Stochastic Gradient Descents Variants Implementations

Dzodzoenyenye Adjowa Senanou, Nguinabe Josue, YEBADOKPO Charmaine Aurette O. A.
dsenanou@aimsammi.org, jnguinabe@aimsammi.org cyebadokpo@aimsammi.org

1 Introduction

In the realm of machine learning and optimization, algorithms designed to efficiently minimize functions play a pivotal role. One of the cornerstone techniques is Stochastic Gradient Descent (SGD) and its various sophisticated extensions. These algorithms are widely employed across different domains due to their effectiveness and scalability in handling large datasets and complex models.

The objective of this lab is to implement and test some of the variants of Stochastic Gradient Descent (SGD) and compare their performance with traditional gradient descent. The following SGD variants were implemented:SGD with constant step sizes, SGD with shrinking step sizes, SGD with sampling with/without replacement, SGD with averaging, and SGD with momentum

These methods were applied to minimize two types of loss functions: Least-squares regression and Logistic regression.

2 Methodology

2.1 Datasets

We generate datasets for the least-squares and the logistic cases. The feature matrix X is generated by drawing n samples from a multivariate normal distribution with a mean vector of zeros (of length d) and the defined covariance matrix. A noise vector is generated by drawing n samples from a normal distribution with mean 0 and standard deviation std. This noise simulates the randomness typically present in real-world data.

2.2 SGD

An iteration of SGD is writen:

for $t = 1, \dots, T$
Pick i uniformly at random in $\{1, \dots, n\}$
$w^{t+1} \leftarrow w^t - \gamma^t \nabla f_i(w^t)$
end for

2.3 Loss functions, gradients and step-sizes

We want to minimize

$$\frac{1}{n} \sum_{i=1}^{n} \ell(x_i^\top w, b_i) + \frac{\lambda}{2} \|w\|_2^2$$

where - $\ell(z, b) = \frac{1}{2}(b - z)^2$ (least-squares regression) - $\ell(z, b) = \log(1 + \exp(-bz))$ (logistic regression).

We write it as a a minimization problem of the form

$$\frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

where

$$f_i(w) = \ell(x_i^\top w, y_i) + \frac{\lambda}{2}\|w\|_2^2.$$

For both cases, the gradients are

$$\nabla f_i(w) = (x_i^\top w - y_i)x_i + \lambda w$$

and

$$\nabla f_i(w) = -\frac{y_i}{1 + \exp(y_i x_i^\top w)}x_i + \lambda w.$$

Denote by L the Lipschitz constant of f and $X = [x_1, \ldots, x_n]$. One can see easily that for linear regression

$$L = \frac{\|\mathbf{X}\mathbf{X}^\top\|_2}{n} + \lambda$$

while for logistic regression it is

$$L = \frac{\|\mathbf{X}\mathbf{X}^\top\|_2}{4n} + \lambda$$

For full-gradient methods, the theoretical step-size is $1/L$.

2.4 SGD with constant step sizes

The learning rate (or step size) remains fixed throughout the training process. It is simple in implementation and tuning. But, it may lead to suboptimal convergence if the step size is too large (causing oscillations) or too small (leading to slow convergence). It is effectively used for smaller datasets and problems where a good initial learning rate can be identified.

2.5 SGD with shrinking step sizes

For this the learning rate decreases over time, often following a predefined schedule. It helps in achieving finer convergence as the algorithm progresses by reducing the step size. However, it requires careful tuning of the decay schedule to balance exploration and convergence. Useful in scenarios where initial fast learning is beneficial but needs to slow down for convergence.

$$\gamma^t = \begin{cases} \frac{1}{2L_{\max}} & \text{for} \quad t \leq 4\lceil \mathcal{K} \rceil \\ \frac{2t+1}{(t+1)^2\mu} & \text{for} \quad t > 4\lceil \mathcal{K} \rceil. \end{cases}$$

2.6 SGD with sampling with/without replacement

- Sampling with Replacement
  Each data point can be chosen multiple times within a single epoch. It is simple and often used in online learning settings. But, may lead to redundant updates and potentially slower convergence. Suitable for streaming data and online learning.
- Sampling without Replacement
  Each data point is chosen exactly once per epoch. It ensures all data points contribute equally within an epoch, often leading to faster and more stable convergence. But, more complex to implement in certain online or streaming data scenarios. Preferred for batch training where full dataset is available.

2.7 SGD with averaging

This SGD maintains an average of the parameter values over iterations rather than using the final parameter values directly. It can lead to more stable and accurate solutions by smoothing out fluctuations. As challenges, additional computational and memory overhead to maintain and compute averages. Effectively employed in reducing variance and achieving more reliable convergence in noisy optimization landscapes.

2.8 SGD with momentum

Introduces a momentum term to the updates, which helps to accelerate convergence, especially in the presence of high curvature, small but consistent gradients, or noise. It reduces oscillations and speeds up convergence by accumulating a velocity vector in directions of persistent reduction. But, it requires tuning of the momentum parameter in addition to the learning rate. Highly effective in training deep neural networks and other complex models.

3 Results

In this section, we present the findings from our implementations. The results include the outcomes of applying different optimizations techniques.

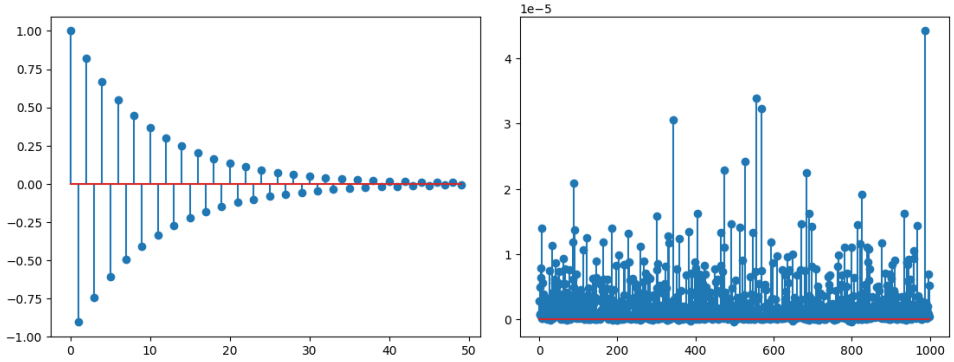

Figure 1: Linear Regression and Gradient Error

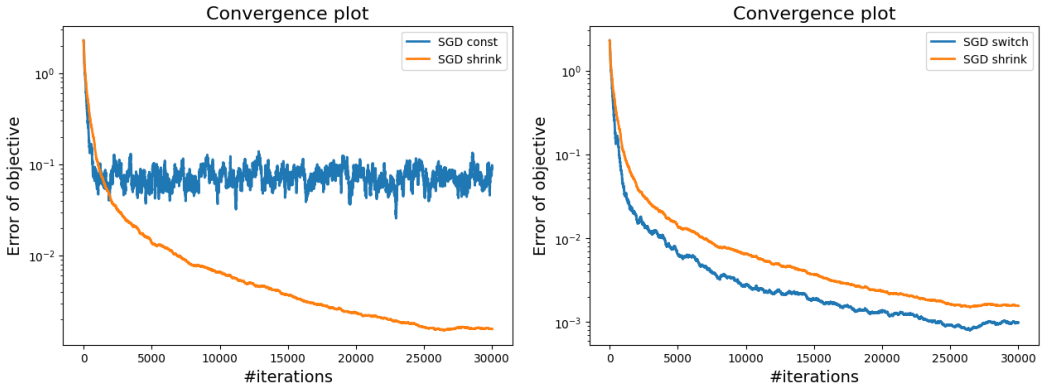

Figure 2: SGD with constant, Shrinking, Switching stepsizes

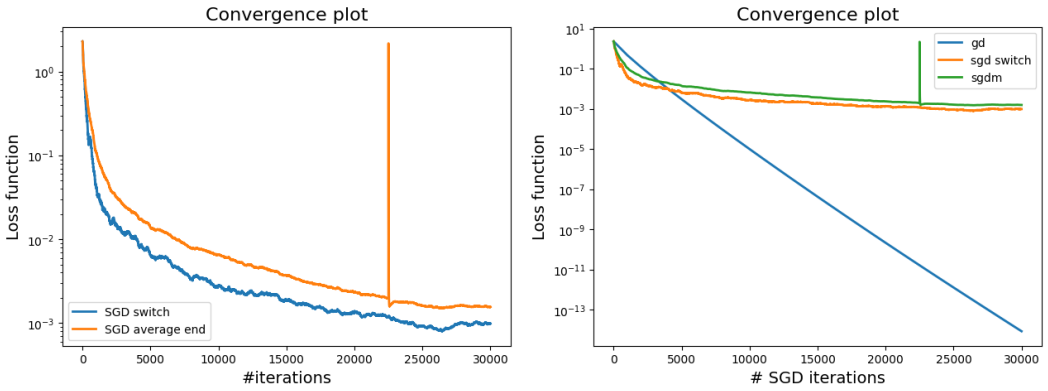

Figure 3: SGD with Shrinking + momentum, Switching stepsizes and GD

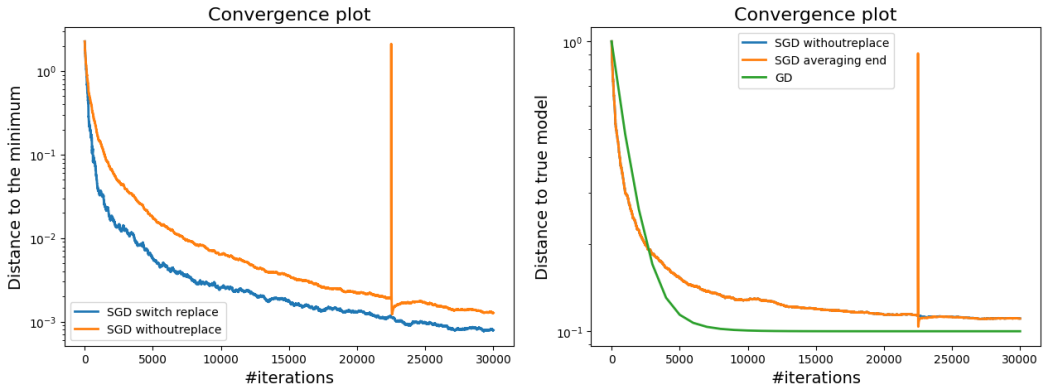

Figure 4: SGD with and Without Replacement; SGD Without replacement, Averaging end and GD

4 Discussions/Comparisons

4.1 SGD with constant stepsizes and SGD with shrinking stepsizes

Figure 2 SGD with constant stepsizes (blue) exhibits erratic behavior and fluctuates around a higher error value, not converging well. SGD with shrinking stepsizes (orange) steadily decreases the error and converges to a lower error value.

Both methods start reducing the error quickly, but SGD with constant stepsizes (blue) appears to reduce the error more rapidly initially.

SGD with shrinking stepsizes (orange) reaches the best solution as it converges to a lower error value compared to the constant stepsize version.

Sampling without replacement (one datapass) generally leads to faster convergence within a single pass as it avoids redundant updates to the same data points. However, the implementation is less straightforward for multiple passes.

4.2 SGD with shrinking stepsizes and SGD with averaging

Figure 2, SGD with shrinking stepsizes (orange in both plots) converges more smoothly but more slowly. SGD with averaging (orange in the first plot) converges initially similarly to the shrinking stepsizes method but exhibits a significant drop at around 25,000 iterations. This suggests an adjustment or switch in the method used, leading to a rapid decrease in error.

If averaging is applied only to the last n iterates (as seen in the sudden drop in the first plot at 25,000 iterations), it can significantly reduce the noise and variance, leading to a sharp decrease in the error. This method can provide a more stable and lower final error.

Averaging is particularly useful in the later stages of training when the algorithm starts to oscillate around the optimal solution. It helps to smooth out these oscillations and achieve a lower final error. It is also beneficial when the learning rate is small, and the updates are noisy, as averaging can reduce the variance of the updates and lead to more stable convergence.

4.3 SGD with shrinking stepsizes and SGD with averaging

Figure 3, The plot shows that GD (Gradient Descent) converges the fastest to a very low loss, but it might be computationally expensive for large datasets. SGD switch (orange) performs better than plain SGD with constant stepsizes and SGD with shrinking stepsizes but has a noticeable improvement at around 25,000 iterations, likely due to an averaging mechanism. SGDm (SGD with momentum) (green) converges slower initially compared to SGD switch but shows consistent improvement and stability over iterations.

SGDm works well with momentum parameters in the range of 0.9 to 0.99. These values help in accelerating the convergence and smoothing out the oscillations by incorporating a fraction of the previous updates into the current update.

Considering all the tricks and variants seen so far, the SGD switch with a late start averaging or SGDm with an appropriate momentum parameter seems to be the best variant for this problem. SGD switch benefits from the averaging mechanism in the later stages, which significantly reduces the error. SGDm offers stability and consistent improvement, making it a robust choice, especially if tuned well with the right momentum parameter.

4.4 SGD with and Without Replacement; SGD Without replacement, Averaging end and GD

In Figure 4

1. Computational cost of gradient descent (GD) vs. stochastic gradient descent (SGD):
   A step of gradient descent (GD) involves computing the gradient over the entire dataset, which has a computational cost of $O(n)$, where n is the number of data points. A step of SGD involves computing the gradient using a single data point, which has a computational cost of $O(1)$.
2. Equivalent steps of gradient descent to SGD: To make the computational complexity equivalent to $datapassesxn$ steps of SGD, you should take $datapasses$ steps of GD. Each pass over the data in SGD is equivalent to one full gradient computation in GD.
3. Comparison of GD and SGD:
   On the x-axis, if you plot the total computational effort, GD converges much faster as it considers the entire dataset for each update, leading to a more precise direction towards the minimum. This is evident from the steep slope of the GD curve compared to the flatter slopes of the SGD variants. SGD, although less precise per step, takes significantly more steps to reach a similar level of convergence due to its lower computational cost per step.
4. Effect of increasing the number of datapasses:
   Increasing the number of datapasses in SGD generally leads to better convergence as the model gets more opportunities to update based on different data points. However, the benefit of additional passes diminishes after a certain point, where the additional computational effort might not result in significant improvements in convergence.

5 Conclusion

1. GD (green): Shows the fastest initial convergence rate. The error drops rapidly within the first few iterations, indicating efficient progress toward the minimum. GD continues to show a consistent and smooth reduction in error. The convergence path is stable, reflecting the deterministic nature of using the entire dataset for gradient computation. GD achieves a very low error level quickly and stabilizes at a low error value. It consistently gets closer to the true model with minimal oscillations.
2. SGD without replacement (blue): Also exhibits a rapid initial convergence but not as steep as GD. This suggests that using each data point exactly once before reusing helps in making

significant progress early on. The convergence path is relatively noisy compared to GD but continues to make steady progress. The noise is due to the stochastic nature of SGD. SGD continues to decrease the error but at a slower rate compared to earlier stages. The noise reduces as more iterations pass, but it still doesn't reach as low an error as GD.

3. SGD with averaging end (orange): Initially converges at a rate similar to SGD without replacement, indicating that the early progress is comparable between the two SGD variants. Progress is steady but shows similar noise patterns as SGD without replacement. The noise is expected due to the stochastic updates in SGD. Around 25,000 iterations, there is a noticeable drop in error. This significant drop indicates the effect of averaging, which reduces the variance of the updates and helps in achieving a lower error. After the drop, the error stabilizes at a much lower level, similar to GD.
4. In practice, combining the benefits of GD and SGD variants, such as using SGD with averaging and careful control over data usage (without replacement), can yield robust and efficient optimization performance.

GitHub Repository:

`https://github.com/Loverighteous1/Optm_SGD_and_Variants.git`