# Stochastic Gradient Descent with Momentum and Line Searches

David Nardi
MSc student in Artificial Intelligence
Univeristy of Florence

2nd 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 Introduction

Different SGD-type algorithms proposed by the literature were implemented and tested on different benchmark datasets for training the $\ell_2$-regularized Logistic Regression model.

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 line search, 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 10 describes the experiments performed for showing the performance of the algorithms on different datasets.

## 1.1 Classification task

Given a dataset as follows

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

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 $\lambda$. There are three regularization possible choices, the $\ell_2$ 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

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

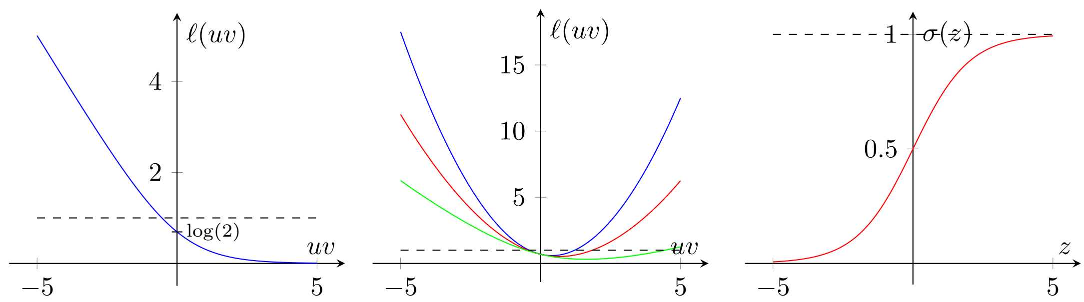
figure 1a on the next page shows a plot of the loss function $\ell(uv) = \log\bigl(1 + \exp(-uv)\bigr)$ where $u = y^{(i)}$ and $v = w^T x^{(i)}$.

### Prediction

The sigmoid function, see figure 1c on the following page, is used for predicting the labels (positive or negative class) of unseen samples as follows

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

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; 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 the Stochastic Gradient Descent (SGD) algorithm variants

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

where $i = 1, \ldots, N$ 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

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

the constant column is meant for the intercept, known as *bias* the $b$ weight in vector $w$. The dataset matrix can be written as $X = (x^{(1)}, x^{(2)}, \ldots, x^{(N)})$.

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

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

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

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

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*, 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.

## 2 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 gradient samples 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 $w^0 \in \mathbb{R}^{(p+1)}$.

### 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.

Other than the stopping criterion, we 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 $\alpha$ in the Armijo method as well as the momentum term $\beta$ in the momentum correction).

### Mini-batch gradient

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

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

the update uses information from the mini-batch $B_t$ when computing the direction $d_t$ and the step-size $\alpha_t$ follows a certain rule.*

The direction involves the gradient, so we want to compute the gradient w.r.t. $z^t$ using the mini-batch $B_t$ whose indices are randomly chosen from the full dataset $i_t \subset \{1, \dots, N\}$

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

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 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.

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

- `SGD-Fixed`: constant step-size s.t. $\alpha_t = \alpha_0$;

* *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.

- SGD-Decreasing: decreasing step-size s.t. $\alpha_t = \frac{\alpha_0}{k+1}$, see figure 1.

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

#### 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 evaluated on 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

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

the coefficient $\gamma$ 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 $\delta$ 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 8. Same as the basic versione, the momentum term is set to $\beta_0 = 0$.

## 2.2 Adding momentum term

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

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

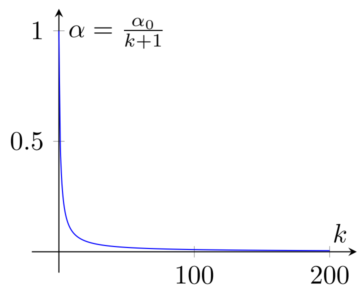
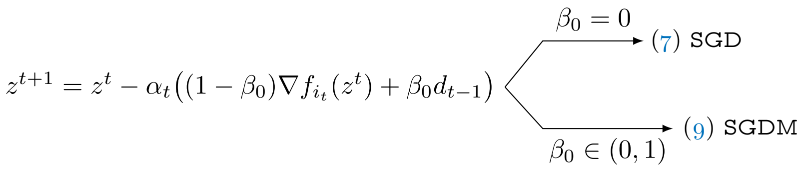



**Figure 1:** Step-size schedule for the SGD-Decreasing algorithm, starting with $\alpha_0 = 1$

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



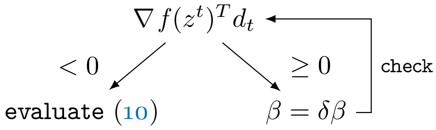

#### 2.2.1 Stochastic line search

As the paper by Fan, Vaswani, Thrampoulidis *et al.* [2] says, when using the momentum term together with a line search, $\beta_0$ complicates the selection of a suitable step-size. The Armijo line search applied to the SGDM algorithm has the following condition

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

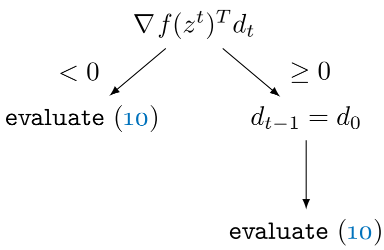
but using only the line search is not robust to the choice of the momentum term as the paper stated.

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




in algorithmic terms, until the direction is descent, damp the momentum term by a factor $\delta$, 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.

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



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. The algorithm that uses this procedure is MSL-SGDM-R.

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

**Algorithm 1:** reset

**Input:** $\alpha$, $\alpha_0$, $M$, $N$, $t$, $a \in \mathbb{R}^+$, $\texttt{opt} \in \{0,1,2\}$

- **if** $t = 0$ **or** $\texttt{opt} = 1$ **then**
  - **return** $\alpha_0$
- **else if** $\texttt{opt} = 0$ **then**
  - $\alpha \leftarrow \alpha$
- **else if** $\texttt{opt} = 2$ **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** $f_{i_t}(z^{t+1}) \leq f_{i_t}(z^t) + \gamma\alpha\nabla f_{i_t}(z^t)^T d_t$ **or** $q \geq q^*$;

**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** $q \geq q^*$;

**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**
  - $d_{t-1} \leftarrow d_0$;
- **end**

**Output:** $d_t$

**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 $B_0, \ldots, B_{N/M-1}$;
  $z^0 \leftarrow w^k$;
  $d_{-1} \leftarrow 0$;
  $\alpha_{-1} \leftarrow \begin{cases} \frac{\alpha_0}{k+1} & \text{if SGD-Decreasing} \\ \alpha_0 & \text{otherwise} \end{cases}$;
  **for** $t = 0$ **to** $N/M - 1$ **do**
    get indices $i_t$ from $B_t$ then get the samples;
    $\nabla f_{i_t}(z^t) \leftarrow \sum_{j \in B_t} \nabla f_j(z^t)$;
    $d_t \leftarrow \begin{cases} -\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big) & \text{if SGD-, SGDM} \\ \texttt{momentum-correction}\big(\beta_0, \nabla f_{i_t}(z^t), d_{t-1}\big) & \text{if MSL-SGDM-C} \\ \texttt{momentum-restart}\big(\beta_0, \nabla f_{i_t}(z^t), d_{t-1}\big) & \text{if MSL-SGDM-R} \end{cases}$;
    **if** *SGD-Armijo, MSL-SGDM-C/R* **then**
      $\alpha \leftarrow \texttt{reset}(\alpha_{t-1}, \alpha_0, M, N, t, a, \texttt{opt})$;
      $\alpha_t \leftarrow \texttt{armijo-method}(z^t, d_t, \alpha)$;
    **end**
    $z^{t+1} \leftarrow z^t + \alpha_t d_t$;
  **end**
  $w^{k+1} \leftarrow z^{N/M}$;
  $k \leftarrow k + 1$;
**end**

# 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 page 12 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.

## 3.1 Solving the optimization problem

In order to solve the optimization problem, an initial guess for the model parameters is given: we set a null *bias* and the other features in range $[-0.5, 0.5]$. Then a *hyper-parameters tuning* is performed. We set fixed values for the $\lambda$ regularization coefficient from (2), the $\varepsilon$ tolerance from the stopping criterion (4), the initial momentum term $\beta_0$ and the aggressiveness of the Armijo condition $\gamma$ to a small value. Regarding failures for exceeding epochs and iterations, $k^*$ and $q^*$ for both Armijo method and momentum correction were set. Follows 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* 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 rule is to stay around the 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

| | | |
|---|---|---|
| $M$ | SGD-, SGDM- | depends on dataset |
| $\alpha_0$ | SGD-Fixed, SGDM | $1,\ 0.5,\ 0.1,\ 0.01,\ 0.001$ and $0.0005$ |
| $\alpha_0$ | SGD-Decreasing | $1,\ 0.8,\ 0.5,\ 0.1,\ 0.05,\ 0.01$ and $0.005$ |
| $\alpha_0$ | SGD-Armijo, MSL-SGDM-C/R | $1,\ 0.5,\ 0.1,\ 0.05,\ 0.01$ and $0.005$ |
| $\delta_a$ | SGD-Armijo, MSL-SGDM-C/R | $0.3,\ 0.5,\ 0.7$ and $0.9$ |
| $\delta_m$ | MSL-SGDM-C | $0.3,\ 0.5$ and $0.7$ |

where $\delta_a$ is the damping for the Armijo line search and $\delta_m$ for the momentum correction, the combinations for the MSL-SGDM-C are twice those of SGD-Armijo and MSL-SGDM-R.

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 2 on page 13 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 a benchmarking purpose 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, however lowering the $\lambda$ coefficient would have led to a smaller solution norm, so the null model.

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 $\varepsilon$ 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.

The full-batch method for these dataset sizes is still a valid choice and could reach also a lower tolerance value, same for the `SciPy` solvers that reach the smallest gradient norm.

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.

## 3.2 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 during time. For this purpose, another grid search is performed, the fixed hyper-parameters are again

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

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

| | | |
|---|---|---|
| $\alpha_0$ | `SGD-`, `SGDM-` | 1 or 0.1 or 0.01 |
| $M$ | `SGD-`, `SGDM-` | depends on dataset |
| $\delta_a$ | `SGD-Armijo`, `MSL-SGDM-C/R` | 0.3, 0.5, 0.7 and 0.9 |
| $\delta_m$ | `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 $\alpha_0$, then objective function for every epoch and for the time took by each epoch is plotted in figures from 2 on page 15 to 4.

Although the first two datasets are unbalanced, the performance of the objective function tends to the minimum without significant fluctuations. In figure 4b the `SGD-Armijo` shows important oscillations, that may also be due to small dataset size.

In figure 3 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 4a the `SGD-Fixed` and `SGDM` have very different behaviours for the values in the $\alpha$ grid, while the momentum term tries to correct this abnormal behaviour.

Among the basic versions, the `SGD-Decreasing` is the least affected by different initial learning rates. However if $\alpha_0$ is too small, the algorithm takes smaller steps and requires more epochs to reach a solution.

**Algorithm 6:** Grid search for hyper-parameters tuning

**Data:** $\alpha_0$, $M$, $\delta_a$, $\delta_m$

prepare the hyper-parameters combinations using the given grids;

**for** *param in params* **do**

  model training with `param`;

  confront metrics with current `best-model` and update `best-param` if necessary;

**end**

**Output:** `best-model`, `best-param`

**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 | Epochs | Run-time | $\|w^*\|$ | $f(w^*)$ | $\|\nabla f(w^*)\|$ | Test score |
|---|---|---|---|---|---|---|
| SGD-Armijo | 600 | 3.2954 | 0.455 603 | 0.536 138 | $3.63 \times 10^{-1}$ | 0.971 400 |
| Newton-CG | 6 | NaN | 0.667 394 | 0.464 614 | $4.60 \times 10^{-5}$ | 0.970 236 |
| CG | 7 | NaN | 0.667 395 | 0.464 614 | $9.00 \times 10^{-6}$ | 0.970 236 |
| L-BFGS-B | 7 | NaN | 0.667 406 | 0.464 614 | $2.30 \times 10^{-5}$ | 0.970 236 |
| BatchGD-Fixed | 12 | 0.0060 | 0.667 389 | 0.464 614 | $5.64 \times 10^{-4}$ | 0.970 236 |
| SGD-Decreasing | 27 | 0.0895 | 0.667 400 | 0.464 614 | $7.92 \times 10^{-4}$ | 0.970 236 |
| SGD-Fixed | 27 | 0.1787 | 0.667 311 | 0.464 615 | $8.52 \times 10^{-4}$ | 0.970 236 |
| SGDM | 386 | 2.9202 | 0.667 383 | 0.464 615 | $9.78 \times 10^{-4}$ | 0.970 236 |
| MSL-SGDM-R | 600 | 3.1091 | 0.665 659 | 0.464 693 | $9.14 \times 10^{-3}$ | 0.970 236 |
| MSL-SGDM-C | 600 | 3.0976 | 0.665 656 | 0.464 693 | $9.15 \times 10^{-3}$ | 0.970 236 |

**Table 3:** w3a dataset

| Solver | Epochs | Run-time | $\|w^*\|$ | $f(w^*)$ | $\|\nabla f(w^*)\|$ | Test score |
|---|---|---|---|---|---|---|
| SGD-Armijo | 600 | 8.0974 | 0.478 986 | 0.500 431 | $2.68 \times 10^{-1}$ | 0.971 006 |
| Newton-CG | 6 | NaN | 0.666 640 | 0.462 742 | $1.10 \times 10^{-5}$ | 0.970 203 |
| CG | 7 | NaN | 0.666 648 | 0.462 742 | $2.20 \times 10^{-5}$ | 0.970 203 |
| L-BFGS-B | 7 | NaN | 0.666 658 | 0.462 742 | $3.30 \times 10^{-5}$ | 0.970 203 |
| BatchGD-Fixed | 12 | 0.0241 | 0.666 635 | 0.462 742 | $5.64 \times 10^{-4}$ | 0.970 203 |
| SGD-Decreasing | 19 | 0.1657 | 0.666 731 | 0.462 743 | $8.76 \times 10^{-4}$ | 0.970 203 |
| SGD-Fixed | 23 | 0.1549 | 0.666 893 | 0.462 743 | $9.49 \times 10^{-4}$ | 0.970 203 |
| SGDM | 45 | 0.4911 | 0.666 594 | 0.462 743 | $8.95 \times 10^{-4}$ | 0.970 203 |
| MSL-SGDM-C | 600 | 3.5907 | 0.667 576 | 0.462 787 | $6.92 \times 10^{-3}$ | 0.970 203 |
| MSL-SGDM-R | 600 | 3.5150 | 0.667 576 | 0.462 787 | $6.92 \times 10^{-3}$ | 0.970 203 |

**Table 4:** Phishing dataset

| Solver | Epochs | Run-time | $\|w^*\|$ | $f(w^*)$ | $\|\nabla f(w^*)\|$ | Test score |
|---|---|---|---|---|---|---|
| SGD-Armijo | 600 | 7.2636 | 0.144 155 | 0.687 736 | $6.65 \times 10^{-2}$ | 0.865 219 |
| MSL-SGDM-C | 600 | 9.4775 | 0.150 617 | 0.686 392 | $4.88 \times 10^{-2}$ | 0.709 181 |
| MSL-SGDM-R | 600 | 17.7458 | 0.152 190 | 0.685 660 | $3.26 \times 10^{-2}$ | 0.568 521 |
| 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 | $8.00 \times 10^{-6}$ | 0.567 616 |
| CG | 6 | NaN | 0.164 214 | 0.685 065 | $2.30 \times 10^{-5}$ | 0.567 616 |
| SGD-Decreasing | 6 | 0.0407 | 0.164 270 | 0.685 065 | $5.08 \times 10^{-4}$ | 0.567 616 |
| SGDM | 22 | 0.2663 | 0.163 872 | 0.685 065 | $5.75 \times 10^{-4}$ | 0.567 616 |
| BatchGD-Fixed | 11 | 0.0448 | 0.164 001 | 0.685 065 | $5.34 \times 10^{-4}$ | 0.567 616 |
| SGD-Fixed | 13 | 0.1701 | 0.163 727 | 0.685 065 | $9.27 \times 10^{-4}$ | 0.567 616 |

**Table 5:** a2a dataset

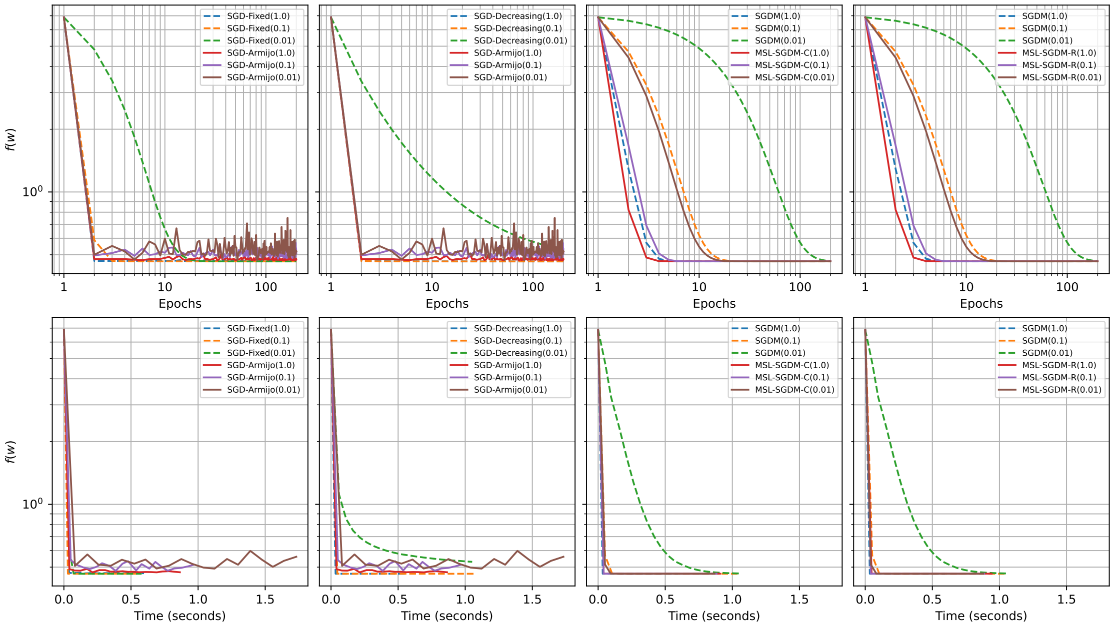
| Solver | Epochs | Run-time | $\|w^*\|$ | $f(w^*)$ | $\|\nabla f(w^*)\|$ | Test score |
|---|---|---|---|---|---|---|
| SGD-Armijo | 600 | 3.9613 | 0.344 862 | 0.603 503 | $3.97 \times 10^{-1}$ | 0.825 191 |
| BatchGD-Fixed | 600 | 0.2068 | 0.355 752 | 0.594 416 | $3.64 \times 10^{-1}$ | 0.822 386 |
| MSL-SGDM-C | 600 | 7.0367 | 0.406 144 | 0.585 491 | $2.90 \times 10^{-1}$ | 0.810 569 |
| SGD-Fixed | 600 | 3.5597 | 0.425 177 | 0.602 741 | $3.56 \times 10^{-1}$ | 0.807 136 |
| MSL-SGDM-R | 600 | 7.6314 | 0.413 366 | 0.577 575 | $2.28 \times 10^{-1}$ | 0.790 236 |
| SGDM | 600 | 1.8107 | 0.438 444 | 0.564 030 | $2.63 \times 10^{-3}$ | 0.760 298 |
| Newton-CG | 5 | NaN | 0.438 972 | 0.564 027 | $4.00 \times 10^{-6}$ | 0.760 265 |
| CG | 12 | NaN | 0.438 961 | 0.564 027 | $1.50 \times 10^{-5}$ | 0.760 265 |
| L-BFGS-B | 8 | NaN | 0.438 969 | 0.564 027 | $1.20 \times 10^{-5}$ | 0.760 265 |
| SGD-Decreasing | 59 | 0.1687 | 0.438 522 | 0.564 028 | $7.26 \times 10^{-4}$ | 0.760 265 |

**Table 6:** Mushrooms dataset

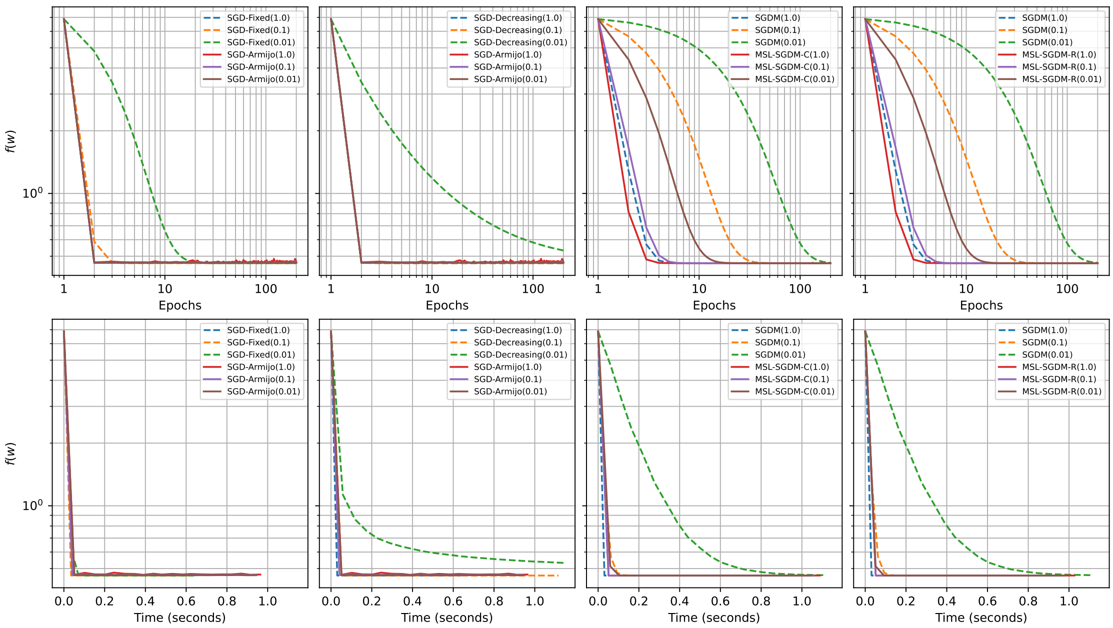
| Solver | Epochs | Run-time | $\|w^*\|$ | $f(w^*)$ | $\|\nabla f(w^*)\|$ | Test score |
|---|---|---|---|---|---|---|
| SGD-Armijo | 600 | 11.4039 | 0.644 938 | 0.535 765 | $2.34 \times 10^{-1}$ | 0.953 231 |
| MSL-SGDM-C | 600 | 6.7131 | 0.652 698 | 0.529 380 | $2.50 \times 10^{-1}$ | 0.942 769 |
| MSL-SGDM-R | 600 | 6.4450 | 0.642 551 | 0.527 069 | $2.32 \times 10^{-1}$ | 0.940 308 |
| SGD-Fixed | 600 | 3.3183 | 0.646 431 | 0.525 499 | $2.00 \times 10^{-1}$ | 0.926 154 |
| SGDM | 600 | 3.6847 | 0.658 660 | 0.557 069 | $4.79 \times 10^{-1}$ | 0.924 308 |
| SGD-Decreasing | 26 | 0.2326 | 0.635 898 | 0.517 727 | $7.79 \times 10^{-4}$ | 0.893 538 |
| Newton-CG | 7 | NaN | 0.635 933 | 0.517 726 | $3.00 \times 10^{-6}$ | 0.892 923 |
| CG | 11 | NaN | 0.635 939 | 0.517 726 | $2.40 \times 10^{-5}$ | 0.892 923 |
| L-BFGS-B | 10 | NaN | 0.635 930 | 0.517 726 | $1.70 \times 10^{-5}$ | 0.892 923 |
| BatchGD-Fixed | 26 | 0.0433 | 0.635 906 | 0.517 727 | $7.57 \times 10^{-4}$ | 0.892 923 |

**Table 7:** German dataset

| Solver | Epochs | Run-time | $\|w^*\|$ | $f(w^*)$ | $\|\nabla f(w^*)\|$ | Test score |
|---|---|---|---|---|---|---|
| MSL-SGDM-C | 600 | 4.9993 | 0.365 021 | 0.629 212 | $3.65 \times 10^{-1}$ | 0.755 000 |
| SGD-Armijo | 600 | 5.0955 | 0.423 356 | 0.632 348 | $3.63 \times 10^{-1}$ | 0.750 000 |
| SGDM | 600 | 2.3890 | 0.313 661 | 0.616 375 | $3.14 \times 10^{-1}$ | 0.745 000 |
| MSL-SGDM-R | 600 | 5.6525 | 0.373 225 | 0.599 779 | $7.32 \times 10^{-2}$ | 0.730 000 |
| SGD-Decreasing | 600 | 2.2918 | 0.348 660 | 0.607 993 | $1.14 \times 10^{-1}$ | 0.720 000 |
| Newton-CG | 5 | NaN | 0.358 504 | 0.597 303 | $1.00 \times 10^{-5}$ | 0.710 000 |
| CG | 12 | NaN | 0.358 506 | 0.597 303 | $4.00 \times 10^{-6}$ | 0.710 000 |
| L-BFGS-B | 7 | NaN | 0.358 506 | 0.597 303 | $1.40 \times 10^{-5}$ | 0.710 000 |
| SGD-Fixed | 58 | 0.2345 | 0.358 265 | 0.597 303 | $6.75 \times 10^{-4}$ | 0.710 000 |
| BatchGD-Fixed | 20 | 0.0050 | 0.358 324 | 0.597 303 | $8.82 \times 10^{-4}$ | 0.710 000 |

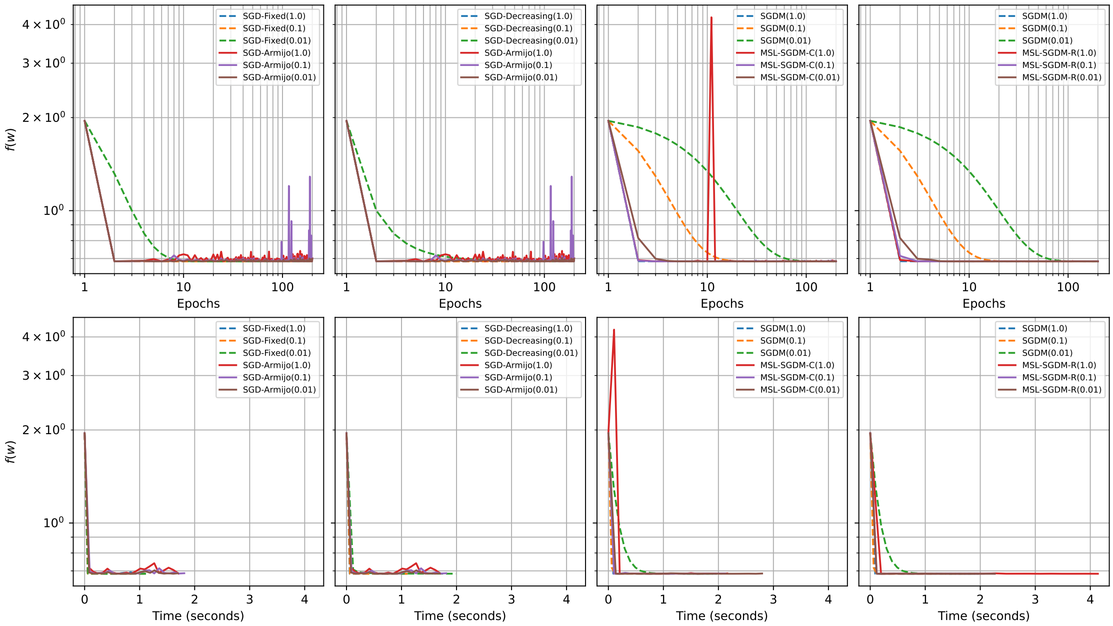



**(a)** *w1a dataset*

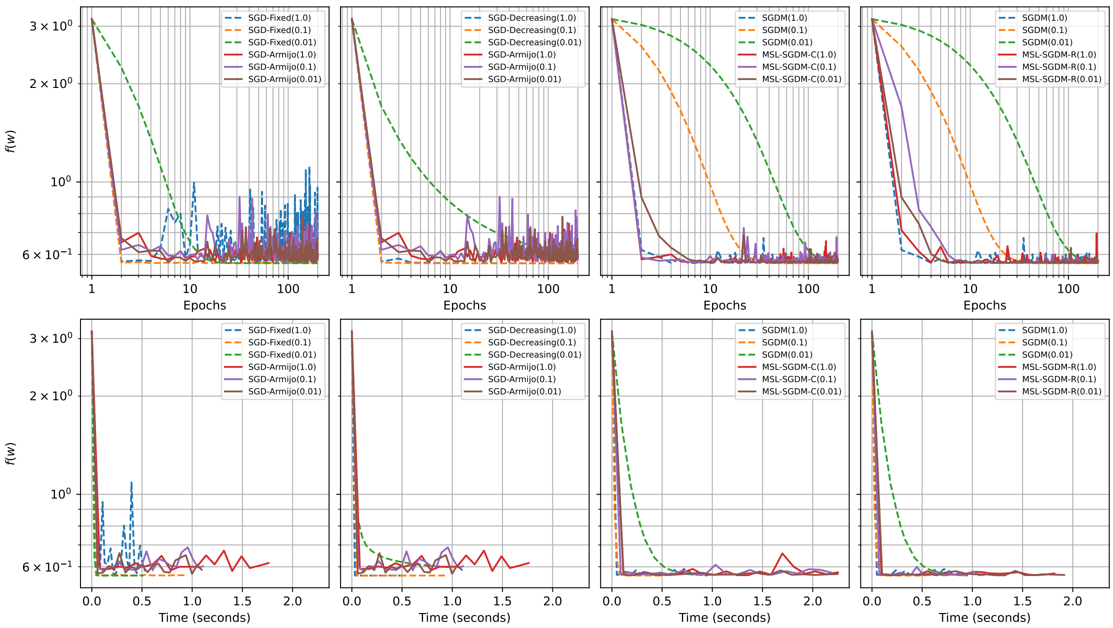



**(b)** *w3a dataset*

**Figure 2:** w1a and w3a datasets

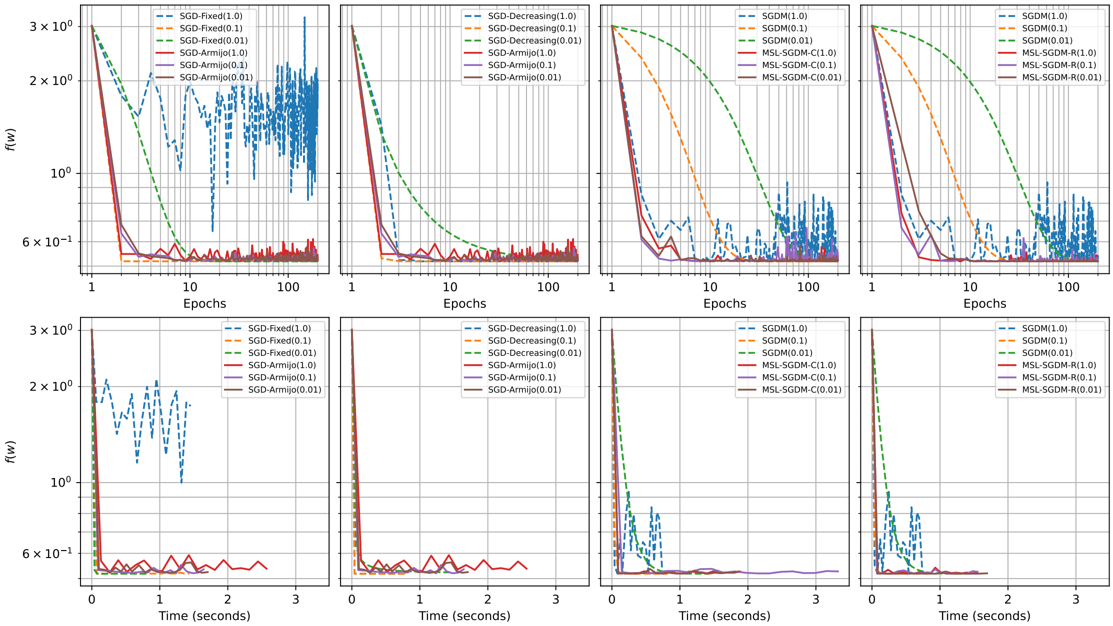



**(a)** *Phishing dataset*

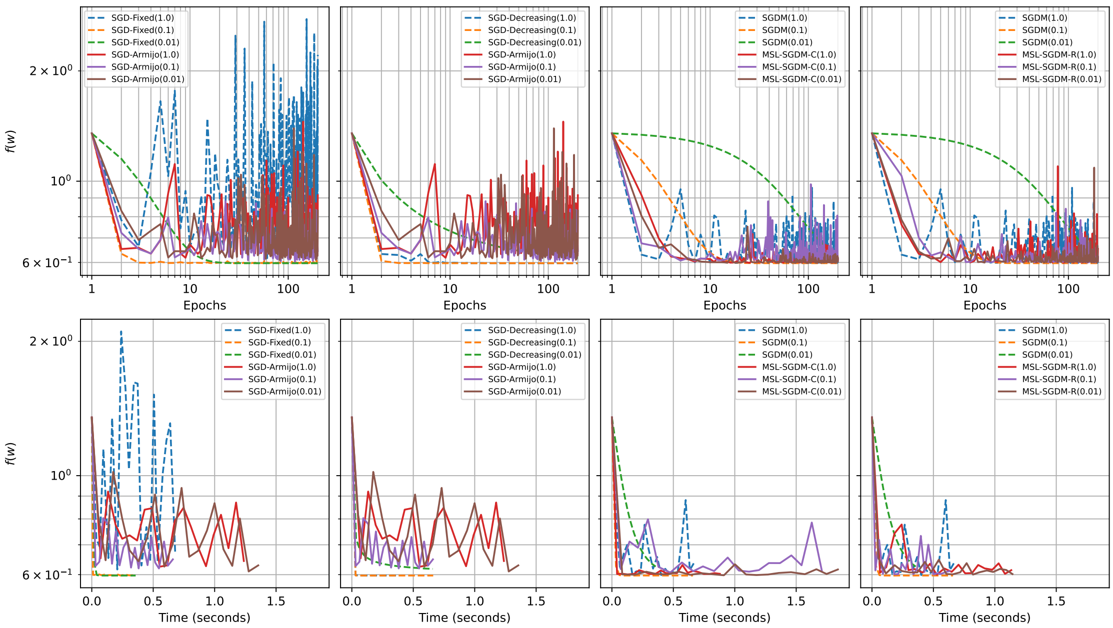



**(b)** *a2a dataset*

**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).