Comparison of optimization strategies: Coordinate Descent, Stochastic Gradient Descent, and L-BFGS

Anay Kulkarni
Department of Computer Science
UC San Diego
ankulkarni@ucsd.edu

Abstract

Coordinate descent is an optimization technique that updates one parameter at a time. This report explores its variations and compares their convergence with stochastic gradient descent (SGD) and L-BFGS on a logistic regression task using the Wine dataset. We implement coordinate descent strategies based on random, steepest, and shallowest gradient selection, finding that the steepest gradient selection improves convergence while the shallowest often stagnates. SGD outperforms both coordinate descent and L-BFGS, achieving the lowest final training loss. We also analyze k-sparse coordinate descent, which enforces sparsity by limiting the number of nonzero parameters. Higher sparsity increases variability in loss, with overly aggressive pruning leading to suboptimal convergence. Our findings emphasize that while coordinate descent is effective, its performance depends on the update strategy, and SGD remains the most efficient method for this task.

1 Steepest-Coordinate Descent

In this approach, we aim to minimize the objective function $L(w)$ by iteratively updating a single coordinate at a time. At each iteration, we follow these steps:

1. **Choosing the Coordinate:** We select the coordinate i corresponding to the steepest gradient, i.e., the one with the largest value of the partial derivative $\frac{\partial L}{\partial w_i}$. This ensures that the most significant direction of change is prioritized in each update.

2. **Updating the Coordinate:** The selected coordinate w_i is updated using a standard gradient descent step:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i}$$

where η is the learning rate, controlling the step size.

This method requires $L(w)$ to be differentiable but does not necessarily require second-order continuity. It is particularly effective when the function is convex or has a well-defined gradient structure.

2 Convergence

The coordinate descent method converges to the optimal loss under certain conditions:

Convexity: If the objective function $L(w)$ is convex, the method is guaranteed to converge to the global minimum. Strong convexity further ensures a unique minimum and faster convergence.

Lipschitz Continuity of the Gradient: If $L(w)$ has a Lipschitz-continuous gradient, meaning that the gradient does not change too abruptly, the updates remain stable and facilitate smooth convergence.

Proper Learning Rate: A well-chosen step size η is crucial. If η is too large, the method may overshoot and diverge; if it is too small, convergence may be slow. A diminishing learning rate can help ensure convergence in non-convex settings.

Selection of Coordinates: Since we choose the coordinate with the steepest gradient, this ensures rapid initial progress. However, for full convergence, all coordinates must be updated sufficiently often.

Differentiability of $L(w)$: Since the method relies on gradient-based updates, differentiability of $L(w)$ is necessary. While second-order continuity is not strictly required, having smooth gradients aids in stable updates.

For convex functions, the method is guaranteed to converge to the global minimum, whereas for non-convex functions, it may converge to a local minimum or a stationary point.

3 Experimental Setup

In this experiment, we compare the convergence behavior of different optimization strategies—Coordinate Descent, Stochastic Gradient Descent (SGD), and Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS).

Dataset

We use the classic Wine dataset, which originally contains three classes. To simplify the classification task into a binary setting, we remove instances belonging to the third class, keeping only data points labeled as 0 or 1.

Baseline Benchmark

To establish a baseline for comparison, we train the logistic regression model implemented in `sklearn` with its default settings, which uses the L-BFGS solver for optimization. The model is trained on 80% of the dataset, and we record the final loss after convergence. Our goal is to assess whether alternative optimization strategies can reach or outperform this benchmark.

Optimization Methods

We implement and compare the following optimization strategies:

- **Coordinate Descent:** We consider three variants:
  1. **Random Coordinate Selection:** The coordinate to be updated is chosen by sampling uniformly at random.
  2. **Steepest Direction Selection:** The coordinate with the highest gradient is chosen for update.
  3. **Shallowest Direction Selection:** The coordinate with the smallest gradient is chosen for update.
- **Mini-batch SGD:** We implement a mini-batch stochastic gradient descent variant, where updates are computed over small batches.

For all methods (except `sklearn`'s L-BFGS), we train for a fixed 300 epochs with a mini-batch size of 16 and a learning rate of 0.01 to ensure a fair comparison of convergence.

Mathematical Formulation

The logistic regression model predicts the probability of the positive class using the sigmoid function:

$$\hat{y} = \sigma(Xw + b) = \frac{1}{1 + e^{-(Xw+b)}}$$

where:

- $X \in \mathbb{R}^{n \times d}$ is the input matrix (batch of size n with d features),
- $w \in \mathbb{R}^d$ is the weight vector,
- $b \in \mathbb{R}$ is the bias term.

The loss function is the binary cross-entropy:

$$L(w, b) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

where y_i represents the ground-truth labels.
The gradients used for optimization are:

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) X_i$$

$$\frac{\partial L}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

where:

- $\frac{\partial L}{\partial w}$ is the gradient of the loss with respect to the weights,
- $\frac{\partial L}{\partial b}$ is the gradient of the loss with respect to the bias,
- The gradients are averaged over the mini-batch.

This setup allows for a direct and fair comparison of convergence behavior across different optimization strategies.

4 Experimental Results

Convergence Analysis

We compare the convergence behavior of different optimization methods based on training loss reduction over 300 epochs.

- **Coordinate Descent (Random Selection)**: The loss reduction is almost linear, showing steady but slow convergence. While it does improve over time, it lags behind other methods in terms of efficiency.

Table 1: Training Loss Comparison Across Different Optimization Methods

Epoch	CD (Random)	CD (Max Gradient)	CD (Min Gradient)	SGD (Mini-batch)
25	0.8090	0.7595	0.5623	0.1830
50	0.7115	0.5668	0.5220	0.1199
75	0.6387	0.4680	0.4992	0.0911
100	0.5674	0.3813	0.4961	0.0780
150	0.5020	0.2847	0.4719	0.0594
200	0.4635	0.2354	0.4814	0.0442
250	0.3863	0.2069	0.4640	0.0369
300	0.3445	0.1849	0.4644	0.0340

Table 2: Final Training Loss Comparison for Different Optimization Methods. *Random selection is the baseline. Arrows depict relative improvement or degradation w.r.t this baseline

Optimization Method	Final Loss
Sklearn (L-BFGS)	0.0570 ↑
CD (Random Selection)*	0.3445
CD (Max Gradient Selection)	**0.1849** ↑
CD (Min Gradient Selection)	0.4644 ↓
SGD (Mini-batch)	**0.0340** ↑

- **Coordinate Descent (Max Gradient Selection)**: This method initially reduces the loss faster than the random selection strategy, demonstrating more efficient updates. However, it eventually asymptotes to a training loss close to that of `sklearn`'s L-BFGS solver, though it never fully reaches the same level of convergence.
- **Coordinate Descent (Min Gradient Selection)**: This method fails to converge effectively, stagnating at a relatively high loss value. Since it updates the coordinate with the smallest gradient, it struggles to make meaningful progress and gets stuck early on.
- **Stochastic Gradient Descent (Mini-batch)**: Among all methods, SGD performs the best, consistently reducing the loss at a rapid rate. It ultimately outperforms `sklearn`'s L-BFGS solver by achieving a lower final training loss.

5 Critical Evaluation

While coordinate descent is a simple and effective optimization method, our results indicate that its performance varies significantly based on the coordinate selection strategy. Below, we discuss potential areas for improvement:

Adaptive Learning Rate: Instead of using a fixed learning rate, an adaptive step size (e.g., using AdaGrad or RMSProp-inspired updates) could help the algorithm make larger updates in the early stages while fine-tuning adjustments as convergence nears.

Hybrid Coordinate Selection: The performance of coordinate descent depends heavily on which coordinate is selected for update. A hybrid approach that dynamically switches between the steepest and random coordinate selection could balance fast initial convergence and overall stability.

Momentum-based Updates: Incorporating momentum into the coordinate updates may help accelerate convergence, especially in cases where coordinate descent stagnates due to small gradient updates.

Block Coordinate Descent: Instead of updating a single coordinate at a time, updating small groups (blocks) of coordinates could lead to more efficient learning while still maintaining computational efficiency.

Variance Reduction Techniques: In stochastic optimization methods like SGD, variance reduction techniques (e.g., SVRG, SAG) have shown significant improvements. Similar techniques could be explored to improve coordinate descent's stability and convergence speed.

Second-order Information: Incorporating second-order information (such as Hessian approximations or curvature-aware updates) may help coordinate descent methods escape local minima more effectively and improve their convergence properties.

Overall, while coordinate descent remains a simple and interpretable approach, integrating adaptive techniques from more advanced optimization methods could significantly improve its performance and bring it closer to the efficiency of methods like

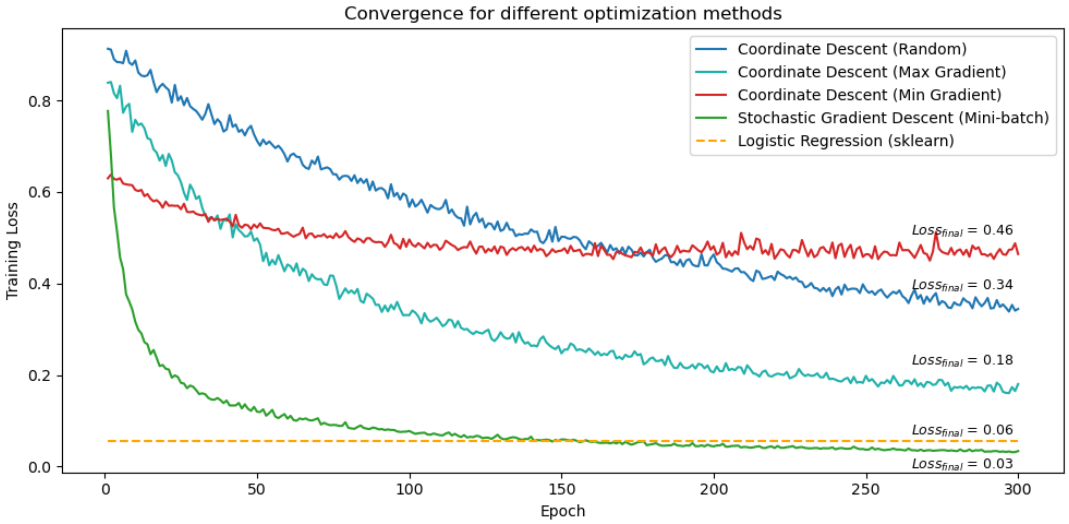

Figure 1: Convergence comparison of different optimization methods over 300 epochs. Coordinate Descent with maximum gradient selection performs better than random selection but does not reach the benchmark. Coordinate Descent with minimum gradient selection stagnates and fails to improve significantly. Stochastic Gradient Descent (SGD) achieves the lowest final training loss, outperforming `sklearn`'s L-BFGS solver.

L-BFGS and SGD.

6 Sparse Coordinate Descent

In this section, we modify our coordinate descent approach to enforce sparsity, meaning that the solution w contains at most k nonzero entries.

6.1 Modified Algorithm for k-Sparse Solutions

To ensure a k-sparse solution, we introduce a constraint where only the top k coordinates are allowed to be updated, while the remaining coordinates are decayed to zero. The modified algorithm proceeds as follows:

1. Initialize $w = 0$.
2. Repeat for a fixed number of iterations:
   - Compute the gradient $\nabla L(w)$.
   - Identify the top k coordinates with the largest gradient values.
   - Update only these k selected coordinates using a coordinate descent step:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i} \quad \text{for } i \in S_k$$

     where S_k is the set of indices corresponding to the top k coordinates.
   - Decay all other coordinates to zero by multiplying a discount factor (decay rate).

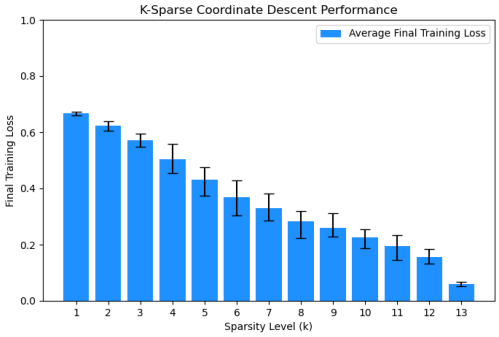

Figure 2: K-Sparse Coordinate Descent Performance: Final training loss across different sparsity levels (k). Error bars indicate the min-max range of loss values. All values are computed over 20 iterations at each sparsity level.

6.2 Optimality of k-Sparse Coordinate Descent

If $L(w)$ is convex, this method guarantees a sparse solution but does not necessarily find the globally optimal k-sparse solution. The restriction to only k coordinates at each step may prevent convergence to the same minimum as an unconstrained approach. However, if $L(w)$ has strong convexity and smoothness properties, the method can still achieve a near-optimal sparse approximation.

6.3 Empirical Evaluation on the Wine Dataset

We apply the sparse coordinate descent approach to the Wine dataset and record the final loss for

different values of k. The results are summarized in the table below.

Table 3: Final Training Loss for Different Sparsity Levels in Coordinate Descent

Sparsity Level (k)	**Final Training Loss**
1	0.6684
2	0.6237
3	0.5702
4	0.5037
5	0.4320
6	0.3693
7	0.3298
8	0.2843
9	0.2595
10	0.2270
11	0.1944
12	0.1563
Full-model	0.0588

As expected, increasing k results in a lower final training loss, approaching the full model's performance. However, small values of k significantly limit the model's expressiveness, leading to higher loss values.

Figure 2. illustrates how the final training loss varies across different levels of sparsity (k) in K-Sparse Coordinate Descent, with error bars representing the min-max range of loss values. As sparsity increases (i.e., as more coordinates are allowed to be nonzero), the mean final loss decreases, indicating improved model performance. At lower values of k (e.g., $k = 1, 2, 3$), the min-max range is relatively small, but as k increases to moderate levels (e.g., $k = 4, 5, 6$), the range expands, reflecting higher variability in performance. This suggests that when only a few coordinates are updated, the selection process introduces fluctuations in loss, leading to less stable convergence.

However, as k continues to increase beyond $k = 7$, the min-max range gradually shrinks, indicating more stable and consistent training outcomes. This trend suggests that a sufficient number of active coordinates allows the model to generalize better and mitigate randomness in updates. At low sparsity levels (e.g., $k \geq 10$), the min-max range becomes very small, and the training loss approaches its lowest value, implying diminishing variability and near-optimal convergence. This pattern confirms that while increasing k improves performance, there exists a threshold beyond which additional nonzero coordinates yield minimal further gains in stability or accuracy.

Acknowledgments

- Most of the LaTex formatting for this paper was prepared using ChatGPT with minor changes
- Content was proof-read and revised using Grammarly with minor changes