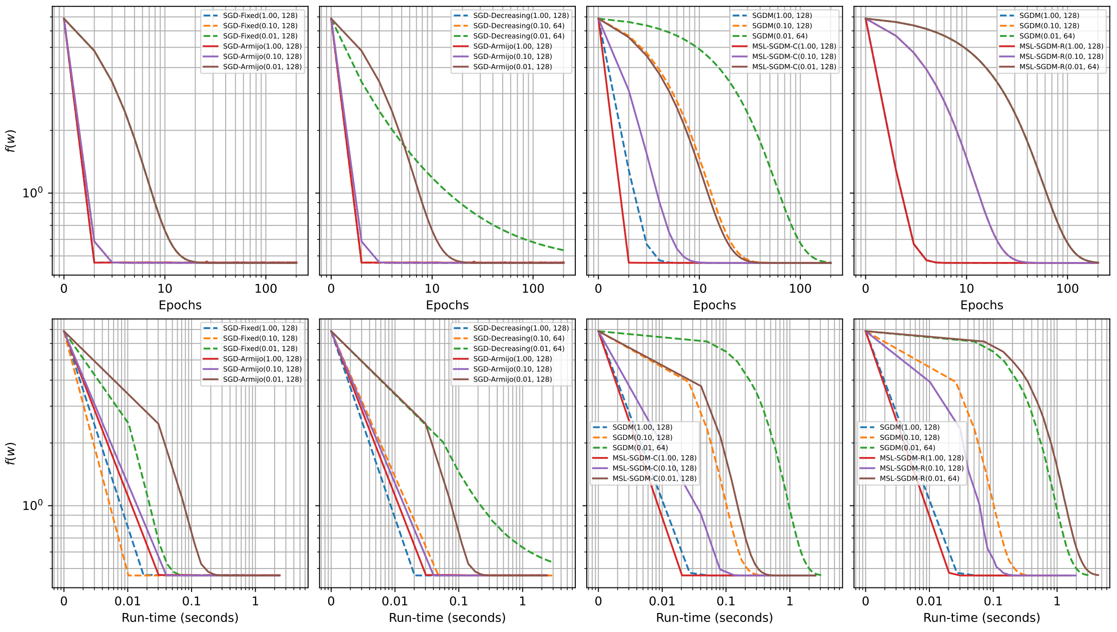
Solver	Epochs	Run-time	$\|w^*\|$	$f(w^*)$	$\|\nabla f(w^*)\|$	Test score
MSL-SGDM-C	600	13.3851	0.631 125	0.541 510	3.66×10^{-1}	0.953 846
SGD-Armijo	600	28.8739	0.638 957	0.530 812	2.45×10^{-1}	0.951 385
SGD-Fixed	600	6.7377	0.638 411	0.536 690	3.23×10^{-1}	0.951 385
SGDM	600	14.7508	0.632 318	0.523 624	1.86×10^{-1}	0.940 308
MSL-SGDM-R	600	28.0490	0.636 756	0.522 395	1.60×10^{-1}	0.926 154
SGD-Decreasing	42	1.0115	0.635 971	0.517 727	8.29×10^{-4}	0.893 538
Newton-CG	7	NaN	0.635 933	0.517 726	3.00×10^{-6}	0.892 923
CG	11	NaN	0.635 939	0.517 726	2.40×10^{-5}	0.892 923
L-BFGS-B	10	NaN	0.635 930	0.517 726	1.70×10^{-5}	0.892 923
BatchGD-Fixed	22	0.0206	0.635 907	0.517 727	7.15×10^{-4}	0.892 923

Table 7: a4a dataset

Solver	Epochs	Run-time	$\|w^*\|$	$f(w^*)$	$\|\nabla f(w^*)\|$	Test score
BatchGD-Fixed	600	0.2072	0.376 931	0.579 971	3.00×10^{-1}	0.809 575
SGDM	600	10.3513	0.394 249	0.566 575	1.78×10^{-1}	0.772 858
SGD-Armijo	600	15.6058	0.380 212	0.574 724	1.88×10^{-1}	0.761 915
Newton-CG	5	NaN	0.450 068	0.558 973	3.00×10^{-6}	0.760 727
L-BFGS-B	8	NaN	0.450 067	0.558 973	1.10×10^{-5}	0.760 727
CG	12	NaN	0.450 062	0.558 973	2.00×10^{-5}	0.760 727
SGD-Decreasing	17	0.2943	0.449 935	0.558 973	5.93×10^{-4}	0.760 727
SGD-Fixed	55	0.3151	0.449 949	0.558 974	4.67×10^{-4}	0.760 727
MSL-SGDM-C	41	1.0661	0.450 326	0.558 974	6.33×10^{-4}	0.760 727
MSL-SGDM-R	399	6.7421	0.450 646	0.558 974	9.62×10^{-4}	0.760 727

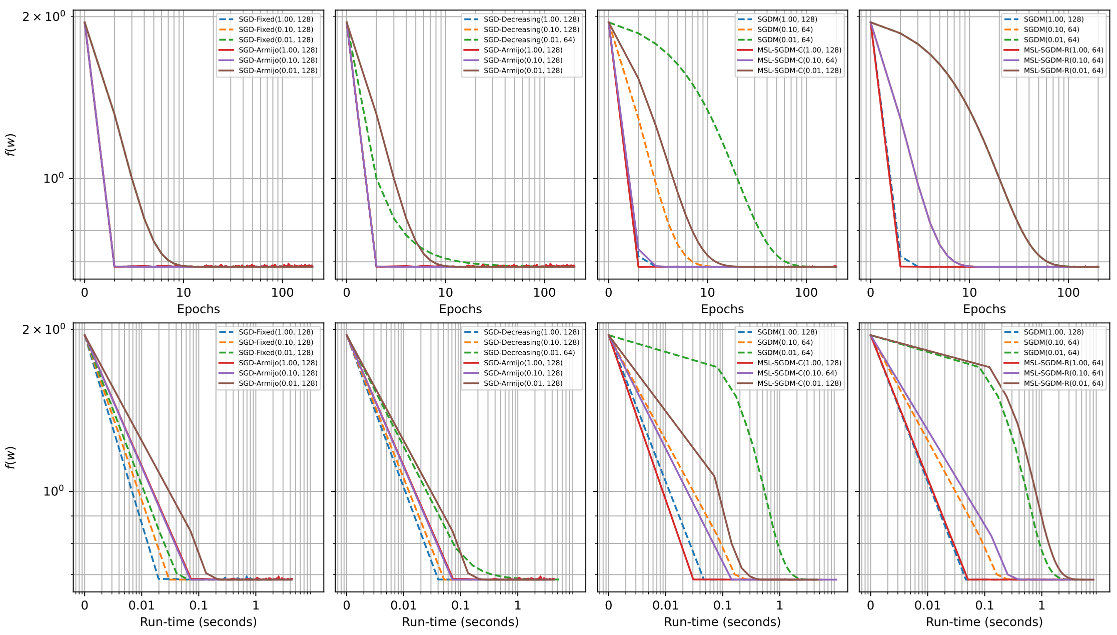


(a) *w1a dataset*

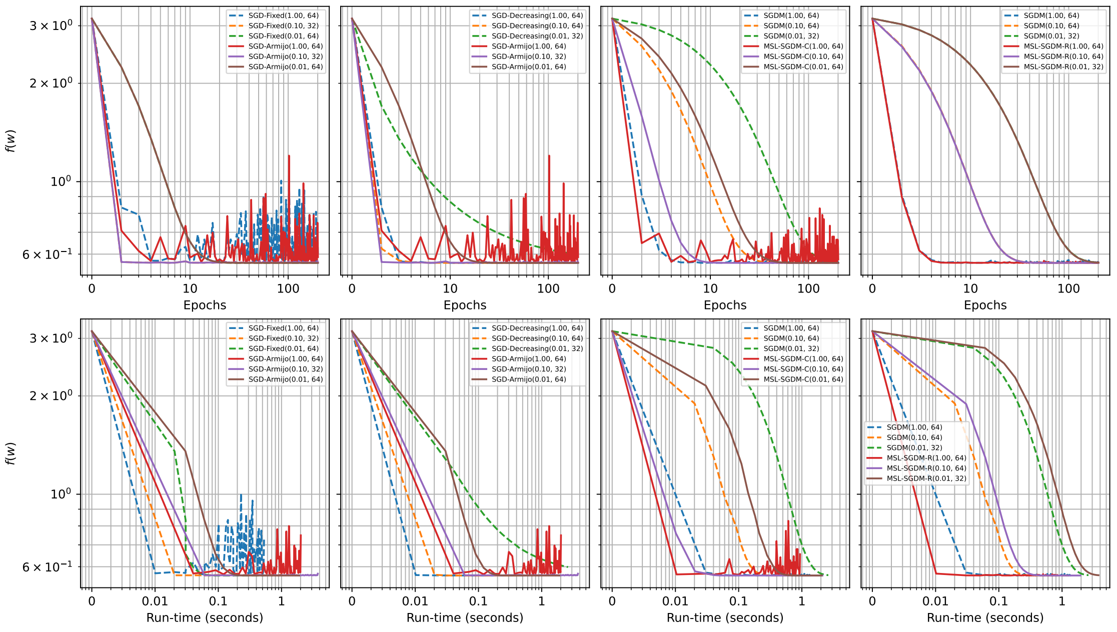


(b) *w3a dataset*

Figure 3: w1a and w3a datasets

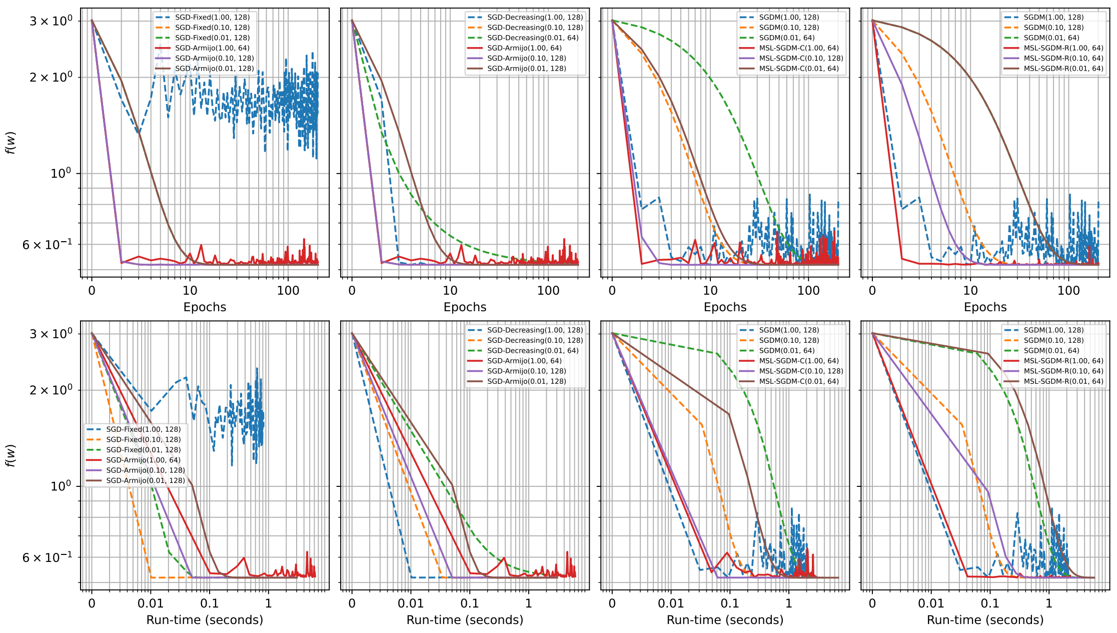


(a) *Phishing dataset*

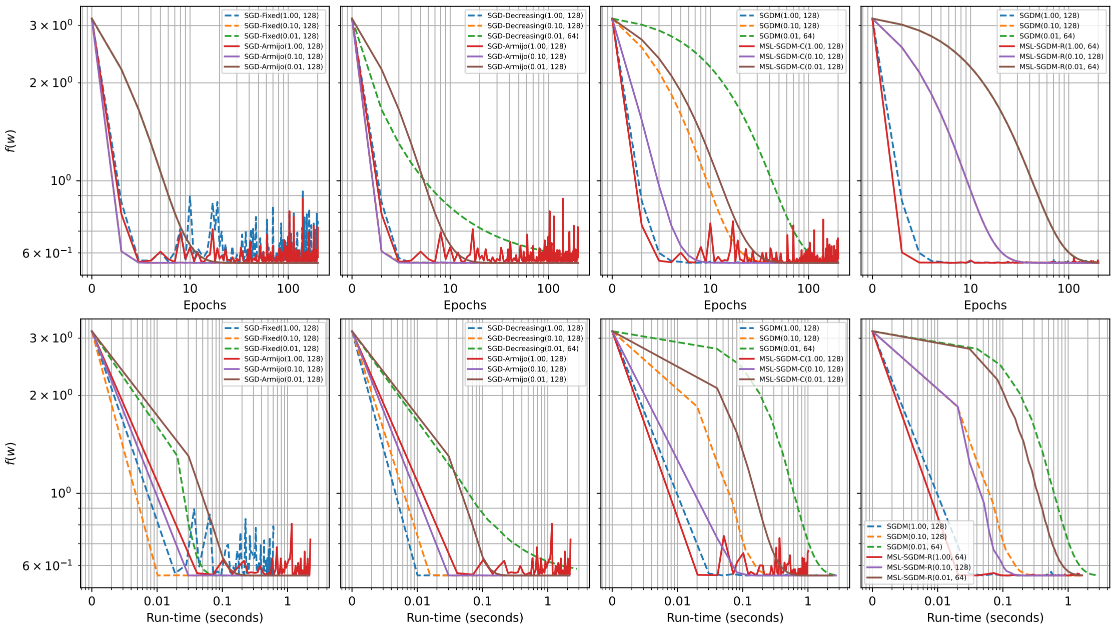


(b) *a2a dataset*

Figure 4: Phishing and a2a datasets



(a) *Mushrooms dataset*



(b) *a4a dataset*

Figure 5: Mushrooms and a4a 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, 8).

[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, 9).