Stochastic Gradient Descent Report

Atou koffi KOUGBANHOUN
AIMS AMMI
Mbour, Sénégal
akougbanhoun@aimsammi.org

Millicent Omondi
AIMS AMMI
Mbour, Sénégal
momondi@aimsammi.org

Aime Ouraga
AIMS AMMI
Mbour, Sénégal
oballou@aimsammi.org

1 Introduction

Stochastic Gradient Descent (SGD) is an algorithm that minimizes the loss function by computing its gradient after each training example as opposed to using the whole dataset as seen in gradient descent. This approach makes SGD computationally efficient, especially when using a large data set. Given that SGD chooses on one training example from the dataset randomly for each iteration, it suffers from noise. Because of this, there are various modifications to SGD like incorporating momentum, adjusting stepsizes, and using different sampling strategies. These help mitigate noisy updates and slow convergence thereby improving performance.

1.1 Problem Statement

The most important step in building machine learning models is to solve the optimization problem as defined by the loss function. While standard gradient descent offers reliable method for finding optimal parameters, it may be computationally expensive.

1.2 Objectives

There are different variants of SGD. The project's objective is look at the following:

- SGD with constant stepsizes.
- SGD with shrinking stepsizes
- SGD with sampling with/without replacement
- SGD with averaging
- SGD with momentum
- comparing the above with gradient descent.

2 Approach

In this part we will look at the loss functions, gradients, stepsizes and the different variants of SGD that we used. We will compare the performance of these SGD variants against each other and with the traditional gradient descent.

2.1 Loss functions, gradients and step-sizes

We implemented linear and logistic regression models. These models fall under supervised learning techniques. Linear regression is used for solving regression problems wheras logistic regression is used for classification problems. Our focus will be applying SGD on the linear regression model.
Loss Functions:
We want to minimize

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

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,

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

2.2 Comparison between SGD with constant stepsizes and SGD with shrinking stepsizes

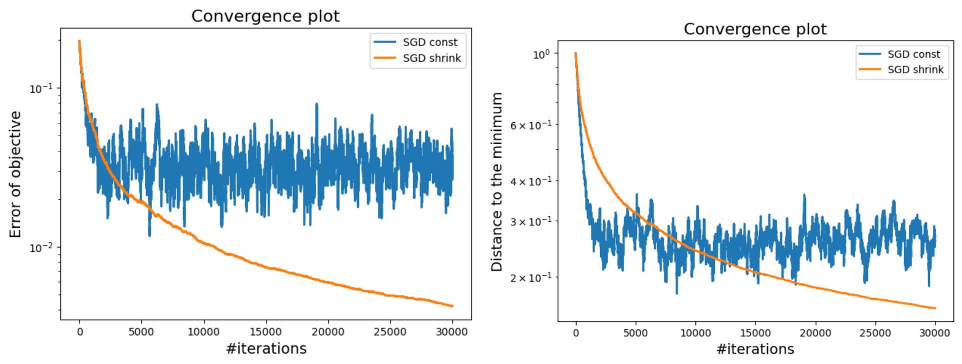

Figure 1: shrink vs constant

From 1, we observe that in the initial stages, SGD with constant step sizes shows faster convergence. This is seen by the rapid decrease in the error of the objective and distance to the minimum. However, it oscillates around the minimum and does not converge smoothly.
For the SGD with shrinking step sizes, we see that it starts slower and then gradually reduces the error steadily. We see that it converges toward the minimum steadily without much oscillation. This shows that it has a more stable and consistent reduction in both the error and the distance to the minimum.
From figure 1, we may conclude that SGD with shrinking stepsizes reaches the "best" solution as it has a lower error and closer distance to the minimum. On the other hand, SGD with constant stepsize finds it hard to reach the optimal solution due to the fixed stepsize thus having continuous oscillations.
Sampling without replacement: In this case, each data point is used only once per iteration. It leads to faster convergence as it reduces the variance of the gradient leading to stability and faster convergence within a single data pass.

2.3 Comparison between SGD with a switching stepsize and SGD with shrinking stepsizes

From figure 2, we observe that SGD with switch starts with steps and switches to smaller ones after some iterations.

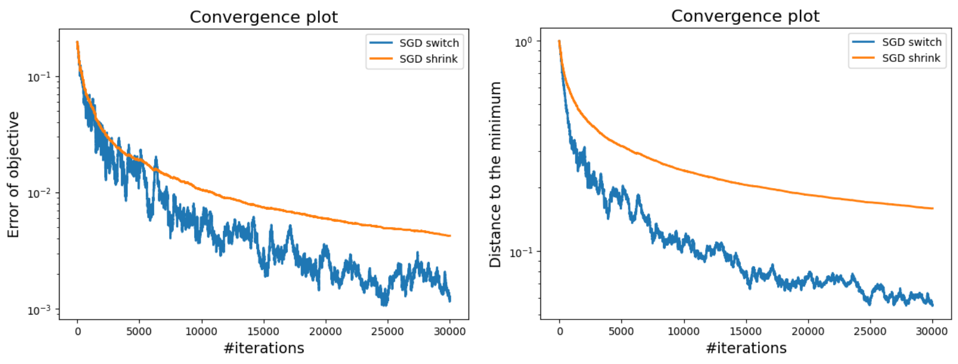

Figure 2: Switch vs shrinking

The SGD with shrinking steps starts with larger steps and then gradually reduces the step sizes. As compared to SGD with switch, we observe that the convergence is more gradual and the final error is lower.
Looking at the two methods, we can deduce that SGD with shrink converges faster at the beginning but reaches a higher final error as compared to SGD with shrink. SGD with shrink appears to be more stable as it has a smoother convergence curve with fewer fluctuations.

2.4 Comparison between SGD with switching stepsizes and SGD with averaging

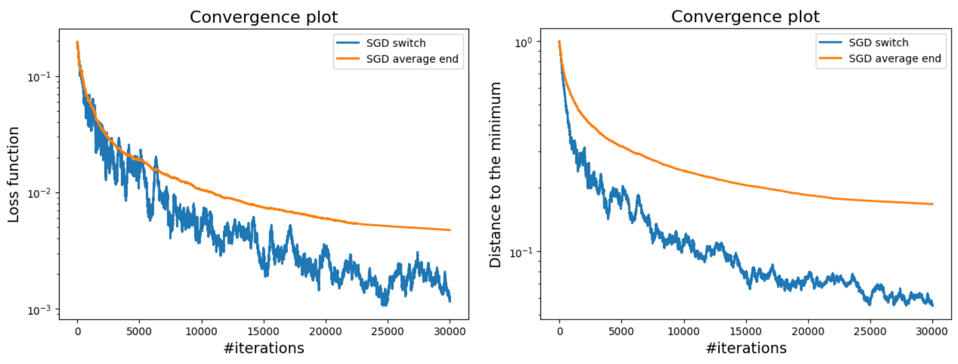

Figure 3: Average vs switch

From figure 3, we observe that SGD with switching stepsizes approach exhibits faster initial covergence at the biginning as large steps allow the algorithm to progress rapidly towards the optimum solution. After the steps switch to the convergence rate slows down therefore making the final solution to have a higer error as compared to SGD with averaged stepsize.
Looking at SGD with averaging, we observe it has a more gradual convergence and has a lower error compared to SGD with switching stepsizes.
Averaging only the last n iterates: Whe we average the last n iterates, we smooth out the noise and obtain a more stable and accurate final solution.
Averaging is useful when we want to improve the final solution incases where we are dealing with noisy gradients near the optimum.

2.5 Comparison between SGD with switching stepsizes , SGD with momentum and gradient descent.

In figure 4, the SGD with switch starts with faster convergence and then reduces. In the end, it’s final solution has a higher error as compared to the other two methods.
As for SGD with momentum, we see that it experiences a sudden drastic reduction in the loss function after some iterations. This is depicted by the vertical green line. This is associated with the momentum term in its update rule. The momentum term accumulates thereby amplifying the

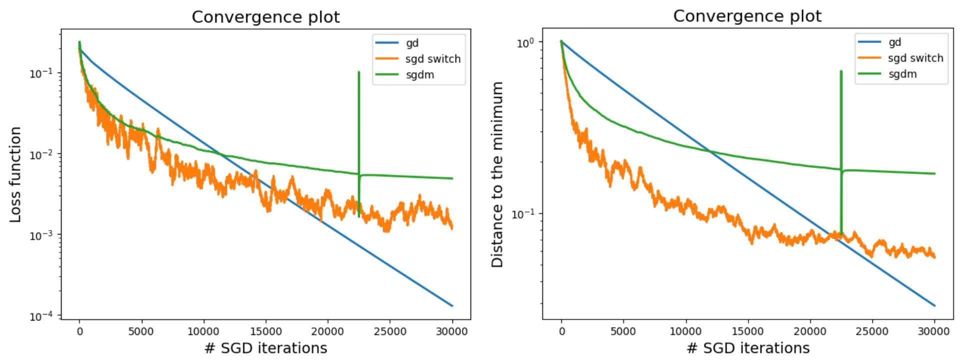

Figure 4: SGD with momentum, decreasing stepsizes and GD

gradients allowing the algorithm to escape shallow local minima or saddle points effectively.
We see that that the gradient descent converges to a lower loss function value as compared to the other two methods. This could be due to the fact that the full gradient was computed over the entire dataset at each iteration. This implies that the gradient information used in the update is more accurate and representative of the true gradient of the objective function. Therefore allowing GD to converge to a better optimum.

For what momentum parameters does SGDm work well?

Combining all the tricks/variants seen so far, what is the best variant of SGD for this problem?

2.6 SGD with replacement vs without replacement

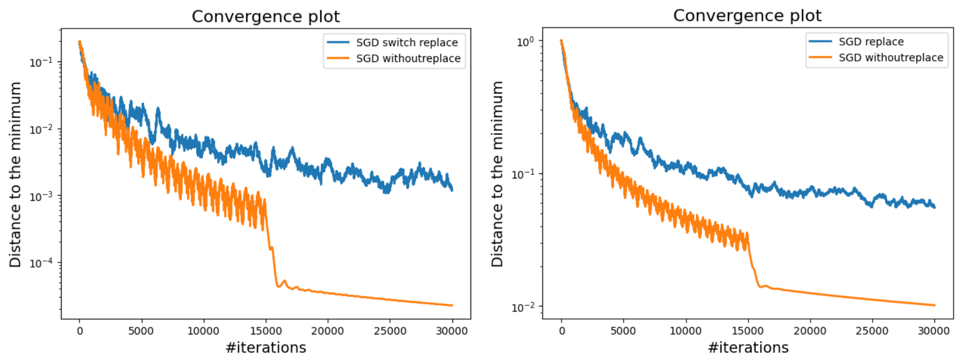

Figure 5: SGD with/without replacement

We can see from figure 5, that SGD with replacement converges faster initially, but in the end SGD without replacement achieves a better final solution. We can say that SGD without replacement shows a more stable and smooth convergence trajectory as compared to SGD with replacement, that is , it has fewer oscillations.

2.7 SGD with replacement vs averaging end

From figure 6, we observe that SGD without replacement exhibits the quickest initial convergence, yet it results in the highest final loss value. In contrast, SGD with averaging begins at a slower pace but ultimately outperforms SGD without replacement, leading to a lower final loss. Gradient Descent (GD) demonstrates the slowest initial convergence; however, it ultimately reaches the lowest final loss value when compared to the other two methods.

How much more is the computational cost of a step of gradient descent with respect to the computational cost of a SGD step?

A step of gradient descent (GD) has a computational cost of $O(n)$, while a step of stochastic gradient descent (SGD) with a single data point has a cost of $O(1)$. Thus, GD is n times more

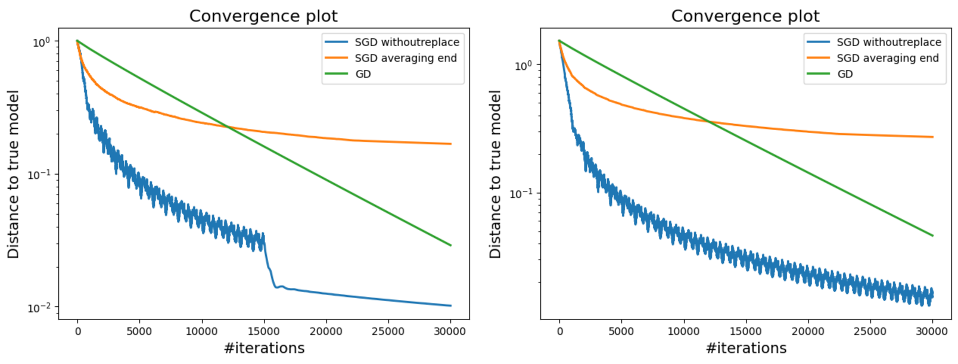

Figure 6: SGD without replacement,averaging end and GD

computationally expensive per step than SGD.

How many steps of gradient descent should you take so that the total computational complexity is equivalent to datapasses$\times N$ steps of SGD ?
Let k be the number steps of gradient descent
So

$$k \times O(n) = datapasses \times N \times O(1) \implies k = \frac{datapasses \times N}{n} \tag{3}$$

that's the number of steps GD should take in order to get the equivalence.

Increasing the number of datapasses helps SGD to converge to a more accurate solution. On the other hand, it leads to increased computational cost as we will need to go through the dataset multiple times.

3 Conclusion

In summary this implementation was amazing exploration of several techniques to improve the Stochastic Gradient Descent. Indeed those techniques really improve SGD. However , the convergence of ML algorithm does not depend only on the optimizer but also on the model architecture as well.