Stochastic Gradient Descent with Momentum and Line Searches

David Nardi
MSc student in Artificial Intelligence, Univeristy of Florence

10th 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.
Those algorithms can be divided in basic SGD and SGD with line search due to common computations, follows a list of the implemented algorithms

- Mini-batch Gradient Descent with fixed step-size and momentum term, and decreasing step-size, algorithm **??** on page ??;
- Mini-batch Gradient Descent with Armijo line search and momentum term restart and correction, algorithm **??** on page ??;

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 9 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

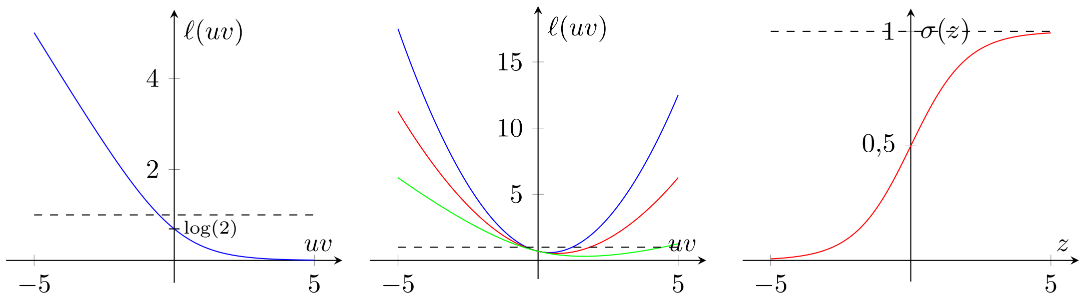
$$\ell_i(w) = \log\bigl(1 + \exp(-y^{(i)} w^T x^{(i)})\bigr) \tag{1}$$

figure 1a on the following 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 next 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

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

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

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

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

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

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

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

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

$$y_{t+1} = y_t + \alpha_t d_t \tag{5}$$

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. y_t on the mini-batch B_t whose indices are randomly chosen $i_t \subset \{1, \dots, N\}$. Knowing that $\nabla f_i(w^k) = x^{(i)} r_i + \lambda w^k$

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

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

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

so the direction is defined as $d_t = -\nabla f_{i_t}(y_t)$ 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. This two version are shown in algorithm **??** on page ?? which is a general version that includes the momentum term (see section 2.2), for this two cases we set $\beta_0 = 0$.

** 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.

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

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

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 2 on the next page.

The SGD with Armijo Line Search `SGD-Armijo` is shown in algorithm **??** on page ??.

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

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

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

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

which is applied as the general update rule in algorithm **??** on page ??, in this case the momentum term is constant $\beta = \beta_0$. To be clear we have the following cases

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

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

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

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

The problem is that $\nabla f_{i_t}(y_t)^T d_t < 0$ 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

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

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 3, the procedure is called *momentum correction*, see algorithm 4 on page 9. 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

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

so if $d_0 = 0$ the direction will be $d_t = -(1-\beta_0)\nabla f_{i_t}(y_t)$ that is a descent direction, see algorithm 5 on page 9. The resulting algorithm is MSL-SGDM-R.

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

Algorithm 2: reset

```
Data: a ∈ ℝ⁺,
      opt ∈ {0, 1, 2}
Input: α, α_0, M, N, t
if t = 0 then
    return α_0
else if opt = 0 then
    α ← α
else if opt = 1 then
    α ← α_0
else if opt = 2 then
    α ← α a^{M/N}
end
Output: α
```

Algorithm 3: armijo

```
Data: γ ∈ (0, 1), δ ∈ (0, 1), q*
Input: y_t, d_t, α
α_t ← α;
q ← 0;
repeat
    α_t ← δ α_t;
    y_{t+1} ← y_t + α_t d_t;
    q ← q + 1;
until f_{i_t}(y_{t+1}) ≤ f_{i_t}(y_t) + γ α_t ∇f_{i_t}(y_{t-1})^T d_t or q ≥ q*;
Output: α_t
```

Algorithm 1: SGD-Fixed, SGD-Decreasing, SGDM

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;
        $\nabla f_{i_t}(z^t) \leftarrow \sum_{j \in B_t} \nabla f_j(z^t)$;
        $d_t \leftarrow \begin{cases} -\big((1-\beta_0)\nabla f_{i_t}(z^t) + \beta_0 d_{t-1}\big) & \text{if SGD-Fix/Decr/Armijo, SGDM} \\ \texttt{correction}\big(\beta_0, \nabla f_{i_t}(z^t), d_{t-1}\big) & \text{if MSL-SGDM-C} \\ \texttt{restart}\big(\beta_0, \nabla f_{i_t}(z^t), d_{t-1}\big) & \text{if MSL-SGDM-R} \end{cases}$;
        if *SGD-Armijo, MSL-SGDM-C/R* **then**
            $\alpha \leftarrow \texttt{reset}(\alpha_{t-1}, \alpha_0, M, N, t)/\delta$;
            $\alpha_t \leftarrow \texttt{armijo}(z^t, d_t, \alpha)$;
        end
        $z^{t+1} \leftarrow z^t + \alpha_t d_t$;
    end
    $w^{k+1} \leftarrow z^{N/M}$;
    $k \leftarrow k + 1$;
end

Algorithm 4: correction

Data: $\delta \in (0,1)$, q^*
Input: β_0, $\nabla f_{i_t}(y_t)$, d_{t-1}

```
β ← β_0;
q ← 0;
repeat
    β ← δβ;
    d_t ← −((1 − β)∇f_{i_t}(y_t) + βd_{t−1});
    q ← q + 1;
until ∇f_{i_t}(y_t)^T d_t < 0 or q ≥ q*;
β_t ← β;
d_t ← −((1 − β_t)∇f_{i_t}(y_t) + β_t d_{t−1});
```

Output: d_t

Algorithm 5: restart

Data: d_0
Input: β_0, $\nabla f_{i_t}(y_t)$, d_{t-1}

```
q ← 0;
d_t ← −((1 − β_0)∇f_{i_t}(y_t) + β_0 d_{t−1});
if not ∇f_{i_t}(y_t)^T d_t < 0 then
    d_{t−1} ← d_0;
    d_t ← −((1 − β_0)∇f_{i_t}(y_t) + β_0 d_{t−1});
end
```

Output: d_t

3 Experiments and results discussion

To test the efficiency the algorithms, a benchmark of six datasets retrieved from LIBSVM, see table 1.

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.

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 value. For a better comparison, the L-BFGS, Conjugate Gradient and Newton-CG algorithms are also tested.

As set 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* and the ε tolerance from (4) is set to 10^{-3}.

Table 1: Benchmark datasets

Name	Train	Test
Diabetes	614	154
Breast cancer	546	137
svmguide1	3089	4000
Australian	552	138
Mushrooms	6499	1625
German	800	200

Table 2: Diabetes dataset

Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
L-BFGS	NaN	6	NaN	0.662 128	1.40×10^{-5}	0.642 857
Newton-CG	NaN	5	NaN	0.662 128	1.00×10^{-6}	0.642 857
CG	NaN	6	NaN	0.662 128	3.00×10^{-6}	0.642 857
BatchGD-Fixed	0.750	6	0.0035	0.662 128	4.89×10^{-4}	0.642 857
SGD-Fixed	0.005	25	0.0383	0.662 128	8.73×10^{-4}	0.642 857
SGD-Decreasing	1.000	155	0.1665	0.662 128	9.53×10^{-4}	0.642 857
SGDM	0.050	82	0.0887	0.662 129	9.27×10^{-4}	0.642 857
SGD-Armijo	0.100	174	1.4736	0.662 128	8.67×10^{-4}	0.642 857
MSL-SGDM-C	1.000	166	1.3840	0.662 128	8.43×10^{-4}	0.642 857
MSL-SGDM-R	0.500	129	1.0333	0.662 129	9.87×10^{-4}	0.642 857

Table 3: Breast cancer dataset

Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
L-BFGS	NaN	7	NaN	0.492 561	3.00×10^{-6}	0.817 518
Newton-CG	NaN	7	NaN	0.492 561	1.00×10^{-6}	0.817 518
CG	NaN	8	NaN	0.492 561	2.00×10^{-6}	0.817 518
BatchGD-Fixed	0.750	12	0.0041	0.492 561	7.99×10^{-4}	0.817 518
SGD-Fixed	0.005	35	0.0409	0.492 561	9.52×10^{-4}	0.817 518
SGD-Decreasing	1.000	156	0.1549	0.492 561	6.66×10^{-4}	0.817 518
SGDM	0.040	76	0.0665	0.492 561	7.88×10^{-4}	0.817 518
SGD-Armijo	0.050	134	0.9551	0.492 561	9.97×10^{-4}	0.817 518
MSL-SGDM-C	0.750	249	1.8371	0.492 561	9.08×10^{-4}	0.817 518
MSL-SGDM-R	0.850	249	1.8281	0.492 561	9.53×10^{-4}	0.817 518

Table 4: svmguide1 dataset

Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
L-BFGS	NaN	5	NaN	0.673 302	1.30×10^{-5}	0.516 250
Newton-CG	NaN	5	NaN	0.673 302	0.00	0.516 750
CG	NaN	8	NaN	0.673 302	8.00×10^{-6}	0.516 750
BatchGD-Fixed	0.750	6	0.0063	0.673 302	2.40×10^{-4}	0.516 250
SGD-Fixed	0.010	9	0.0089	0.673 303	8.23×10^{-4}	0.517 000
SGD-Decreasing	1.000	108	0.2141	0.673 303	9.35×10^{-4}	0.516 000
SGDM	0.050	30	0.0555	0.673 303	8.81×10^{-4}	0.516 250
SGD-Armijo	0.250	32	1.3755	0.673 303	9.92×10^{-4}	0.517 000
MSL-SGDM-C	0.100	70	3.1332	0.673 303	7.38×10^{-4}	0.517 000
MSL-SGDM-R	0.500	84	3.6222	0.673 303	6.16×10^{-4}	0.516 000

Table 5: Australian dataset

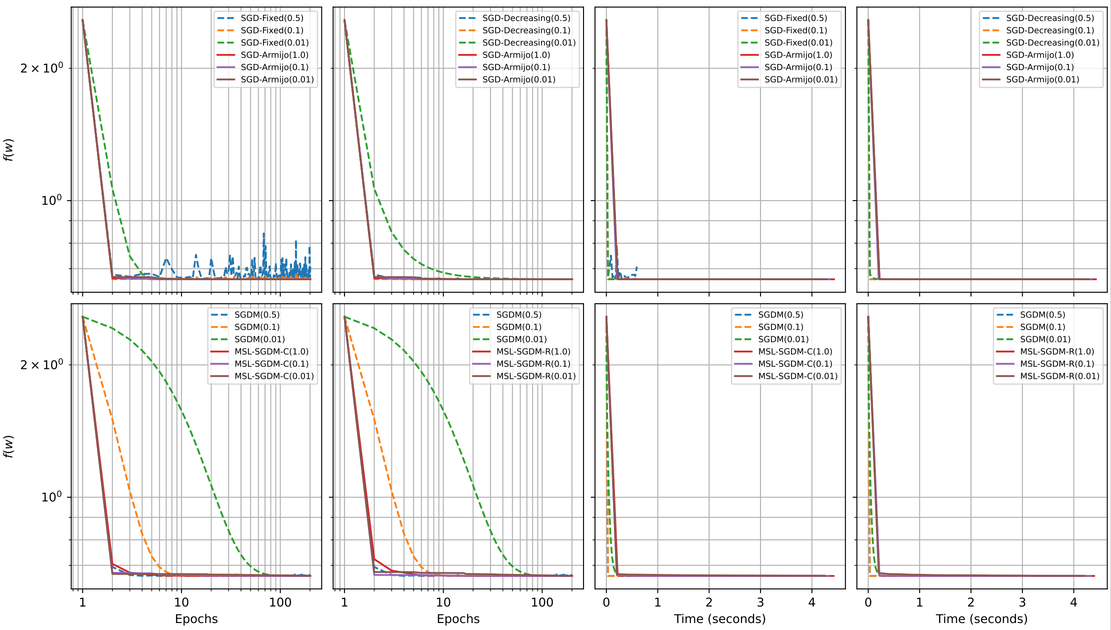
Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	7	NaN	0.615 582	1.00×10^{-6}	0.876 812
L-BFGS	NaN	7	NaN	0.615 582	4.00×10^{-6}	0.876 812
CG	NaN	8	NaN	0.615 582	5.00×10^{-6}	0.876 812
BatchGD-Fixed	0.200	34	0.0150	0.615 582	8.02×10^{-4}	0.876 812
SGD-Decreasing	0.050	18	0.0509	0.615 582	9.05×10^{-4}	0.876 812
SGDM	0.020	190	0.5137	0.615 582	9.39×10^{-4}	0.876 812
SGD-Fixed	0.001	108	0.2945	0.615 582	9.51×10^{-4}	0.876 812
SGD-Armijo	0.020	400	8.8846	0.615 592	4.57×10^{-3}	0.876 812
MSL-SGDM-R	0.500	400	8.9440	0.615 603	6.57×10^{-3}	0.876 812
MSL-SGDM-C	0.500	400	9.0916	0.615 629	9.88×10^{-3}	0.876 812

Table 6: Mushrooms dataset

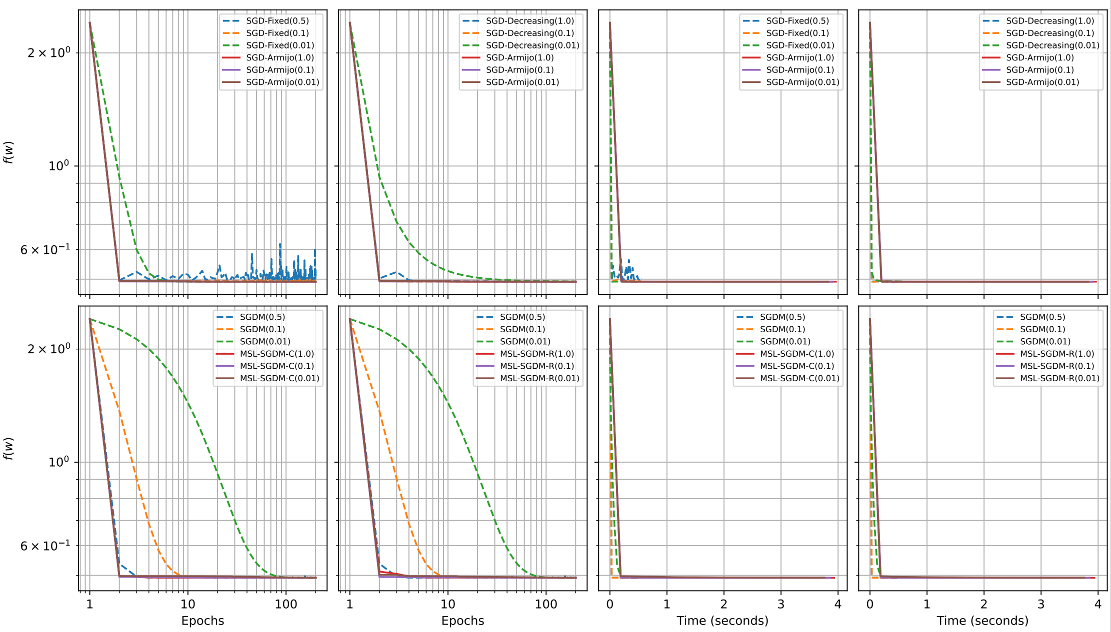
Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	8	NaN	0.580 925	1.38×10^{-4}	0.886 154
L-BFGS	NaN	9	NaN	0.580 925	6.00×10^{-6}	0.886 154
CG	NaN	10	NaN	0.580 925	2.20×10^{-5}	0.886 154
SGD-Fixed	0.001	90	0.2452	0.580 925	7.55×10^{-4}	0.886 154
SGDM	0.030	72	0.1953	0.580 925	9.88×10^{-4}	0.886 154
SGD-Decreasing	0.100	26	0.0895	0.580 925	9.96×10^{-4}	0.886 154
BacthGD-Fixed	0.050	168	0.6020	0.580 925	9.77×10^{-4}	0.886 154
SGD-Armijo	0.250	400	28.4355	0.580 927	2.23×10^{-3}	0.886 154
MSL-SGDM-R	0.500	400	28.5444	0.580 966	9.29×10^{-3}	0.886 769
MSL-SGDM-C	0.200	400	28.3637	0.581 541	3.56×10^{-2}	0.886 769

Table 7: German dataset

Solver	α_0	Epochs	Run-time	$f(w)$	$\nabla f(w)$	Test score
Newton-CG	NaN	7	NaN	0.619 120	1.00×10^{-6}	0.700 000
L-BFGS	NaN	10	NaN	0.619 120	9.00×10^{-6}	0.700 000
CG	NaN	9	NaN	0.619 120	1.70×10^{-5}	0.700 000
BatchGD-Fixed	0.500	12	0.0080	0.619 120	6.13×10^{-4}	0.700 000
SGD-Decreasing	1.000	161	0.1269	0.619 120	8.08×10^{-4}	0.700 000
SGD-Fixed	0.005	63	0.0908	0.619 121	9.63×10^{-4}	0.700 000
SGD-Armijo	0.250	400	3.2284	0.619 122	1.77×10^{-3}	0.700 000
SGDM	0.050	400	0.2840	0.619 130	4.81×10^{-3}	0.700 000
MSL-SGDM-R	1.000	400	3.1536	0.619 143	6.86×10^{-3}	0.700 000
MSL-SGDM-C	1.000	400	3.0884	0.619 170	1.03×10^{-2}	0.700 000

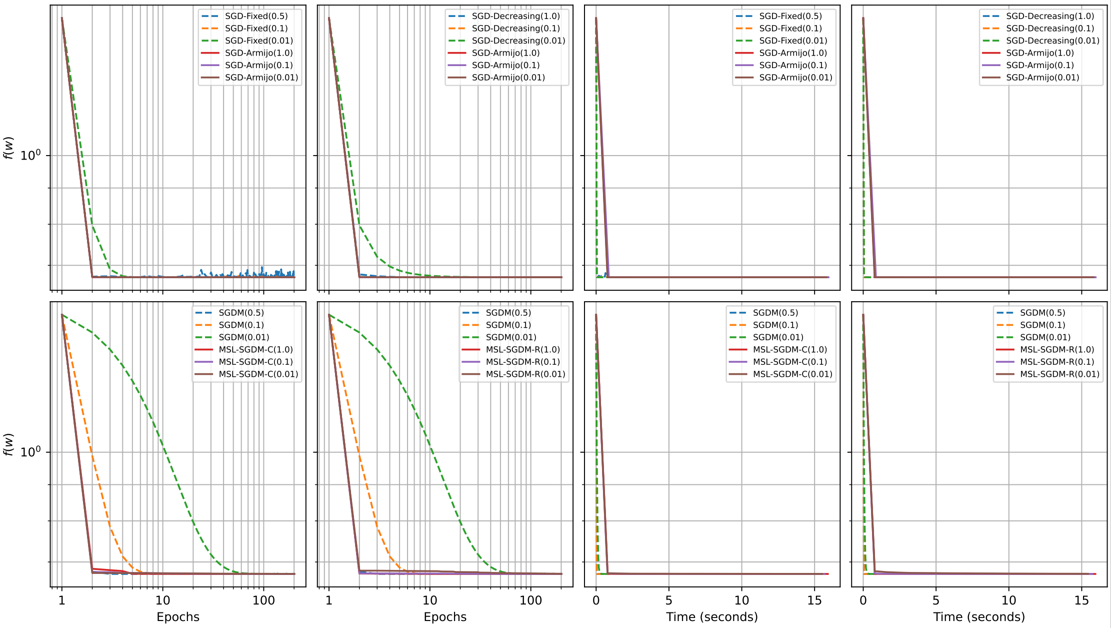


(a) *Diabetes dataset*

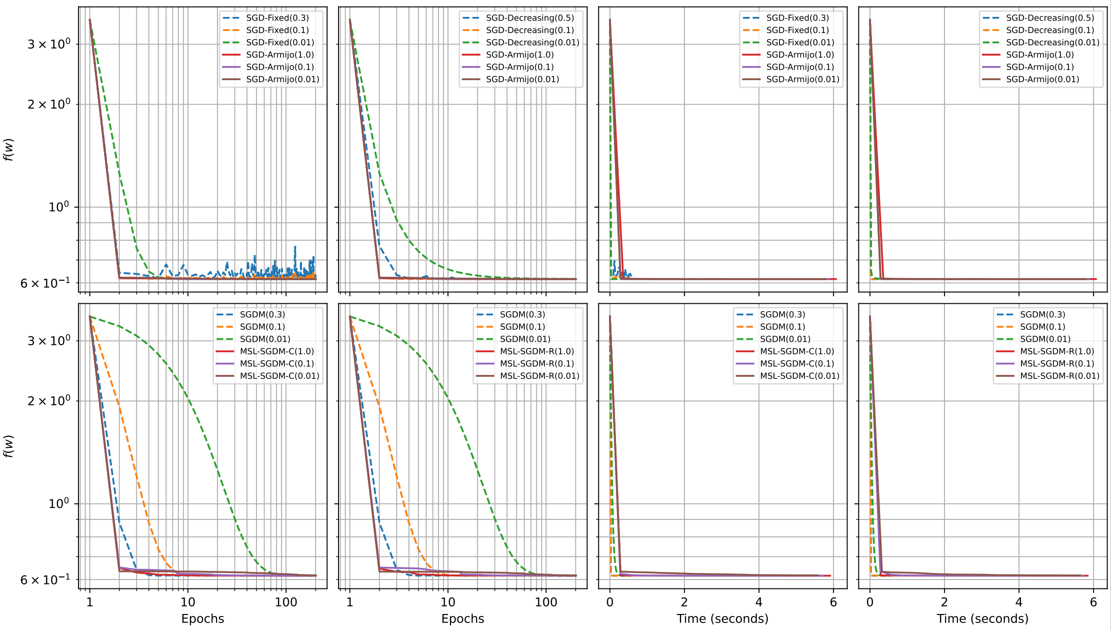


(b) *Breast cancer dataset*

Figure 1: Diabetes and Breast cancer

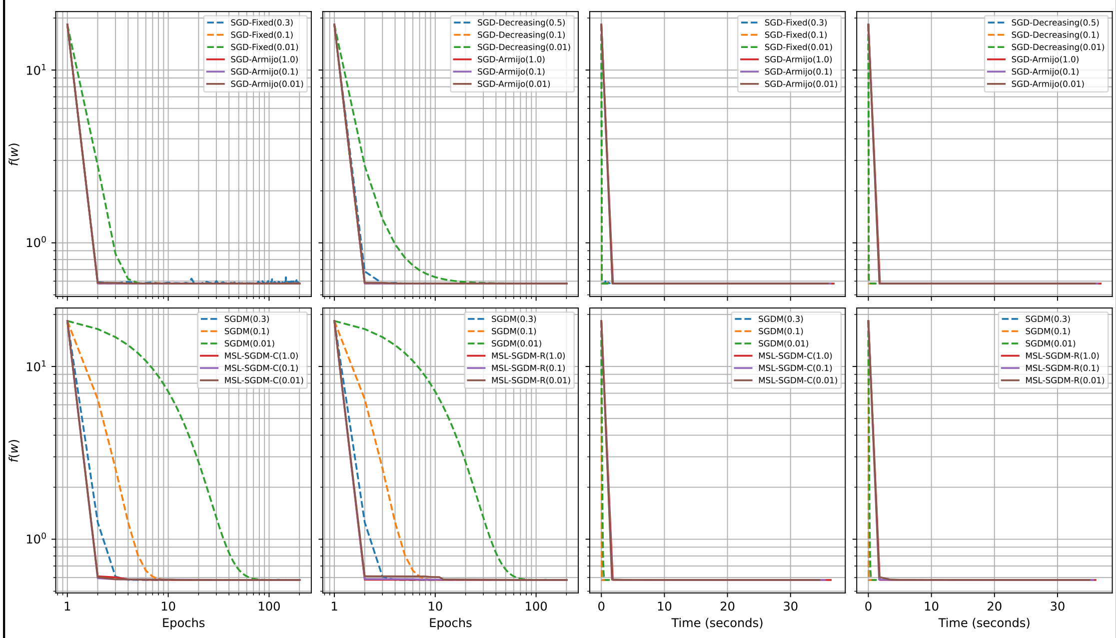


(a) *svmguide1 dataset*

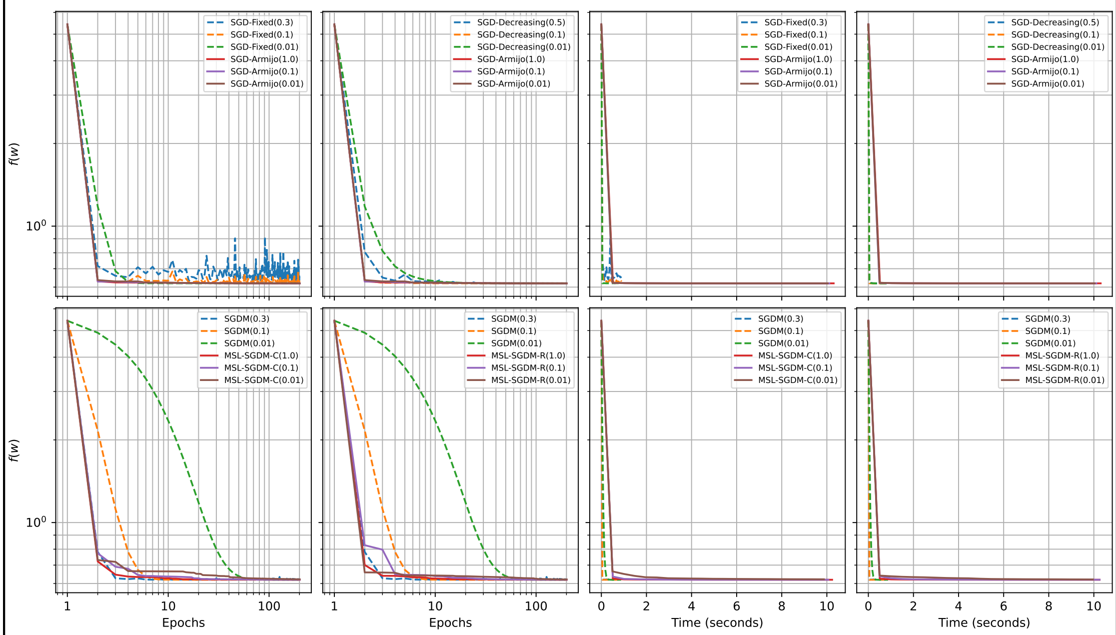


(b) *Australian dataset*

Figure 2: svmguide1 and Australian datasets



(a) *Mushrooms dataset*



(b) *German dataset*

Figure 3: 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, 6, 9).