Examining Methods for Coordinate Descent

Ryosuke Oguchi
roguchi@ucsd.edu

Abstract

The purpose of this project is to examine a coordinate descent method that is developed via a selection rule in order to reduce the loss for an unconstrained optimization problem. More specifically, the project will examine a Logistic Regression problem on a Wine Dataset to see how alternative methods to coordinate descent will lead to different rates of weights convergence. We will see that a for this specific dataset, convergence of the weights will be better for a maximum gradient coordinate descent method rather than mere random selection. However, when considering high dimensionality and large number of points, we will see that the number of epochs will be taxing if the update step is arbitrarily chosen to be small. Overall, considering the works of Nesterov and Nutini et. al., we will see that considering the curvature of the loss function will lead to improved performance of convergence.

1 Coordinate Descent Method

The method chosen for this project was a maximum gradient coordinate descent where coordinate $i \in \{1, 2, \ldots, d\}$ is picked by $\max_i(|\nabla L(w)|)$. Then, a basic update rule is applied:

$$w_{d,t+1} = w_{d,t} - \eta \cdot \max_i(|\nabla L(w)|)$$

Here η is considered to be an arbitrarily small constant of 0.001 to ensure convergence. In general, we do not have a strict requirement that the loss function be differentiable at all points, but we do require that the function be continuous. This is because for our method to work, we need a gradient vector to return a value and the update step to occur. As for the second-order derivative, we do not require a continuous second-order derivative; however, it is crucial that the loss function is convex. Without a convex function, the update step will not converge to a local (or global) optimum.

1.1 Basic Pseudocode

1. **Input:**
   - Objective function $f(x)$ to minimize
   - Gradient $\nabla f(x)$
   - Initial guess $x_0 = (x_1, x_2, ..., x_n)$
   - Learning rate α
   - Convergence threshold ε
   - Maximum iterations max_iter (optional)
2. **Initialize:** $x = x_0$
3. **Repeat until convergence or max_iter:**
   (a) Compute gradient $g = \nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}\right)$
   (b) Identify coordinate $i_{\max}$ with largest absolute gradient:
   $$i_{\max} = \arg\max_i |g_i| \quad \text{for } i \in \{1, 2, ..., n\}$$
   (c) Update only the selected coordinate:
   $$x_{i_{\max}} = x_{i_{\max}} - \alpha g_{i_{\max}}$$
   (d) Check stopping condition:
   - If $\|\nabla f(x)\| < \varepsilon$, stop.
   - Otherwise, continue.
4. **Return:** Optimized solution x

2 Convergence Conditions

As mentioned in the previous section, we need a convex function, ensure a local optima exists. Due to the nature of the update step and coordinate selection system that is developed, it forcefully updates until a stop threshold is achieved. Hence, non-convex functions will return w^* values that are not optimal. When considering the "efficiency" of convergence, we have only set an arbitrarily small η, so the number of epochs will be dependent on how "strong" the convexity of the loss function is.

3 Experimental Results

Before experimental results can be specifically examined, there must be some general conditions that ought to be established. First, is what type of loss function that will be used for the data being used. Against UCI's wine dataset, we will use logistic regression; hence, we will be using logistic loss for this problem. The formulas will be the following:

$$L(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^{(i)} w^T x^{(i)})$$

$$\nabla f(w) = \sum_{i=1}^{n} \frac{-y^{(i)} x^{(i)}}{1 + \exp(y^{(i)} w^T x^{(i)})}$$

$$\nabla^2 f(w) = \sum_{i=1}^{n} \frac{y^{(i)2} x^{(i)} x^{(i)T} \exp(-y^{(i)} w^T x^{(i)})}{(1 + \exp(-y^{(i)} w^T x^{(i)}))^2}$$

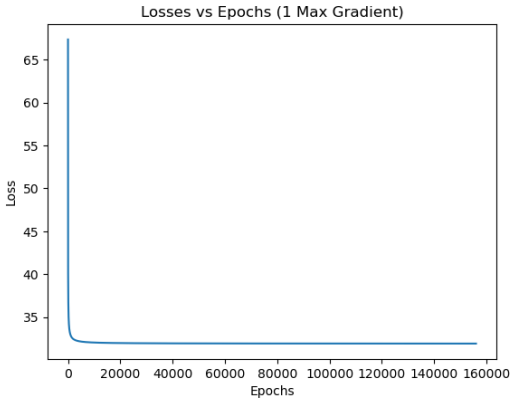
These formulas are taken directly from the Convergence Lecture.

3.1 Baseline

As a baseline, we will be running a regular logistic regression to note the minimum loss L^*. Using sklearn's LogisticRegression solver. This turns out to be $L^* = 31.89404$; for any future coordinate descent algorithm, we will use this as the stop threshold of convergence.

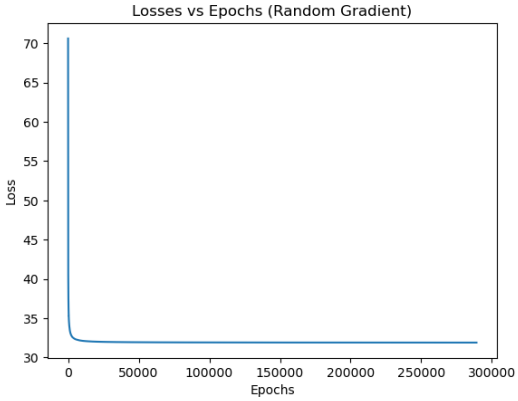
3.2 Max Gradient Coordinate Descent

As stated earlier, this method considers the direction to move as $\max_i(|\nabla L(w)|)$ where the coordinate is $i \in \{1, 2, \ldots, d\}$. Then, a standard weight updating scheme used to move in the direction of the greatest negative gradient direction. In order for the descent algorithm to converge to L^*, it required 156026 epochs on the training set. Although seen in the image below, the number of epochs it requires to be close to L^* appears relatively low, it requires many epochs to fully converge to L^*.

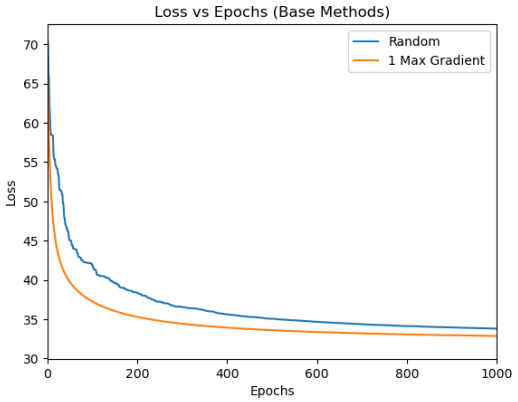


3.3 Random-Feature Coordinate Descent

In order to compare the effectiveness of the Max Gradient Coordinate Descent, we will implement a Random-Feature method that will randomly sample 1 gradient and implement the update algorithm on that randomly selected point. We see here that it required 288136 epochs to converge to L^*, while exhibiting similar behaviors to the Max Gradient method. We can attribute the extremely long asymptotic behavior to the fact that the gradient after numerous epochs will already be very small, so the update step provide little value to w_{t+1}.



We can also examine how the two models perform against each other when there is a low amount of epochs.



We see a rather smoother decrease in loss when

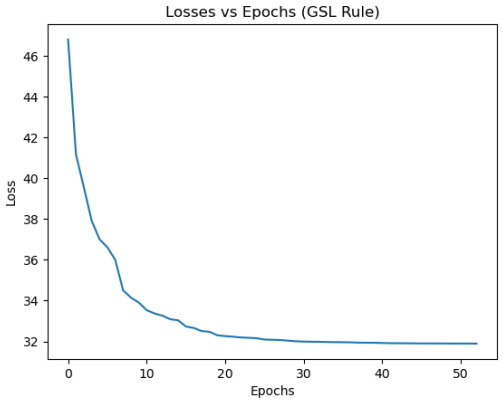
we have targeted feature descent, because we are still moving in a good direction. The random gradient is not smooth because the update step will only provide some sort of decrease, and that decrease is not said to be optimal. Nevertheless, as the loss approaches $L*$, we will recognize that the incremental change is so minimal that regardless of direction, it will only provide marginal benefit for convergence.

4 Critical Evaluation

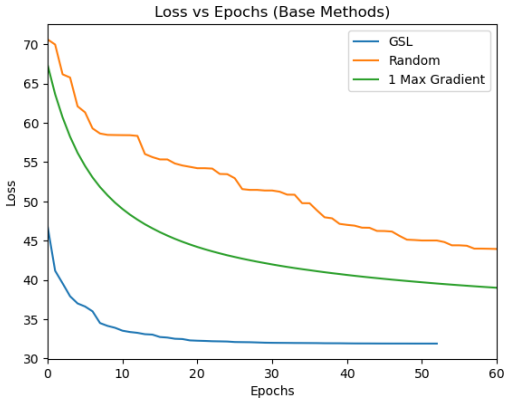
Although we see from our experiment that maximum gradient coordinate descent is an improvement to random-feature selection, we must recognize this may be a mere coincidence in how the data was split for training and testing purposes and the original data source. These beliefs have been validated in Nesterov's 2012 paper (Yu, 2012) that exhibited random-coordinate selection was no better than the Gauss-Southwell selection rule (Maximum Gradient Coordinate Descent). This is a valid when considering high-dimensional data as finding the "best" gradient is considered to be very expensive. Moreover, when viewing that it requires more than 100000 epochs to find optimal convergence for a mere 178×13 matrix will show that this cost may exponentially grow with more data points and features. Hence, to make reduce the number of epochs and achieve faster convergence, we will use a variant of the GS method by adapting the update constant to be dynamic. Proposed by Nutini et. al. (Nutini et al., 2015), they introduce the usage of the Lipschitz constant that is derived by the Hessian matrix of the Loss.

To compute the constant we are using to adaptively update our weights, we use the Hessian computation as mentioned in the conditions as the beginning of this section. Moreover, the specific coordinate that is chosen is the eigenvector relative to the dimension of the coordinate of the greatest magnitude of the gradient. The update formula is adapted to the following:

$$w_{t+1} = w_t - \frac{1}{H(w_t)_{i,i}} \cdot \max_i(|\nabla L(w_t)|)$$



Overall, we see an astronomical improvement to just 53 epochs to converge to reach L^*. This shows that there is a general benefit to using an adaptive learning rate rather than $\eta = 0.001$.



Moreover, we see that even in the early stages, considering the curvature of the loss function allows for faster convergence. Even at roughly 20 epochs, loss appears to be asymptotic to L^*.

Although we have concluded that an adaptive learning rate have reduced the time it takes to converge, we also can consider other methods such as L_2 regularization. Although it requires the data to be manipulated by $\lambda||w||_2^2$, there is implied stability with a m-strongly convexity. As per the theorem in the lecture slides,

$$w_1 = argmin_w f(w)$$

$$w_2 = argmin_w f(w) + g(w)$$

Then:

$$||w_1 - w_2|| \leq \frac{\max_w ||\nabla g(w)||}{m}$$

With this transformation and theorem, there is opportunity for faster convergence due to steeper gradients, and then less epochs without deviating the final weights.

5 Conclusions

As per the experiments we have conducted with various coordinate descent techniques, we see that without considering a fleshed out learning rate, we will see slow convergence and high number of epochs. Depending on the data and how the testing and training is built, we will see that convergence may wildly vary. As per Nesterov, we may see a case where a GS algorithm may not even perform better than random feature selection. Hence, introducing a Lipschitz constant in determining a learning rate will radically improve the efficiency of the learning algorithm. Moreover, we can consider the usage of regularization to create stronger convexity to allow for faster convergence as well. But naively believing that a greedy descent mechanism is the "best" will not serve people well when trying to find the weights of a learning algorithm.

References

Julie Nutini, Mark Schmidt, Issam Laradji, Michael Friedlander, and Hoyt Koepke. 2015. Coordinate decsent converges faster with the gauss-southwell rule than random selection. *JMLR: W&CP*, 37.

Nesterov Yu. 2012. Efficiency of coordinate descent methods on huge-scale optimization problems. *Society of Industrial and Applied Mathematics*, pages 341–362.