SONIA: A Symmetric Blockwise Truncated Optimization Algorithm

Majid Jahani[1], Mohammadreza Nazari[2], Rachael Tappenden[3], Albert S. Berahas[1], and Martin Takáč[1]

[1]Department of Industrial and Systems Engineering, Lehigh University

[2]SAS Institute

[3]University of Canterbury

ISE Technical Report 20T-011

SONIA: A Symmetric Blockwise Truncated Optimization Algorithm

Majid Jahani
Lehigh University
Bethlehem, PA 18015
maj316@lehigh.edu

Mohammadreza Nazari
SAS Institute
Cary, NC 27513
mrza.nazari@gmail.com

Rachael Tappenden
University of Canterbury
Christchurch 8041, New Zealand
rachael.tappenden@canterbury.ac.nz

Albert S. Berahas
Lehigh University
Bethlehem, PA 18015
albertberahas@gmail.com

Martin Takáč
Lehigh University
Bethlehem, PA 18015
Takac.MT@gmail.com

Abstract

This work presents a new algorithm for empirical risk minimization. The algorithm bridges the gap between first- and second-order methods by computing a search direction that uses a second-order-type update in one subspace, coupled with a scaled steepest descent step in the orthogonal complement. To this end, partial curvature information is incorporated to help with ill-conditioning, while simultaneously allowing the algorithm to scale to the large problem dimensions often encountered in machine learning applications. Theoretical results are presented to confirm that the algorithm converges to a stationary point in both the strongly convex and nonconvex cases. A stochastic variant of the algorithm is also presented, along with corresponding theoretical guarantees. Numerical results confirm the strengths of the new approach on standard machine learning problems.

1 Introduction

This paper presents a novel optimization algorithm for empirical risk minimization:

$$\min_{w\in\mathbb{R}^d} F(w) := \tfrac{1}{n}\textstyle\sum_{i=1}^n f(w;x^i,y^i) = \tfrac{1}{n}\sum_{i=1}^n f_i(w), \tag{1.1}$$

where $\{(x^i,y^i)\}_{i=1}^n$ are training examples (observations), and $f_i : \mathbb{R}^d \to \mathbb{R}$ is the composition of a prediction function (parameterized by $w \in \mathbb{R}^d$) and a loss function associated with the ith training observation (sample). Problems of the form (1.1) arise in a wide variety of machine learning applications [5, 14, 22, 28]. The main challenge of solving these problems stems from the fact that they are often high-dimensional and nonlinear, and may be nonconvex.

For many machine learning applications, a common approach is to employ first-order methods such as the Stochastic Gradient method (SGD). SGD and its variance-reduced, adaptive and distributed variants [17, 23, 26, 27, 38, 41, 42, 45, 47] are popular because they are simple to implement and have low per-iteration cost. However, these methods often require significant tuning efforts (for each problem) to ensure practical performance, and they struggle on ill-conditioned problems.

Preprint. Under review.

One avenue for mitigating the aforementioned issues is the use of second-order and quasi-Newton methods [16, 19, 39]. These methods, in the deterministic setting, are relatively insensitive to their associated hyper-parameters and are able to alleviate the effects of ill-conditioning. Unfortunately, a drawback of these methods is that they often do not scale sufficiently well with the high dimensionality (both n and d) typical in machine learning and deep learning problems. Thus, the computational burden of using deterministic higher-order methods is often deemed to be too high.

Recently attention has shifted towards stochastic second-order [7, 10, 32, 46] and quasi-Newton [3, 4, 12, 15, 34, 48] methods. These methods attempt to combine the speed of Newton's method and the scalability of first-order methods by incorporating curvature information in a judicious manner, and have proven to work well for several machine learning tasks [1, 51]. However, the question of how to balance the accuracy in the gradient and Hessian approximation is yet unresolved, and as such these methods often perform on par with their first-order variants.

Several other attempts have been made to balance the first- versus second-order trade-off, in order to find ways of incorporating partial curvature information to help with ill-conditioning at an acceptable cost. For example, variants of coordinate descent methods that perform second-order-type updates restricted to a low dimensional subspace, are prototypical methods in this niche [20, 43, 44, 49]. However, while progress has been made, the gap between first- and second-order methods remains.

In this paper, we propose the **S**ymmetric bl**O**ckwise tru**N**cated optim**I**zation **A**lgorithm (`SONIA`). `SONIA` aims to bridge the gap between first- and second-order methods, but is different in nature to coordinate descent methods because at every iteration a step in the *full dimensional space* is generated. The search direction consists of two components. The first component lies in an m-dimensional subspace (where $m \ll d$ is referred to as the 'memory' and is user defined), and is generated using a second-order approach. The second component of the update lies in the orthogonal complement, and is an inexpensive scaled steepest descent update. The combination of the two components allows for the overall search direction to explore the full-dimensional space at every iteration.

Contributions Our contributions can be summarized as follows:

- *Novel Optimization Algorithm*. We propose `SONIA` for solving empirical risk minimization problems that attempts to bridge the gap between first- and second-order methods. The algorithm judiciously incorporates curvature information in one subspace (whose dimension is determined by the user) and takes a gradient descent step in the complement of that subspace. As such, at every iteration, `SONIA` takes a step in full dimensional space while retaining a low per-iteration cost and storage, similar to that of limited memory quasi-Newton methods.
- *Theoretical Analysis*. We derive convergence guarantees for `SONIA`, both in deterministic and stochastic regimes, for strongly convex and nonconvex optimization problems. These guarantees match those of popular quasi-Newton methods such as L-BFGS.
- *Stochastic Variant of `SONIA`*. We develop and analyze a stochastic variant of `SONIA` that uses stochastic gradient and Hessian approximations in lieu of the true gradient and Hessian.
- *Competitive Numerical Results*. We investigate the empirical performance of the deterministic and stochastic variants of `SONIA` on strongly convex (logistic regression) and nonconvex (nonlinear least squares) problems that arise in machine learning. Our proposed methods are competitive with the algorithms of choice in both the deterministic and stochastic settings.

Organization Related works are described in Section 2. Section 3 presents our proposed algorithm, `SONIA`, and its stochastic variant. We show the theoretical properties of our proposed method in Section 4. Numerical results on deterministic and stochastic problems are reported in Section 5. Finally, in Section 6 we provide some final remarks and discuss possible avenues for future research.

2 Related Work

As in this work, the following works employ iterate updates of the form

$$w_{k+1} = w_k + \alpha_k p_k, \tag{2.1}$$

where $p_k \in \mathbb{R}^d$ is the search direction and $\alpha_k > 0$ is the step length or learning rate.

The work in [40] proposes a Newton-type algorithm for nonconvex optimization problems. At each iteration the construction and eigenvalue decomposition (full dimensional) of the Hessian is required, small (in modulus) eigenvalues are truncated, and a Newton-like search direction is generated using the truncated inverse Hessian instead of the true inverse Hessian. The method works well in practice and is guaranteed to converge to local minima, but is expensive.

Quasi-Newton methods–methods that compute search directions using (inverse) Hessian approximations that are constructed using past iterate and gradient information–represent some of the most effective algorithms for minimizing nonlinear objective functions. This class of nonlinear optimization algorithms includes BFGS, DFP and SR1; see [16, 19, 39] and the references therein.

The Symmetric Rank One (SR1) update is a special case of a rank one quasi-Newton method [16]. It is the unique symmetric rank-1 update that satisfies the secant condition $B_{k+1}s_k = y_k$, where $s_k = w_k - w_{k-1}$ and $y_k = \nabla F(w_k) - \nabla F(w_{k-1})$ are the curvature pairs and the Hessian approximation is updated at every iteration via: $B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{s_k^T(y_k - B_k s_k)}$. Hessian approximations using SR1 updates are not guaranteed to be positive definite, and while this was originally seen as a drawback, it is arguably viewed as an advantage in the context of nonconvex optimization. Limited memory variants exist where there is a fixed memory size m, and only the last m curvature pairs are kept and used to construct the Hessian approximation. Let $S_k = [s_{k-m+1}, \dots, s_k] \in \mathbb{R}^{d\times m}$ and $Y_k = [y_{k-m+1}, \dots, y_k] \in \mathbb{R}^{d\times m}$ denote matrices consisiting of the m most recent curvature pairs. As studied in [13], the compact form of L-SR1 is as follows:

$$B_k = B_0 + (Y_k - B_0 S_k)(L_k + D_k + L_k^T - S_k^T B_0 S_k)^{-1}(Y_k - B_0 S_k)^T, \tag{2.2}$$

where $S_k^T Y_k = L_k + D_k + U_k$, L_k denotes the strictly lower triangular part, D_k is the diagonal and U_k denotes the strictly upper triangular part of $S_k^T Y_k$, respectively, and B_0 is an initial approximation (usually set as $B_0 = \eta I$, $\eta > 0$). A key observation is that while B_{k+1} is a full dimensional $d \times d$ matrix, the inverse in (2.2) is a small $m \times m$ matrix. Recent works that employ the compact L-SR1 update include [2, 9, 18], where (2.2) defines the quadratic model within a trust region algorithm.

Rather than maintaining a history of the m most recent curvature pairs, the work [2] proposes a *sampled* variant of the L-SR1 update. In that work, at each iteration $k \geq 0$, m directions $\{s_1, \dots, s_m\}$ are sampled around the current iterate w_k and stored as $S_k = [s_1, \dots, s_m] \in \mathbb{R}^{d\times m}$. Next, the gradient displacement vectors are computed via

$$Y_k = \nabla^2 F(w_k) S_k, \tag{2.3}$$

and the matrix B_0 is set to zero. In this way, previous curvature information is 'forgotten', and local curvature information is 'sampled' around the current iterate. Moreover, depending on the way the vectors $\{s_1, \dots, s_m\}$ are sampled, one can view (2.3) as a sketch of the Hessian [50].

The approach proposed here combines quasi-Newton updates for indefinite Hessians [18] with sampled curvature pairs [2]. Moreover, a truncation step (as in [40]) allows us to avoid checking conditions on the curvature pairs, and ensures that the Hessian approximations constructed are positive definite. The subspace generation is based on the user-defined hyper-parameter m (as such the user has full control over the computational cost of each step), and an eigenvalue decomposition step (which is performed in reduced dimension as so is cheap).

3 Symmetric blOckwise truNcated optimIzation Algorithm (`SONIA`)

In this section, we present our proposed algorithm. We begin by motivating and describing the deterministic variant of the method and then discuss its stochastic counterpart. We end this section by discussing the per iteration complexity of `SONIA`.

3.1 Deterministic `SONIA`

The `SONIA` algorithm generates iterates according to the update (2.1). The search direction p_k consists of two components; the first component lies in one subspace and is a second-order based update, while the second component lies in the orthogonal complement and is a scaled steepest descent direction. We now describe how the subspaces are built at each iteration as well as how to compute the second-order component of the search direction.

The algorithm is initialized with a user defined memory size $m \ll d$. At each iteration $k \geq 0$ of `SONIA`, m directions $\{s_1, \dots, s_m\}$ are randomly sampled, and curvature pair matrices S_k and Y_k are constructed via (2.3). Setting $B_0 = 0$, and substituting into (2.2) gives the compact form of the

Hessian approximation used in this work[1]:

$$B_k = Y_k(Y_k^T S_k)^\dagger Y_k^T. \tag{3.1}$$

Similar to the strategy in [18], using the 'thin' QR factorization of $Y_k = Q_k R_k$, where $Q_k \in \mathbb{R}^{d\times m}$ has orthonormal columns and $R_k \in \mathbb{R}^{m\times m}$ is an upper triangular matrix, (3.1) gives

$$B_k = Q_k R_k (Y_k^T S_k)^\dagger R_k^T Q_k^T. \tag{3.2}$$

Note that B_k is symmetric because, by (2.3), $(Y_k^T S_k)^\dagger = (S_k^T \nabla^2 F(w_k) S_k)^\dagger \in \mathbb{R}^{m\times m}$ is symmetric. Thus, by the spectral decomposition, $R_k(Y_k^T S_k)^\dagger R_k^T = V_k \Lambda_k V_k^T$, where the columns of $V_k \in \mathbb{R}^m$ form an orthonormal basis (of eigenvectors), and $\Lambda_k \in \mathbb{R}^{m\times m}$ is a diagonal matrix containing the corresponding eigenvalues. Substituting this into (3.2) gives $B_k = Q_k V_k \Lambda_k V_k^T Q_k^T$. Since Q_k has orthonormal columns and V_k is an orthogonal matrix, it is clear that

$$\tilde{V}_k := Q_k V_k \in \mathbb{R}^{d\times m} \tag{3.3}$$

has orthonormal columns. Finally, the low rank decomposition of the Hessian approximation B_k is

$$B_k = \tilde{V}_k \Lambda_k \tilde{V}_k^T. \tag{3.4}$$

The following definition is motivated by [40, Definition 2.1].

Definition 3.1. *Let B_k, $\tilde{V}_k$ and Λ_k be the matrices in (3.4), and let $\epsilon > 0$. The truncated inverse Hessian approximation of B_k is $A_k := \tilde{V}_k |\Lambda_k|_\epsilon^{-1} \tilde{V}_k^T$, where $(|\Lambda_k|_\epsilon)_{ii} = \max\{|\Lambda_k|_{ii}, \epsilon\}$.*

Definition 3.1 explains that any eigenvalues in Λ_k below the threshold ϵ are truncated and set to ϵ. This is useful for several reasons. Firstly, it ensures that the search direction consists only of directions with non-negligible curvature. Moreover, unlike classical quasi-Newton methods that enforce conditions on the curvature pairs to guarantee that the (inverse) Hessian approximations are well-defined and that the updates are stable, `SONIA` utilizes a truncation step (as described in Definition 3.1) and as such avoids the need for any such safe-guards. The reason for this is that even if $Y_k^T S_k$ is rank deficient, the truncation step ensures that A_k has full rank. This is especially important with SR1-type methods that require matrix-vector products in the checks.

Before we proceed, we make a few more comments about our algorithmic choice of constructing the gradient differencing curvature pairs via (2.3). As mentioned above, this ensures that $Y_k^T S_k$ is symmetric which is a fundamental component of our approach for three main reasons. Firstly, it ensures that the Hessian approximations constructed are symmetric. Secondly, it allows us to utilize the spectral decomposition. And, thirdly, unlike the classical SR1 method that utilizes only the lower triangular part of the $Y_k^T S_k$ matrix to construct Hessian approximations (see (2.2)) and as such throws away possibly useful curvature information, our approach allows us to use all curvature information collected at every iteration. Moreover, one can show that constructing curvature pairs in this manner guarantees that the secant equations hold for all curvature pairs, and that the Hessian approximations are scale invariant.

Next we discuss the subspace decomposition. The gradient is orthogonally decomposed as:

$$\nabla F(w_k) = g_k + g_k^\perp, \quad \text{where} \quad g_k = \tilde{V}_k \tilde{V}_k^T \nabla F(w_k) \quad \text{and} \quad g_k^\perp = (I - \tilde{V}_k \tilde{V}_k^T)\nabla F(w_k). \tag{3.5}$$

Clearly, $g_k \in \text{range}(\tilde{V}_k \tilde{V}_k^T)$ and $g_k^\perp \in \ker(\tilde{V}_k \tilde{V}_k^T) \equiv \text{range}(I - \tilde{V}_k \tilde{V}_k^T)$. Vectors g_k and $g_k^\perp$ are orthogonal because the subspaces $\text{range}(\tilde{V}_k \tilde{V}_k^T)$ and $\text{range}(I - \tilde{V}_k \tilde{V}_k^T)$ are orthogonal complements (i.e., $g_k^T g_k^\perp = \nabla F(w_k)^T \tilde{V}_k \tilde{V}_k^T (I - \tilde{V}_k \tilde{V}_k^T)\nabla F(w_k) = 0$).

The `SONIA` search direction is

$$p_k := -\tilde{V}_k |\Lambda_k|_\epsilon^{-1} \tilde{V}_k^T \nabla F(w_k) - \rho_k (I - \tilde{V}_k \tilde{V}_k^T)\nabla F(w_k) = -\tilde{V}_k |\Lambda_k|_\epsilon^{-1} \tilde{V}_k^T g_k - \rho_k g_k^\perp, \tag{3.6}$$

where for all $k \geq 0$,

$$\rho_k \in (0, \lambda_k^{\min}], \qquad \text{and} \qquad \lambda_k^{\min} := \min_i \{[|\Lambda_k|_\epsilon^{-1}]_{ii}\}. \tag{3.7}$$

The first component of the search direction lies in the subspace $\text{range}(\tilde{V}_k \tilde{V}_k^T)$, while the second component lies in the orthogonal complement.

Lemma 3.2. *The search direction p_k in (3.6) is equivalent to $p_k = -\mathcal{A}_k \nabla F(w_k)$, where*

$$\mathcal{A}_k := \tilde{V}_k |\Lambda_k|_\epsilon^{-1} \tilde{V}_k^T + \rho_k (I - \tilde{V}_k \tilde{V}_k^T). \tag{3.8}$$

[1]If $S_k^T Y_k$ has full rank, then the pseudo-inverse in (3.1) is simply the inverse.

The search direction p_k in (3.6) can be interpreted as follows. If the memory is chosen as $m = 0$, then p_k is simply a scaled steepest descent direction (in this setting, ρ_k can be any positive number). On the other hand, if $m = d$, then p_k incorporates curvature information in the full dimensional space. If $0 < m < d$, then the algorithm is a hybrid of a second-order method in $\text{range}(\tilde{V}_k \tilde{V}_k^T)$ and steepest descent in the orthogonal complement $\text{null}(\tilde{V}_k \tilde{V}_k^T)$. Thus, this algorithm bridges the gap between first- and second-order methods.

Remark 3.3. *The following remarks are made regarding the search direction p_k.*

- *The first component of the search direction vanishes only if (i) the memory size is $m = 0$, or if (ii) $\nabla F(w) \in \text{null}(\tilde{V}\tilde{V}^T)$.*
- *The second component of the search direction vanishes only if (i) the memory size is $m = d$, or if (ii) $\text{range}(\tilde{V}\tilde{V}^T) \equiv \mathbb{R}^d$.*

The SONIA algorithm is presented in Algorithm 1.

Algorithm 1 SONIA

```
Input: $w_0$ (initial iterate), $m$ (memory), $\epsilon$ (truncation parameter).
 1: for $k = 0, 1, 2, ...$ do
 2:     Compute the gradient $\nabla F(w_k)$
 3:     Construct a random matrix $S_k \in \mathbb{R}^{d \times m}$ and set $Y_k$ via (2.3)
 4:     Compute the $QR$ factorization of $Y_k (= Q_k R_k)$
 5:     Compute the spectral decomposition of $R_k (Y_k^T S_k)^\dagger R_k^T (= V_k \Lambda_k V_k^T)$
 6:     Construct $\tilde{V}_k (= Q_k V_k)$ via (3.3)
 7:     Truncate the eigenvalues of $\Lambda_k$ to form $|\Lambda_k|_\epsilon$ and set $\rho_k$ via (3.7)
 8:     Decompose the gradient $\nabla F(w_k) (= g_k + g_k^\perp)$ via (3.5)
 9:     Compute the search direction $p_k$ via (3.6)
10:     Select the steplength $\alpha_k > 0$, and set $w_{k+1} = w_k + \alpha_k p_k$
11: end for
```

3.2 Stochastic SONIA

The SONIA algorithm presented in Section 3.1, requires a gradient evaluation and a Hessian-matrix product (to construct Y_k) at every iteration. For many machine learning applications n and d are large, and thus the required computations can be prohibitively expensive. To overcome these difficulties, we present a stochastic variant of the SONIA algorithm that employs a mini-batch approach.

Stochastic SONIA chooses a set $\mathcal{I}_k \subset [n]$, and the new iterate is computed as follows:

$$w_{k+1} = w_k - \alpha_k \mathcal{A}_k \nabla F_{\mathcal{I}_k}(w_k), \quad \text{where } \nabla F_{\mathcal{I}_k}(w_k) = \frac{1}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla F_i(w_k) \tag{3.9}$$

and $\mathcal{A}_k$ is the stochastic inverse truncated Hessian approximation. Stochastic SONIA uses stochastic Hessian-matrix products to construct Y_k, i.e., $Y_k = \nabla^2 F_{\mathcal{J}_k}(w_k) S_k$, where $\mathcal{J}_k \subset [n]$. It is important to note that for the theory (see Section 4) the sample sets $\mathcal{I}_k$ and $\mathcal{J}_k$ need to be chosen independently.

3.3 Discussion about Complexity of SONIA

The per iteration complexity of SONIA consists of: (1) a Hessian-matrix product ($\mathcal{O}(mnd)$); (2) a QR factorization of an $d \times m$ matrix ($\mathcal{O}(dm^2)$); (3) a pseudo-inverse of an $m \times m$ matrix ($\mathcal{O}(m^3)$); and, (4) a spectral decomposition of an $m \times m$ matrix ($\mathcal{O}(m^3)$). The computational cost and storage requirement for the SONIA algorithm are presented in Table 1[2], where we compare the cost and storage to popular limited-memory quasi-Newton methods and the NCN method [40]. Note that the SONIA algorithm was developed for the regime where $m \ll d, n$. As is clear from Table 1, SONIA has similar cost and storage to LBFGS and LSR1, and is significantly more efficient, in both regards, to the NCN method. We should note that the computational cost and storage requirements for stochastic SONIA are $\mathcal{O}(d)$.

Table 1: Summary of Computational Cost and Storage (per iteration).

method	computational cost	storage
NCN [40]	$\mathcal{O}(nd^2 + d^3)$	$\mathcal{O}(d^2)$
LBFGS [29]	$\mathcal{O}(nd)$	$\mathcal{O}(d)$
LSR1 [30]	$\mathcal{O}(nd)$	$\mathcal{O}(d)$
SONIA	$\mathcal{O}(nd)$	$\mathcal{O}(d)$

[2]Note, these computations are on top of the function/gradient evaluations that are common to all methods.

4 Theoretical Analysis

Here, theoretical results for `SONIA` are presented, in the deterministic and stochastic settings, for both strongly convex and nonconvex objective functions. Before we present the main theorems, we state two preliminary Lemmas that are used throughout this section. Proofs can be found in Appendix A.

Assumption 4.1. *The function F is twice continuously differentiable.*

Lemma 4.2. *The matrix $\mathcal{A}_k$ in (3.8) is positive definite for all $k \geq 0$.*

Lemma 4.3. *If Assumption 4.1 holds, there exist constants $0 < \mu_1 \leq \mu_2$ such that the inverse truncated Hessian approximations $\{\mathcal{A}_k\}$ generated by Algorithm 1 satisfy,*

$$\mu_1 I \preceq \mathcal{A}_k \preceq \mu_2 I, \qquad \text{for all } k = 0, 1, \ldots. \tag{4.1}$$

4.1 Deterministic Setting

Strongly Convex Functions The following assumption is standard for strongly convex functions.

Assumption 4.4. *There exist positive constants μ and L such that $\mu I \preceq \nabla^2 F(w) \preceq LI, \forall w \in \mathbb{R}^d$.*

Theorem 4.5. *Suppose that Assumptions 4.1 and 4.4 hold, and let $F^\star = F(w^\star)$, where $w^\star$ is the minimizer of F. Let $\{w_k\}$ be the iterates generated by Algorithm 1, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for all $k \geq 0$, $F(w_k) - F^\star \leq (1 - \alpha\mu\mu_1)^k [F(w_0) - F^\star]$.*

Theorem 4.5 shows that `SONIA` converges at a linear rate to the optimal solution of (1.1). The step length range prescribed by `SONIA` depends on μ_1 and μ_2, as does the rate. This is typical for limited memory quasi-Newton methods [2, 29]. In the worst-case, the matrix $\mathcal{A}_k$ can make the limit in Theorem 4.5 significantly worse than that of the first-order variant if the update has been unfortunate and generates ill-conditioned matrices. However, this is rarely observed in practice.

Nonconvex Functions The following assumptions are needed for the nonconvex case.

Assumption 4.6. *The function F is bounded below by a scalar $\widehat{F}$.*

Assumption 4.7. *The gradients of F are L-Lipschitz continuous for all $w \in \mathbb{R}^d$.*

Theorem 4.8. *Suppose that Assumptions 4.1, 4.6 and 4.7 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 1, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for any $T > 1$, $\frac{1}{T}\sum_{k=0}^{T-1} \|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \hat{F}]}{\alpha\mu_1 T} \xrightarrow{T\to\infty} 0$.*

Theorem 4.8 bounds the average norm squared of the gradient of F, and shows that the iterates spend increasingly more time in regions where the objective function has small gradient. From this result, one can show that the iterates, in the limit, converge to a stationary point of F.

4.2 Stochastic Setting

Here, we present theoretical convergence results for the stochastic variant of `SONIA`. Note that, in this section $\mathbb{E}_{\mathcal{I}_k}[\cdot]$ denotes the conditional expectation given w_k, whereas $\mathbb{E}[\cdot]$ denotes the total expectation over the full history. In this setting me make the following standard assumptions.

Assumption 4.9. *There exist a constant γ such that $\mathbb{E}_{\mathcal{I}}[\|\nabla F_{\mathcal{I}}(w) - \nabla F(w)\|^2] \leq \gamma^2$.*

Assumption 4.10. *$\nabla F_{\mathcal{I}}(w)$ is an unbiased estimator of the gradient, i.e., $\mathbb{E}_{\mathcal{I}}[\nabla F_{\mathcal{I}}(w)] = \nabla F(w)$, where the samples $\mathcal{I}$ are drawn independently.*

Strongly Convex Functions

Theorem 4.11. *Suppose that Assumptions 4.1, 4.4, 4.9 and 4.10 hold, and let $F^\star = F(w^\star)$, where $w^\star$ is the minimizer of F. Let $\{w_k\}$ be the iterates generated by Algorithm 1, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for all $k \geq 0$,*
$\mathbb{E}[F(w_k) - F^\star] \leq (1 - \alpha\mu_1\lambda)^k (F(w_0) - F^\star - \frac{\alpha\mu_2^2\gamma^2 L}{2\mu_1\lambda}) + \frac{\alpha\mu_2^2\gamma^2 L}{2\mu_1\lambda}$.

The bound in Theorem 4.11 has two components: (1) a term decaying linearly to zero, and (2) a term identifying the neighborhood of convergence. Notice that a larger step length yields a more favorable constant in the linearly decaying term, at the cost of an increase in the size of the neighborhood of convergence. As in the deterministic case, the step length range prescribed by `SONIA` depends on μ_1

and μ_2, as does the rate. Thus, this result is weaker than that of its first-order variant if the update has been unfortunate and generates ill-conditioned matrices. This is seldom observed in practice.

One can establish convergence of `SONIA` to the optimal solution $w^\star$ by employing a sequence of step lengths that converge to zero (see [45]), but at the slower, sub-linear rate. Another way to achieve exact convergence is to employ variance reduced gradient approximations [26, 47], and achieve linear convergence, at the cost of computing the full gradient every so often, or increased storage.

Non-convex Functions

Theorem 4.12. *Suppose that Assumptions 4.1, 4.6, 4.7, 4.9 and 4.10 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 1, where $0 < \alpha_k = \alpha \le \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point.*

Then, for all $k \ge 0$, $\mathbb{E}[\frac{1}{T}\sum_{k=0}^{T-1}\|\nabla F(w_k)\|^2] \le \frac{2[F(w_0)-\widehat{F}]}{\alpha\mu_1 T} + \frac{\alpha\mu_2^2\gamma^2 L}{\mu_1} \xrightarrow{T\to\infty} \frac{\alpha\mu_2^2\gamma^2 L}{\mu_1}$.

Theorem 4.12 bounds the average norm squared of the gradient of F, in expectation, and shows that, in expectation, the iterates spend increasingly more time in regions where the objective function has small gradient. The difference with the deterministic setting is that one cannot show convergence to a stationary point; this is due to the variance in the gradient approximation employed. One can establish such convergence under an appropriate step length schedule (diminishing step lengths).

5 Numerical Experiments

In this section, we present numerical experiments on several standard machine learning problems, and compare the empirical performance of `SONIA` with that of state-of-the-art first- and second-order methods[3], in both the stochastic and deterministic settings[4]. We considered 4 different classes of problems: (1) deterministic and stochastic logistic regression (stronlgy convex); and, (2) deterministic and stochastic nonlinear least squares (nonconvex), and report results on 2 standard machine learning datasets[5]. For brevity we report only a subset of the results here and defer the rest to Appendix D.

We compared the performance of `SONIA` to algorithms with computational cost and storage requirements linear in both n and d. As such, we did not compare against NCN [40] and full-memory quasi-Newton methods. Our metric for comparison was the number of effective passes (or epochs), which we calculated as the number of function, gradient and Hessian-vector (or matrix) evaluations; see Appendix B for more details. We tuned the hyper-parameters of each method individually for every instance; see Appendix C.3 for a complete description of the tuning efforts. Where applicable, the regularization parameter was chosen from the set $\lambda \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. The memory size was set to $\min\{d, 64\}$. Finally, the truncation parameter was set to $\epsilon = 10^{-5}$ and $\rho_k = \max_i\{[|\Lambda_k|_\epsilon^{-1}]_{ii}\}$; we found that these choices gave the best performance. We also performed sensitivity analysis for `SONIA`; see Appendices D.1.1 and D.1.2.

5.1 Deterministic Setting

In the deterministic setting, we compared the performance of `SONIA` to that of Gradient Descent, L-BFGS [29], L-SR1 [30], NEST+ [35] and Newton CG [39]. We implemented the algorithms with adaptive procedures for selecting the steplength (e.g., Armijo backtracking procedure [39]) and/or computing the step (e.g., trust-region subroutine [39]). Note, Newton CG was implemented with a line search for strongly convex problems and with a trust region for nonconvex problems.

Deterministic Logistic Regression We considered ℓ_2 regularized logistic regression problems, $F(w) = \frac{1}{n}\sum_{i=1}^n \log(1 + e^{-y_i x_i^T w}) + \frac{\lambda}{2}\|w\|^2$. Figure 1a shows the performance of the methods in terms of optimality gap ($F(w) - F^\star$[6]) and testing accuracy versus number of effective passes. As is clear, the performance of `SONIA` is on par or better than that of the other methods. Similar behavior was observed on other datasets; see Appendix D.1.

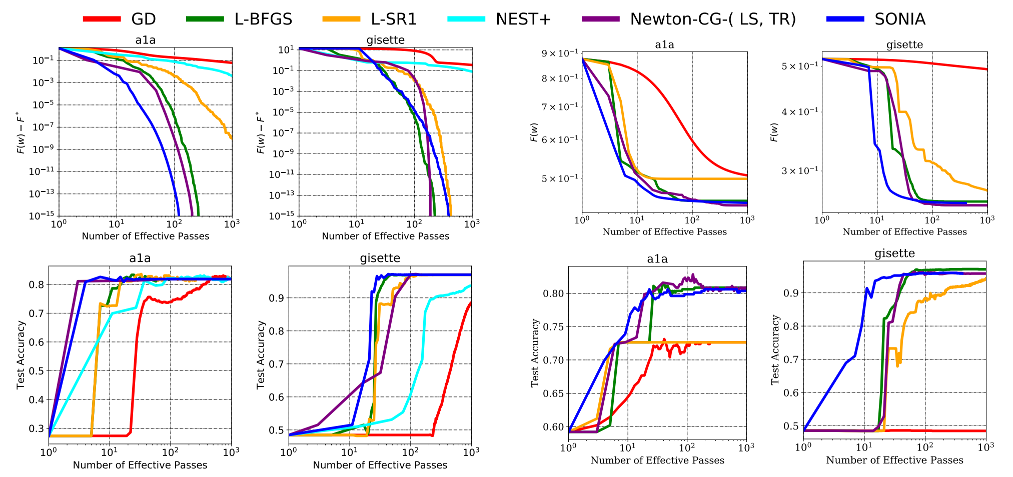
Deterministic Non-linear Least Square We considered non-linear least squares problems, $F(w) = \frac{1}{n}\sum_{i=1}^n (y_i - \frac{1}{1+e^{-x_i^T w}})^2$, described in [51]. Figure 1b shows the performance of the methods in terms of objective function and testing accuracy versus number of effective passes. As

[3]See Section C.1 for details about all algorithms considered in this section.

[4]All the codes to reproduce the experimental results will be released upon publication.

[5]`a1a` and `gistte`. Available at: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

[6]To find $w^\star$ we ran the ASUESA algorithm [31]; see Section C.1.



(a) Comparison of optimality gap ($F(w) - F^{\star}$) and Test Accuracy for different algorithms on Logistic Regression Problems.

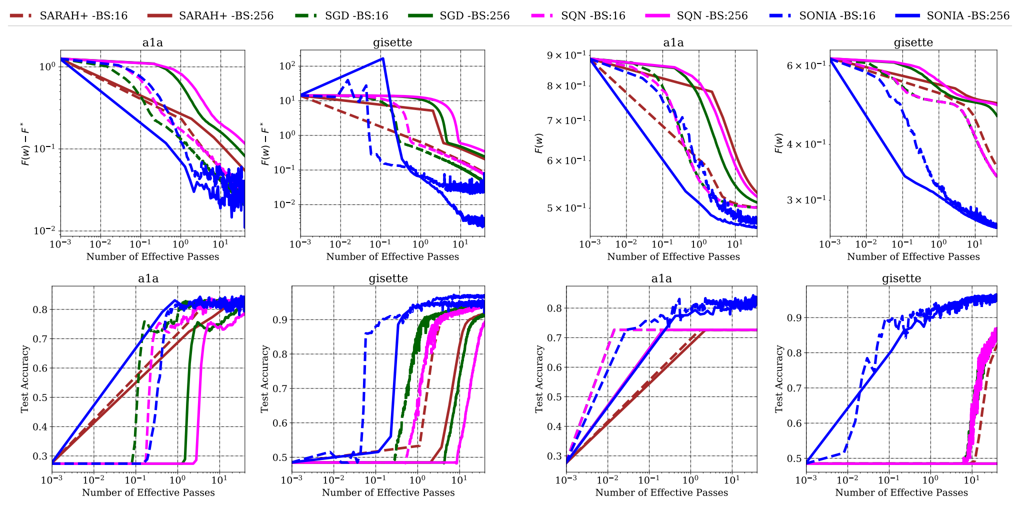
(b) Comparison of objective function ($F(w)$) and Test Accuracy for different algorithms on Non-Linear Least Squares Problems.

Figure 1: Deterministic Problems. Datasets: `a1a`, and `gistte`.

is clear, the performance of `SONIA` is always better than the other methods in the initial stages of training, and the final objective and testing accuracy is comparable to the best method for each problem. We should note that in Figure 1b we report results for a single starting point (as is done in [51]), but report that the performance of `SONIA` was stable with respect to the starting point.

5.2 Stochastic Setting

In the stochastic setting, we compared the performance of `SONIA` to that of SGD [8], SARAH [38] and SQN [12]. We implemented the algorithms with fixed steplength rules, and tuned this parameter as well as the batch size for every problem; see Section C.3 for more details.



(a) Comparison of optimality gap ($F(w) - F^{\star}$) and Test Accuracy for different algorithms on Logistic Regression Problems.

(b) Comparison of objective function ($F(w)$) and Test Accuracy for different algorithms on Non-Linear Least Squares Problems.

Figure 2: Stochastic Problems. Datasets: `a1a` and `gistte`.

Stochastic Logistic Regression Figure 2a shows the performance of the stochastic methods on logistic regression problems. We show results for every method in the small batch regime (16) and in the large batch regime (256). As is clear, the stochastic variant of `SONIA` is competitive with the other methods. We should also mention that as predicted by the theory, using a larger batch size (lower variance in the stochastic gradient approximation) allows for SONIA to convergence to a smaller neighborhood around the optimal solution. For more results see Section D.3.

Stochastic Non-Linear Least Square Figure 2b shows the performance of the stochastic methods on (nonconvex) nonlinear least squares problems. As is clear, the stochastic variant of SONIA outperforms the other methods for all problems reported. Within a very small number of epochs, SONIA is able to achieve high testing accuracy. We attribute the success of SONIA in the stochastic nonconvex regime to the fact that useful curvature information is incorporated in the search direction. For more results see Section D.4.

6 Final Remarks and Future Works

This paper describes a deterministic and stochastic variant of a novel optimization method, SONIA, for empirical risk minimization. The method attempts to bridge the gap between first- and second-order methods by computing a search direction that uses a second-order-type update in one subspace, coupled with a scaled steepest descent step in the orthogonal complement. Numerical results show that the method is efficient in both the deterministic and stochastic settings, and theoretical guarantees confirm that SONIA converges to a stationary point for both strongly convex and nonconvex functions.

Future research directions include: (1) developing adaptive memory variants of SONIA, (2) exploring stochastic SONIA variants that use an adaptive number of samples for gradient/Hessian approximations (following the ideas from [6, 11, 21, 24, 33]), or that employ variance reduced gradients, and (3) a thorough numerical investigation for deep learning problems to test the limits of the methods.

Acknowledgements

This work was partially supported by the U.S. National Science Foundation, under award numbers NSF:CCF:1618717, NSF:CMMI:1663256 and NSF:CCF:1740796.

References

[1] Berahas, A.S., Bollapragada, R., Nocedal, J.: An investigation of newton-sketch and subsampled newton methods. arXiv preprint arXiv:1705.06211 (2017)

[2] Berahas, A.S., Jahani, M., Takáč, M.: Quasi-newton methods for deep learning: Forget the past, just sample. arXiv preprint arXiv:1901.09997 (2019)

[3] Berahas, A.S., Nocedal, J., Takác, M.: A multi-batch l-bfgs method for machine learning. In: Advances in Neural Information Processing Systems, pp. 1055–1063 (2016)

[4] Berahas, A.S., Takáč, M.: A robust multi-batch l-bfgs method for machine learning. Optimization Methods and Software **35**(1), 191–219 (2020)

[5] Bishop, C.M.: Pattern recognition and machine learning. springer (2006)

[6] Bollapragada, R., Byrd, R., Nocedal, J.: Adaptive sampling strategies for stochastic optimization. SIAM Journal on Optimization **28**(4), 3312–3343 (2018)

[7] Bollapragada, R., Byrd, R.H., Nocedal, J.: Exact and inexact subsampled newton methods for optimization. IMA Journal of Numerical Analysis (2016)

[8] Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. Siam Review **60**(2), 223–311 (2018)

[9] Brust, J., Erway, J.B., Marci, R.F.: On solving l-sr1 trust-region subproblems. Computational Optimization and Applications **66**, 245—-266 (2017)

[10] Byrd, R.H., Chin, G.M., Neveitt, W., Nocedal, J.: On the use of stochastic hessian information in optimization methods for machine learning. SIAM Journal on Optimization **21**(3), 977–995 (2011)

[11] Byrd, R.H., Chin, G.M., Nocedal, J., Wu, Y.: Sample size selection in optimization methods for machine learning. Mathematical programming **134**(1), 127–155 (2012)

[12] Byrd, R.H., Hansen, S.L., Nocedal, J., Singer, Y.: A stochastic quasi-newton method for large-scale optimization. SIAM Journal on Optimization **26**(2), 1008–1031 (2016)

[13] Byrd, R.H., Nocedal, J., Schnabel, R.B.: Representations of quasi-newton matrices and their use in limited memory methods. Math. Program. **63**, 129–156 (1994)

[14] Chang, C.C., Lin, C.J.: Libsvm: A library for support vector machines. ACM transactions on intelligent systems and technology (TIST) **2**(3), 1–27 (2011)

[15] Curtis, F.: A self-correcting variable-metric algorithm for stochastic optimization. In: International Conference on Machine Learning, pp. 632–641 (2016)

[16] Dennis Jr, J.E., Moré, J.J.: Quasi-newton methods, motivation and theory. SIAM review **19**(1), 46–89 (1977)

[17] Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research **12**(Jul), 2121–2159 (2011)

[18] Erway, J.B., Griffin, J., Marcia, R.F., Omheni, R.: Trust-region algorithms for training responses: machine learning methods using indefinite hessian approximations. Optimization Methods and Software pp. 1–28 (2019)

[19] Fletcher, R.: Practical Methods of Optimization, 2 edn. John Wiley & Sons, New York (1987)

[20] Fountoulakis, K., Tappenden, R.: A flexible coordinate descent method. Computational Optimization and Applications **70**, 351—-394 (2018)

[21] Friedlander, M.P., Schmidt, M.: Hybrid deterministic-stochastic methods for data fitting. SIAM Journal on Scientific Computing **34**(3), A1380–A1405 (2012)

[22] Friedman, J., Hastie, T., Tibshirani, R.: The elements of statistical learning, vol. 1. Springer series in statistics New York (2001)

[23] Harikandeh, R., Ahmed, M.O., Virani, A., Schmidt, M., J., K., Sallinen, S.: Stop wasting my gradients: Practical svrg. In: Advances in Neural Information Processing Systems, p. 2242–2250 (2015)

[24] Jahani, M., He, X., Ma, C., Mokhtari, A., Mudigere, D., Ribeiro, A., Takáč, M.: Efficient distributed hessian free algorithm for large-scale empirical risk minimization via accumulating sample strategy. arXiv preprint arXiv:1810.11507 (2018)

[25] Jahani, M., Nazari, M., Rusakov, S., Berahas, A.S., Takáč, M.: Scaling up quasi-newton algorithms: Communication efficient distributed sr1. arXiv preprint arXiv:1905.13096 (2019)

[26] Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. In: Advances in neural information processing systems, pp. 315–323 (2013)

[27] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[28] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)

[29] Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Mathematical programming **45**(1-3), 503–528 (1989)

[30] Lu, X.: A study of the limited memory SR1 method in practice. University of Colorado at Boulder (1996)

[31] Ma, C., Gudapati, N.V.C., Jahani, M., Tappenden, R., Takáč, M.: Underestimate sequences via quadratic averaging. arXiv preprint arXiv:1710.03695 (2017)

[32] Martens, J.: Deep learning via hessian-free optimization. In: ICML, vol. 27, pp. 735–742 (2010)

[33] Mokhtari, A., Daneshmand, H., Lucchi, A., Hofmann, T., Ribeiro, A.: Adaptive newton method for empirical risk minimization to statistical accuracy. In: Advances in Neural Information Processing Systems, pp. 4062–4070 (2016)

[34] Mokhtari, A., Ribeiro, A.: Global convergence of online limited memory bfgs. The Journal of Machine Learning Research **16**(1), 3151–3181 (2015)

[35] Nesterov, Y.: Introductory Lectures on Convex Optimization: A Basic Course, *Applied Optimization*, vol. 87. Springer (Originally published by Kluwer Academic Publishers) (2004). Doi:10.1007/978-1-4419-8853-9

[36] Nesterov, Y.: Gradient methods for minimizing composite functions. Mathematical Programming **140**(1), 125–161 (2013)

[37] Nesterov, Y.: Introductory lectures on convex optimization: A basic course, vol. 87. Springer Science & Business Media (2013)

[38] Nguyen, L.M., Liu, J., Scheinberg, K., Takáč, M.: SARAH: a novel method for machine learning problems using stochastic recursive gradient. In: Advances in neural information processing systems, vol. 70, p. 2613–2621 (2017)

[39] Nocedal, J., Wright, S.J.: Numerical Optimization, second edn. Springer Series in Operations Research. Springer (2006)

[40] Paternain, S., Mokhtari, A., Ribeiro, A.: A newton-based method for nonconvex optimization with fast evasion of saddle points. SIAM Journal on Optimization **29**(1), 343–368 (2019)

[41] Recht, B., Re, C., Wright, S., Niu, F.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in neural information processing systems, pp. 693–701 (2011)

[42] Reddi, S.J., Hefny, A., Sra, S., Póczos, B., Smola, A.: Stochastic variance reduction for nonconvex optimization. In: International conference on machine learning, pp. 314–323 (2016)

[43] Richtárik, P., Takáč, M.: Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. Mathematical Programming **144**(1-2), 1–38 (2014)

[44] Richtárik, P., Takáč, M.: Parallel coordinate descent methods for big data optimization. Mathematical Programming **156**(1-2), 433–484 (2016)

[45] Robbins, H., Monro, S.: A stochastic approximation method. The annals of mathematical statistics pp. 400–407 (1951)

[46] Roosta-Khorasani, F., Mahoney, M.W.: Sub-sampled newton methods. Mathematical Programming (2018)

[47] Schmidt, M., Le Roux, N., Bach, F.: Minimizing finite sums with the stochastic average gradient. Mathematical Programming **162**(1-2), 83–112 (2017)

[48] Schraudolph, N.N., Yu, J., Günter, S.: A stochastic quasi-newton method for online convex optimization. In: Artificial Intelligence and Statistics, pp. 436–443 (2007)

[49] Tappenden, R., Richtárik, P., Gondzio, J.: Inexact coordinate descent: Complexity and preconditioning. Journal of Optimization Theory and Applications **170**, 144—-176 (2016)

[50] Woodruff, D.P.: Sketching as a tool for numerical linear algebra. Foundations and Trends® in Theoretical Computer Science **10**(1–2), 1–157 (2014)

[51] Xu, P., Roosta, F., Mahoney, M.W.: Second-order optimization for non-convex machine learning: An empirical study. In: Proceedings of the 2020 SIAM International Conference on Data Mining, pp. 199–207. SIAM (2020)

A Theoretical Results and Proofs

A.1 Assumptions

Assumption 4.1. *F is twice continuously differentiable.*

Assumption 4.4. *There exist positive constants μ and L such that*

$$\mu I \preceq \nabla^2 F(w) \preceq LI, \quad \text{for all } w \in \mathbb{R}^d.$$

Assumption 4.6. *The function $F(w)$ is bounded below by a scalar $\widehat{F}$.*

Assumption 4.7. *The gradients of F are L-Lipschitz continuous for all $w \in \mathbb{R}^d$.*

Assumption 4.9. *There exist a constant γ such that $\mathbb{E}_{\mathcal{I}}[\|\nabla F_{\mathcal{I}}(w) - \nabla F(w)\|^2] \leq \gamma^2$.*

Assumption 4.10. *$\nabla F_{\mathcal{I}}(w)$ is an unbiased estimator of the gradient, i.e., $\mathbb{E}_{\mathcal{I}}[\nabla F_{\mathcal{I}}(w)] = \nabla F(w)$, where the samples $\mathcal{I}$ are drawn independently.*

A.2 Proof of Lemma 3.2

Lemma 3.2. *Let $\mathcal{A}_k := \tilde{V}_k |\Lambda_k|_\epsilon^{-1} \tilde{V}_k^T + \rho_k(I - \tilde{V}_k \tilde{V}_k^T)$. Then the search direction p_k in (3.6) is equivalent to $p_k = -\mathcal{A}_k \nabla F(w_k)$.*

Proof. Define

$$A_k := \tilde{V}_k |\Lambda_k|_\epsilon^{-1} \tilde{V}_k^T, \quad \text{and} \quad A_k^\perp := \rho_k(I - \tilde{V}_k \tilde{V}_k^T), \tag{A.1}$$

so that by (3.8) and (A.1),

$$\mathcal{A}_k = A_k + A_k^\perp. \tag{A.2}$$

By (3.5), $g_k^\perp = (I - \tilde{V}\tilde{V}^T) g_k^\perp$. Then

$$\begin{aligned}
p_k &\overset{(3.6)+(A.1)}{=} -A_k g_k - A_k^\perp g_k^\perp \\
&\overset{(3.5)}{=} -A_k g_k - A_k^\perp (\nabla F(w_k) - g_k) \\
&\overset{(3.5)}{=} -A_k \tilde{V}_k \tilde{V}_k^T \nabla F(w_k) - A_k^\perp (I - \tilde{V}_k \tilde{V}_k^T) \nabla F(w_k) \\
&\overset{(A.1)}{=} -\tilde{V}_k |\Lambda_k|_\epsilon^{-1} \tilde{V}_k^T \tilde{V}_k \tilde{V}_k^T \nabla F(w_k) - \rho_k (I - \tilde{V}_k \tilde{V}_k^T)(I - \tilde{V}_k \tilde{V}_k^T) \nabla F(w_k) \\
&= -\tilde{V}_k |\Lambda_k|_\epsilon^{-1} \tilde{V}_k^T \nabla F(w_k) - \rho_k (I - \tilde{V}_k \tilde{V}_k^T) \nabla F(w_k) \\
&\overset{(A.1)+(A.2)}{=} -\mathcal{A}_k \nabla F(w_k).
\end{aligned} \tag{A.3}$$

□

A.3 Proof of Lemma 4.2

Lemma 4.2. *The matrix $\mathcal{A}_k$ in (A.2) is positive definite.*

Proof. Let $y \in \mathbb{R}^d$ be any nonzero vector. One can then write $y = y_1 + y_2$, where $y_1 \in \text{range}\{\tilde{V}\}$ and $y_2 \in \text{range}\{\tilde{V}\}^\perp$. Note that this implies that $\tilde{V}^T y_2 = 0$ and $(I - \tilde{V}\tilde{V}^T) y_1 = 0$. Also notice that $I - \tilde{V}\tilde{V}^T$ is a projection matrix, thus $(I - \tilde{V}\tilde{V}^T)^2 = I - \tilde{V}\tilde{V}^T$.

Let $D_1 = |\Lambda_k|_\epsilon^{-1}$ and $D_2 = \rho I_d$, where $\rho > 0$. Thus, we have

$$\begin{aligned}
y^T \mathcal{A}_k y &= (y_1 + y_2)^T \mathcal{A}_k (y_1 + y_2) \\
&= (y_1 + y_2)^T \left(A_k + A_k^\perp\right)(y_1 + y_2) \\
&= (y_1 + y_2)^T \left(\tilde{V}_k D_1 \tilde{V}_k^T + D_2(I - \tilde{V}\tilde{V}^T)\right)(y_1 + y_2) \\
&= (y_1 + y_2)^T \left(\tilde{V}_k D_1 \tilde{V}_k^T + (I - \tilde{V}\tilde{V}^T) D_2 (I - \tilde{V}\tilde{V}^T)\right)(y_1 + y_2) \\
&= y_1^T \tilde{V}_k D_1 \tilde{V}_k^T y_1 + y_2^T (I - \tilde{V}\tilde{V}^T) D_2 (I - \tilde{V}\tilde{V}^T) y_2 \\
&> 0,
\end{aligned}$$

The final strict inequality is due to the fact that since y is a nonzero vector, this implies that at least one of the two vectors y_1 or y_2 are also nonzero, the specific decompositions of y and the fact that both matrices D_1 and D_2 are positive definite. □

A.4 Proof of Lemma 4.3

Lemma 4.3. *Let Assumption 4.1 hold, and let ρ_k satisfy (3.7) for all $k \geq 0$. Then there exist constants $0 < \mu_1 \leq \mu_2$ such that the sequence of matrices $\{\mathcal{A}_k\}_{k\geq 0}$ generated by Algorithm 1 satisfies,*

$$\mu_1 I \preceq \mathcal{A}_k \preceq \mu_2 I, \qquad \textit{for } k = 0, 1, 2, \ldots.$$

Proof. By (A.2), $\mathcal{A}_k = A_k + A_k^{\perp}$, so we can write

$$\begin{aligned} \mathcal{A}_k &= A_k + A_k^{\perp} \\ &= \tilde{V}_k |\Lambda_k|_{\epsilon}^{-1} \tilde{V}_k^T + \rho_k (I - \tilde{V}_k \tilde{V}_k^T) \\ &= \tilde{V}_k (|\Lambda_k|_{\epsilon}^{-1} - \rho_k I) \tilde{V}_k^T + \rho_k I \\ &\overset{(3.7)}{\succeq} \rho_k I. \end{aligned}$$

Furthermore, $|\Lambda_k|_{\epsilon}^{-1} \preceq \frac{1}{\epsilon} I$ so that $\mathcal{A}_k \preceq \frac{1}{\epsilon} I + \rho_k I \preceq \frac{2}{\epsilon}$. Defining $\mu_1 := \min_{0\leq i\leq k} \rho_i$ and $\mu_2 := \frac{2}{\epsilon}$, and noting that $0 < \mu_1 \leq \mu_2$, gives the result.

□

A.5 Proof of Theorem 4.5

Theorem 4.5. *Suppose that Assumptions 4.1 and 4.4 hold, and let $F^\star = F(w^\star)$, where $w^\star$ is the minimizer of F. Let $\{w_k\}$ be the iterates generated by Algorithm 1, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for all $k \geq 0$,*

$$F(w_k) - F^\star \leq \left(1 - \alpha\mu\mu_1\right)^k [F(w_0) - F^\star].$$

Proof. We have that

$$\begin{aligned} F(w_{k+1}) &= F(w_k - \alpha \mathcal{A}_k \nabla F(w_k)) \\ &\leq F(w_k) + \nabla F(w_k)^T (-\alpha \mathcal{A}_k \nabla F(w_k)) + \frac{L}{2} \|\alpha \mathcal{A}_k \nabla F(w_k)\|^2 \\ &\leq F(w_k) - \alpha\mu_1 \|\nabla F(w_k)\|^2 + \frac{\alpha^2 \mu_2^2 L}{2} \|\nabla F(w_k)\|^2 \\ &= F(w_k) - \alpha \left(\mu_1 - \alpha \frac{\mu_2^2 L}{2}\right) \|\nabla F(w_k)\|^2 \\ &\leq F(w_k) - \alpha \frac{\mu_1}{2} \|\nabla F(w_k)\|^2, \end{aligned} \tag{A.4}$$

where the first inequality is due to Assumption 4.4, the second inequality arises as a consequence of Lemma 4.3 and the last inequality is due to the choice of the steplength. By strong convexity [37], we have $2\mu(F(w) - F^\star) \leq \|\nabla F(w)\|^2$, and thus

$$F(w_{k+1}) \leq F(w_k) - \alpha\mu\mu_1 (F(w_k) - F^\star).$$

Subtracting $F^\star$ from both sides,

$$F(w_{k+1}) - F^\star \leq (1 - \alpha\mu\mu_1)(F(w_k) - F^\star).$$

Recursive application of the above inequality yields the desired result. □

A.6 Proof of Theorem 4.8

Theorem 4.8. *Suppose that Assumptions 4.1, 4.6 and 4.7 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 1, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for any $T > 1$,*

$$\frac{1}{T} \sum_{k=0}^{T-1} \|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \hat{F}]}{\alpha\mu_1 T} \xrightarrow{T\to\infty} 0.$$

Proof. We start with (A.4)

$$F(w_{k+1}) \leq F(w_k) - \alpha \frac{\mu_1}{2} \|\nabla F(w_k)\|^2.$$

Summing both sides of the above inequality from $k = 0$ to $T - 1$,

$$\sum_{k=0}^{T-1}(F(w_{k+1}) - F(w_k)) \leq -\sum_{k=0}^{T-1}\alpha\frac{\mu_1}{2}\|\nabla F(w_k)\|^2.$$

The left-hand-side of the above inequality is a telescopic sum and thus,

$$\sum_{k=0}^{T-1}[F(w_{k+1}) - F(w_k)] = F(w_T) - F(w_0) \geq \widehat{F} - F(w_0),$$

where the inequality is due to $\hat{F} \leq F(w_T)$ (Assumption 4.6). Using the above, we have

$$\sum_{k=0}^{T-1}\|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1}. \tag{A.5}$$

Dividing (A.5) by T we conclude

$$\frac{1}{T}\sum_{k=0}^{T-1}\|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1 T}.$$

□

A.7 Proof of Theorem 4.11

Theorem 4.11. *Suppose that Assumptions 4.1, 4.4, 4.9 and 4.10 hold, and let $F^\star = F(w^\star)$, where $w^\star$ is the minimizer of F. Let $\{w_k\}$ be the iterates generated by Algorithm 1, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for all $k \geq 0$,*

$$\mathbb{E}[F(w_k) - F^\star] \leq (1 - \alpha\mu_1\lambda)^k\left(F(w_0) - F^\star - \frac{\alpha\mu_2^2\gamma^2 L}{2\mu_1\lambda}\right) + \frac{\alpha\mu_2^2\gamma^2 L}{2\mu_1\lambda}.$$

Proof. We have that

$$\begin{aligned}
F(w_{k+1}) &= F(w_k - \alpha\mathcal{A}_k\nabla F_{\mathcal{I}_k}(w_k)) \\
&\leq F(w_k) + \nabla F(w_k)^T(-\alpha\mathcal{A}_k\nabla F_{\mathcal{I}_k}(w_k)) + \frac{L}{2}\|\alpha\mathcal{A}_k\nabla F_{\mathcal{I}_k}(w_k)\|^2 \\
&\leq F(w_k) - \alpha\nabla F(w_k)^T\mathcal{A}_k\nabla F_{\mathcal{I}_k}(w_k) + \frac{\alpha^2\mu_2^2 L}{2}\|\nabla F_{\mathcal{I}_k}(w_k)\|^2
\end{aligned}$$

where the first inequality is due to Assumption 4.4 and the second inequality is due to Lemma 4.3. Taking the expectation over the sample $\mathcal{I}_k$, we have

$$\begin{aligned}
\mathbb{E}_{\mathcal{I}_k}[F(w_{k+1})] &\leq F(w_k) - \alpha\mathbb{E}_{\mathcal{I}_k}[\nabla F(w_k)^T\mathcal{A}_k\nabla F_{\mathcal{I}_k}(w_k)] + \frac{\alpha^2\mu_2^2 L}{2}\mathbb{E}_{\mathcal{I}_k}[\|\nabla F_{\mathcal{I}_k}(w_k)\|^2] \\
&= F(w_k) - \alpha\nabla F(w_k)^T\mathcal{A}_k\nabla F(w_k) + \frac{\alpha^2\mu_2^2 L}{2}\mathbb{E}_{\mathcal{I}_k}[\|\nabla F_{\mathcal{I}_k}(w_k)\|^2] \\
&\leq F(w_k) - \alpha\left(\mu_1 - \frac{\alpha\mu_2^2 L}{2}\right)\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\gamma^2 L}{2} \\
&\leq F(w_k) - \frac{\alpha\mu_1}{2}\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\gamma^2 L}{2},
\end{aligned} \tag{A.6}$$

where the second inequality is due to Lemma 4.3 and Assumption 4.9 and the third inequality is due to the choice of the step length. Since F is strongly convex [37], we have

$$\mathbb{E}_{\mathcal{I}_k}[F(w_{k+1})] \leq F(w_k) - \alpha\mu_1\lambda(F(w_k) - F^\star) + \frac{\alpha^2\mu_2^2\gamma^2 L}{2}.$$

Taking the total expectation over all batches $\mathcal{I}_0$, $\mathcal{I}_1$, $\mathcal{I}_2$,... and all history starting with w_0, and subtracting $F^\star$ from both sides, we have

$$\mathbb{E}[F(w_{k+1}) - F^\star] \leq (1 - \alpha\mu_1\lambda)\,\mathbb{E}[F(w_k) - F^\star] + \frac{\alpha^2\mu_2^2\gamma^2 L}{2},$$

where $0 \leq (1-\alpha\mu_1\lambda) \leq 1$ by the step length choice. Subtracting $\frac{\alpha\mu_2^2\gamma^2 L}{2\mu_1\lambda}$ from both sides yields

$$\mathbb{E}[F(w_{k+1}) - F^\star] - \frac{\alpha\mu_2^2\gamma^2 L}{2\mu_1\lambda} \leq (1-\alpha\mu_1\lambda)\left(\mathbb{E}[F(w_k) - F^\star] - \frac{\alpha\mu_2^2\gamma^2 L}{2\mu_1\lambda}\right).$$

Recursive application of the above completes the proof. □

A.8 Proof of Theorem 4.12

Theorem 4.12. *Suppose that Assumptions 4.1, 4.6, 4.7, 4.9 and 4.10 hold, and let $F^\star = F(w^\star)$, where $w^\star$ is the minimizer of F. Let $\{w_k\}$ be the iterates generated by Algorithm 1, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for all $k \geq 0$,*

$$\mathbb{E}\left[\frac{1}{T}\sum_{k=0}^{T-1}\|\nabla F(w_k)\|^2\right] \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1 T} + \frac{\alpha\mu_2^2\gamma^2 L}{\mu_1} \xrightarrow{T\to\infty} \frac{\alpha\mu_2^2\gamma^2 L}{\mu_1}.$$

Proof. Starting with (A.6) and taking the total expectation over all batches $\mathcal{I}_0$, $\mathcal{I}_1$, $\mathcal{I}_2$,... and all history starting with w_0

$$\mathbb{E}[F(w_{k+1}) - F(w_k)] \leq -\frac{\alpha\mu_1}{2}\mathbb{E}[\|\nabla F(w_k)\|^2] + \frac{\alpha^2\mu_2^2\gamma^2 L}{2}.$$

Summing both sides of the above inequality from $k=0$ to $T-1$,

$$\begin{aligned}\sum_{k=0}^{T-1}\mathbb{E}[F(w_{k+1}) - F(w_k)] &\leq -\frac{\alpha\mu_1}{2}\sum_{k=0}^{T-1}\mathbb{E}[\|\nabla F(w_k)\|^2] + \frac{\alpha^2\mu_2^2\gamma^2 LT}{2} \\ &= -\frac{\alpha\mu_1}{2}\mathbb{E}\left[\sum_{k=0}^{T-1}\|\nabla F(w_k)\|^2\right] + \frac{\alpha^2\mu_2^2\gamma^2 LT}{2}.\end{aligned}$$

The left-hand-side of the above inequality is a telescopic sum and thus,

$$\sum_{k=0}^{T-1}\mathbb{E}\left[F(w_{k+1}) - F(w_k)\right] = \mathbb{E}[F(w_T)] - F(w_0) \geq \widehat{F} - F(w_0),$$

where the inequality is due to $\hat{F} \leq F(w_T)$ (Assumption 4.6). Using the above, we have

$$\mathbb{E}\left[\sum_{k=0}^{T-1}\|\nabla F(w_k)\|^2\right] \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1} + \frac{\alpha\mu_2^2\gamma^2 LT}{\mu_1}.$$

Dividing by T we conclude completes the proof. □

B Efficient Hessian-Matrix Computations

The SONIA algorithm requires the computation of Hessian-maxtrix products for the construction of the curvature pairs. In this section, we describe how one can efficiently compute Hessian-matrix products for the problems studied in this paper. Moreover, we describe an efficient distributed algorithm for computing curvature pairs; see [25] for more details.

Assume that $\odot$ is the operator for component-wise product, $*$ is the standard multiplication of matrices, $\mathbb{1}_n$ is the vector of ones with size $1 \times n$, X is the feature matrix and Y is the label matrix.

In the following, firstly, we present the efficient calculation of objective function, gradient and Hessian-matrix products for logistic regression problems. Next, we do the same for non-linear least square problems. Moreover, we describe a simple distributed methodology for computing Hessian-matrix products. Finally, further discussion is provided for the efficient computation of Hessian-matrix products.

B.1 Logistic Regression

The objective function, gradient, Hessian and Hessian-matrix product for logistic regression problems are calculated efficiently as follows:

$$F(w) = \frac{1}{n}\left(\mathbb{1}_n * \log\left(1 + e^{-Y \odot X^T w}\right)\right) + \frac{\lambda}{2}\|w\|^2, \tag{B.1}$$

$$\nabla F(w) = \frac{1}{n}\left(X^T * \frac{-Y \odot e^{-Y \odot X^T w}}{1 + e^{-Y \odot X^T w}}\right) + \lambda w, \tag{B.2}$$

$$\nabla^2 F(w) = \frac{1}{n}\left(X^T * \left[\frac{Y \odot Y \odot e^{-Y \odot X^T w}}{\left(1 + e^{-Y \odot X^T w}\right)^2}\right] \odot X\right) + \lambda I_d, \tag{B.3}$$

$$\nabla^2 F(w) * S = \frac{1}{n}\left(X^T * \left[\frac{Y \odot Y \odot e^{-Y \odot X^T w}}{\left(1 + e^{-Y \odot X^T w}\right)^2}\right] \odot X * S\right) + \lambda S. \tag{B.4}$$

B.2 Non-Linear Least Squares

The objective function, gradient, Hessian and Hessian-matrix product for non-linear least square are calculated efficiently as follows (where $\phi(z) = \frac{1}{1+e^{-z}}$):

$$F(w) = \frac{1}{2n}\|Y - \phi(X^T w)\|^2 \tag{B.5}$$

$$\nabla F(w) = \frac{1}{n}\left(-X^T * [\phi(X^T w) \odot (1 - \phi(X^T w)) \odot (Y - \phi(X^T w))]\right) \tag{B.6}$$

$$\begin{aligned}\nabla^2 F(w) = &\frac{1}{n}\left(-X^T * \left[\phi(X^T w) \odot (1 - \phi(X^T w)) \odot (Y - 2(1+Y) \odot \phi(X^T w) \;+\right.\right.\\ &\left.\left. 3\phi(X^T w) \odot \phi(X^T w))\right] \odot X\right)\end{aligned} \tag{B.7}$$

$$\begin{aligned}\nabla^2 F(w) * S = &\frac{1}{n}\left(-X^T * \left[\phi(X^T w) \odot (1 - \phi(X^T w)) \odot (Y - 2(1+Y) \odot \phi(X^T w) \;+\right.\right.\\ &\left.\left. 3\phi(X^T w) \odot \phi(X^T w))\right] \odot X * S\right)\end{aligned} \tag{B.8}$$

Based on the above equations, one can note that the cost of objective function and gradient computations is $\mathcal{O}(nd)$ due to the calculation of $X^T w$. The cost of Hessian-matrix products is $\mathcal{O}(mnd)$, and the cost is dominated by the calculation of $X * S$. Furthermore, by considering the fact that `SONIA` was developed for the regime where $m \ll d, n$, the effective cost of Hessian-matrix product is $\mathcal{O}(nd)$. In summary, the cost of Hessian-matrix products is similar to the cost of objective function and gradient evaluations.

B.3 Distributed Algorithm

For the cases where the Hessian has a compact representation (e.g., logistic regression and non-linear least square problems), we showed that the Hessian-matrix products can be efficiently calculated. However, this is not always the case. In the rest of this section, we justify that for general non-linear problems, such as deep learning, the Hessian-matrix product can be efficiently computed in a distributed environment. By following the study in [25], one can note that in order to construct curvature information (S_k, Y_k), a Hessian-matrix calculation is required in order to form $Y_k = \nabla^2 F(w_k) S_k$. The aforementioned Hessian-matrix products can be calculated efficiently in master-worker framework, summarized in Algorithm 2. Each worker has a portion of the dataset, performs local computations, and then reduces the locally calculated information to the master node. This method is matrix-free (i.e., the Hessian approximation is never explicitly constructed).

Algorithm 2 Construct new (S_k, Y_k) curvature pairs

```
Input: $w_k$ (iterate), $m$ (memory), $S_k = [\,]$, $Y_k = [\,]$ (curvature pair containers).
Master Node:                                              Worker Nodes ($i = 1, 2, \dots, \mathcal{K}$):
1: Broadcast: $S_k$ and $w_k$                  $\longrightarrow$        Compute $Y_{k,i} = \nabla^2 F_i(w_k) S_k$
2: Reduce: $Y_{k,i}$ to $Y_k$ and $S_k^T Y_{k,i}$ to $Y_k^T S_k$    $\longleftarrow$    Compute $S_k^T Y_{k,i}$
Output: $S^T Y$, $Y_k$
```

C Method and Problem Details

C.1 Table of Algorithms

In this section, we summarize the implemented algorithms in Section 5 in the Table 2.

Algorithm	Description and Reference
NEST+	Algorithm described in [35, Chapter 2] with adaptive Lipschitz constant
L-BFGS	Limited memory BFGS [29]
L-SR1	Limited memory SR1 [30]
Newton-CG-TR	Newton method with conjugate gradient (CG) utilizing trust region (TR)[39]
Newton-CG-LS	Newton method with conjugate gradient (CG) utilizing line search (LS) [39]
SARAH+	Practical variant of SARAH [38]
SQN	Stochastic Quasi-Newton [12]
SGD	Stochastic gradient method [45]
GD	Gradient descent
ASUESA	Accelerated Smooth Underestimate Sequence Algorithm with adaptive Lipschitz constant [31]
SONIA	**S**ymmetric bl**O**ckwise tru**N**cated optim**I**nation **A**lgorithm

Table 2: Description of implemented algorithms

In order to find $w^\star$ for the strongly convex problems, we used the ASUESA algorithm [31]. ASUESA constructs a sequence of lower bounds, and at each iteration of ASUESA the gap between the aforementioned lower bounds and objective function goes to zero at an optimal linear rate. One of the most important advantages of ASUESA is the natural stopping condition, which provides the user with a certificate of optimality. In other words, when the gap between objective function and the lower bounds is small enough, ASUESA is close enough to the optimal solution $w^\star$.

C.2 Problem Details

Table 3: Summary of two binary classification datasets and two multi-labels classification datasets

Dataset	# of samples	# of features	# of categories
`rcv1`	20,242	47,326	2
`gisette`	6000	5,000	2
`a1a`	1605	119	2
`ijcnn1`	35000	22	2

C.3 Implementation Details

In the following sections, we describe the way that the algorithms (Table 2) were implemented and tuned. In order to have fair comparisons, we considered the same number of hyper-parameter choices for a given problem for the algorithms that needed tuning. We consider the set of hyper-parameters for each single algorithm for the optimization problems discussed in Section 5.

C.3.1 Deterministic Strongly Convex Case

- **GD**: No tuning is needed. The learning rate is chosen by Armijo backtracking line search.
- **L-BFGS**: Memory is chosen from the set $\{4, 16, 32, 64\}$; $\epsilon_{\text{L-BFGS}} = 10^{-8}$ (ϵ for checking the curvature condition in L-BFGS method) and the learning rate is chosen by Armijo backtracking line search.
- **L-SR1**: Memory is chosen from the set $\{4, 16, 32, 64\}$ and $\epsilon_{\text{L-SR1}} = 10^{-8}$($\epsilon$ for checking the curvature condition in L-SR1 method). The search direction is calculated by trust region solver by the default setting reported in Algorithm 6.1 in [39].
- **Newton-CG-LS**: The learning rate is chosen by Armijo backtracking line search. The Newton system is solved according to Algorithm 7.1 in [39].
- **NEST+**: For adaptive Lipschitz constant, we set the parameters $d_{\text{decrease}} \in \{1.1, 2\}$ and $u_{\text{increase}} \in \{1.1, 2\}$ according to the Algorithm 4.1 in [36].

- **SONIA**: Memory is chosen from the set $\{4, 16, 32, 64\}$; $\epsilon_{\text{SONIA}} = 10^{-5}$ (truncated ϵ) and the learning rate is chosen by Armijo backtracking line search.

C.3.2 Deterministic Non-convex Case

- **GD**: No tuning is needed. The learning rate is chosen by Armijo backtracking line search.
- **L-BFGS**: Memory is chosen from the set $\{4, 16, 32, 64\}$; $\epsilon_{\text{L-BFGS}} = 10^{-8}$($\epsilon$ for checking the curvature condition in L-BFGS method) and the learning rate is chosen by Armijo backtracking line search.
- **L-SR1**: Memory is chosen from the set $\{4, 16, 32, 64\}$ and $\epsilon_{\text{L-SR1}} = 10^{-8}$($\epsilon$ for checking the curvature condition in L-SR1 method). The search direction is calculated by trust region solver by the default setting reported in Algorithm 6.1 in [39].
- **Newton-CG-TR**: The Newton system is solved according to CG-Steihaug method (Algorithm 7.2) in [39].
- **SONIA**: Memory is chosen from the set $\{4, 16, 32, 64\}$; $\epsilon_{\text{SONIA}} = 10^{-5}$ (truncated ϵ) and the learning rate is chosen by Armijo backtracking line search.

C.3.3 Stochastic Strongly Convex Case

- **SGD**: The learning rate is chosen from the set $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$ and the batch size is from the set $\{16, 256\}$.
- **SQN**: The learning rate is chosen from the set $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$ and the batch size is from the set $\{16, 256\}$. Moreover, we set $L_{\text{SQN}} = 1$, meaning that it checks to accept/reject the curvature information at every iteration. Also, we set $\epsilon_{\text{SQN}} = 10^{-8}$ (ϵ for checking the curvature condition in SQN method). By checking the sensitivity analysis of SQN w.r.t different memories, we notice SQN is not sensitive to the choice of memory, then we set memory $m = 64$.
- **SARAH+**: The learning rate is chosen from the set $\{4, 2, 1, 0.1, 0.01, 0.001\}$[7]. Also, we consider the batch sizes 16 and 256.
- **SONIA**: The learning rate is chosen from the set $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$ and the batch size is from the set $\{16, 256\}$. Also, we set $\epsilon_{\text{SONIA}} = 10^{-5}$ (truncated ϵ) and memory $m = 64$.

C.3.4 Stochastic Non-convex Case

- **SGD**: The learning rate is chosen from the set $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$.
- **SQN**: The tuning for this case is similar to the stochastic strongly convex case. The candidate learning rate set is $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$ and the batch size is from the set $\{16, 256\}$. Moreover, we set $L_{\text{SQN}} = 1$. Also, we set $\epsilon_{\text{SQN}} = 10^{-8}$ (ϵ for checking the curvature condition in SQN method) and memory $m = 64$.
- **SARAH+**: The learning rate is chosen from the set $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$. Also, we consider the batch sizes 16 and 256.
- **SONIA**: Similar to the previous case, the learning rate is chosen from the set $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$ and the batch size is from the set $\{16, 256\}$. Also, we set $\epsilon_{\text{SONIA}} = 10^{-5}$ (truncated ϵ) and memory $m = 64$.

C.3.5 Required Hardware and Software

All the algorithms are implemented in Python 3 and ran on Intel(R) Xeon(R) CPUs.

[7]The reason for this choice is that the learning for SARAH is selected to be approximately by $\frac{1}{L}$ where L is the Lipschitz constant.

D Additional Numerical Experiments

In this section, we present additional numerical results in order to compare the performance of `SONIA` with the state-of-the-art first- and second-order methods described in Table 2 on the datasets reported in Table 3.

1. Section D.1: deterministic strongly convex case (ℓ_2 regularized logistic regression).
2. Section D.2: deterministic nonconvex case (non-linear least squares).
3. Section D.3: stochastic strongly convex case (ℓ_2 regularized logistic regression).
4. Section D.4: stochastic nonconvex (non-linear least squares).

Moreover, we investigated the sensitivity of `SONIA` to its associated hyper-parameters (i.e., the memory size m, and the truncation parameter ϵ). Sections D.1.1 and D.1.2 show sensitivity results for deterministic logistic regression problems. The key take-aways are that `SONIA` is robust with respect to m and ϵ (the variation in performance is small for different choices of the hyper-parameters) and under reasonable choices of these hyper-parameters ($m \in [0, d]$ and $\epsilon > 0$), the `SONIA` algorithm always converges, albeit at a slower rate for some choices. This of course is in contrast to certain methods that may diverge is the hyper-parameters are not chosen appropriately (e.g., the learning rate for the SGD method).

D.1 Additional Numerical Experiments: Deterministic Strongly Convex Functions

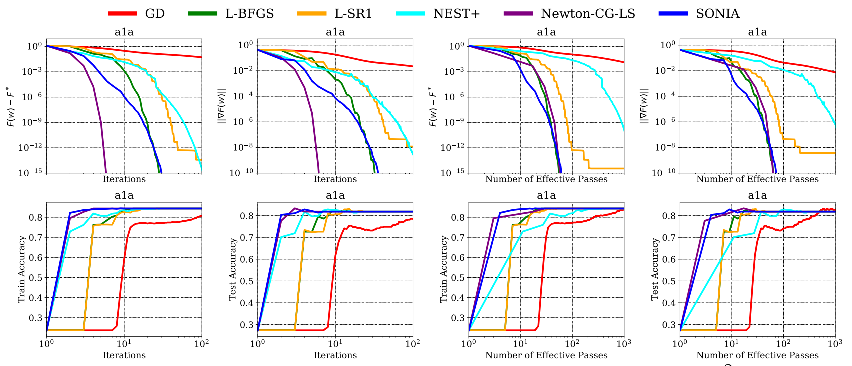


Figure 3: `a1a`: Deterministic Logistic Regression with $\lambda = 10^{-3}$.

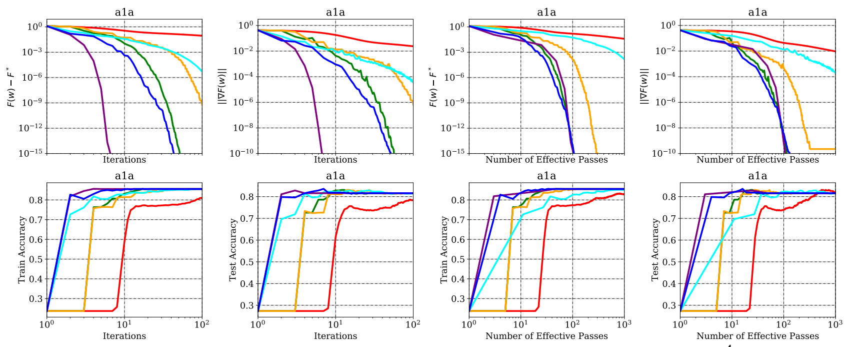


Figure 4: `a1a`: Deterministic Logistic Regression with $\lambda = 10^{-4}$.

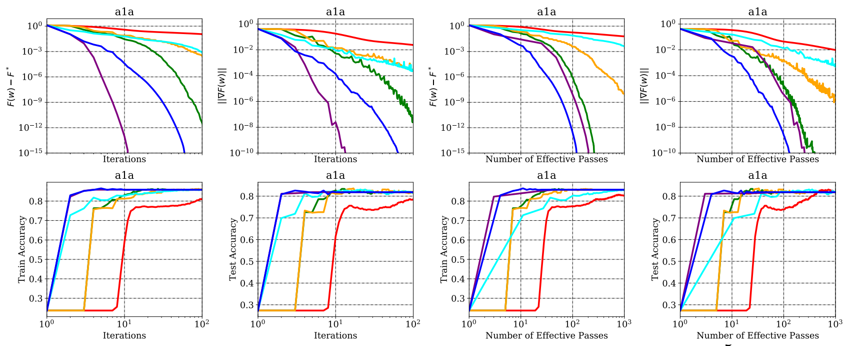


Figure 5: `a1a`: Deterministic Logistic Regression with $\lambda = 10^{-5}$.

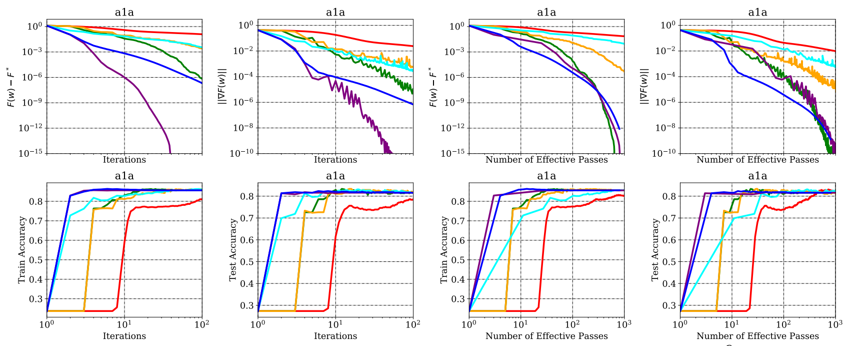


Figure 6: `a1a`: Deterministic Logistic Regression with $\lambda = 10^{-6}$.

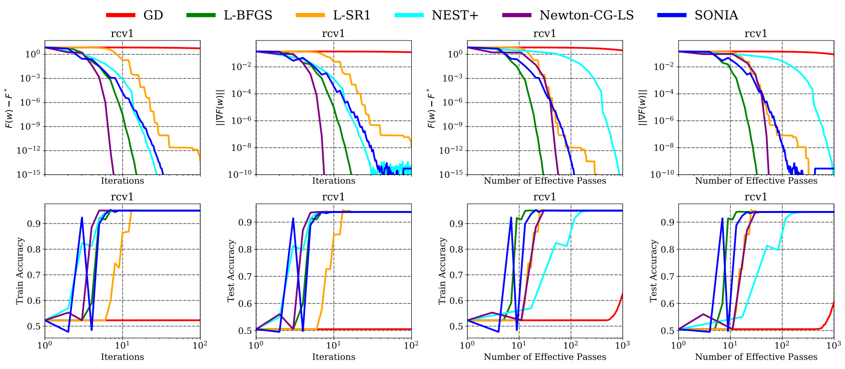


Figure 7: `rcv1`: Deterministic Logistic Regression with $\lambda = 10^{-3}$.

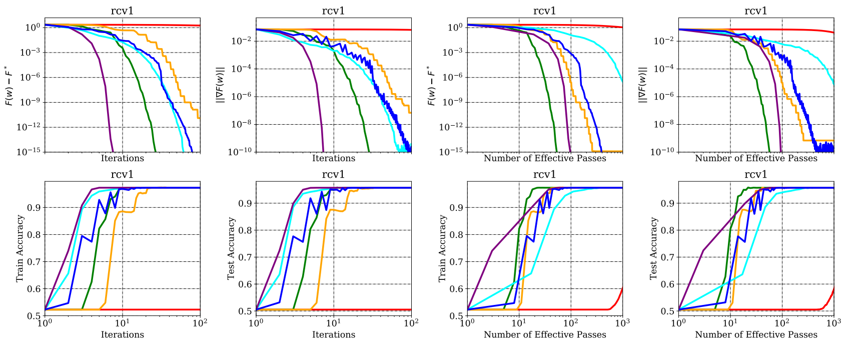


Figure 8: `rcv1`: Deterministic Logistic Regression with $\lambda = 10^{-4}$.

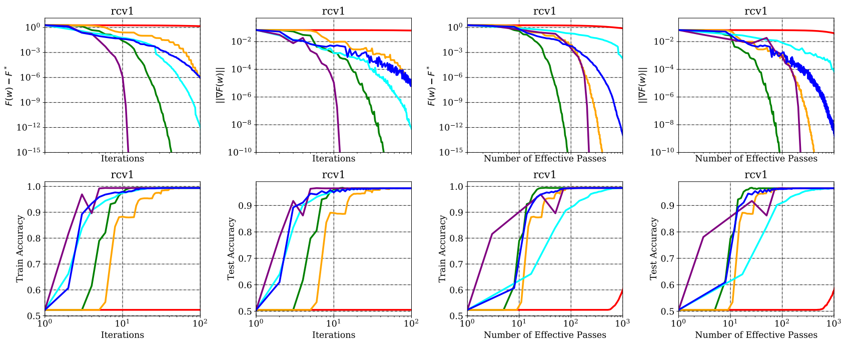


Figure 9: `rcv1`: Deterministic Logistic Regression with $\lambda = 10^{-5}$.

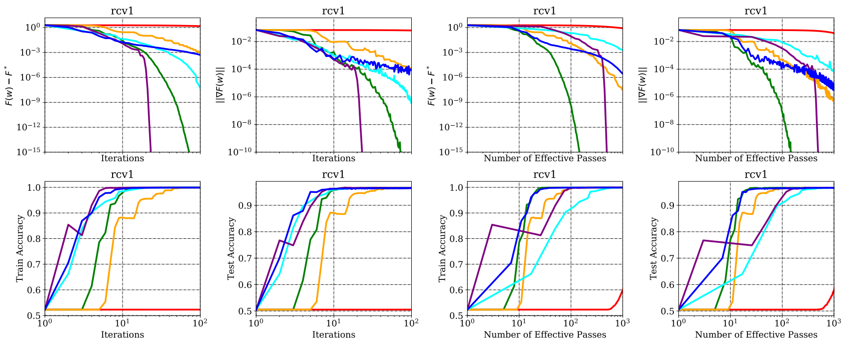


Figure 10: `rcv1`: Deterministic Logistic Regression with $\lambda = 10^{-6}$.

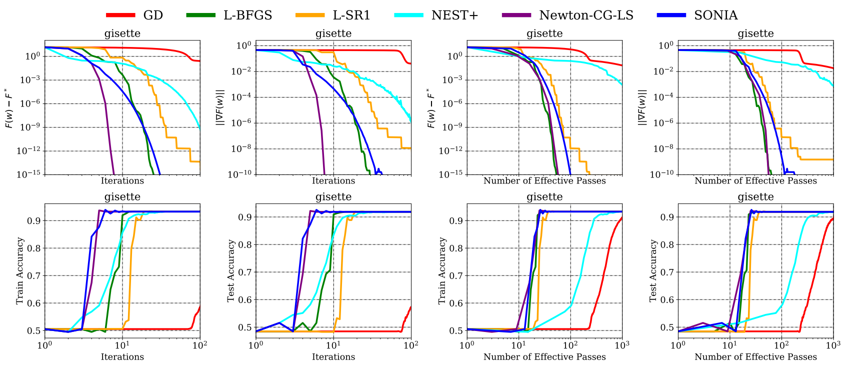


Figure 11: `gisette`: Deterministic Logistic Regression with $\lambda = 10^{-3}$.

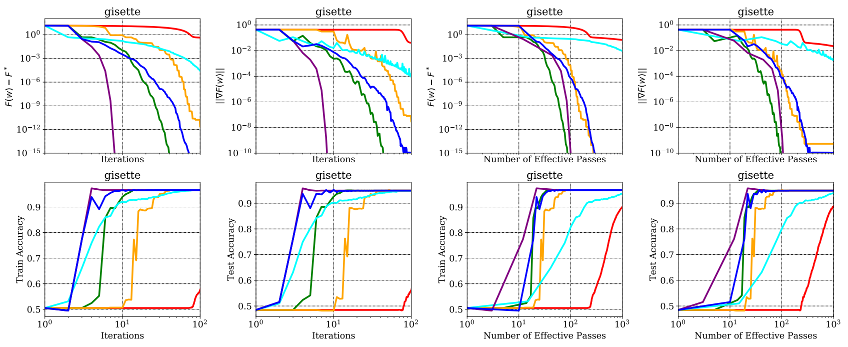


Figure 12: `gisette`: Deterministic Logistic Regression with $\lambda = 10^{-4}$.

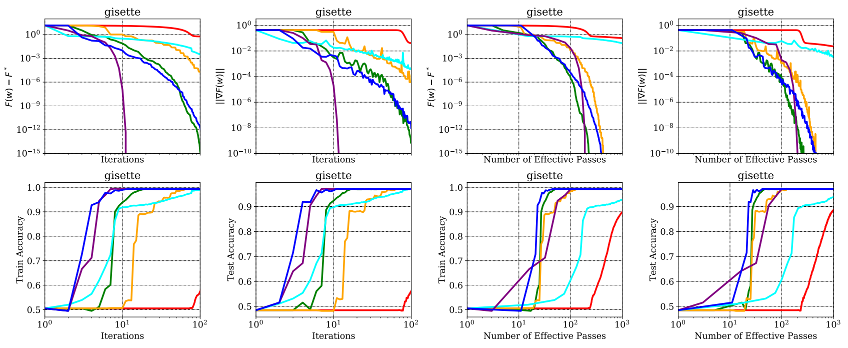


Figure 13: `gisette`: Deterministic Logistic Regression with $\lambda = 10^{-5}$.

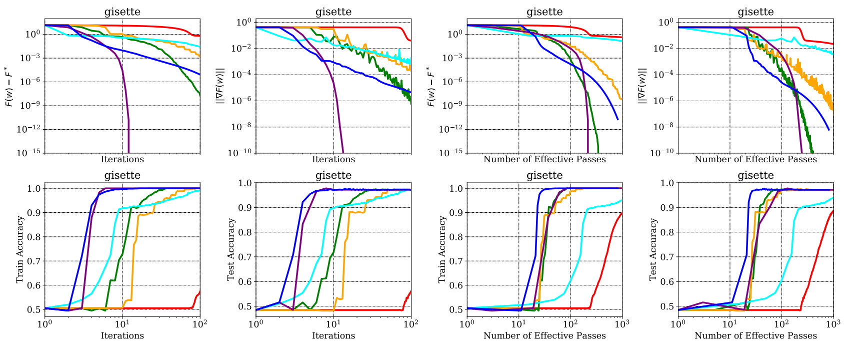


Figure 14: `gisette`: Deterministic Logistic Regression with $\lambda = 10^{-6}$.

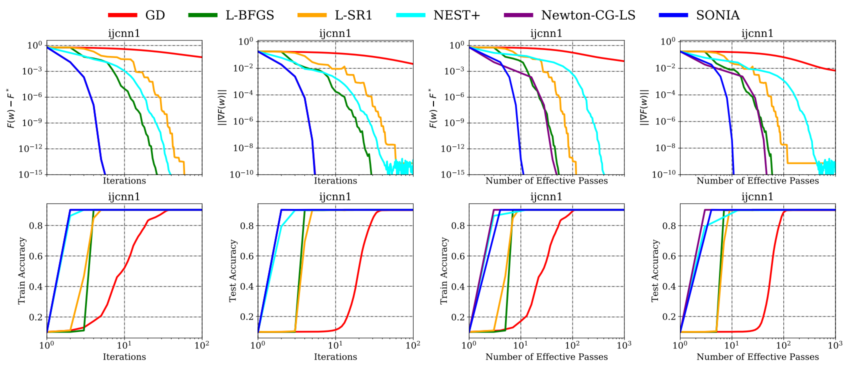


Figure 15: `ijcnn1`: Deterministic Logistic Regression with $\lambda = 10^{-3}$.

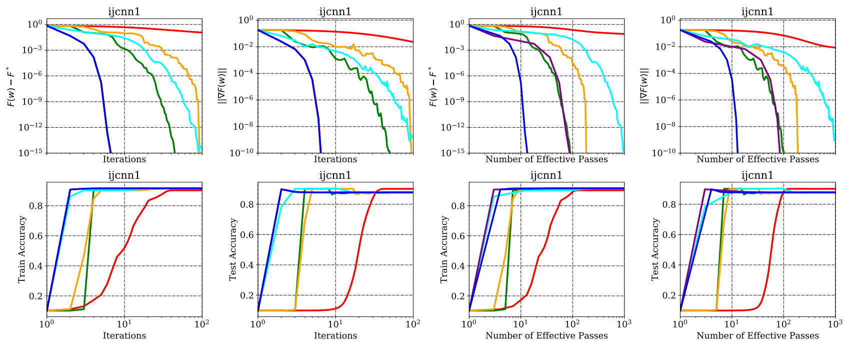


Figure 16: `ijcnn1`: Deterministic Logistic Regression with $\lambda = 10^{-4}$.

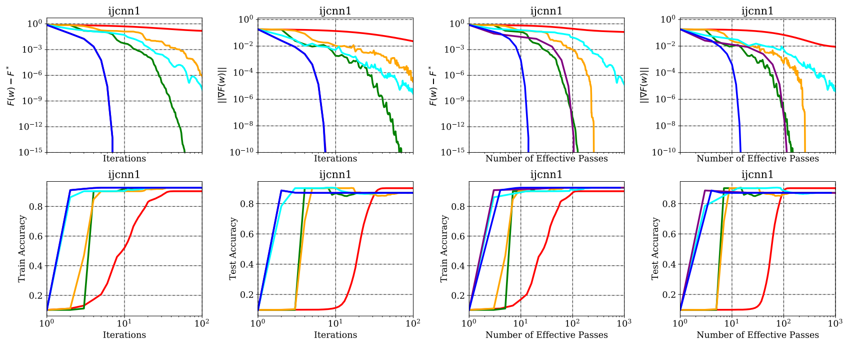


Figure 17: `ijcnn1`: Deterministic Logistic Regression with $\lambda = 10^{-5}$.

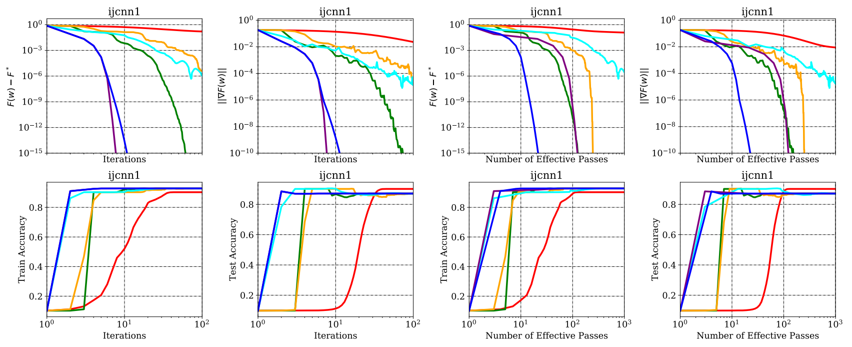


Figure 18: `ijcnn1`: Deterministic Logistic Regression with $\lambda = 10^{-6}$.

D.1.1 Sensitivity of SONIA to memory hyper-parameter

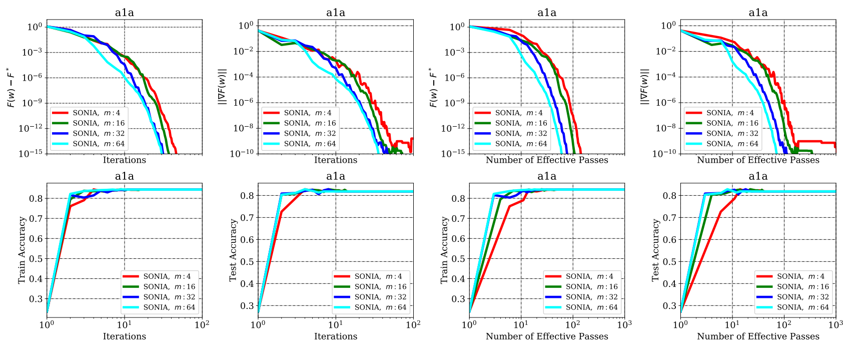


Figure 19: a1a: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

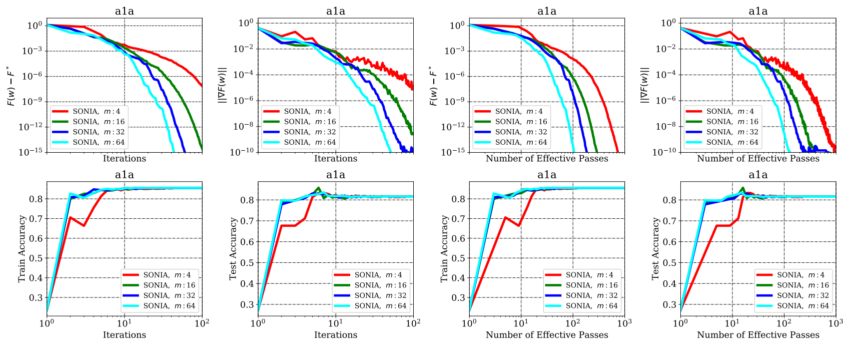


Figure 20: a1a: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

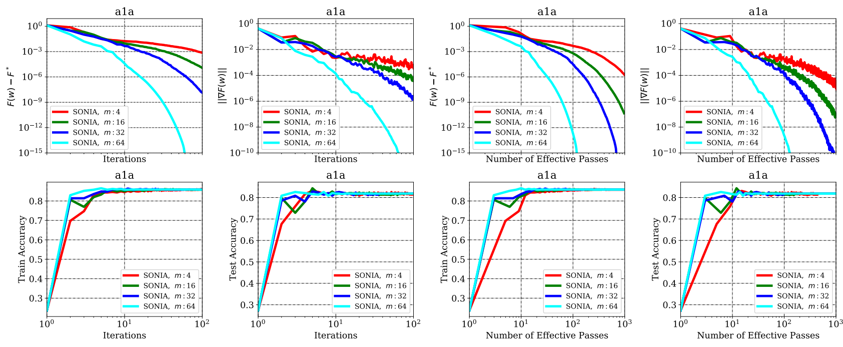


Figure 21: a1a: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

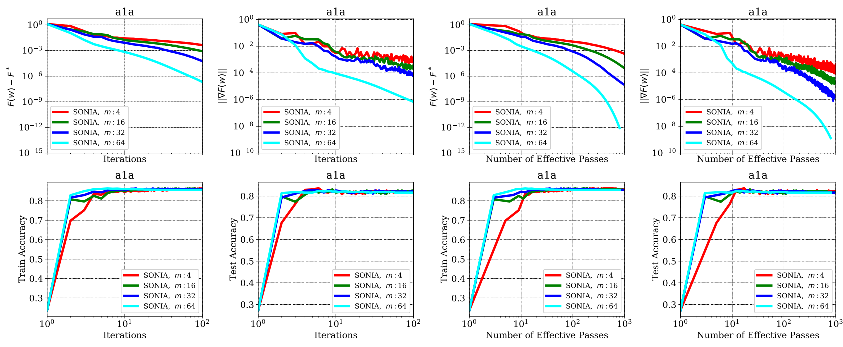


Figure 22: a1a: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

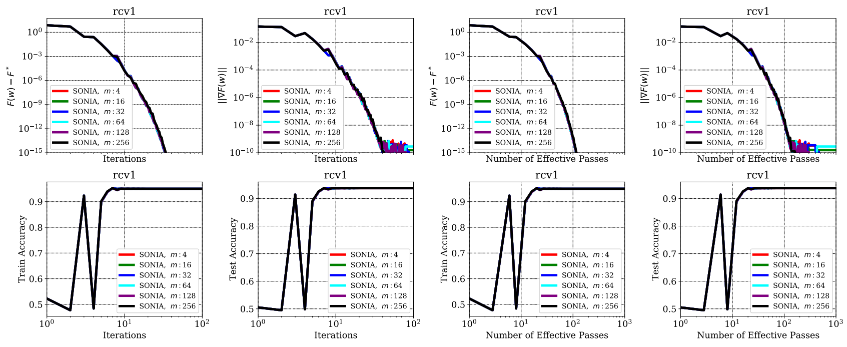


Figure 23: rcv1: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

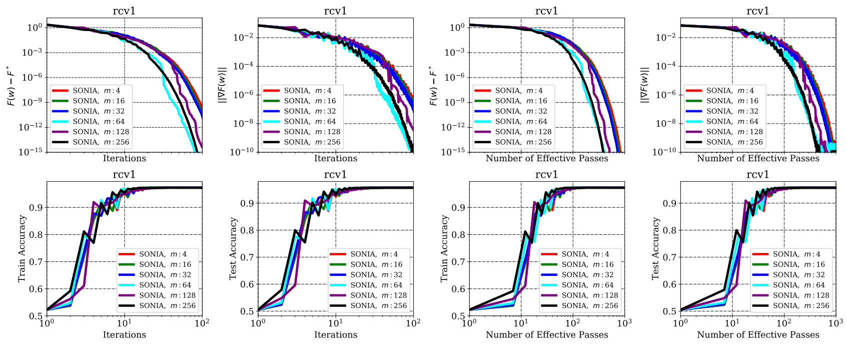


Figure 24: rcv1: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

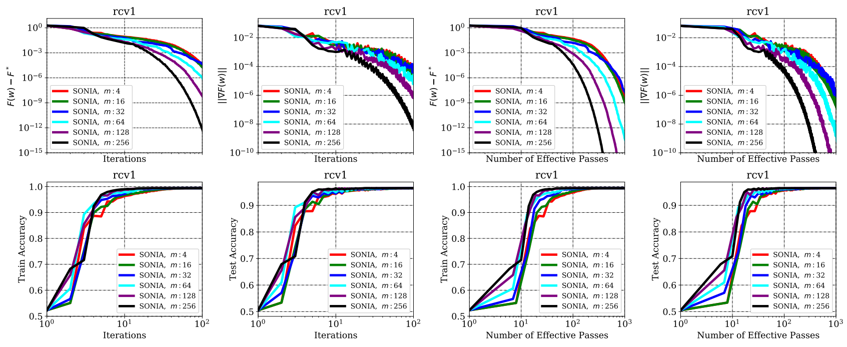


Figure 25: rcv1: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

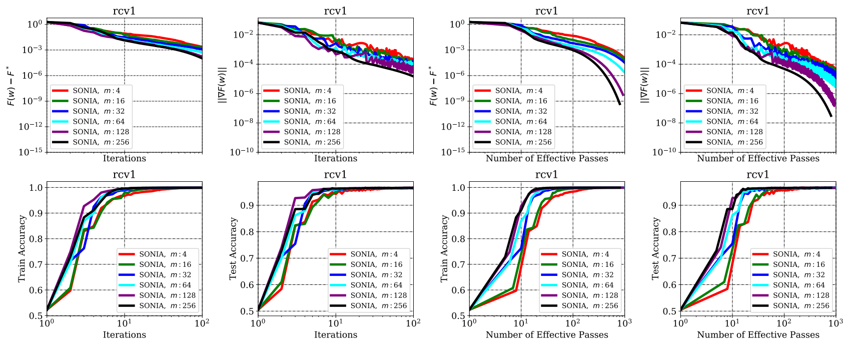


Figure 26: rcv1: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

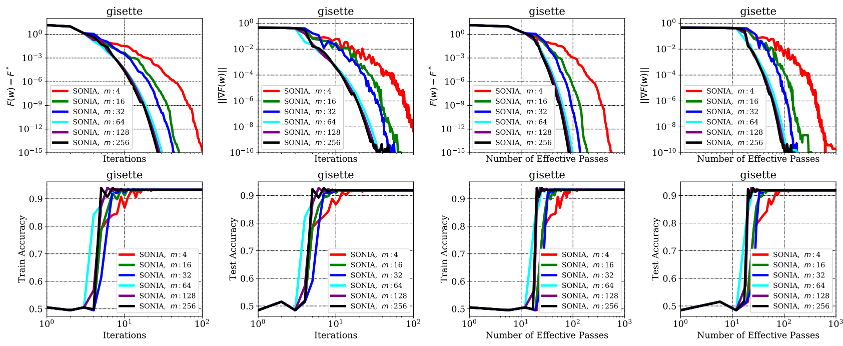


Figure 27: `gisette`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

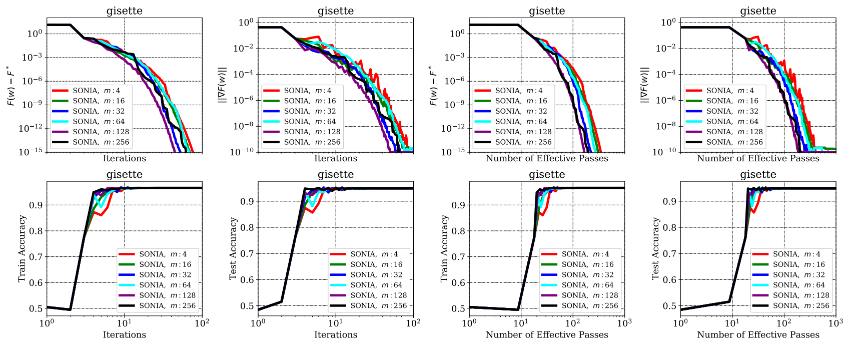


Figure 28: `gisette`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

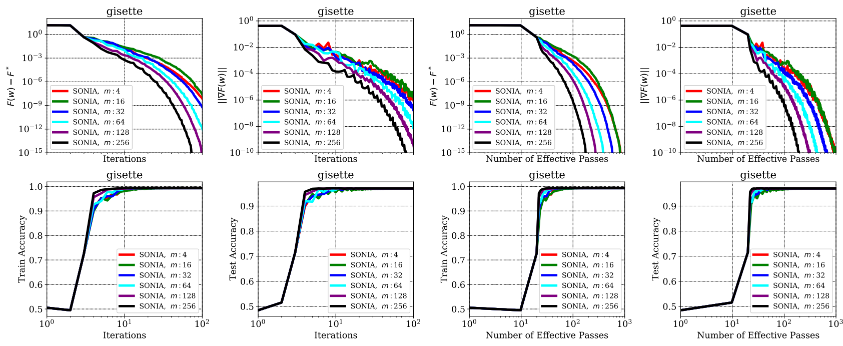


Figure 29: `gisette`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

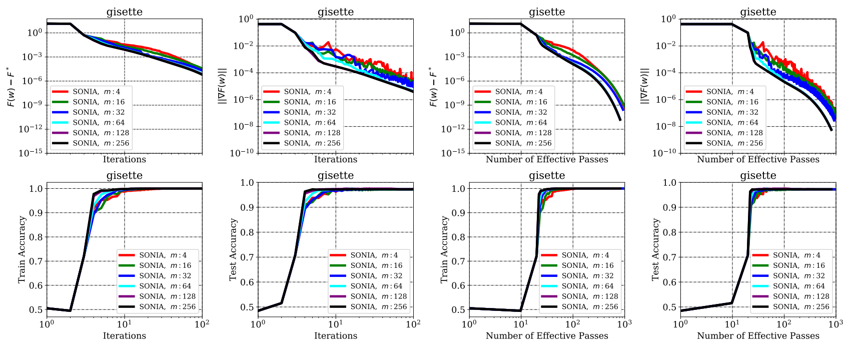


Figure 30: `gisette`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

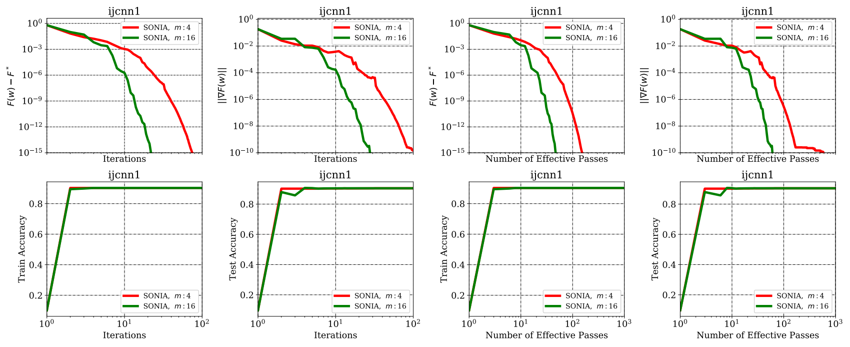


Figure 31: `ijcnn1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

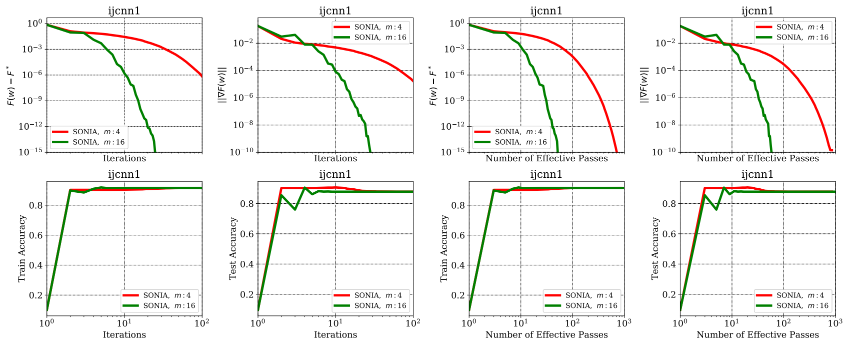


Figure 32: `ijcnn1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

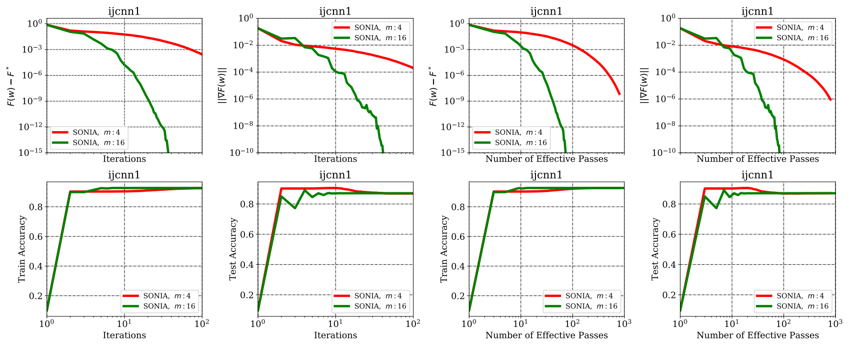


Figure 33: `ijcnn1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

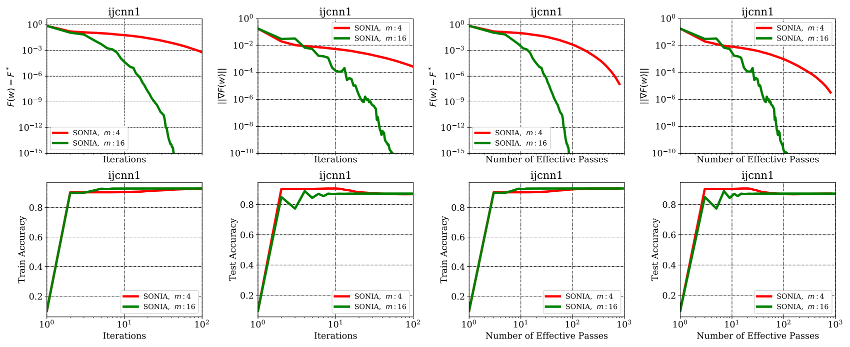


Figure 34: `ijcnn1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

D.1.2 Sensitivity of SONIA to truncation hyper-parameter

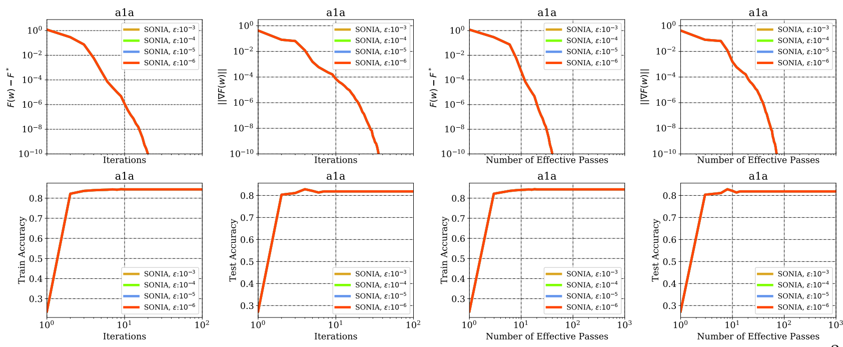


Figure 35: a1a: Sensitivity of SONIA to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

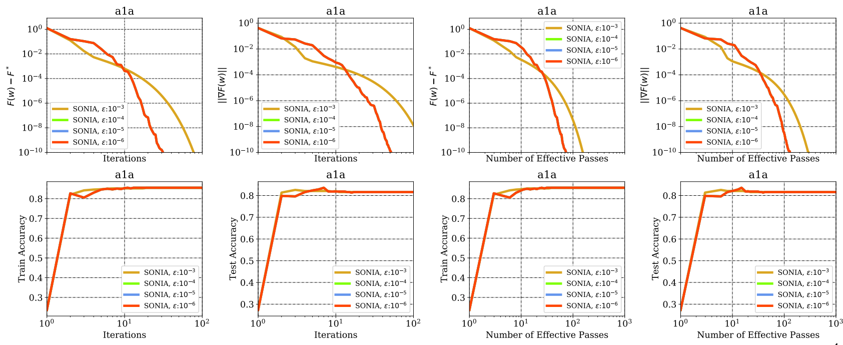


Figure 36: a1a: Sensitivity of SONIA to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

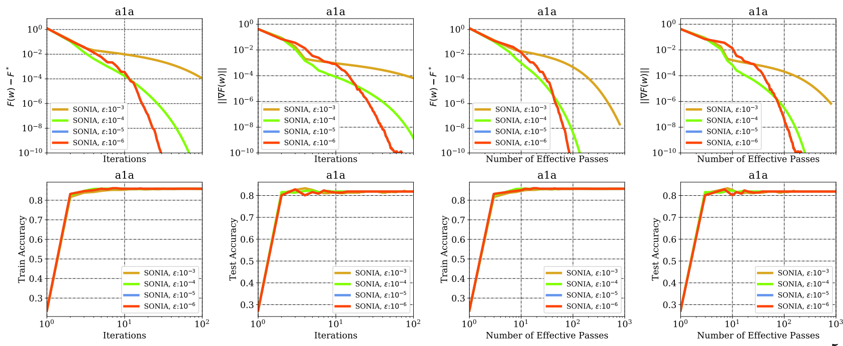


Figure 37: a1a: Sensitivity of SONIA to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

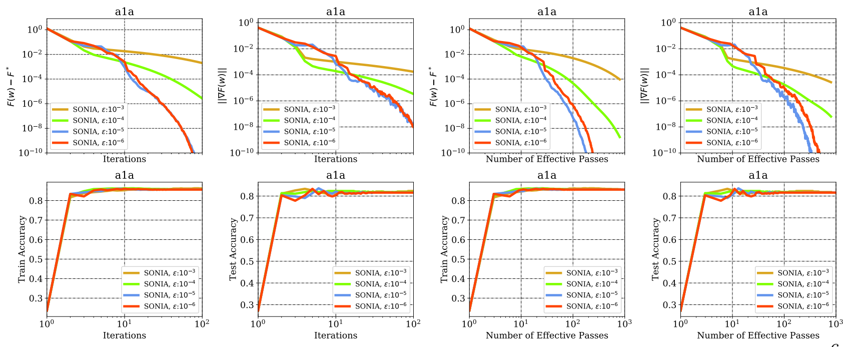


Figure 38: a1a: Sensitivity of SONIA to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

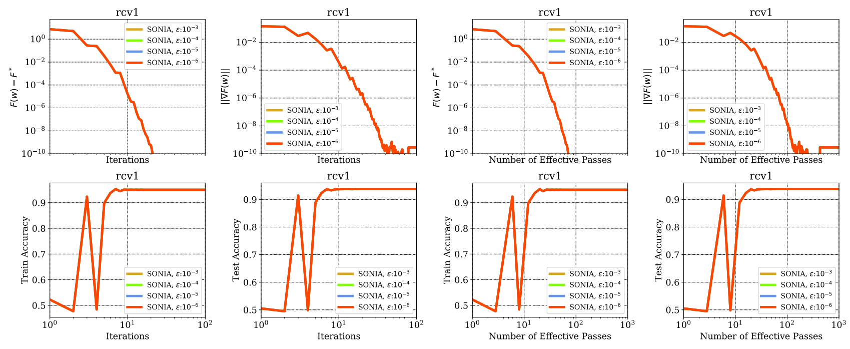


Figure 39: `rcv1`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

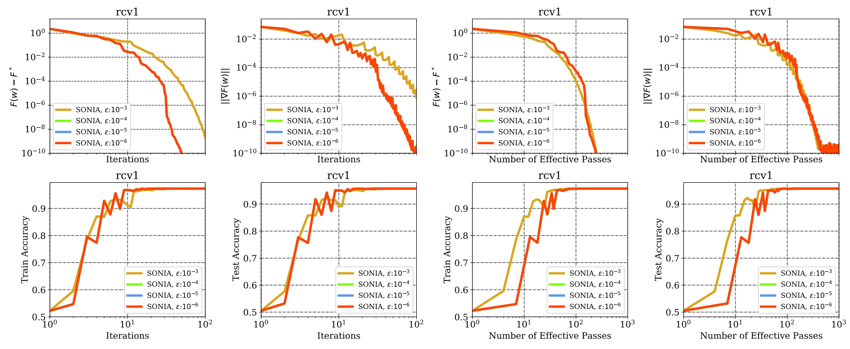


Figure 40: `rcv1`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

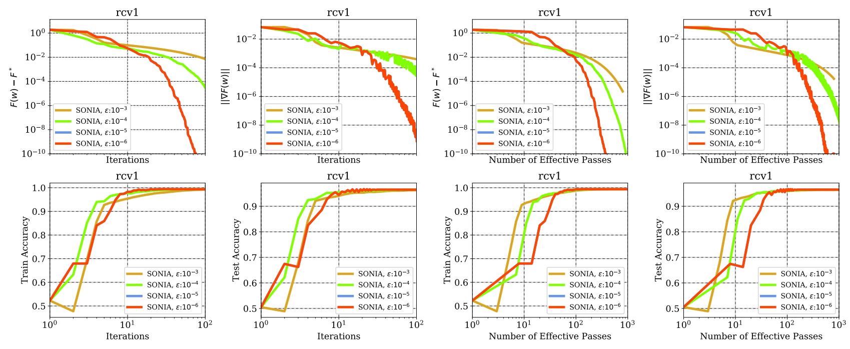


Figure 41: `rcv1`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

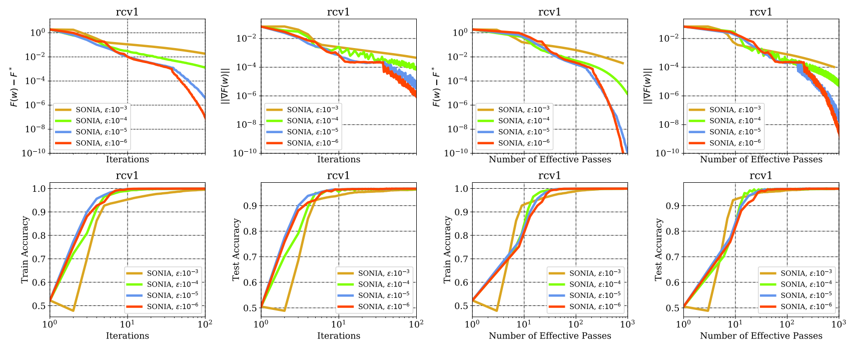


Figure 42: `rcv1`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

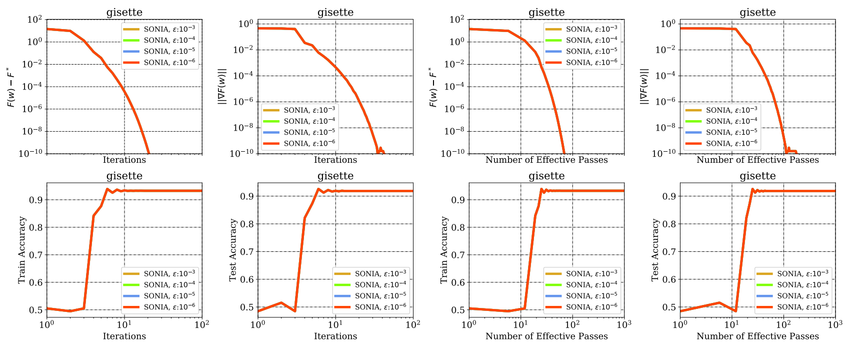


Figure 43: `gisette`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

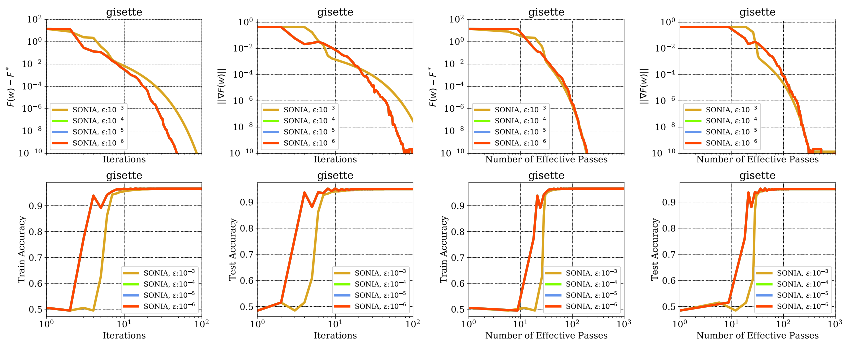


Figure 44: `gisette`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

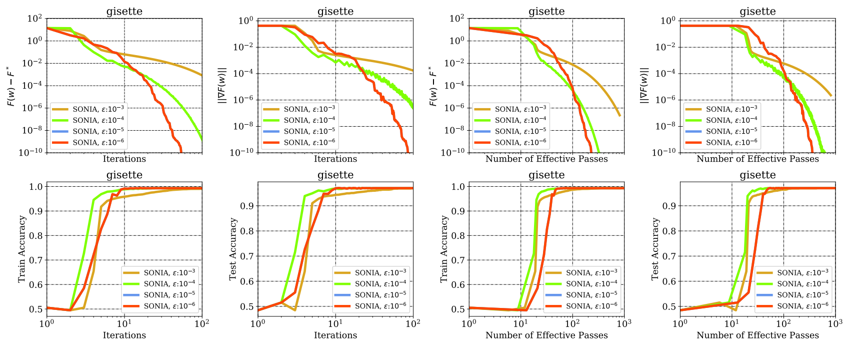


Figure 45: `gisette`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

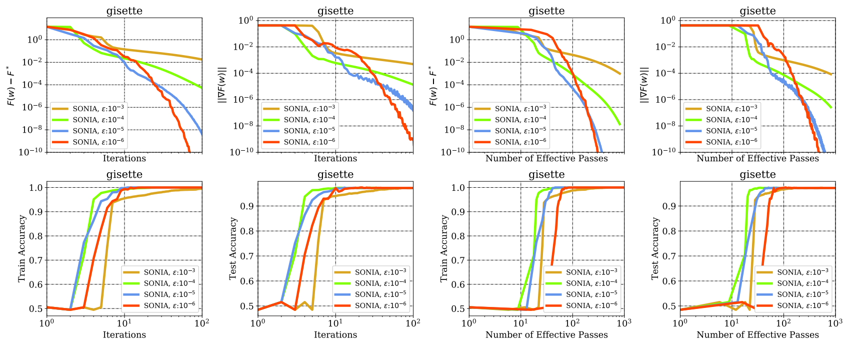


Figure 46: `gisette`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

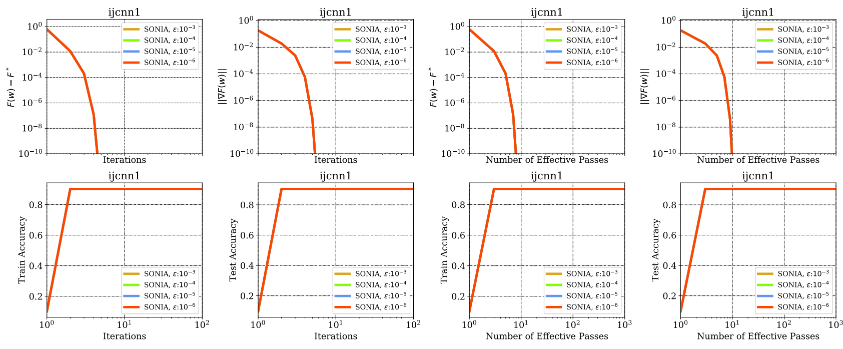


Figure 47: `ijcnn1`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

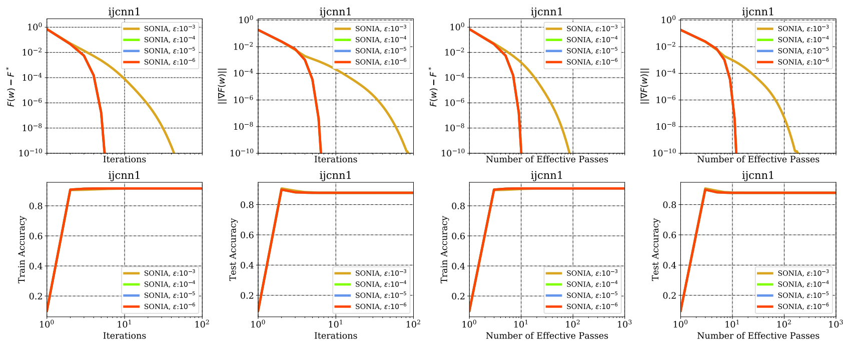


Figure 48: `ijcnn1`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

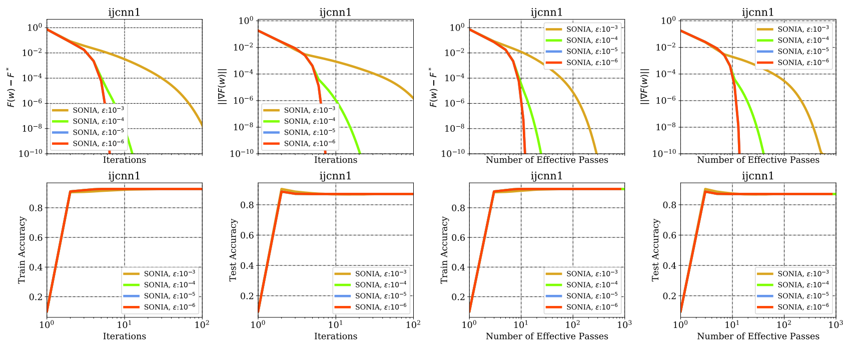


Figure 49: `ijcnn1`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

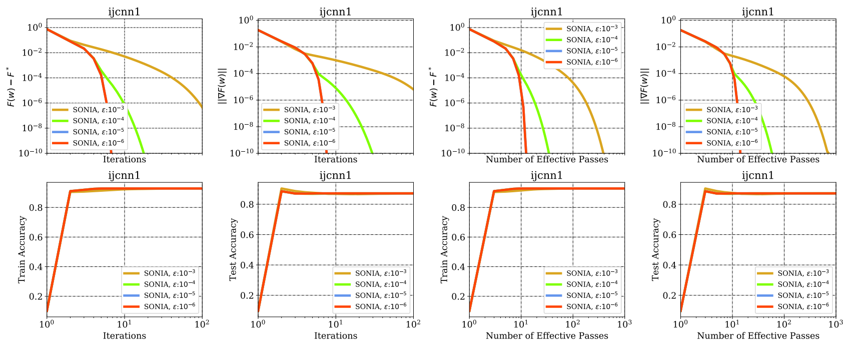


Figure 50: `ijcnn1`: Sensitivity of `SONIA` to ϵ, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

D.2 Additional Numerical Experiments: Deterministic Nonconvex Functions

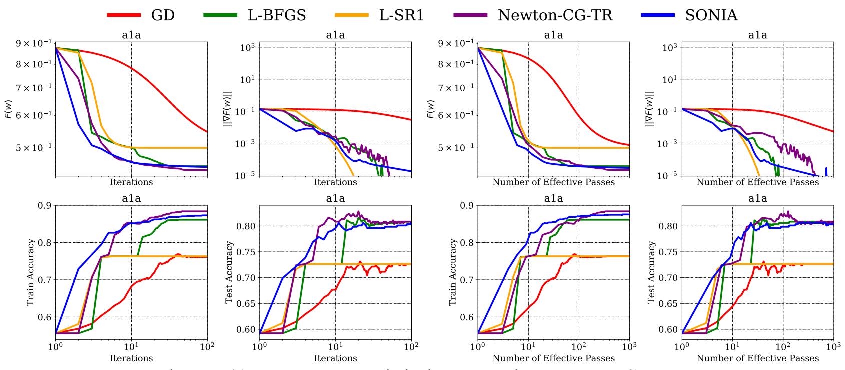


Figure 51: a1a: Deterministic Non-Linear Least Square

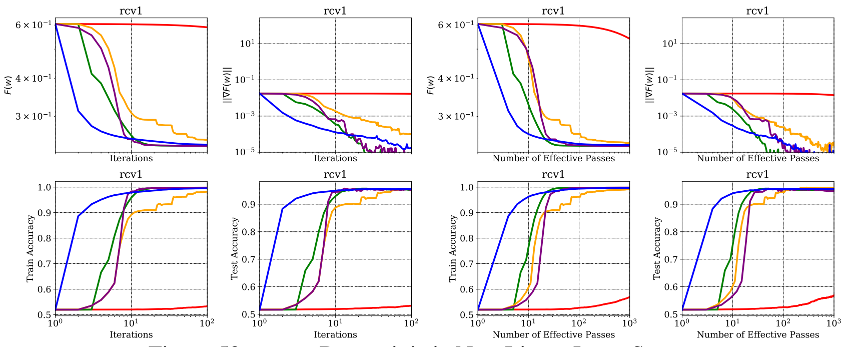


Figure 52: rcv1: Deterministic Non-Linear Least Square

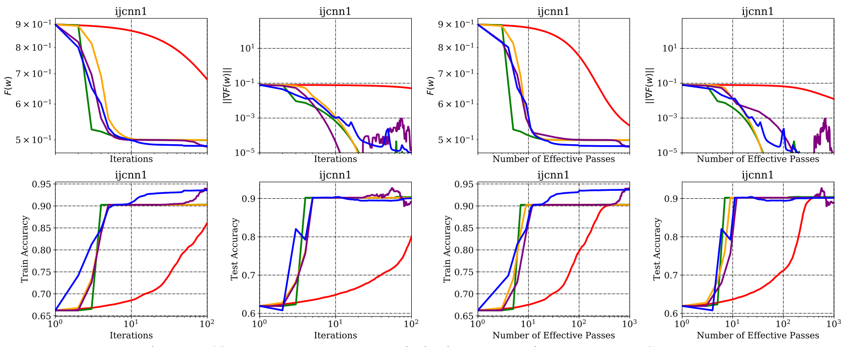


Figure 53: ijcnn1: Deterministic Non-Linear Least Square

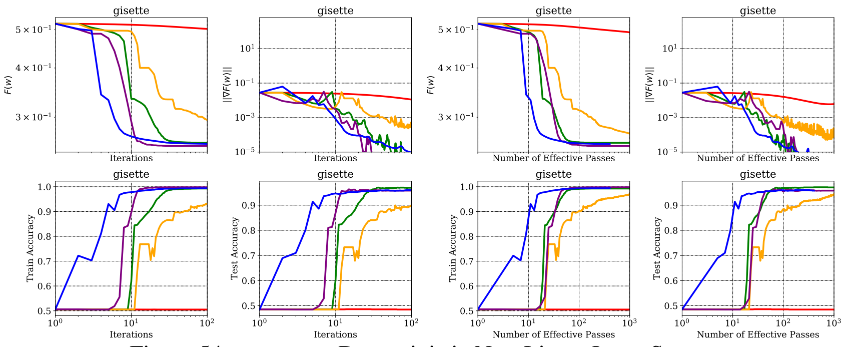


Figure 54: gisette: Deterministic Non-Linear Least Square

D.3 Additional Numerical Experiments: Stochastic Strongly Convex Functions

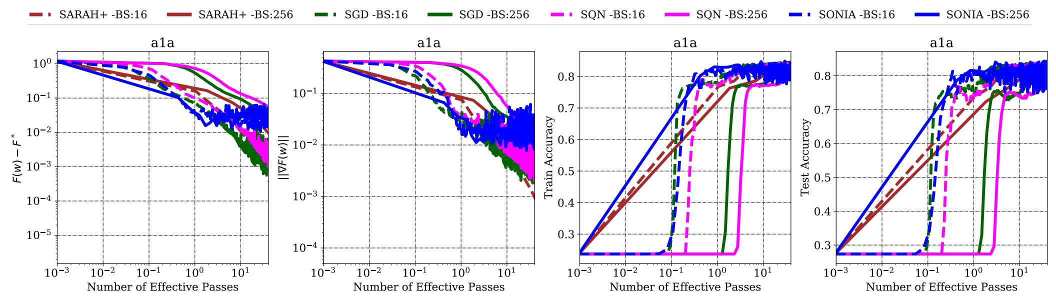


Figure 55: `a1a`: Stochastic Logistic Regression ($\lambda = 10^{-3}$).

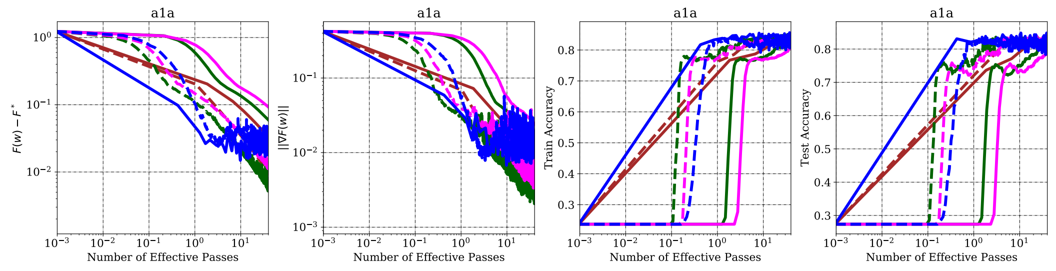


Figure 56: `a1a`: Stochastic Logistic Regression ($\lambda = 10^{-4}$).

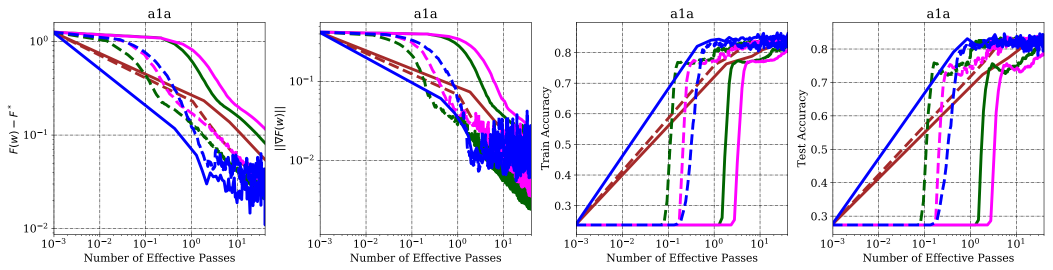


Figure 57: `a1a`: Stochastic Logistic Regression ($\lambda = 10^{-5}$).

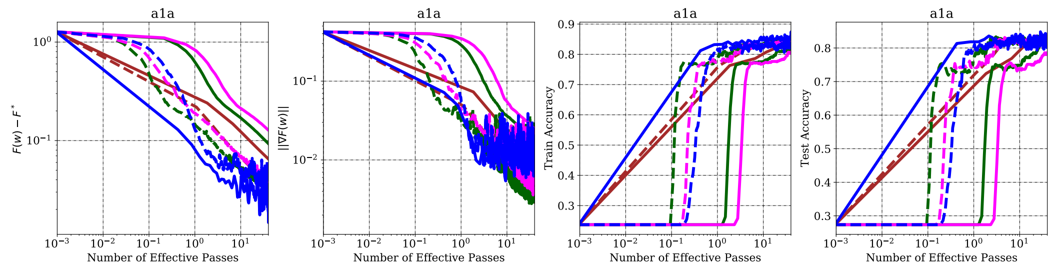


Figure 58: `a1a`: Stochastic Logistic Regression ($\lambda = 10^{-6}$).

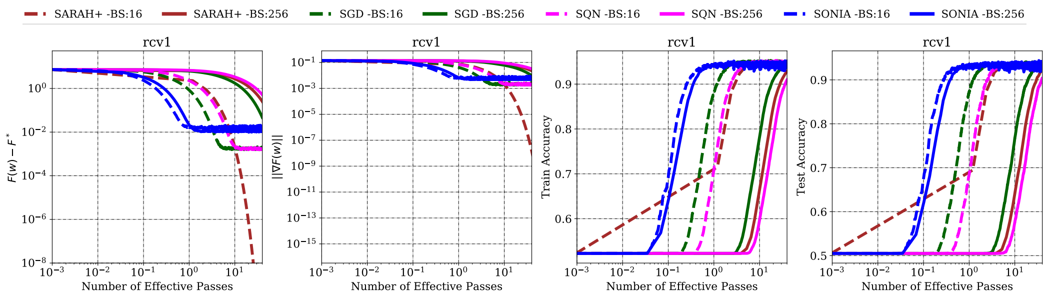


Figure 59: `rcv1`: Stochastic Logistic Regression ($\lambda = 10^{-3}$).

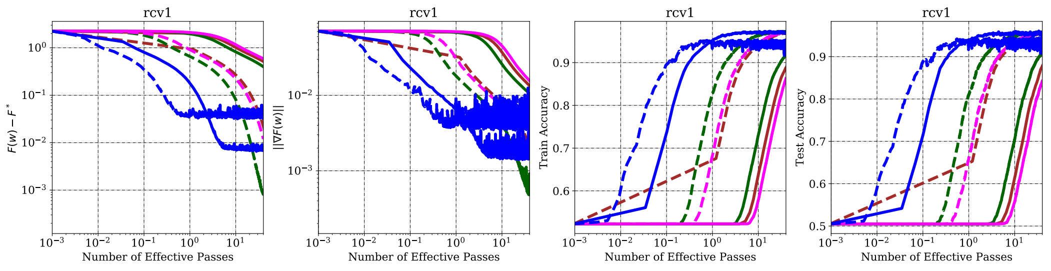


Figure 60: `rcv1`: Stochastic Logistic Regression ($\lambda = 10^{-4}$).

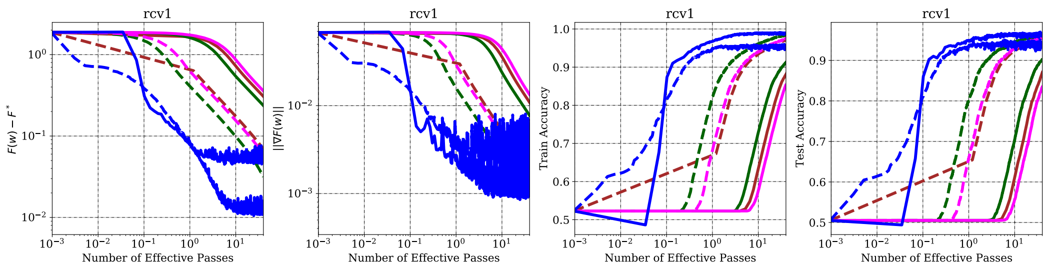


Figure 61: `rcv1`: Stochastic Logistic Regression ($\lambda = 10^{-5}$).

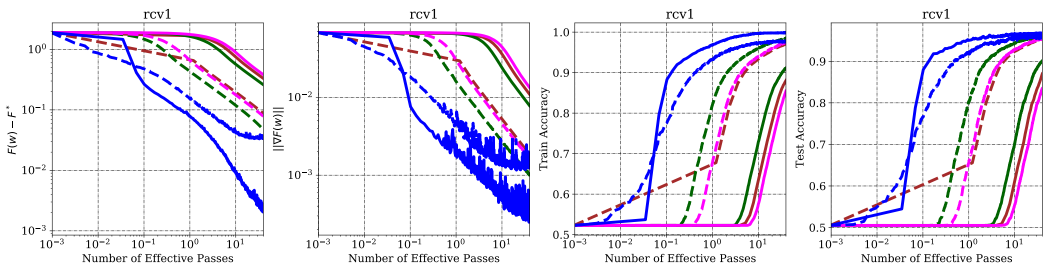


Figure 62: `rcv1`: Stochastic Logistic Regression ($\lambda = 10^{-6}$).

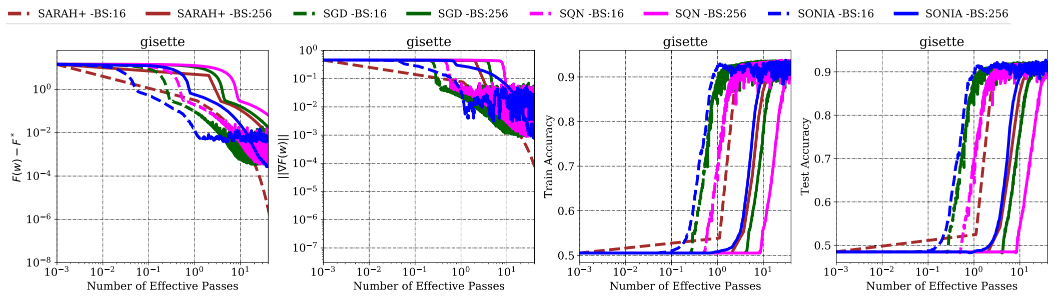


Figure 63: `gisette`: Stochastic Logistic Regression ($\lambda = 10^{-3}$).

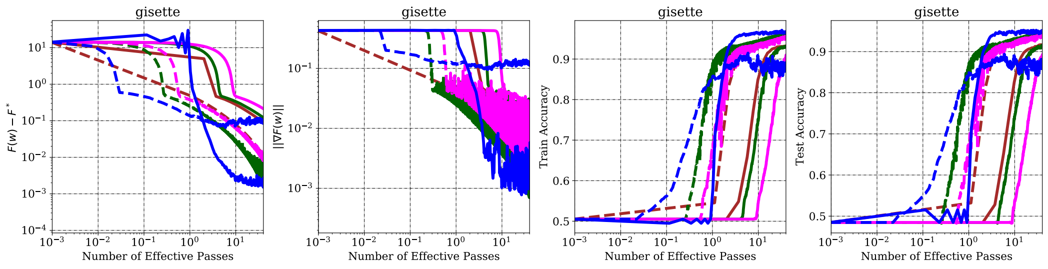


Figure 64: `gisette`: Stochastic Logistic Regression ($\lambda = 10^{-4}$).

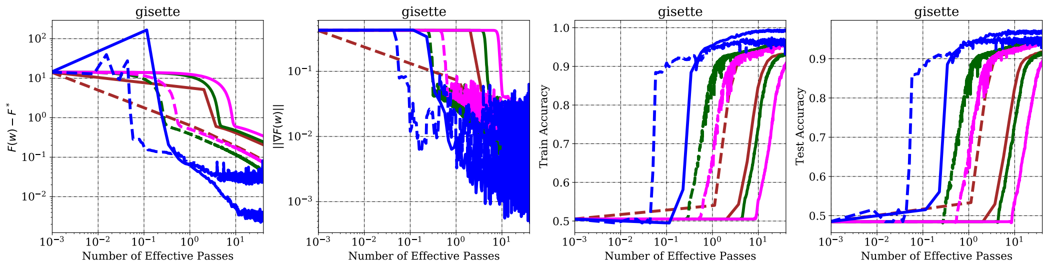


Figure 65: `gisette`: Stochastic Logistic Regression ($\lambda = 10^{-5}$).

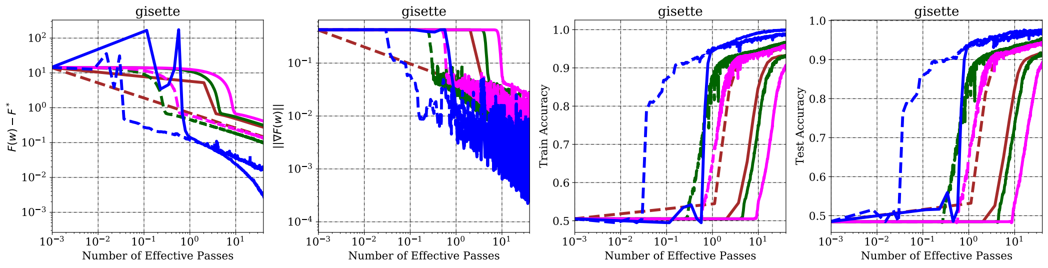


Figure 66: `gisette`: Stochastic Logistic Regression ($\lambda = 10^{-6}$).

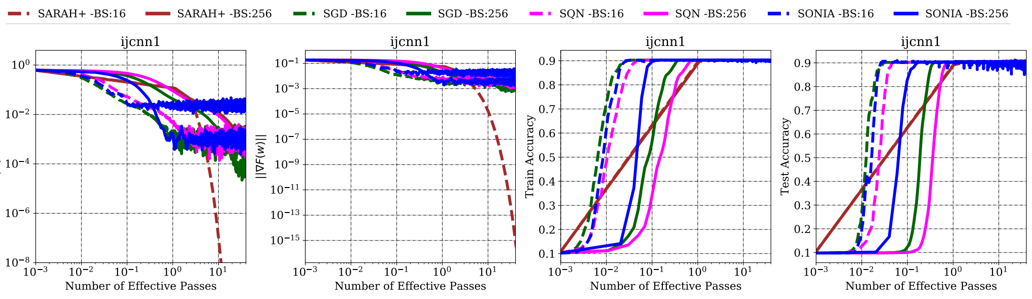


Figure 67: ijcnn1: Stochastic Logistic Regression ($\lambda = 10^{-3}$).

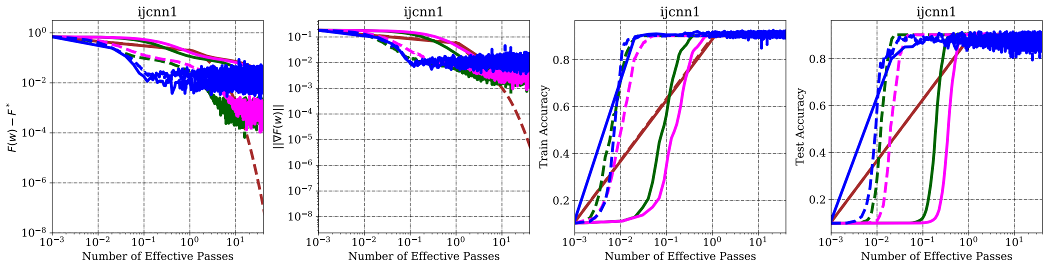


Figure 68: ijcnn1: Stochastic Logistic Regression ($\lambda = 10^{-4}$).

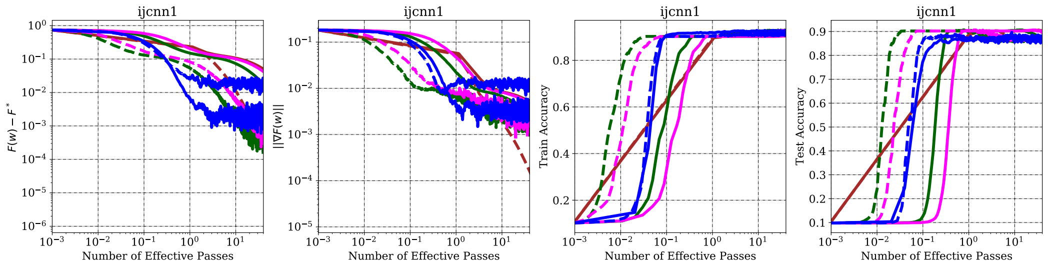


Figure 69: ijcnn1: Stochastic Logistic Regression ($\lambda = 10^{-5}$).

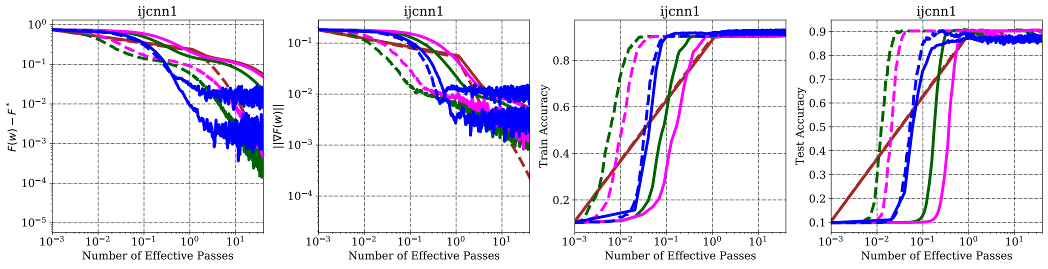


Figure 70: ijcnn1: Stochastic Logistic Regression ($\lambda = 10^{-6}$).

D.4 Additional Numerical Experiments: Stochastic Nonconvex Functions

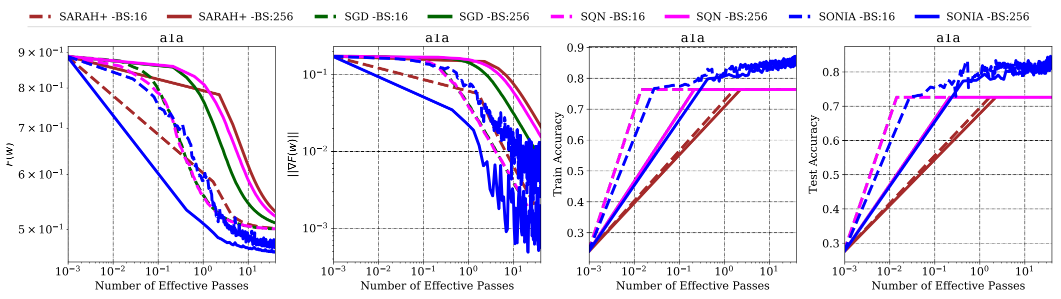


Figure 71: `a1a`: Stochastic Nonlinear Least Squares.

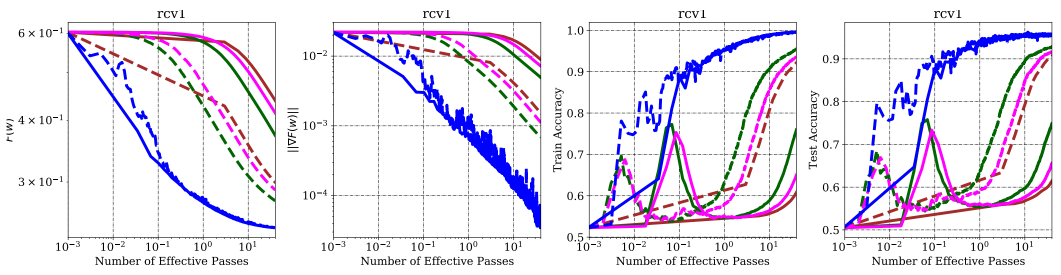


Figure 72: `rcv1`: Stochastic Nonlinear Least Squares.

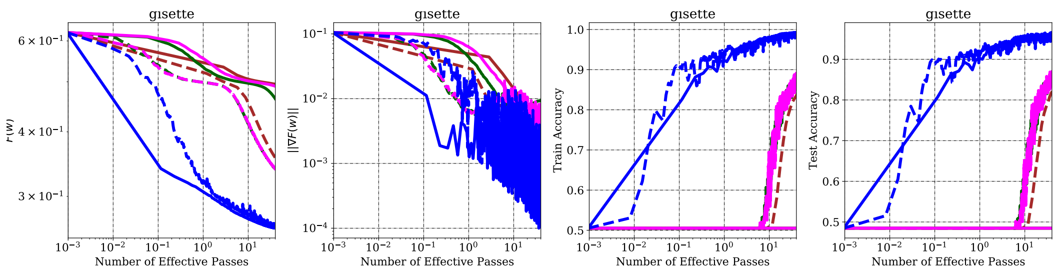


Figure 73: `gisette`: Stochastic Nonlinear Least Squares.

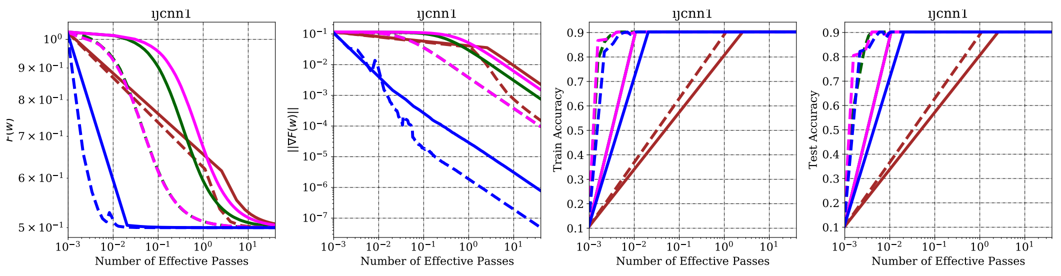


Figure 74: `ijcnn1`: Stochastic Nonlinear Least Squares.