09 appendix

ORCID: 0000-0000-0000-1234 Email: author@example.com DOI: 10.5281/zenodo.12345678

November 21, 2025

Contents

1 Appendix

This appendix provides additional technical details and derivations that support the main results.

1.1 A. Detailed Proofs

1.1.1 A.1 Proof of Convergence (Theorem 1)

The convergence rate established in (??) follows from the following detailed analysis.

Proof: Let x_k be the iterate at step k. From the update rule (??), we have:

$$x_{k+1} = x_k \ \ _k f(x_k) + {}_k(x_k \ \ x_{k1}) \tag{1.1}$$

By the Lipschitz continuity of f, there exists a constant $L > 0$ such that:

$$f(x) \ \ f(y) \ \ Lx \ \ y, \quad x, y \ \ X \tag{1.2}$$

Using strong convexity with parameter > 0 [? ?]:

$$f(y)\ \ f(x) + f(x)^T(y\ \ x) + \frac{}{2}y\ \ x^2 \tag{1.3}$$

Combining these properties with the adaptive step size rule (??), following the analysis framework in [? ?], we obtain the linear convergence rate with $= 1\ \ /L$. □

1.1.2 A.2 Complexity Analysis

The computational complexity per iteration is derived as follows:

1. Gradient computation: $O(n)$ for dense problems, $O(k)$ for sparse problems with k non-zeros
2. Update rule: $O(n)$ for vector operations
3. Adaptive step size: $O(1)$ for the update in (??)
4. Momentum term: $O(n)$ for the momentum computation

Total per-iteration complexity: $O(n)$ for dense problems.

For structured problems, we can exploit the separable structure of (??) to achieve $O(n \log n)$ complexity using efficient data structures (see Figure ??).

1.2 B. Additional Experimental Details

1.2.1 B.1 Hyperparameter Tuning

The following hyperparameters were used in our experiments:

Parameter	Symbol	Value	Range Tested
Learning rate	$_0$	0.01	[0.001, 0.1]
Momentum		0.9	[0.5, 0.99]
Regularization		0.001	[0, 0.01]
Tolerance		10^6	$[10^8, 10^4]$

Table 1. Hyperparameter settings used in experiments

1.2.2 B.2 Computational Environment

All experiments were conducted on: - CPU: Intel Xeon E5-2690 v4 @ 2.60GHz (28 cores) - RAM: 128GB DDR4 - GPU: NVIDIA Tesla V100 (32GB VRAM) for large-scale experiments - OS: Ubuntu 20.04 LTS - Python: 3.10.12 - NumPy: 1.24.3 - SciPy: 1.10.1

1.2.3 B.3 Dataset Preparation

Datasets were preprocessed using standard normalization:

$$x_i = \frac{x_i}{} \tag{1.4}$$

where and are the mean and standard deviation computed from the training set.

1.3 C. Extended Results

1.3.1 C.1 Additional Benchmark Comparisons

Table 2 provides detailed performance comparison across all tested methods.

Method	Time (s)	Iterations	Final Error	Memory (MB)
Our Method	12.3	245	1.2 Œ 10^6	156
Gradient Descent	18.7	412	1.5 Œ 10^6	312
Adam	15.4	358	1.4 Œ 10^6	298
L-BFGS	16.2	198	1.1 Œ 10^6	425

Table 2. Extended performance comparison with computational details

1.3.2 C.2 Sensitivity Analysis

Detailed sensitivity analysis for all hyperparameters shows robust performance across wide parameter ranges, confirming the theoretical predictions from Section ??.

1.4 D. Implementation Details

1.4.1 D.1 Pseudocode

```
\KeywordTok{def}\NormalTok{ optimize(f, x0, alpha0, beta, max\_iter, tol):}
    \CommentTok{"""}
\CommentTok{    Optimization algorithm implementation.}
\CommentTok{    }
\CommentTok{    Args:}
\CommentTok{        f: Objective function}
\CommentTok{        x0: Initial point}
\CommentTok{        alpha0: Initial learning rate}
\CommentTok{        beta: Momentum coefficient}
\CommentTok{        max\_iter: Maximum iterations}
\CommentTok{        tol: Convergence tolerance}
\CommentTok{    }
\CommentTok{    Returns:}
\CommentTok{        x\_opt: Optimal solution}
\CommentTok{        history: Convergence history}
\CommentTok{    """}
\NormalTok{    x }\OperatorTok{=}\NormalTok{ x0}
\NormalTok{    x\_prev }\OperatorTok{=}\NormalTok{ x0}
```

```
\NormalTok{    history }\OperatorTok{=}\NormalTok{ []}
\NormalTok{    grad\_sum\_sq }\OperatorTok{=} \DecValTok{0}

    \ControlFlowTok{for}\NormalTok{ k }\KeywordTok{in} \BuiltInTok{range}\NormalTok{(max\_iter):}
        \CommentTok{\# Compute gradient}
\NormalTok{        grad }\OperatorTok{=}\NormalTok{ compute\_gradient(f, x)}
\NormalTok{        grad\_sum\_sq }\OperatorTok{+=}\NormalTok{ np.linalg.norm(grad)}\OperatorTok{**}

        \CommentTok{\# Adaptive step size}
\NormalTok{        alpha }\OperatorTok{=}\NormalTok{ alpha0 }\OperatorTok{/}\NormalTok{ np.sqrt()}\D

        \CommentTok{\# Update with momentum}
\NormalTok{        x\_new }\OperatorTok{=}\NormalTok{ x }\OperatorTok{{-}}\NormalTok{ alpha }\Opera

        \CommentTok{\# Check convergence}
        \ControlFlowTok{if}\NormalTok{ np.linalg.norm(x\_new }\OperatorTok{{-}}\NormalTok{ x) }\Ope
            \ControlFlowTok{break}

        \CommentTok{\# Update history}
\NormalTok{        history.append(\{}\StringTok{\textquotesingle{}iter\textquotesingle{}}\NormalTok

        \CommentTok{\# Prepare next iteration}
\NormalTok{        x\_prev }\OperatorTok{=}\NormalTok{ x}
\NormalTok{        x }\OperatorTok{=}\NormalTok{ x\_new}

    \ControlFlowTok{return}\NormalTok{ x, history}
```

1.4.2 D.2 Performance Optimizations

Key performance optimizations implemented: 1. Vectorized operations using NumPy 2. Sparse matrix representations when applicable 3. In-place updates to reduce memory allocation 4. Parallel gradient computations for separable problems