A Theoretical Results and Proofs

A.1 Assumptions

Assumption 4.1. *F is twice continuously differentiable.*

Assumption 4.4. *There exist positive constants $0 < \mu \leq L$ 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 $\mathcal{A}_k := \tilde{V}_k(|\Lambda_k|_\omega^\Omega)^{-1}\tilde{V}_k^T + \rho_k(I - \tilde{V}_k\tilde{V}_k^T)$. Then the search direction p_k in (3.7) is equivalent to $p_k = -\mathcal{A}_k\nabla F(w_k)$.*

Proof. Define

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

so that by (3.10) and (A.1),

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

By (3.6), $g_k^\perp = (I - \tilde{V}\tilde{V}^T)g_k^\perp$. Then

$$\begin{aligned}
p_k &\overset{(3.7)+(A.1)}{=} -A_k g_k - A_k^\perp g_k^\perp \\
&\overset{(3.6)}{=} -A_k g_k - A_k^\perp(\nabla F(w_k) - g_k) \\
&\overset{(3.6)}{=} -A_k\tilde{V}_k\tilde{V}_k^T\nabla F(w_k) - A_k^\perp(I - \tilde{V}_k\tilde{V}_k^T)\nabla F(w_k) \\
&\overset{(A.1)}{=} -\tilde{V}_k(|\Lambda_k|_\omega^\Omega)^{-1}\tilde{V}_k^T\tilde{V}_k\tilde{V}_k^T\nabla F(w_k) - \rho_k(I - \tilde{V}_k\tilde{V}_k^T)(I - \tilde{V}_k\tilde{V}_k^T)\nabla F(w_k) \\
&= -\tilde{V}_k(|\Lambda_k|_\omega^\Omega)^{-1}\tilde{V}_k^T\nabla F(w_k) - \rho_k(I - \tilde{V}_k\tilde{V}_k^T)\nabla F(w_k) \\
&\overset{(A.1)+(A.2)}{=} -\mathcal{A}_k\nabla F(w_k).
\end{aligned} \tag{A.3}$$

□

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 $D_1 = (|\Lambda_k|_\omega^\Omega)^{-1}$ 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 (3.8) 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,*

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

Proof. Note, during the proof steps, the notations I_m and I_d are used which simply mean the identity matrices with sizes memory m and dimension d respectively.
By (A.2), $\mathcal{A}_k = A_k + A_k^{\perp}$, so we can write

$$\begin{aligned} \mathcal{A}_k &= A_k + A_k^{\perp} \\ &= \tilde{V}_k(|\Lambda_k|_{\omega}^{\Omega})^{-1}\tilde{V}_k^T + \rho_k(I_d - \tilde{V}_k\tilde{V}_k^T) \end{aligned} \tag{A.4}$$

By (3.8), and the point that $(|\Lambda_k|_{\omega}^{\Omega})^{-1} \succeq \frac{1}{\Omega} I_m$ we have:

$$\begin{aligned} \tilde{V}_k(|\Lambda_k|_{\omega}^{\Omega})^{-1}\tilde{V}_k^T + \rho_k(I_d - \tilde{V}_k\tilde{V}_k^T) &\succeq \tilde{V}_k\frac{1}{\Omega}I_m\tilde{V}_k^T + \frac{1}{\Omega}(I_d - \tilde{V}_k\tilde{V}_k^T) \\ &= \frac{1}{\Omega}I_d \end{aligned} \tag{A.5}$$

Furthermore, by (3.8) it is clear that $\rho_k \leq \frac{1}{\omega}$. Since $I_d - \tilde{V}_k\tilde{V}_k^T$ is positive semi-definite, and $(|\Lambda_k|_{\omega}^{\Omega})^{-1} \preceq \frac{1}{\omega} I_m$, we have

$$\begin{aligned} \tilde{V}_k(|\Lambda_k|_{\omega}^{\Omega})^{-1}\tilde{V}_k^T + \rho_k(I_d - \tilde{V}_k\tilde{V}_k^T) &\preceq \tilde{V}_k\frac{1}{\omega}I_m\tilde{V}_k^T + \frac{1}{\omega}(I_d - \tilde{V}_k\tilde{V}_k^T) \\ &= \frac{1}{\omega}I_d. \end{aligned} \tag{A.6}$$

Defining $0 < \mu_1 \leq \frac{1}{\Omega} \leq \frac{1}{\omega} \leq \mu_2$, 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 $k \geq 0$,*

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

Proof. We have that

$$\begin{aligned}
F(w_{k+1}) &= F(w_k - \alpha \mathcal{A}_k \nabla F(w_k)) \\
&\leq F(w_k) + \nabla F(w_k)^T(-\alpha \mathcal{A}_k \nabla F(w_k)) + \frac{L}{2}\|\alpha \mathcal{A}_k \nabla F(w_k)\|^2 \\
&\leq F(w_k) - \alpha\mu_1\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2 L}{2}\|\nabla F(w_k)\|^2 \\
&= F(w_k) - \alpha\left(\mu_1 - \alpha\frac{\mu_2^2 L}{2}\right)\|\nabla F(w_k)\|^2 \\
&\leq F(w_k) - \alpha\frac{\mu_1}{2}\|\nabla F(w_k)\|^2, \qquad \text{(A.7)}
\end{aligned}$$

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 [Nesterov, 2013b], 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.7)

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

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

$$\sum_{k=0}^{T-1} \|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1}. \tag{A.8}$$

Dividing (A.8) 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 $k \geq 0$,*

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

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

$$\begin{aligned}
\mathbb{E}_{\mathcal{I}_k}[F(w_{k+1})] &\leq F(w_k) - \alpha\mathbb{E}_{\mathcal{I}_k}[\nabla F(w_k)^T\mathcal{A}_k\nabla F_{\mathcal{I}_k}(w_k)] + \frac{\alpha^2\mu_2^2 L}{2}\mathbb{E}_{\mathcal{I}_k}[\|\nabla F_{\mathcal{I}_k}(w_k)\|^2] \\
&= F(w_k) - \alpha\nabla F(w_k)^T\mathcal{A}_k\nabla F(w_k) + \frac{\alpha^2\mu_2^2 L}{2}\mathbb{E}_{\mathcal{I}_k}[\|\nabla F_{\mathcal{I}_k}(w_k)\|^2] \\
&\leq F(w_k) - \alpha\left(\mu_1 - \frac{\alpha\mu_2^2 L}{2}\right)\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\gamma^2 L}{2} \\
&\leq F(w_k) - \frac{\alpha\mu_1}{2}\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\gamma^2 L}{2},
\end{aligned} \tag{A.9}$$

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 [Nesterov, 2013b], we have

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

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

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

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

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

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.9) 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 [Jahani et al., 2019] 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 $\phi(z) = \frac{1}{1 + e^{-z}}$):

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

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

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

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

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 [Jahani et al., 2019], 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 ($i = 1, 2, \ldots, K$):**
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 [Nesterov, 2004, Chapter 2] with adaptive Lipschitz constant
L-BFGS	Limited memory BFGS [Liu and Nocedal, 1989]
L-SR1	Limited memory SR1 [Lu, 1996]
Newton-CG-TR	Newton method with conjugate gradient (CG) utilizing trust region (TR)[Nocedal and Wright, 2006]
Newton-CG-LS	Newton method with conjugate gradient (CG) utilizing line search (LS) [Nocedal and Wright, 2006]
SARAH+	Practical variant of SARAH [Nguyen et al., 2017]
SQN	Stochastic Quasi-Newton [Byrd et al., 2016]
SGD	Stochastic gradient method [Robbins and Monro, 1951]
GD	Gradient descent
ASUESA	Accelerated Smooth Underestimate Sequence Algorithm with adaptive Lipschitz constant [Ma et al., 2017]
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 [Ma et al., 2017]. 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 [Nocedal and Wright, 2006].
- **Newton-CG-LS**: The learning rate is chosen by Armijo backtracking line search. The Newton system is solved according to Algorithm 7.1 in [Nocedal and Wright, 2006].

- **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 [Nesterov, 2013a].
- **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 [Nocedal and Wright, 2006].
- **Newton-CG-TR**: The Newton system is solved according to CG-Steihaug method (Algorithm 7.2) in [Nocedal and Wright, 2006].
- **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

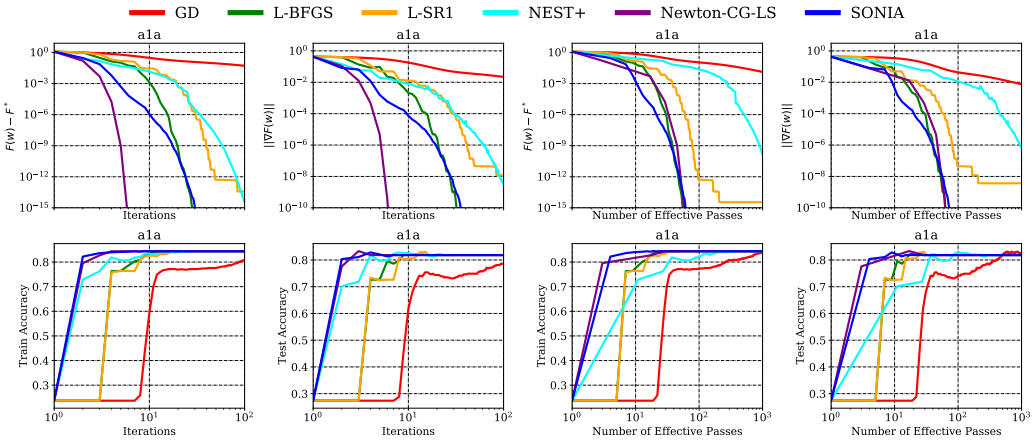


Figure 5: `a1a`: Deterministic Logistic Regression with $\lambda = 10^{-3}$.

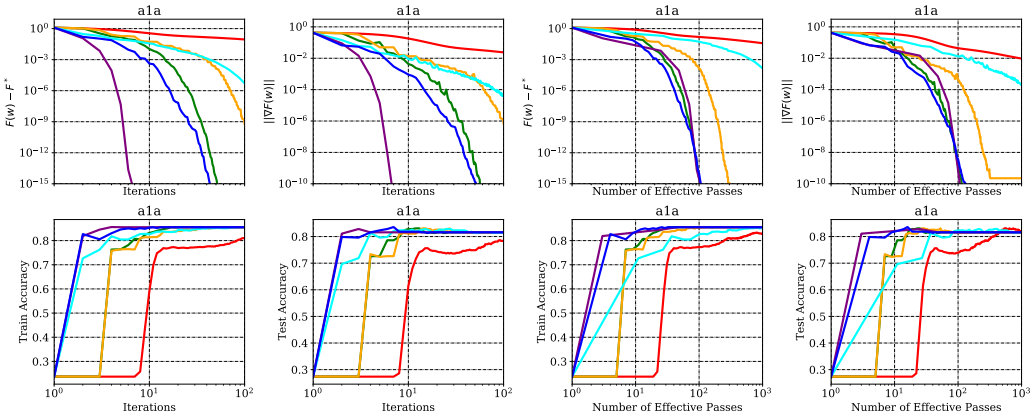


Figure 6: `a1a`: Deterministic Logistic Regression with $\lambda = 10^{-4}$.

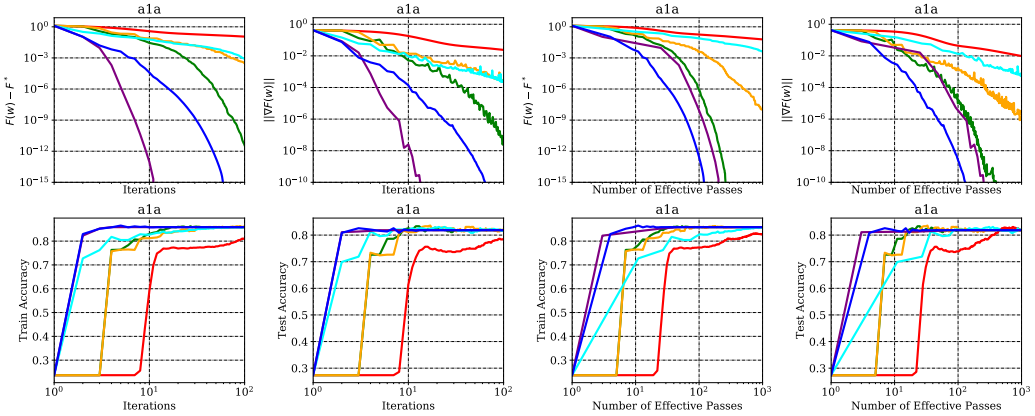


Figure 7: `a1a`: Deterministic Logistic Regression with $\lambda = 10^{-5}$.

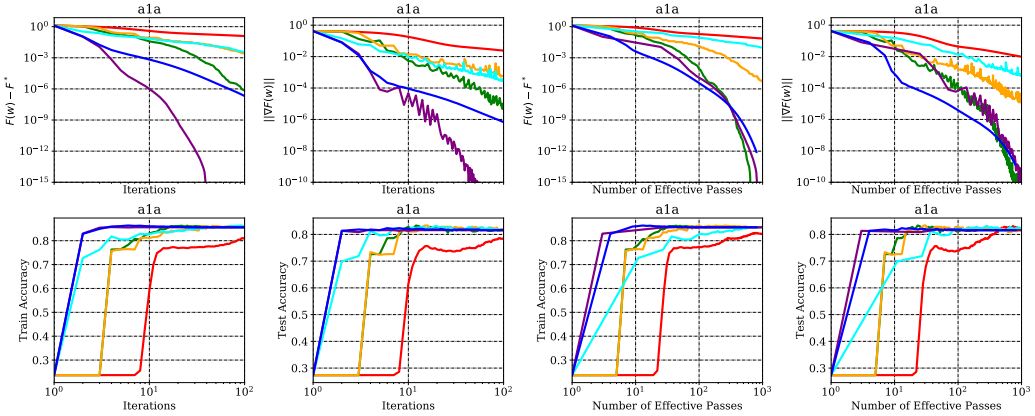


Figure 8: `a1a`: Deterministic Logistic Regression with $\lambda = 10^{-6}$.

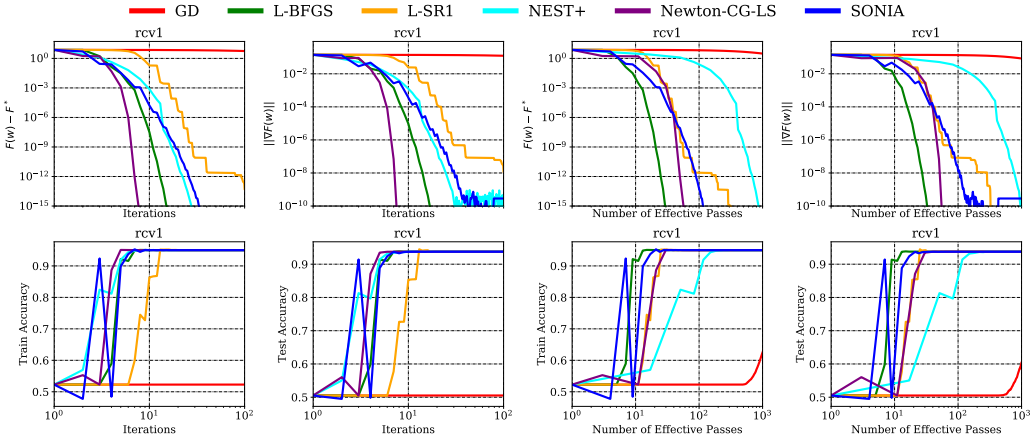


Figure 9: `rcv1`: Deterministic Logistic Regression with $\lambda = 10^{-3}$.

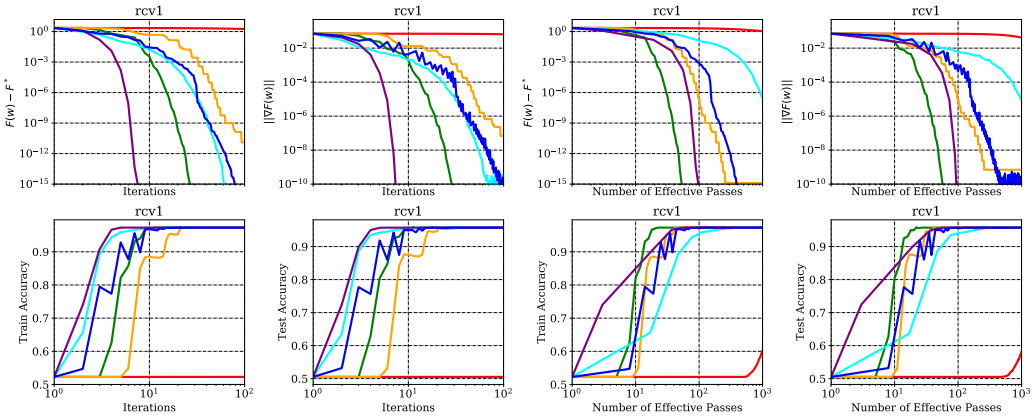


Figure 10: `rcv1`: Deterministic Logistic Regression with $\lambda = 10^{-4}$.

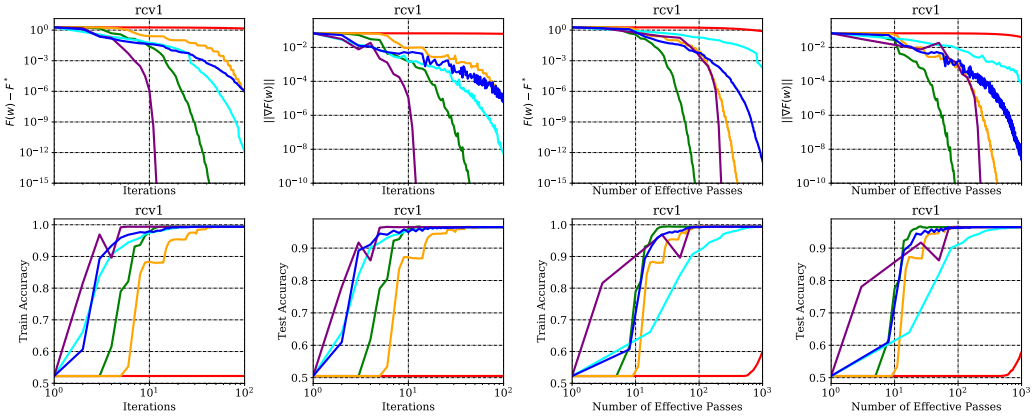


Figure 11: `rcv1`: Deterministic Logistic Regression with $\lambda = 10^{-5}$.

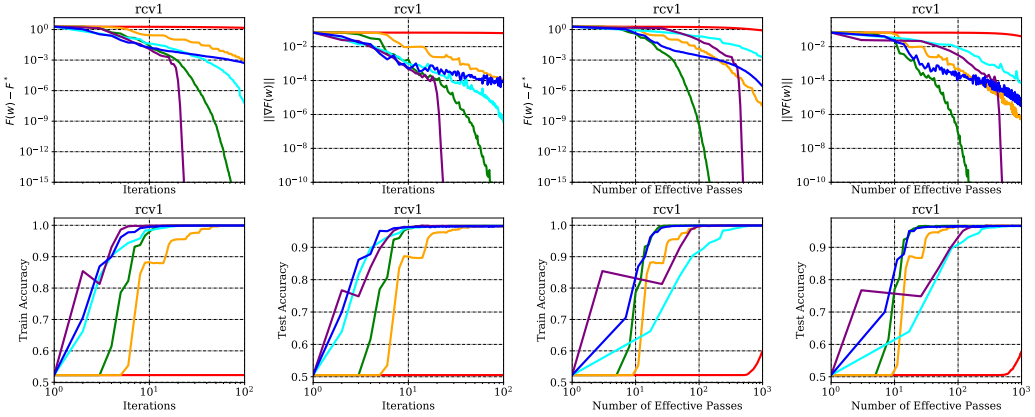


Figure 12: `rcv1`: Deterministic Logistic Regression with $\lambda = 10^{-6}$.

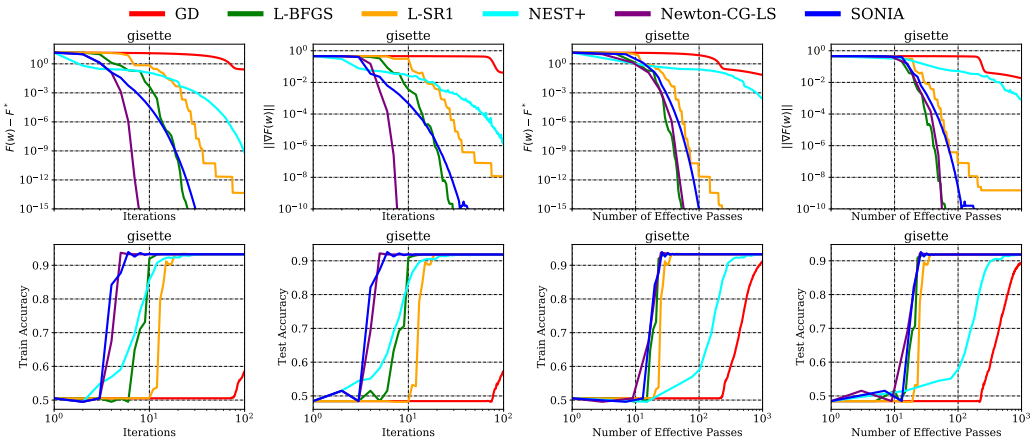


Figure 13: `gisette`: Deterministic Logistic Regression with $\lambda = 10^{-3}$.

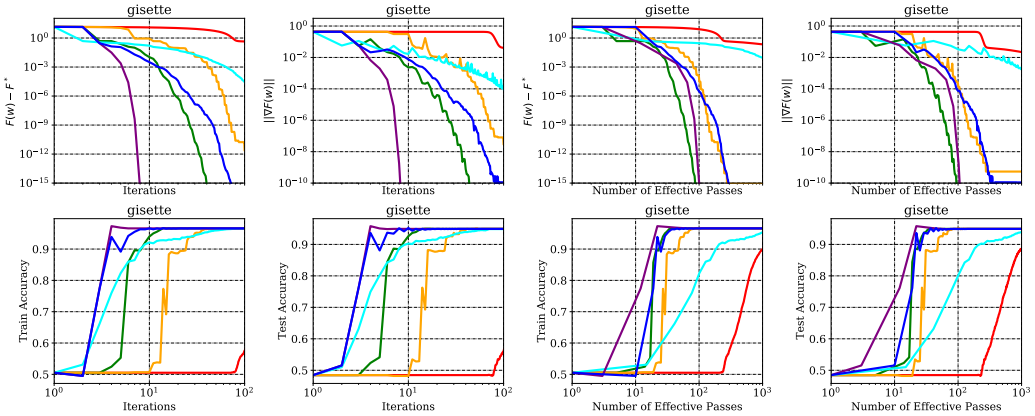


Figure 14: `gisette`: Deterministic Logistic Regression with $\lambda = 10^{-4}$.

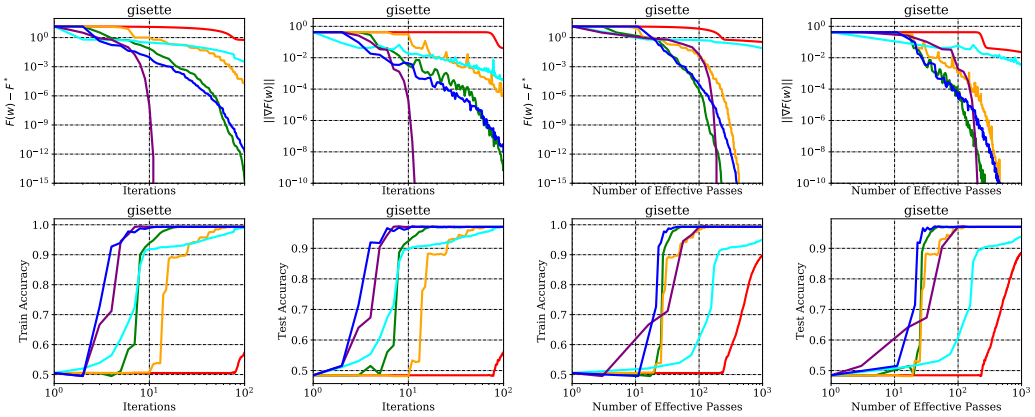


Figure 15: `gisette`: Deterministic Logistic Regression with $\lambda = 10^{-5}$.

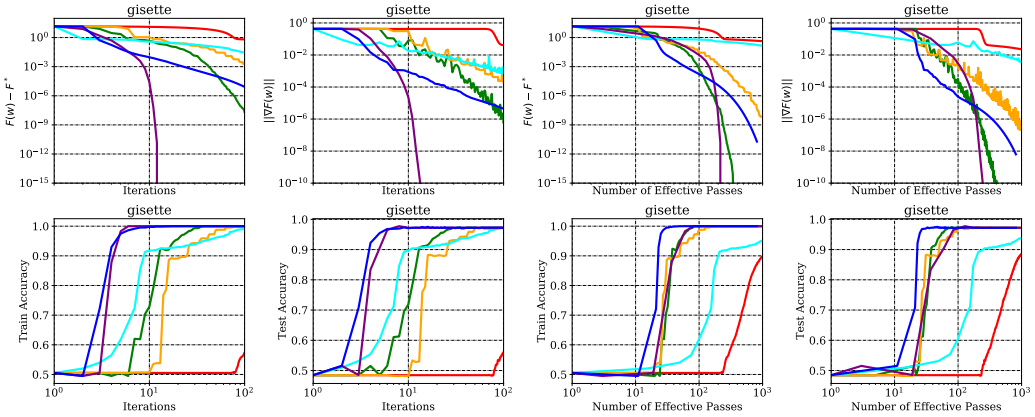


Figure 16: `gisette`: Deterministic Logistic Regression with $\lambda = 10^{-6}$.

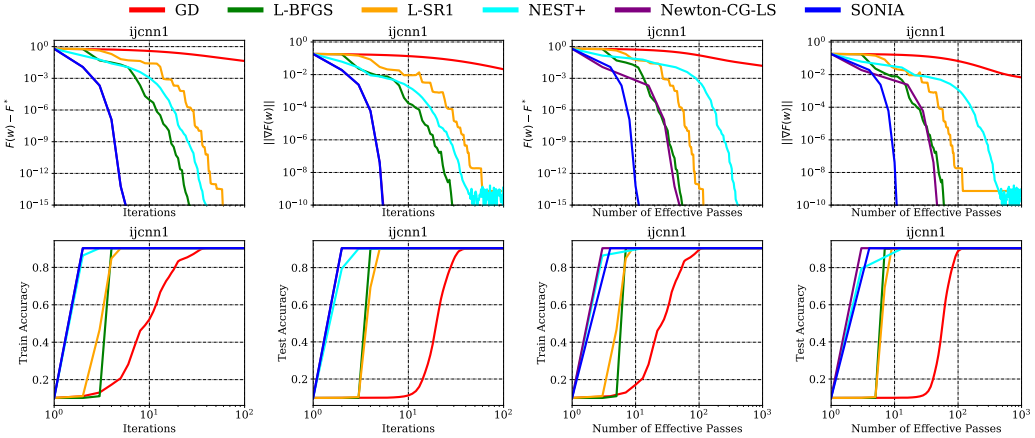


Figure 17: `ijcnn1`: Deterministic Logistic Regression with $\lambda = 10^{-3}$.

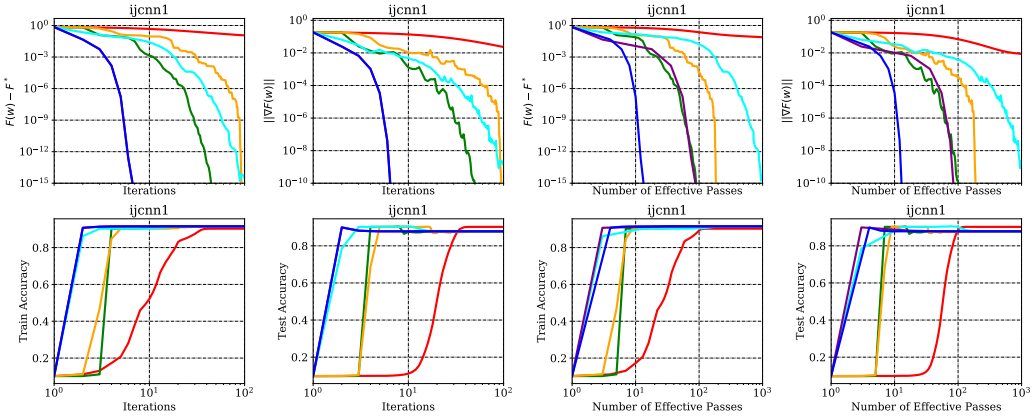


Figure 18: `ijcnn1`: Deterministic Logistic Regression with $\lambda = 10^{-4}$.

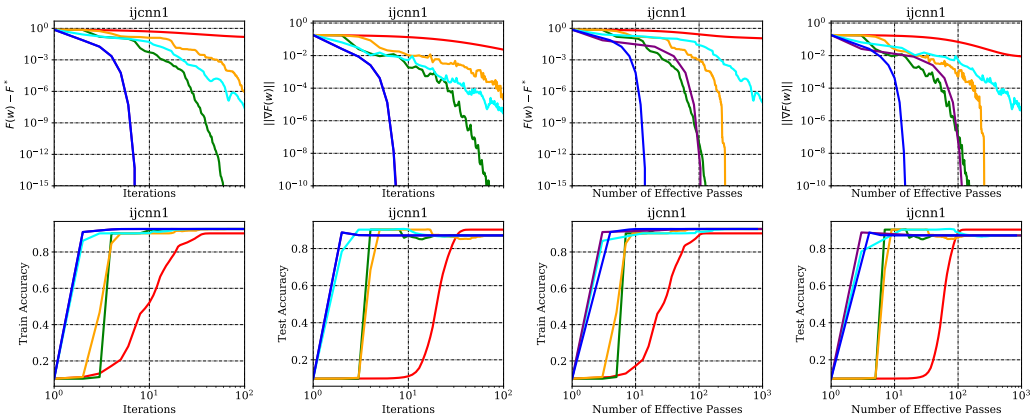


Figure 19: `ijcnn1`: Deterministic Logistic Regression with $\lambda = 10^{-5}$.

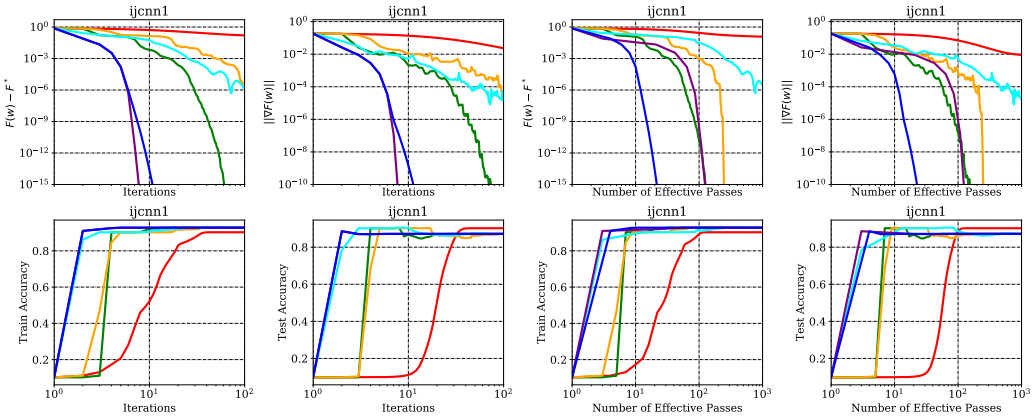


Figure 20: `ijcnn1`: Deterministic Logistic Regression with $\lambda = 10^{-6}$.

D.1.1 Sensitivity of SONIA to memory hyper-parameter

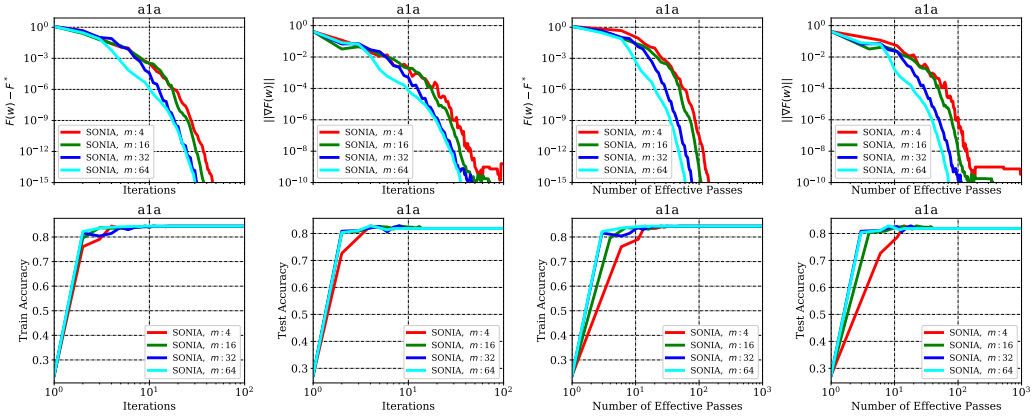


Figure 21: a1a: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

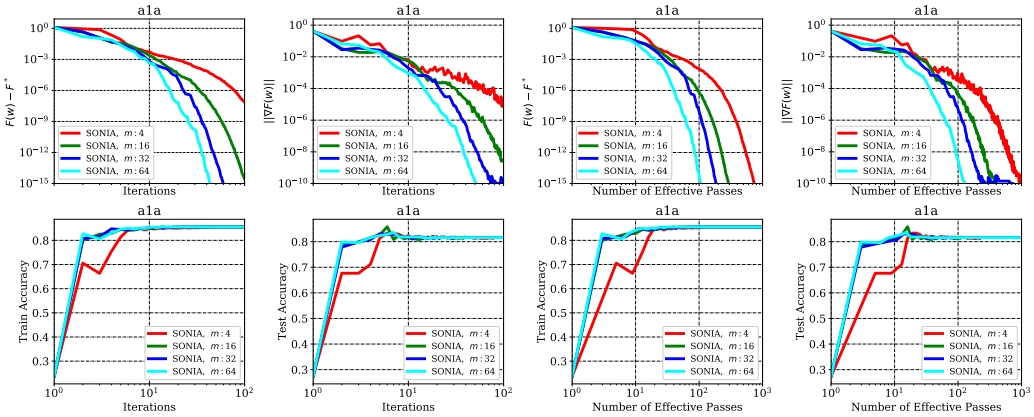


Figure 22: a1a: Sensitivity of SONIA to memory, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

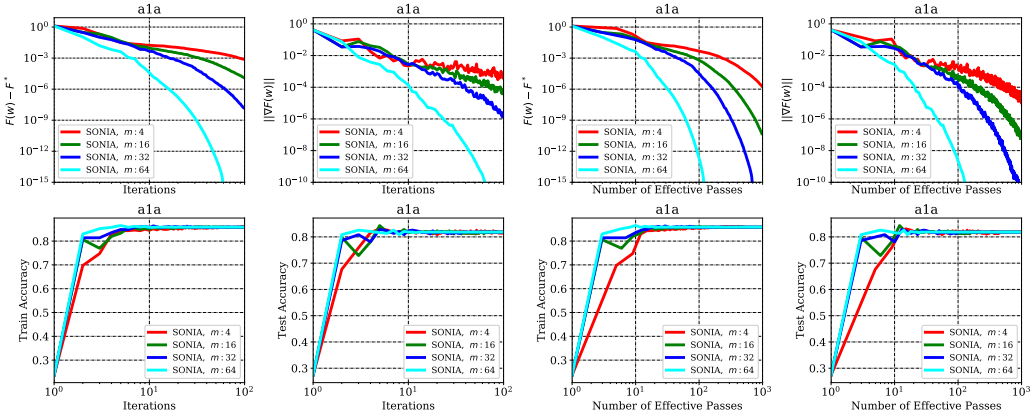


Figure 23: `a1a`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

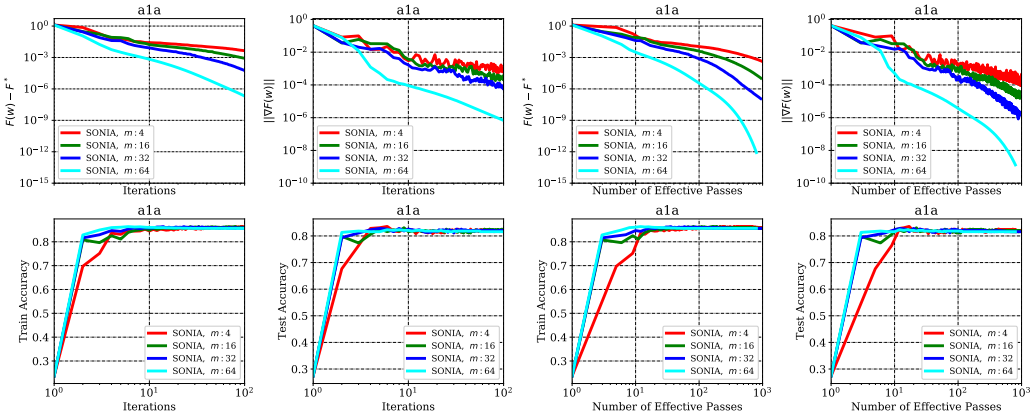


Figure 24: `a1a`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

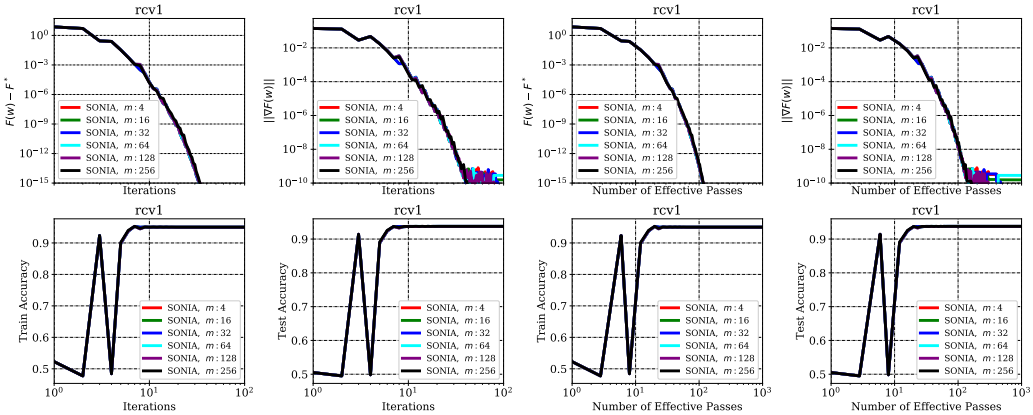


Figure 25: `rcv1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

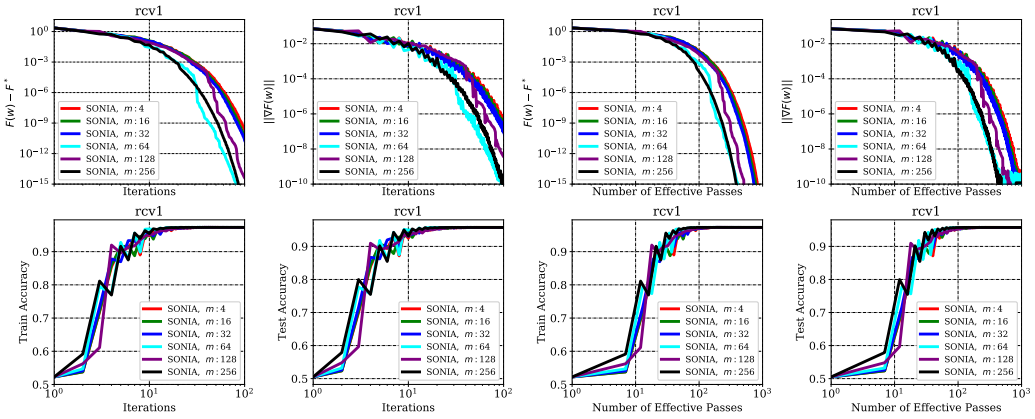


Figure 26: `rcv1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

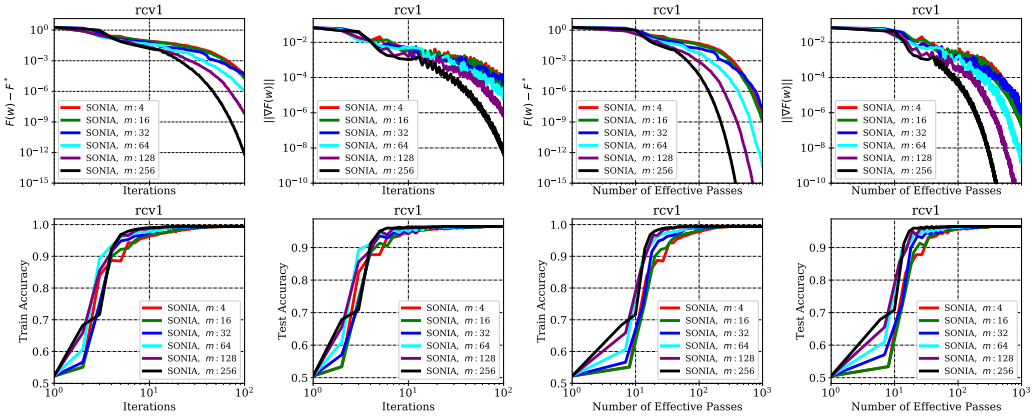


Figure 27: `rcv1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

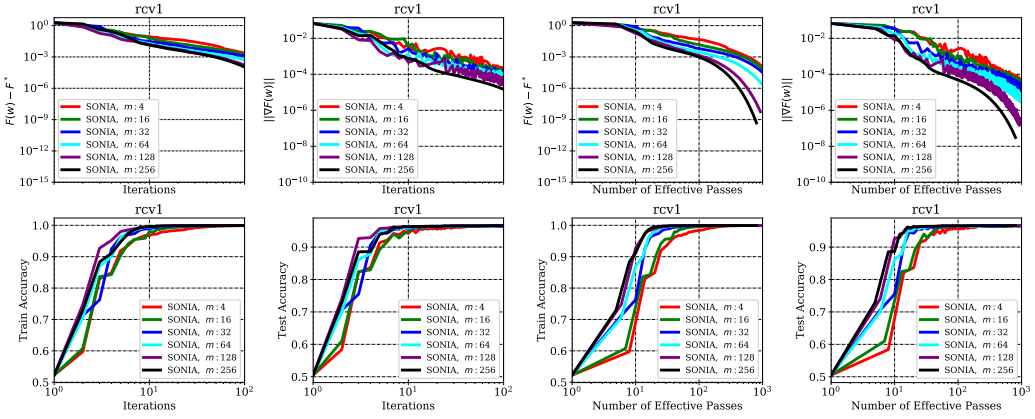


Figure 28: `rcv1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

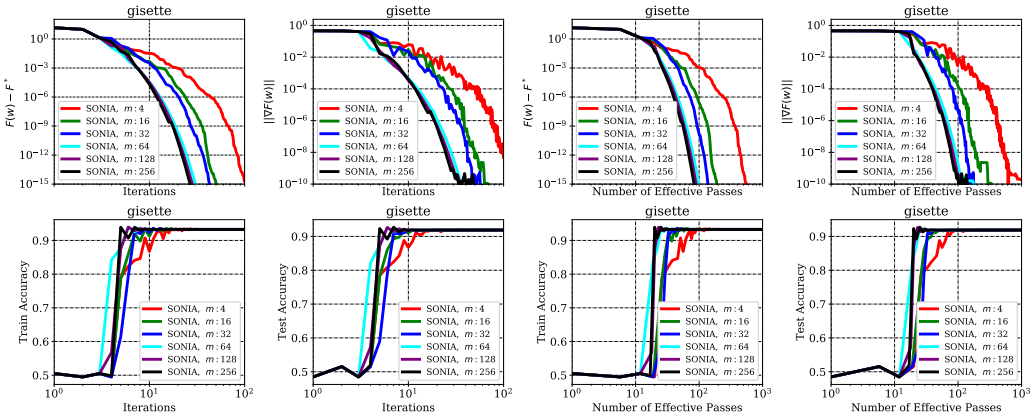


Figure 29: `gisette`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

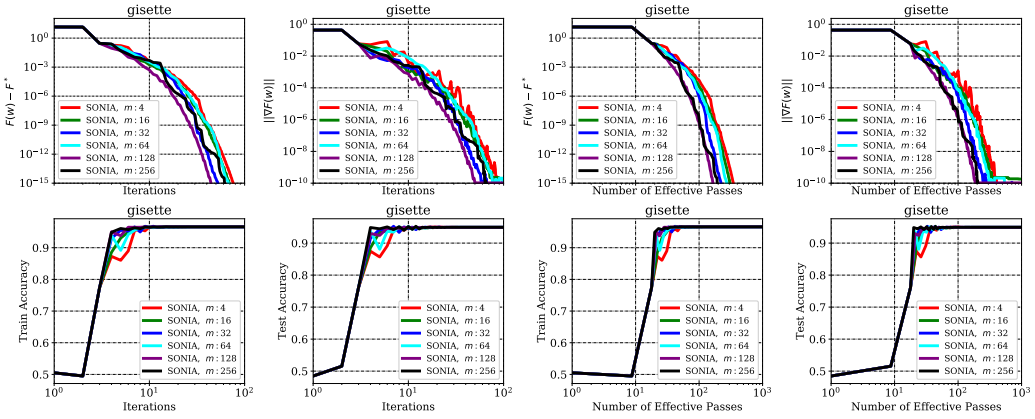


Figure 30: `gisette`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

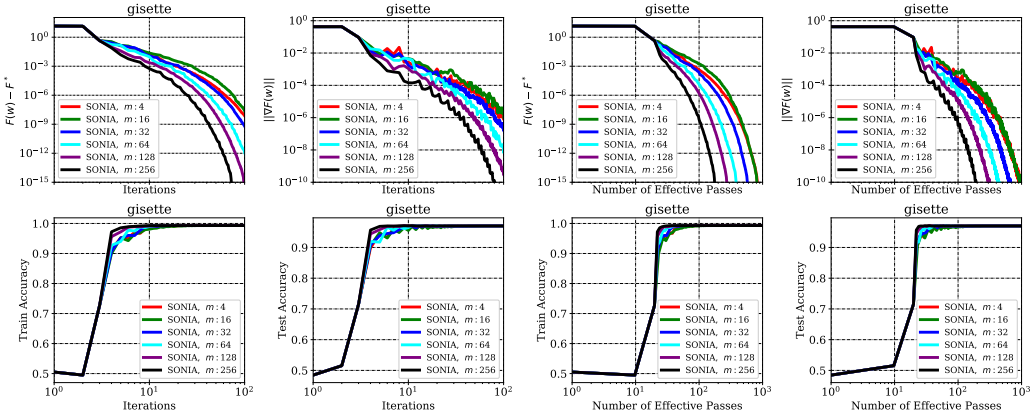


Figure 31: `gisette`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

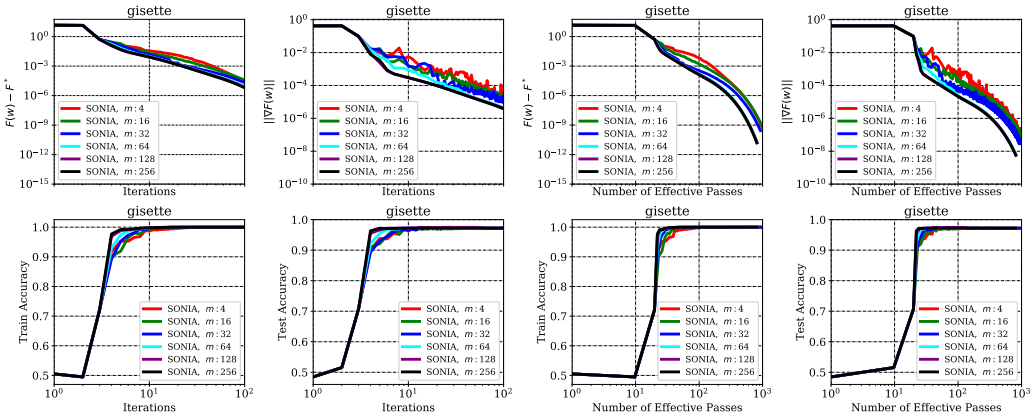


Figure 32: `gisette`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

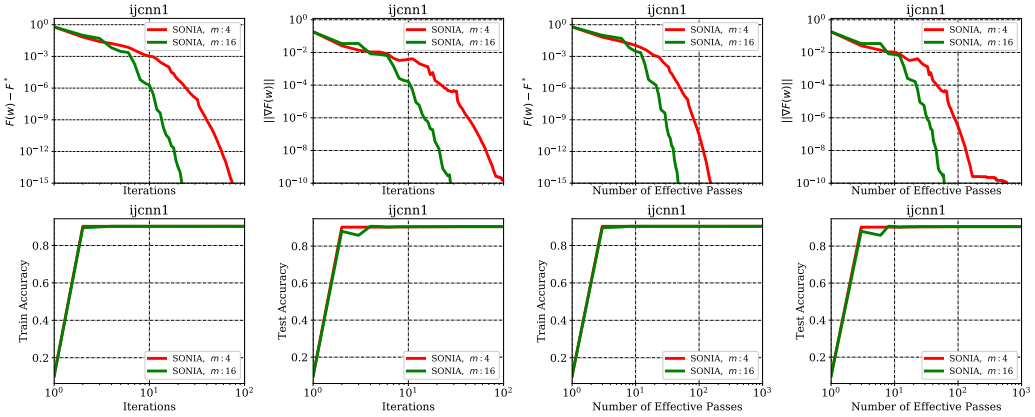


Figure 33: `ijcnn1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

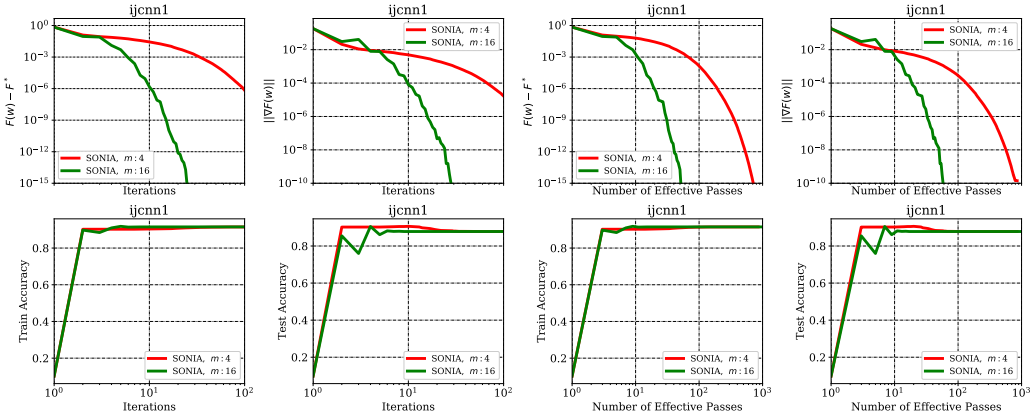


Figure 34: `ijcnn1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

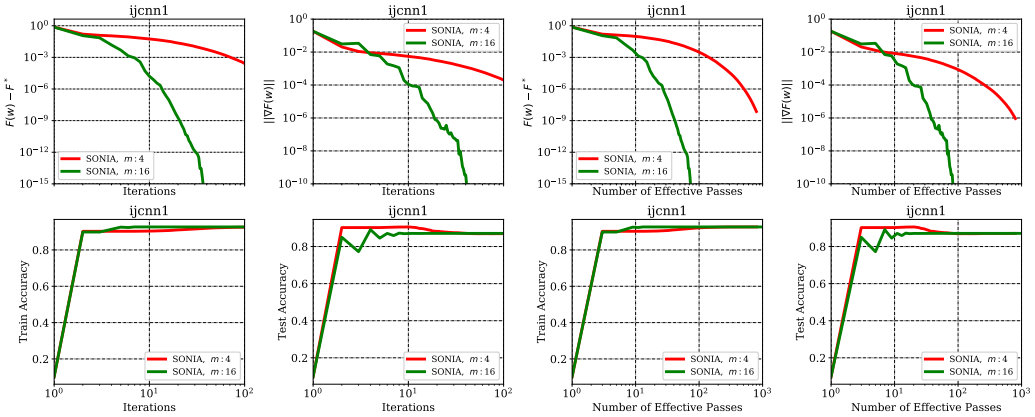


Figure 35: `ijcnn1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

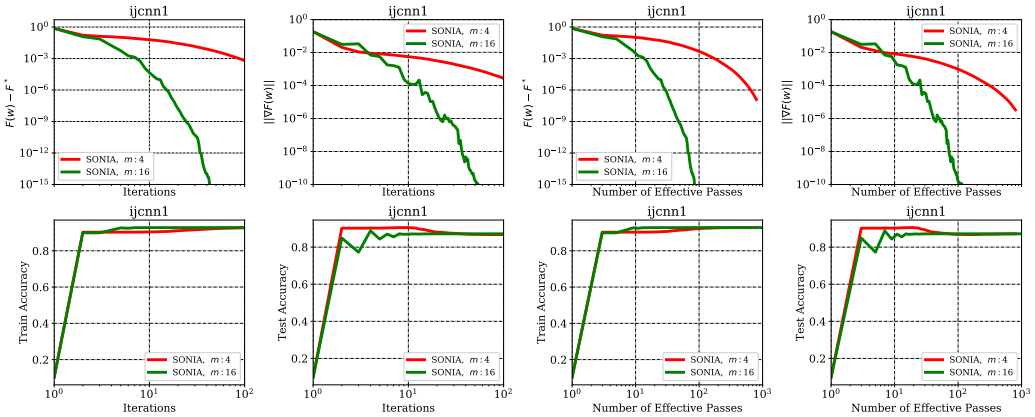


Figure 36: `ijcnn1`: Sensitivity of `SONIA` to memory, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

D.1.2 Sensitivity of SONIA to truncation hyper-parameter

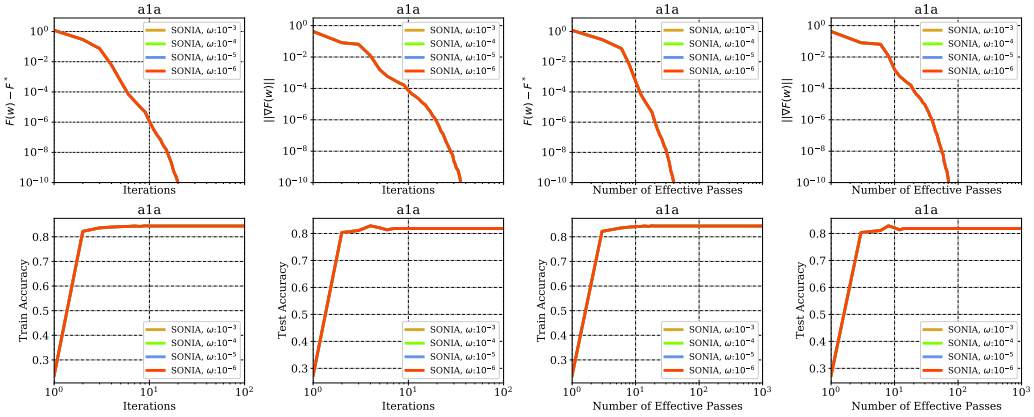


Figure 37: a1a: Sensitivity of SONIA to ω, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

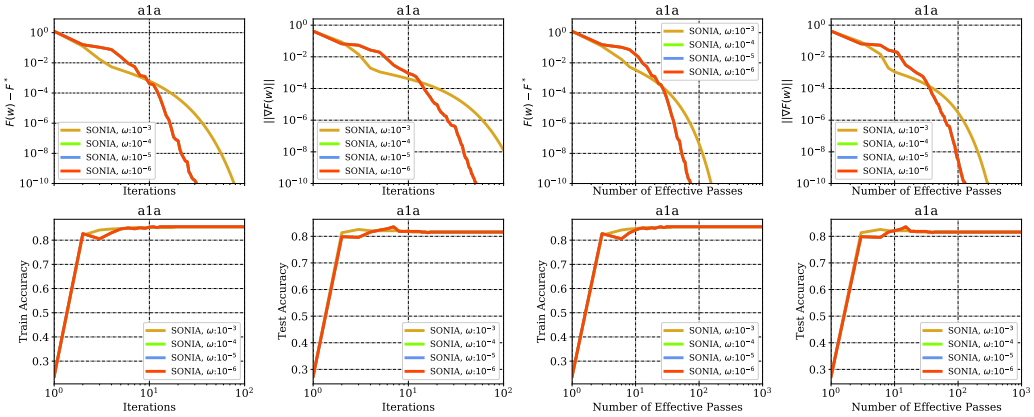


Figure 38: a1a: Sensitivity of SONIA to ω, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

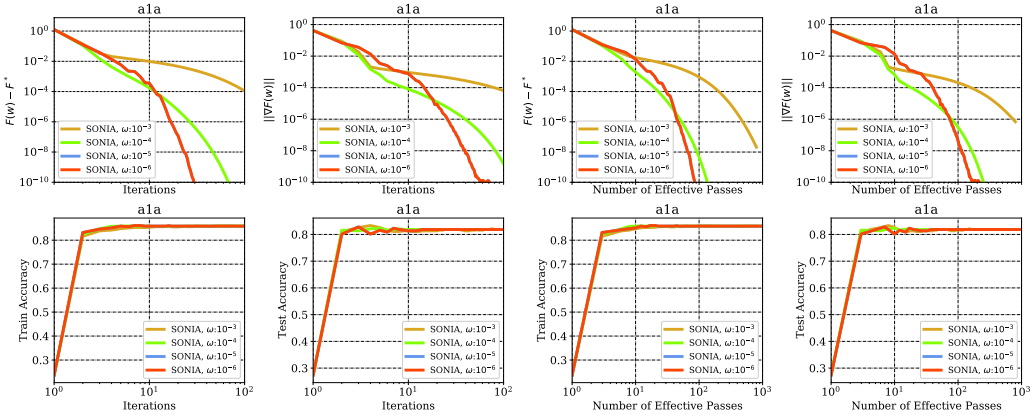


Figure 39: `a1a`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

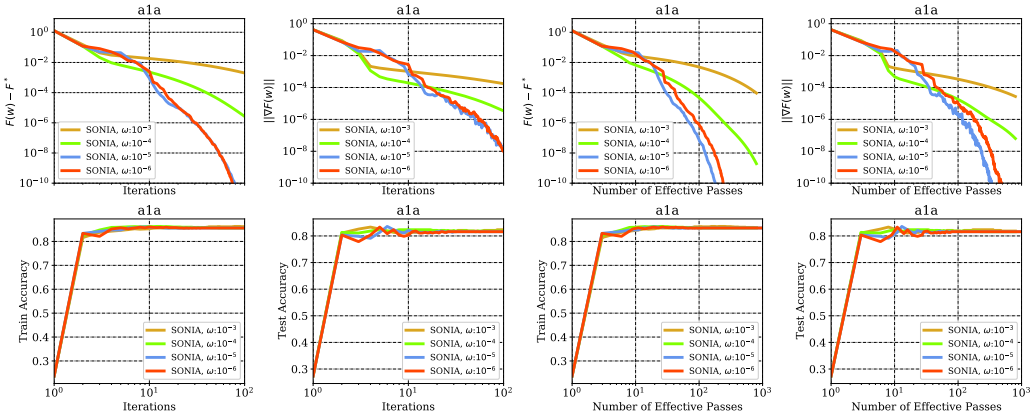


Figure 40: `a1a`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

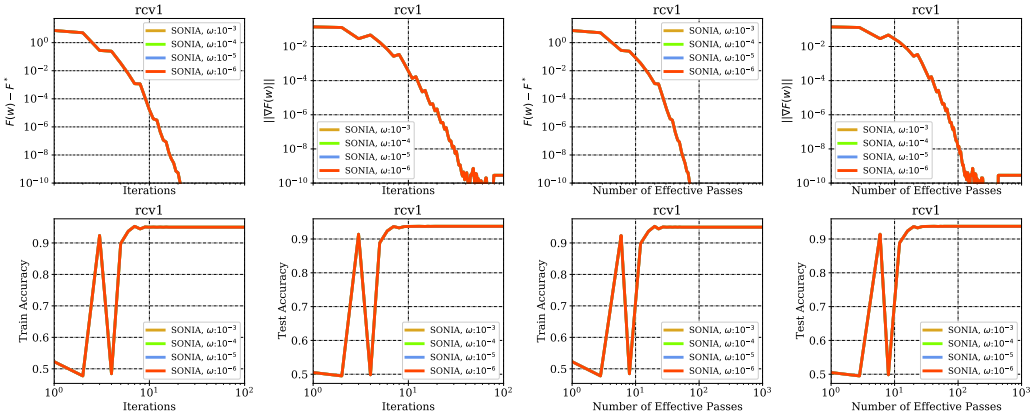


Figure 41: `rcv1`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

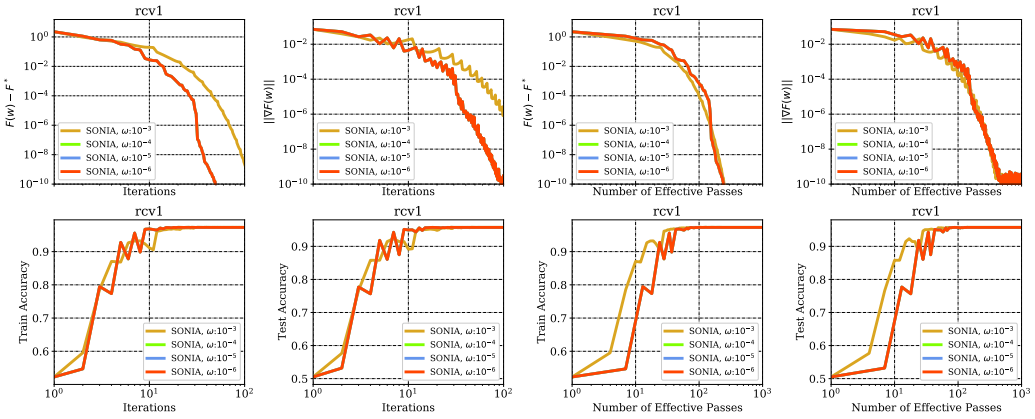


Figure 42: `rcv1`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

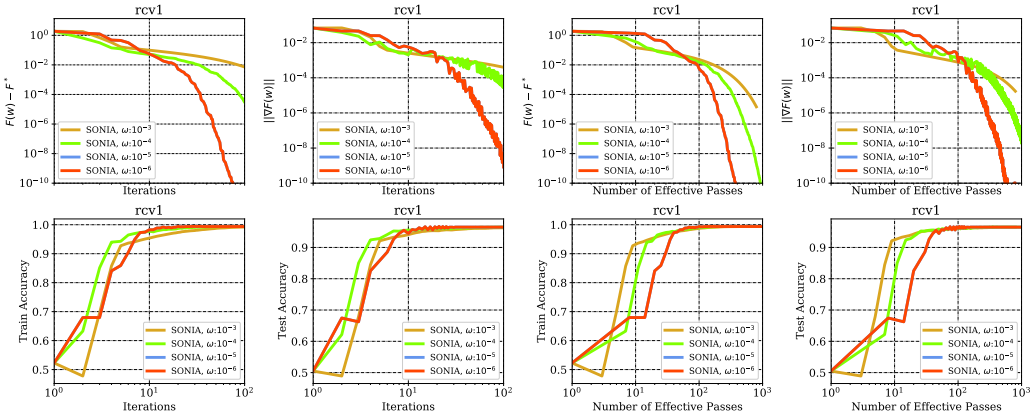


Figure 43: `rcv1`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

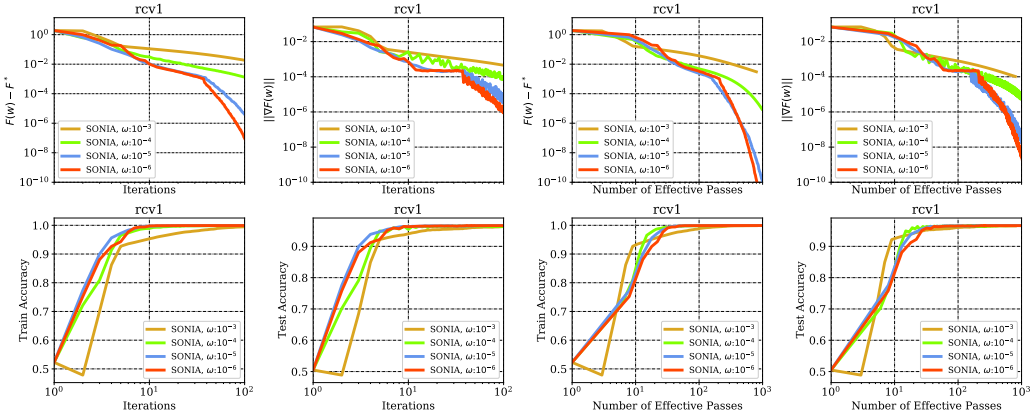


Figure 44: `rcv1`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

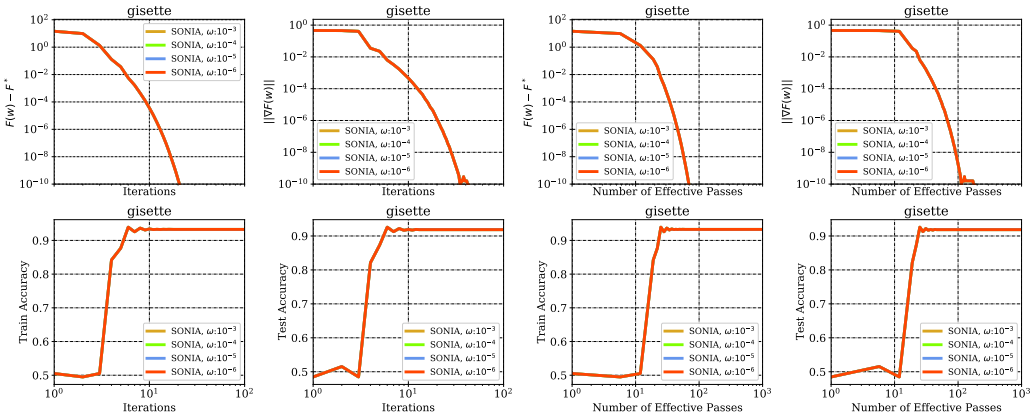


Figure 45: `gisette`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

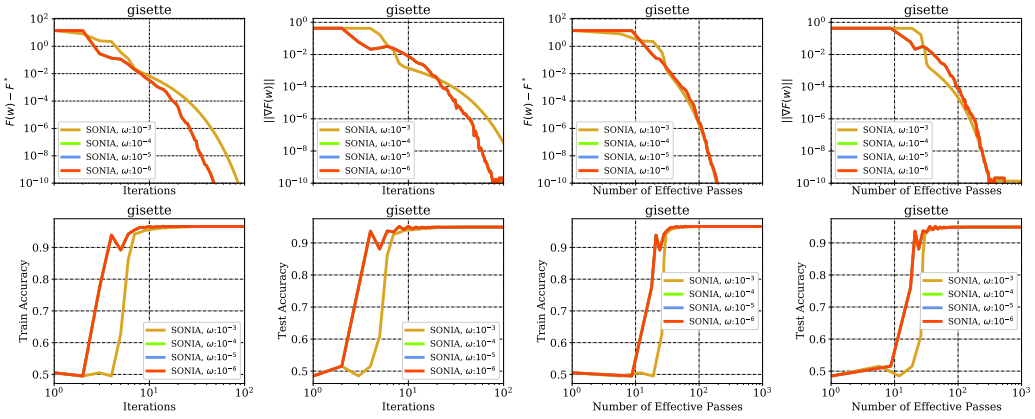


Figure 46: `gisette`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

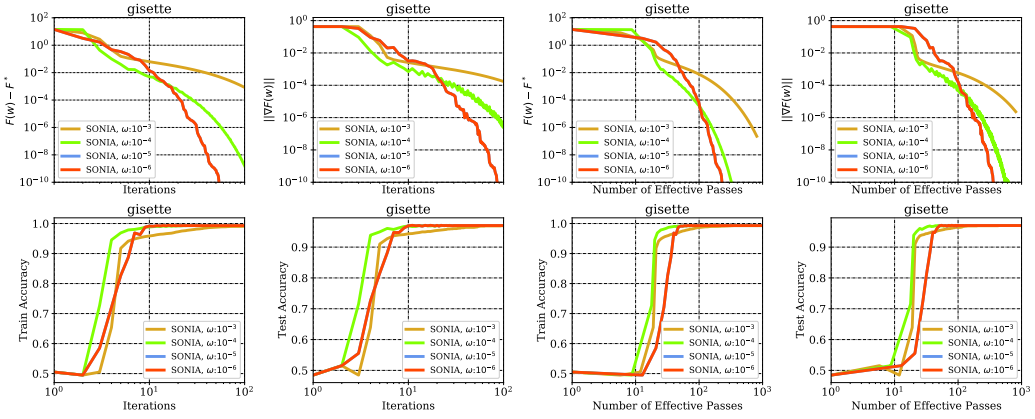


Figure 47: `gisette`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

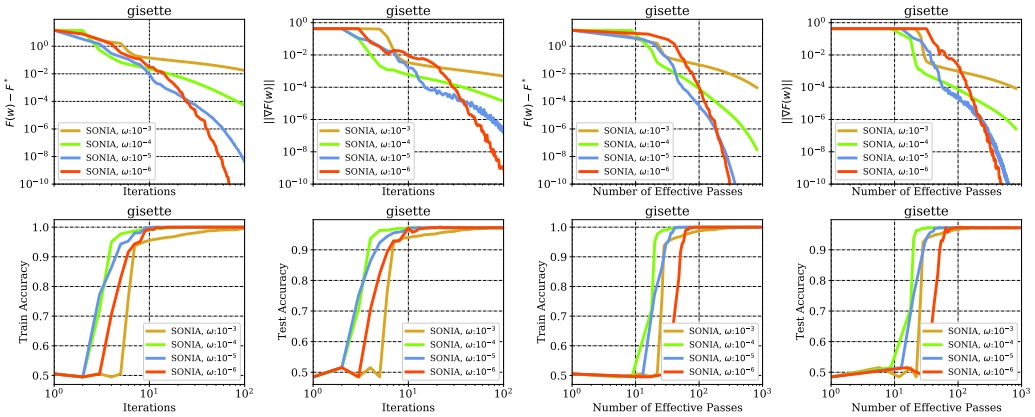


Figure 48: `gisette`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

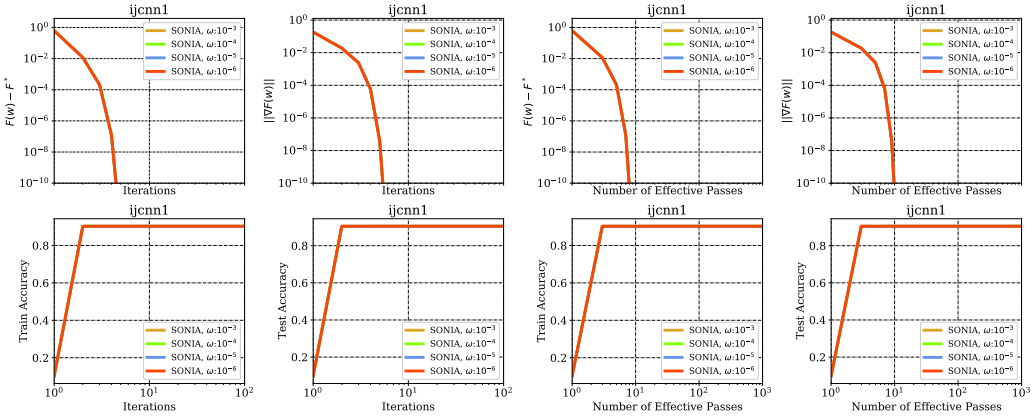


Figure 49: `ijcnn1`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-3}$).

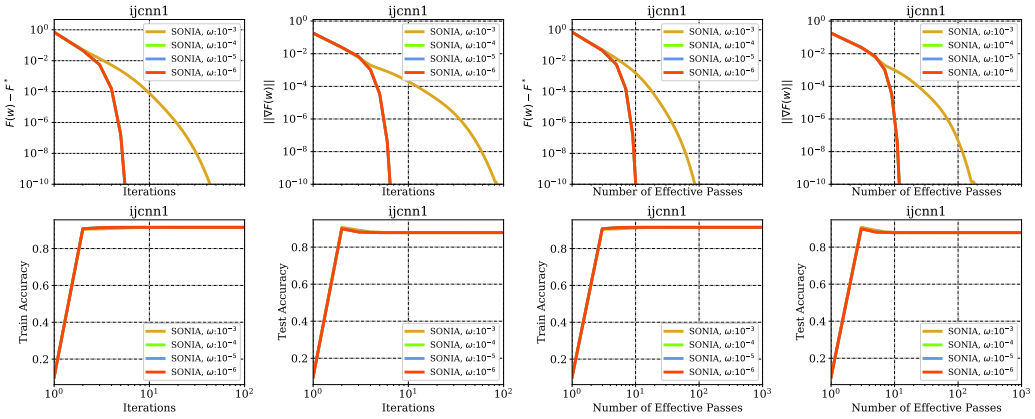


Figure 50: `ijcnn1`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-4}$).

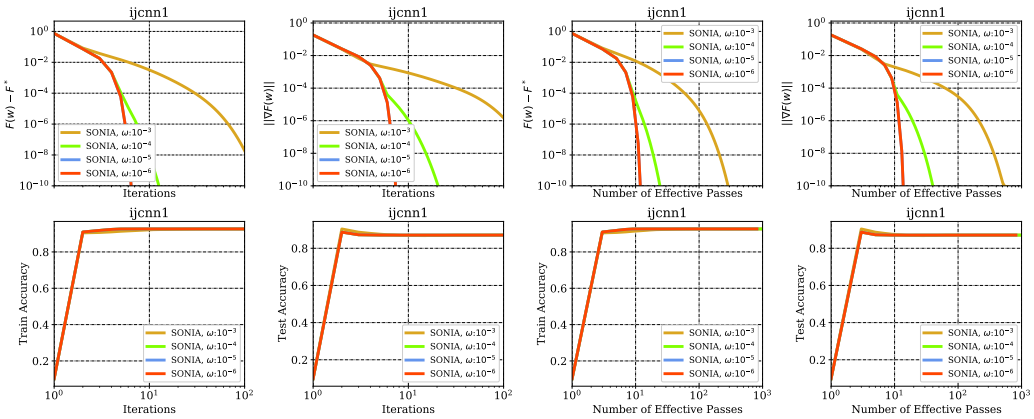


Figure 51: `ijcnn1`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-5}$).

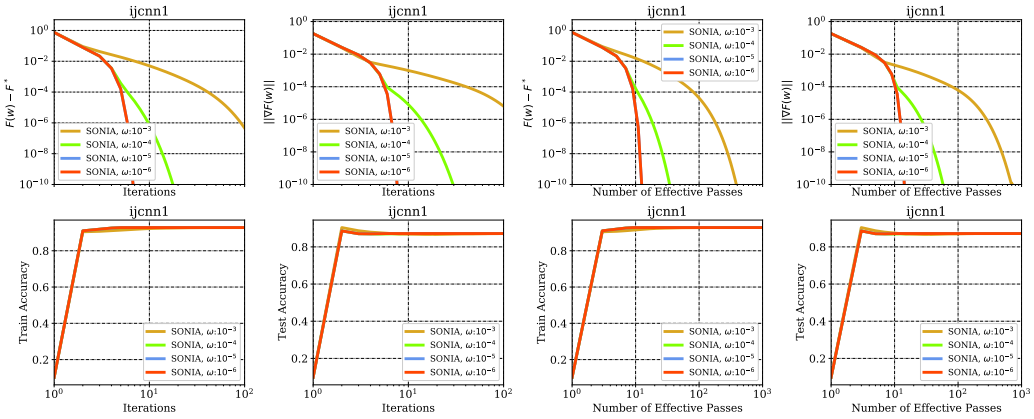


Figure 52: `ijcnn1`: Sensitivity of `SONIA` to ω, Deterministic Logistic Regression ($\lambda = 10^{-6}$).

D.2 Additional Numerical Experiments: Deterministic Nonconvex Functions

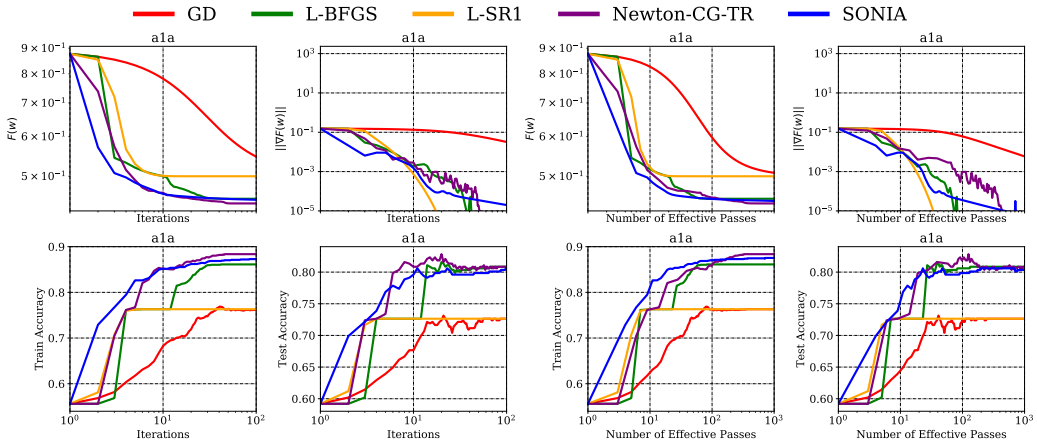


Figure 53: `a1a`: Deterministic Non-Linear Least Square

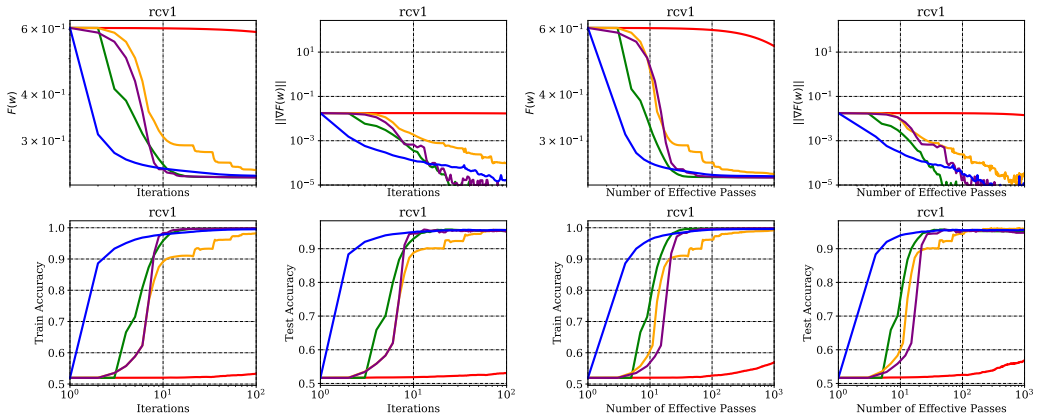


Figure 54: `rcv1`: Deterministic Non-Linear Least Square

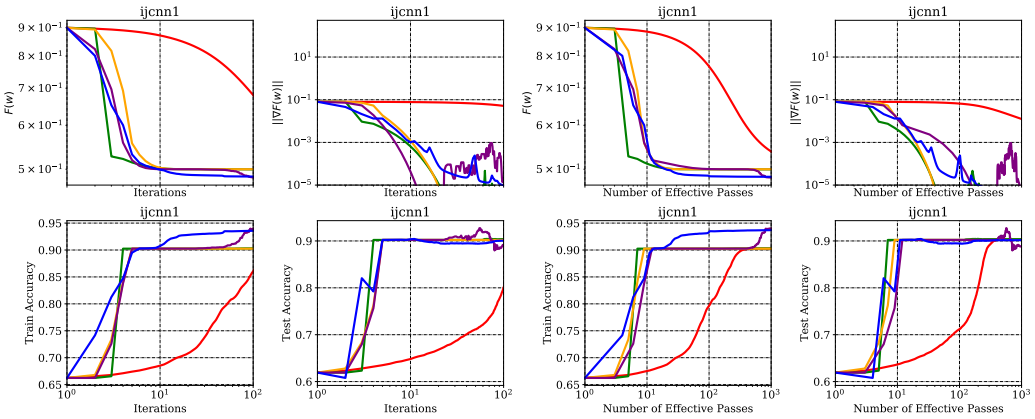


Figure 55: `ijcnn1`: Deterministic Non-Linear Least Square

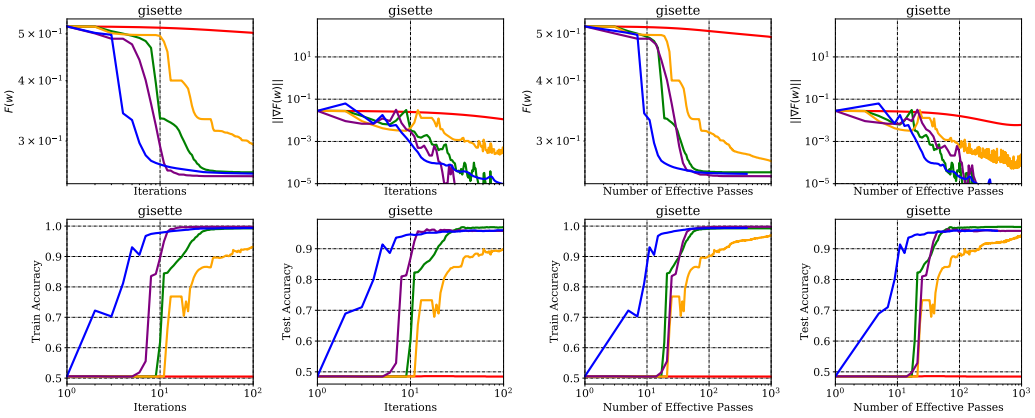


Figure 56: `gisette`: Deterministic Non-Linear Least Square

D.3 Additional Numerical Experiments: Stochastic Strongly Convex Functions

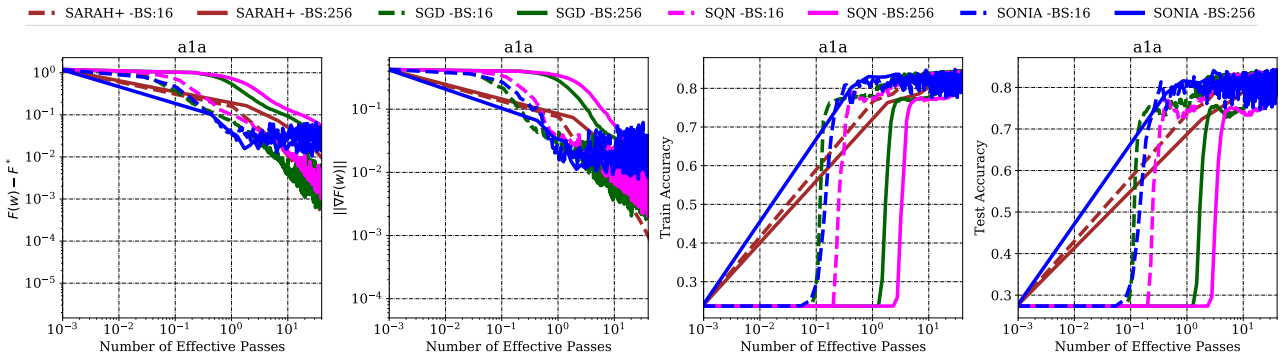


Figure 57: `a1a`: Stochastic Logistic Regression ($\lambda = 10^{-3}$).

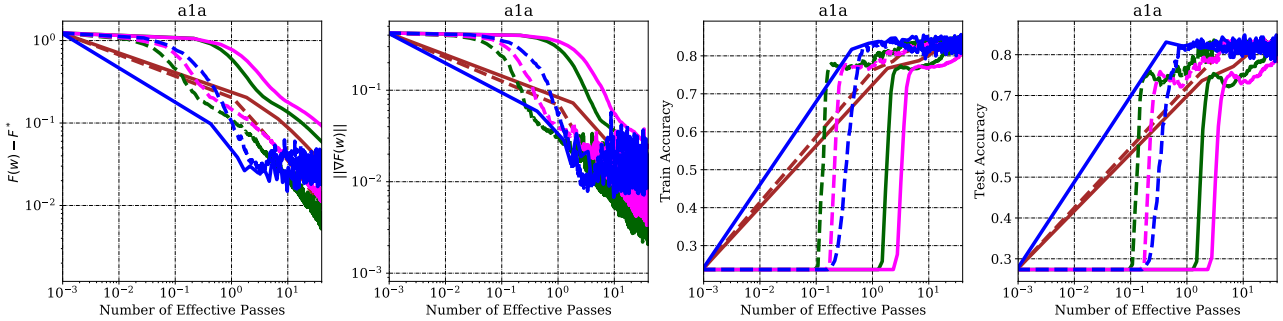


Figure 58: `a1a`: Stochastic Logistic Regression ($\lambda = 10^{-4}$).

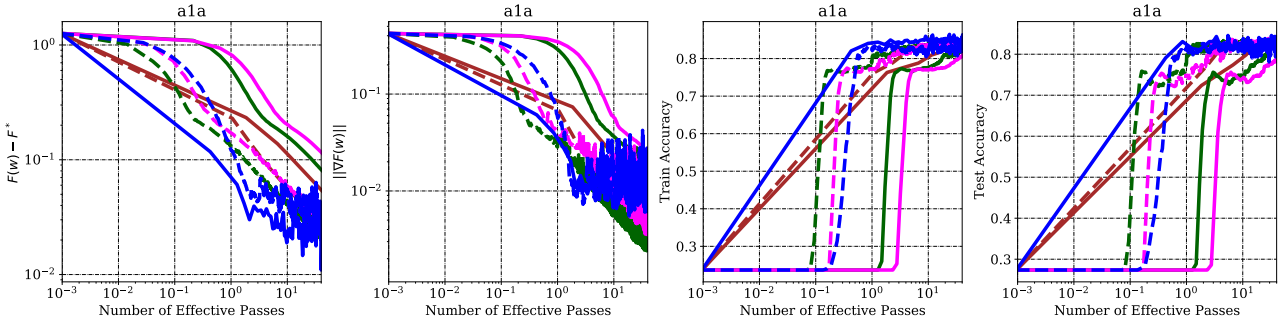


Figure 59: `a1a`: Stochastic Logistic Regression ($\lambda = 10^{-5}$).

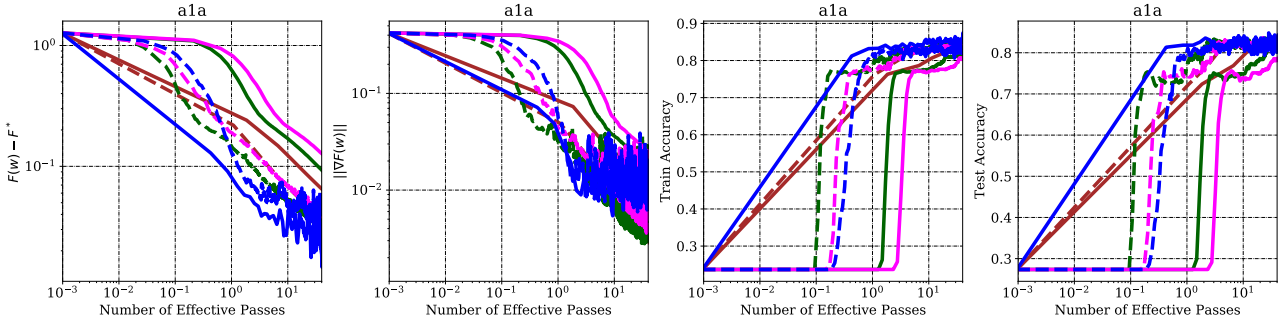


Figure 60: `a1a`: Stochastic Logistic Regression ($\lambda = 10^{-6}$).

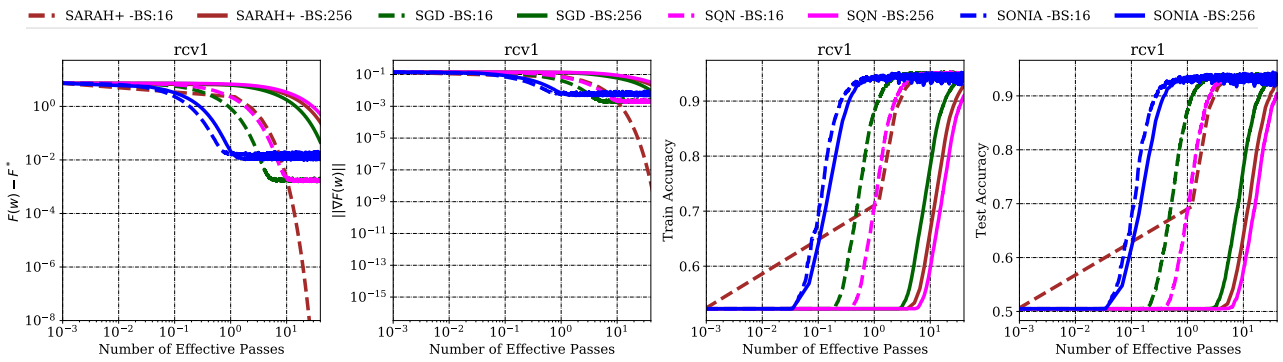


Figure 61: `rcv1`: Stochastic Logistic Regression ($\lambda = 10^{-3}$).

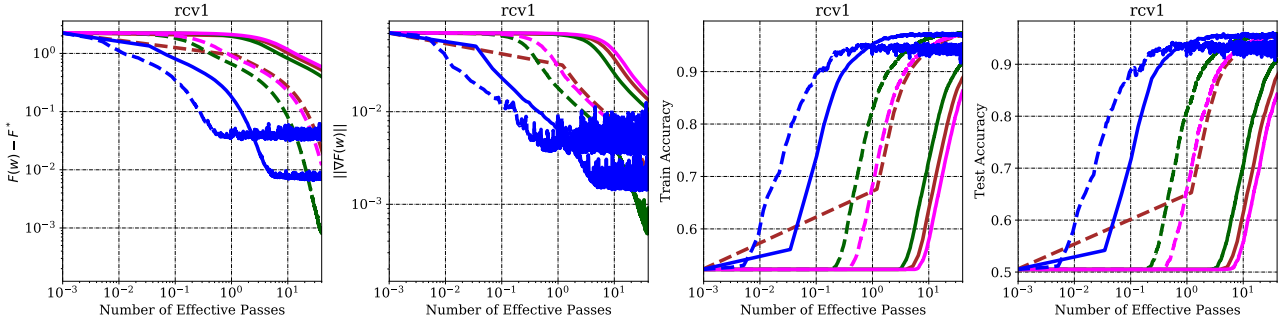


Figure 62: `rcv1`: Stochastic Logistic Regression ($\lambda = 10^{-4}$).

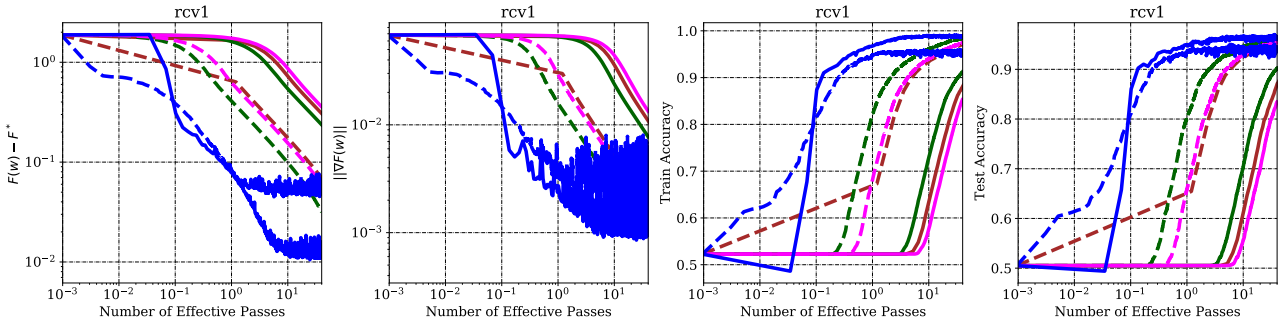


Figure 63: `rcv1`: Stochastic Logistic Regression ($\lambda = 10^{-5}$).

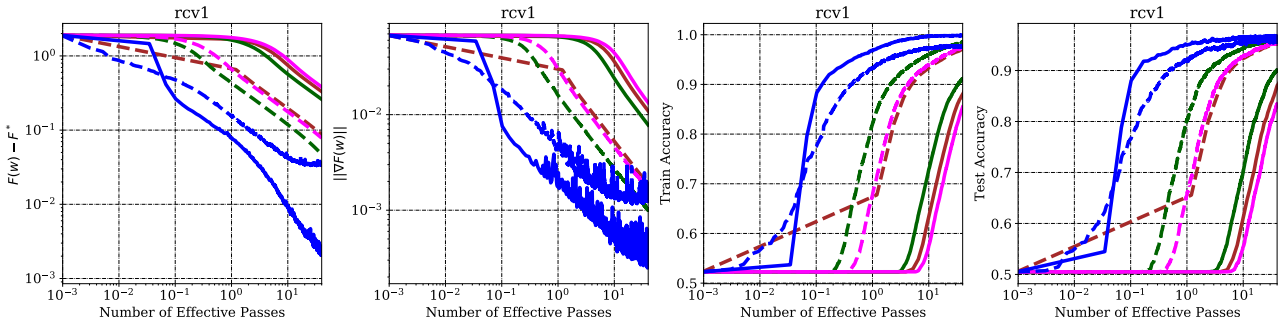


Figure 64: `rcv1`: Stochastic Logistic Regression ($\lambda = 10^{-6}$).

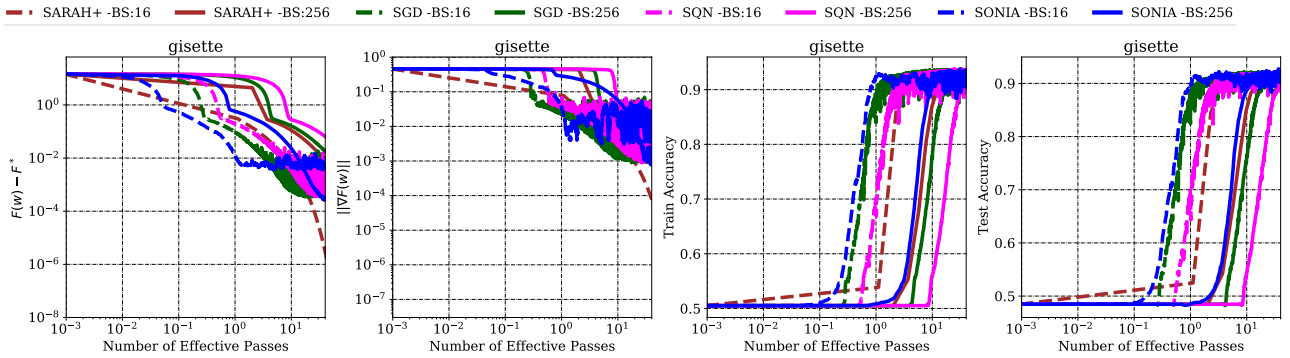


Figure 65: `gisette`: Stochastic Logistic Regression ($\lambda = 10^{-3}$).

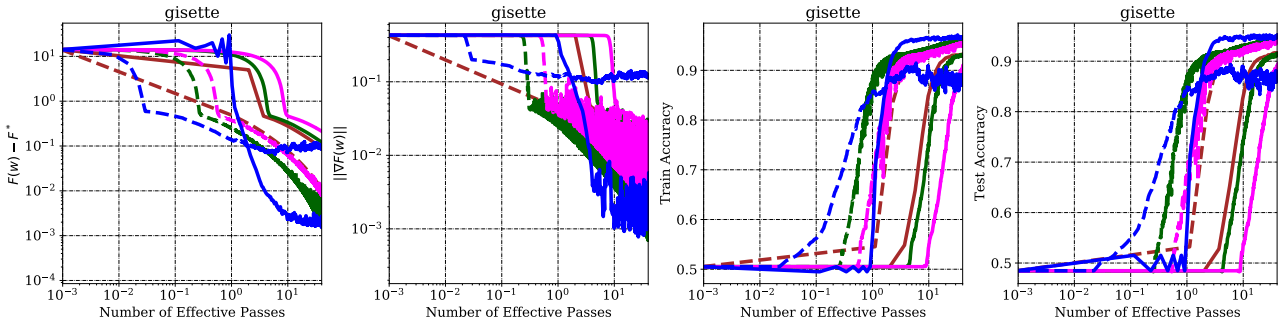


Figure 66: `gisette`: Stochastic Logistic Regression ($\lambda = 10^{-4}$).

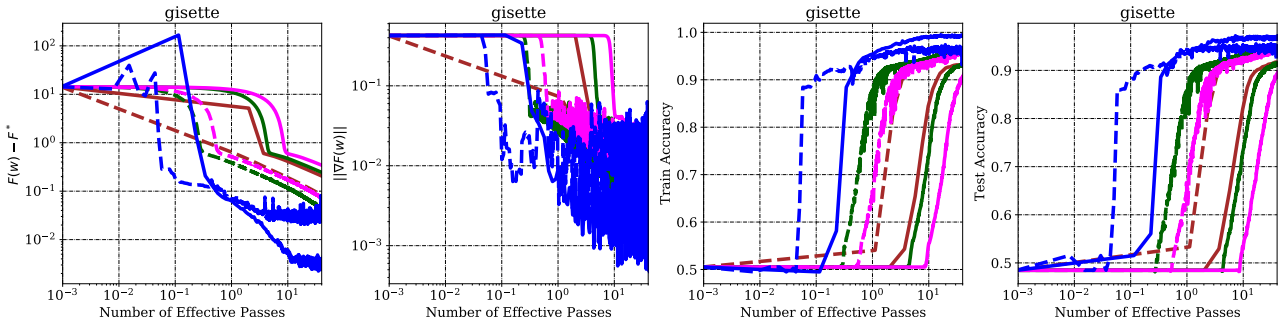


Figure 67: `gisette`: Stochastic Logistic Regression ($\lambda = 10^{-5}$).

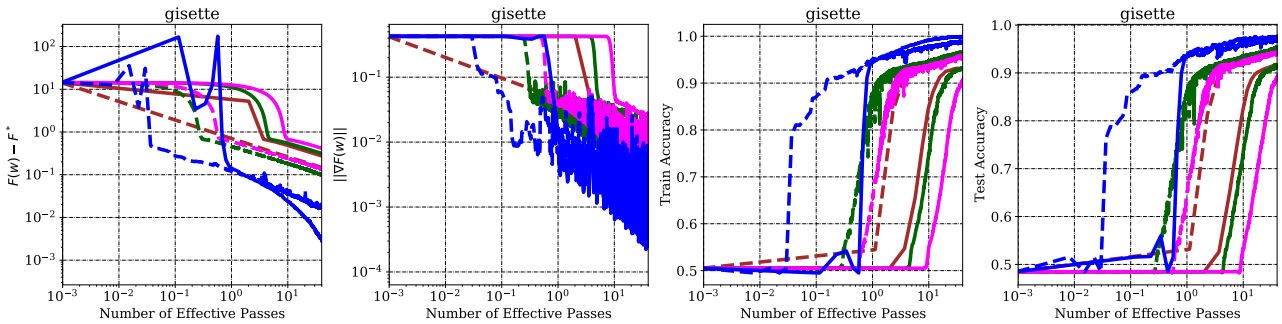


Figure 68: `gisette`: Stochastic Logistic Regression ($\lambda = 10^{-6}$).

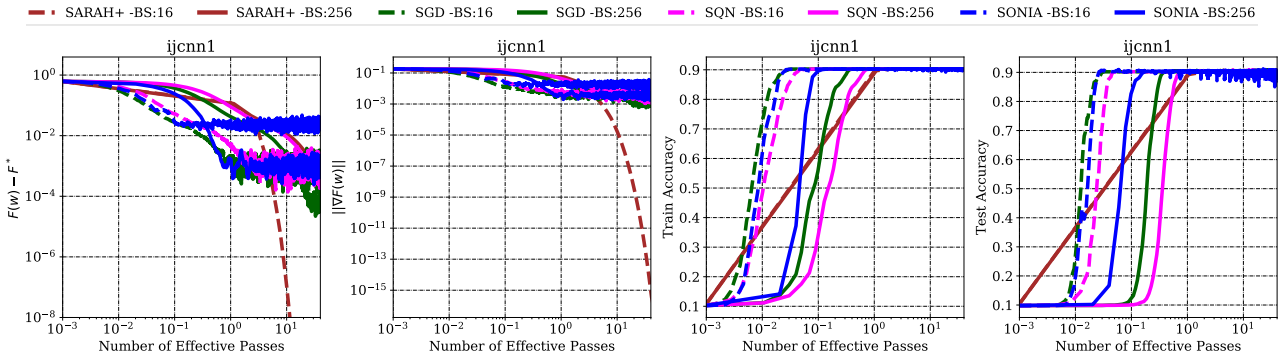


Figure 69: `ijcnn1`: Stochastic Logistic Regression ($\lambda = 10^{-3}$).

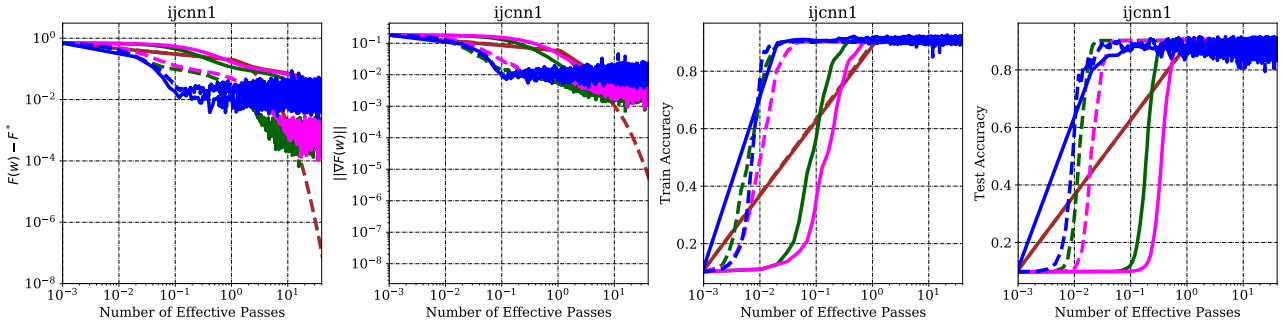


Figure 70: `ijcnn1`: Stochastic Logistic Regression ($\lambda = 10^{-4}$).

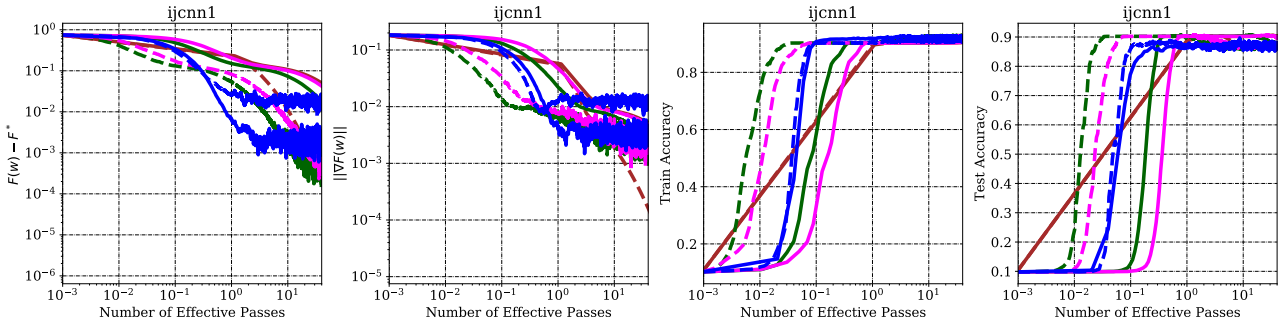


Figure 71: `ijcnn1`: Stochastic Logistic Regression ($\lambda = 10^{-5}$).

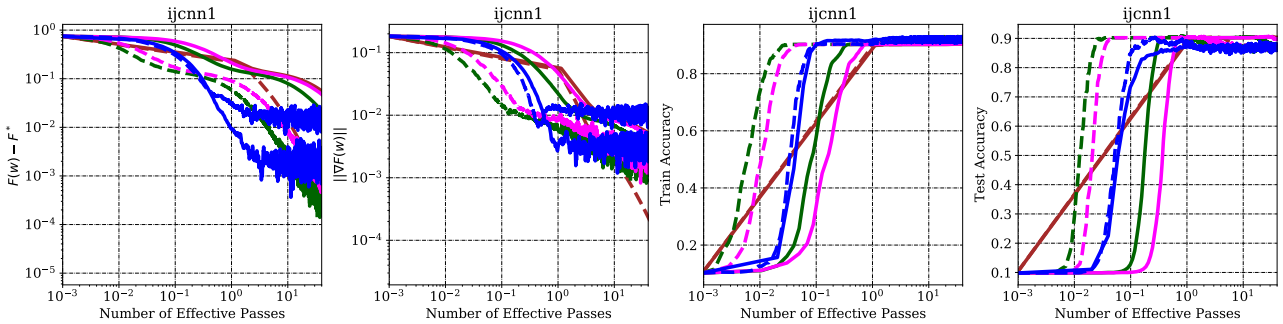


Figure 72: `ijcnn1`: Stochastic Logistic Regression ($\lambda = 10^{-6}$).

D.4 Additional Numerical Experiments: Stochastic Nonconvex Functions

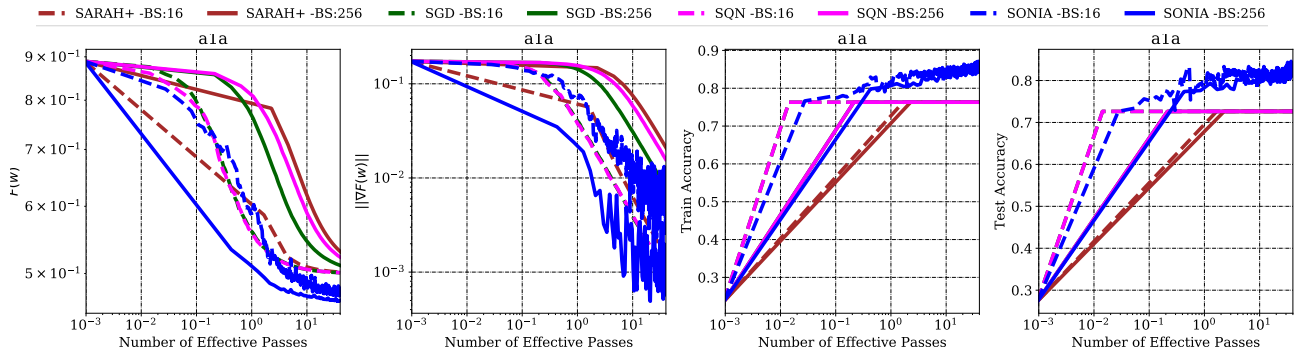


Figure 73: `a1a`: Stochastic Nonlinear Least Squares.

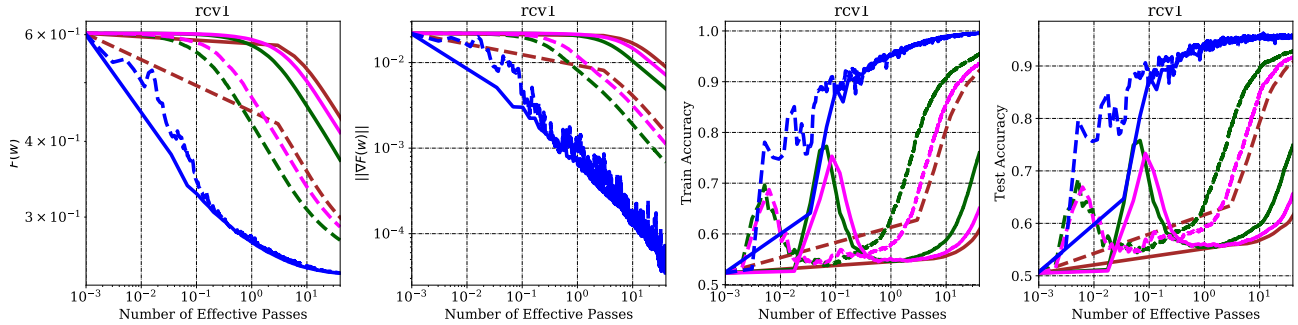


Figure 74: `rcv1`: Stochastic Nonlinear Least Squares.

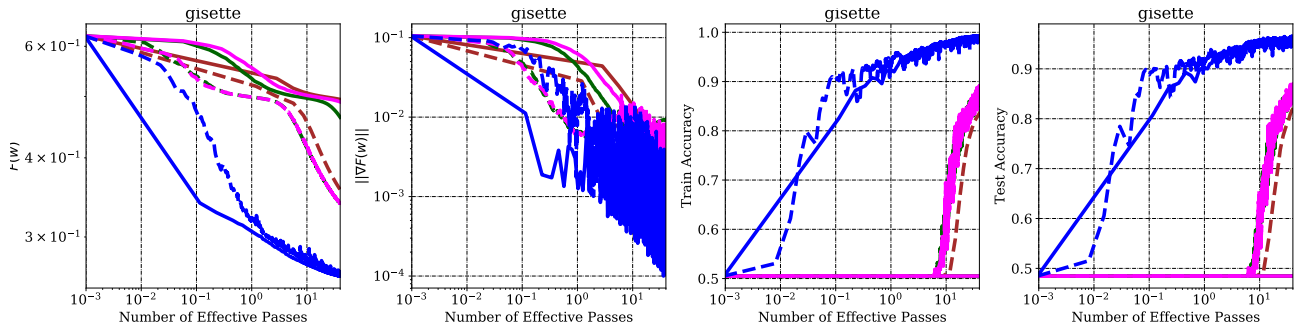


Figure 75: `gisette`: Stochastic Nonlinear Least Squares.

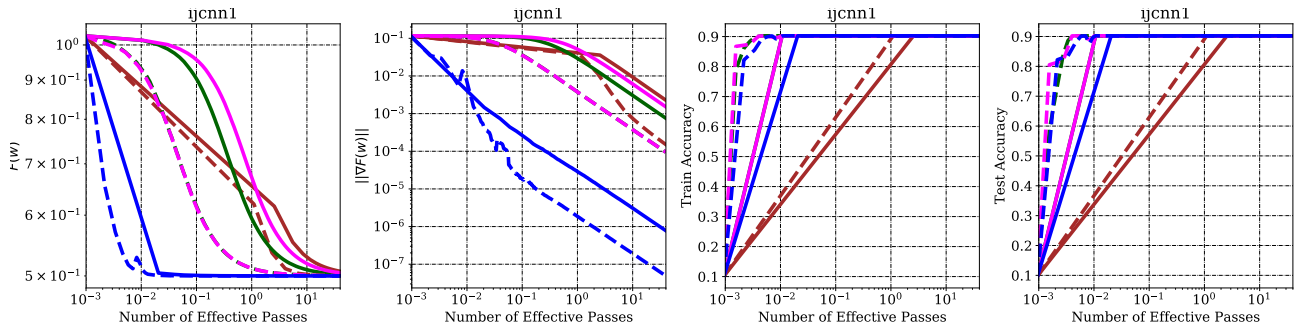
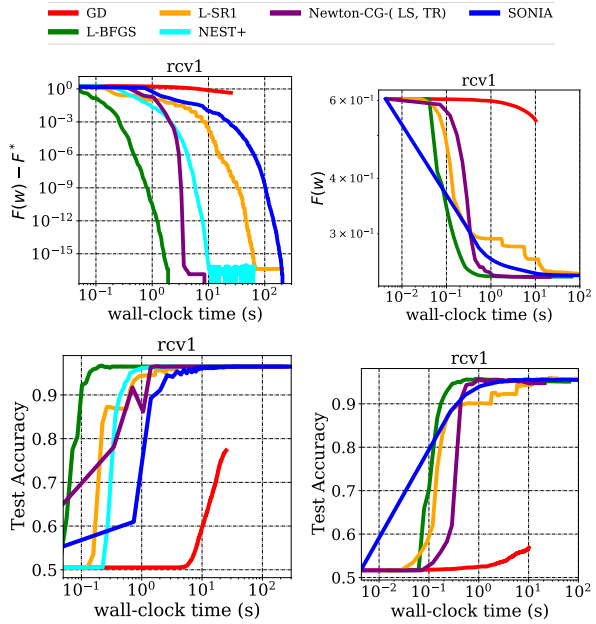
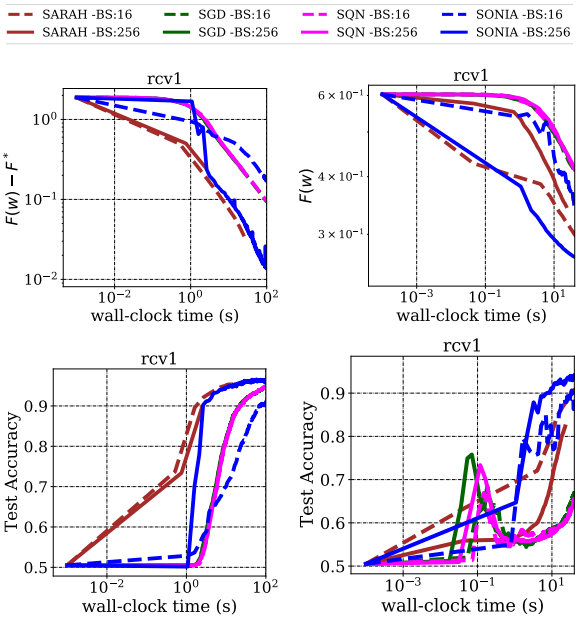


Figure 76: `ijcnn1`: Stochastic Nonlinear Least Squares.

D.5 Additional Numerical Experiments: Wall-clock Time Comparison



(a) Comparison of objective function and Test Accuracy for different algorithms on Deterministic Logistic Regression (first column) and Non-Linear Least Squares (second column) Problems.



(b) Comparison of objective function and Test Accuracy for different algorithms on Stochastic Logistic Regression (first column) and Non-Linear Least Squares (second column) Problems.

Figure 77: Comparison with respect to wall-clock time time. Dataset: `rcv1`.