Final Project

1 Introduction

Stochastic Gradient Descent, or SGD, plays a large role in the field of optimization. However, it is not free of errors. In their paper, "Towards stability and optimality in stochastic gradient descent"[1], researchers from Harvard University highlighted two main problems with SGD. First, SGD is numerically instable because the mean-squared error of the parameters $\|\theta_n - \theta_*\|^2$ can diverge arbitrarily when γ_n is misspecified with respect to the problem parameters. Second, SGD lacks statistical efficiency due to its loss of statistical information. This information loss is tied to the misspecification of γ_n as well as the spectral gap of the Fisher information matrix. To solve this, the authors introduce a new SGD algorithm, called Averaged Implicit SGD (AI-SGD) that solves both numerical instability and statistical inefficiency while being similar to classic SGD in its ease of implementation and computational cost.

The authors present multiple experiments comparing AI-SGD to other contemporary algorithms—A-SGD, I-SGD, SGD—in stability and statistical efficiency. However, one set of experiments that was missing was one comparing the computational costs of the algorithms. Thus, in this paper, our first aim is to test AI-SGD against the same set of contemporary algorithms to measure the added computational costs from running AI-SGD as well as to assess if this added computational cost can be justified by the improvements in accuracy. The second point of consideration will be AI-SGD's comparison with ADAM. As ADAM stands out in its remarkable results at a low computational cost, we also aim to conduct empirical tests comparing ADAM and AI-SGD. We will run our experiments on two of the datasets found in the original paper: MNIST and Covtype.

Throughout this paper, we show that AI-SGD takes a significantly higher time to run each epoch when compared with all other algorithms tested. Furthermore, we show that AI-SGD is inconsistent in its loss minimization between the two datasets. Due to this, we will find that ADAM is holistically more dependable as an algorithm, as it provides low losses with efficient runtimes.

2 Background

2.1 Paper Findings

For many optimization problems, the task can be phrased as finding the optimal parameter θ_* such that for a data input ξ and loss function L

$$\theta_* = \arg\min \mathbb{E}(L(\theta, \xi)).$$

Stochastic gradient descent (SGD) is a common way of finding this optimal parameter, using the update step

$$\theta_n = \theta_{n-1} - \gamma_n \nabla L(\theta_{n-1}, \xi_n),$$

for some sequence of non-increasing step-sizes $\{\gamma_n\}$ to generate parameter estimators $\{\theta_1, \theta_2, \ldots, \theta_n\}$ that minimize the error $\|\theta_n - \theta_*\|^2$.

However, traditional SGD has some flaws which can be improved upon. In the paper "Towards stability and optimality in stochastic gradient descent" [1], the authors claim that SGD can suffer from numerical instability and statistical inefficiency due to misspecification of γ with respect to the convexity of the loss function and the Fisher information matrix respectively.

As a result, the paper proposes a new algorithm which addresses both the instability and inefficiency of traditional SGD. It introduces *Averaged Implicit SGD* (AI-SGD) which re-defines the parameter updates to be

$$\theta_n = \theta_{n-1} - \gamma_n L(\theta_n, \xi),$$
$$\bar{\theta}_n = \frac{1}{n} \sum_{i=0}^{n} \theta_i,$$

taking $\bar{\theta}_n$ as its new parameter at each iteration. The first equation represents the implicit step, as the parameter θ_n is defined in terms of itself. The paper shows this step leads to improved numerical stability because under strong convexity assumptions the step can be proven to contract the square error $\|\theta_n - \theta_*\|^2$ almost surely. The second equation represents the averaging of past parameters, which increases statistical efficiency because when $\gamma \propto n^{-1}$, then $\bar{\theta}_n$ can be seen as a weighted average of the previous error terms $\nabla L(\theta_i, \xi)$ and thus holds prior information about the gradient.

The paper then continues by proving theoretically and experimentally the benefits of the proposed algorithm. Theoretically they prove that AI-SGD is

stable for any choice of γ_n, reducing instability and hyper-parameter tuning requirements relative to other algorithms. In their experiments, they test AI-SGD against a number of algorithms, including averaging SGD (A-SGD), implicit SGD (I-SGD), SGD, AdaGrad, Prox-SAG, and Prox-SVRG. Their tests span a number of datasets including MNIST, CovType, RCV1 and the PASCAL DELTA synthetic set. In all of these instances, they show that AI-SGD performs similarly or better to the above algorithms, and also shows that AI-SGD is less sensitive to choice of parameter γ than the alternatives. The results convincingly show that AI-SGD meets the authors' claims. It is less sensitive to choice of hyper-parameters than alternatives, and converges in as many epochs as alternatives or fewer.

Another topic addressed in the theory section was that of computational efficiency. As presented, the implicit step involved recursively recalculating the gradient and parameter at each iteration which would represent a large computational cost that would significantly impact algorithm performance. However in section (3.1) of the paper, they address this issue by presenting an alternative process for the calculation. They show that almost surely the gradient at θ_n can be calculated as

$$\nabla L(\theta_n, \xi) = s_n \nabla L(\theta_{n-1}, \xi),$$

where s_n is a scalar which can be found using a numerical root-finding procedure. This finding drastically reduces the computational cost by reducing the implicit step from a recursive multi-dimensional equation to a non-recursive single dimension one.

2.2 Weaknesses in paper's overall argument

While the paper's finding reduces the computational costs when compared to a brute force AI-SGD, it fails to address the added cost of solving for s_n at each iteration. This weakens the papers overall argument, as AI-SGD might do as well as or better than alternatives relative to the number of epochs, but could do so at a substantial computational cost. In such a case, other alternatives might converge to optimal solutions more quickly in terms of computational time, and thus be more effective in real-world applications.

Moreover, while the paper shows its optimality in comparison to a number of common SGD optimizations, one algorithm which it is not compared to is ADAM. ADAM is very commonly chosen as an optimization of SGD for its use

of moments and adaptive step-sizes which—like AI-SGD—can help address the statistical inefficiency of traditional SGD. However, unlike AI-SGD, the additional costs of ADAM are a small set of arithmetic operations which may not be as computationally taxing. If AI-SGD can be shown to perform as well or better than ADAM, and have comparable computational costs, it would do even more to strengthen the paper's argument in favor of AI-SGD.

3 Methods

The weaknesses of the paper open its argument to two questions: how great is the difference in performance between AI-SGD and its alternatives, and how does AI-SGD's performance compare to the strong alternative algorithm ADAM? To answer these questions, we implement two experiments.

The first experiment measures the time-elapsed and loss over a range of epochs for the four algorithms compared in the article whose original implementations were publicly available (AI-SGD, A-SGD, Implicit-SGD, SGD). It does so using a largely unaltered implementation of the algorithms in the language R, acquired from a repository by the authors of the paper.

The second experiment implements SGD and ADAM in Python, and compares it against AI-SGD's implementation in R. For this experiment, we use the implementation of SGD in both R and Python as a baseline for comparison. We then measure time elapsed as well as the loss for the algorithms for a range of epochs. In these cases, the analysis is done on the time and loss efficiency relative to the respective SGD implementation.

Each experiment was repeated on two binary classification tasks that were used in the paper. The first was classifying the 9 digit in the MNIST dataset which contains 70,000 28x28 hand-drawn images of the digits 0-9. The second was classifying cover type 2 in the CoverType dataset which consists of 52 features that classified different types of forest coverage for a much larger set of 460,000 samples. For the first, experiments were run for 20 epochs for loss measurements and 60 epochs for time efficiency measurements, and for the second all experiments were run for 60 epochs.

3.1 Limitations

Though the experiments laid out here provide meaningful insight into the questions posed above, there are a number of factors—stemming both from our

work as well as the original AI-SGD implementation—that lead to important limitations to our results.

First, due to a lack of background knowledge of the underlying procedures and implementation, it was neither possible to implement ADAM in R, nor to implement AI-SGD in Python. The use of SGD as a baseline comparison will in part account for differences in implementation type. Regardless, this difference represents a meaningful limitation to this paper's argument that could be addressed in future works.

Second, some aspects of the implementation of the four SGD algorithms have impacted the effectiveness of our experiments. For instance, the paper uses two error values in its analyses: mean-square error and inverse accuracy (i.e. 1− prediction accuracy). The error values returned from the repository's functions are not clearly labeled, making it difficult to discern which of the two is being returned. As a result, graphs in this paper showing the error of these methods have not been appropriately labeled.

Lastly, neither the documentation nor a thorough investigation of the code clarified the choice of learning rate and batch size that were used in the repository's implementation of SGD. As a result, we chose parameters that performed best on a small set of experiments.

4 Experiments

4.1 MNIST Dataset

Our first result compares the computational cost across the different algorithms. As explained in the methods section, computational cost was measured as the time taken per epoch by each algorithm. In Figure 1 (on next page), we plot the ratio of the time taken by each algorithm over the time taken by SGD. It is clear that after 10 epochs, AI-SGD has the highest computational time out of the three algorithms—30% higher than SGD. In the conclusion of their paper [1], the authors mention that AI-SGD comes at "virtually no computationally cost" compared to SGD; however, from our experiments, the magnitude of the difference appears more substantial. In large scale applications, a 30% increase can add a significant amount of computational time. AI-SGD was followed by I-SGD and then A-SGD. In contrast, ADAM is only 10% more computationally expensive than SGD in the later epochs, relatively similar to A-SGD and Implicit-SGD.

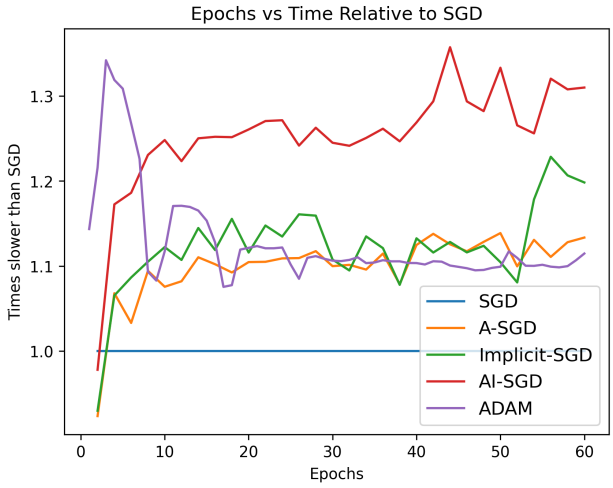

Figure 1

Next we compare the loss of each of those algorithms against the time taken by them. As mentioned in the methods section, since ADAM has been implemented in Python and the other algorithms were implemented in R, it would not be accurate to graph their loss by time directly since this does not take into account language differences. In Figure 2, we compare losses for the different algorithms—except for ADAM—relative to time taken to run an identical number of epochs.

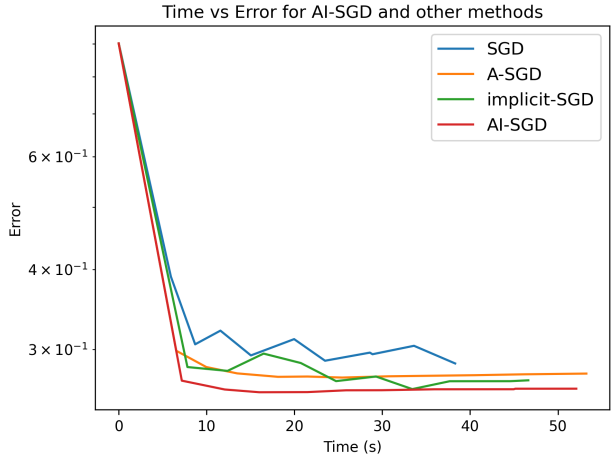

Figure 2

Looking at Figure 2, we can see that AI-SGD achieves the lowest error compared to the other algorithms. So, from a measure of accuracy, AI-SGD performs the best, followed by I-SGD, A-SGD and SGD. We can also compare the algorithms by the rate at which they converge below a certain loss threshold. From the graph, a suitable value for such an analysis is 0.3. Here again, we see that AI-SGD leads with the fastest time to convergence, followed by A-SGD and then by I-SGD.

In an earlier experiment, it was seen that AI-SGD took longer to run each epoch compared to other algorithms. However, from this experiment, we can see that it is the fastest to reach a threshold for convergence. Using the two findings, we reach the conclusion that though AI-SGD takes more time per epoch, it is taking far fewer epochs, and less time, to reach a loss threshold than the alternatives compared. This is also confirmed in Figure 2. Here we compare the loss of each algorithm as the ratio of their loss to the loss of SGD given the same number of epochs:

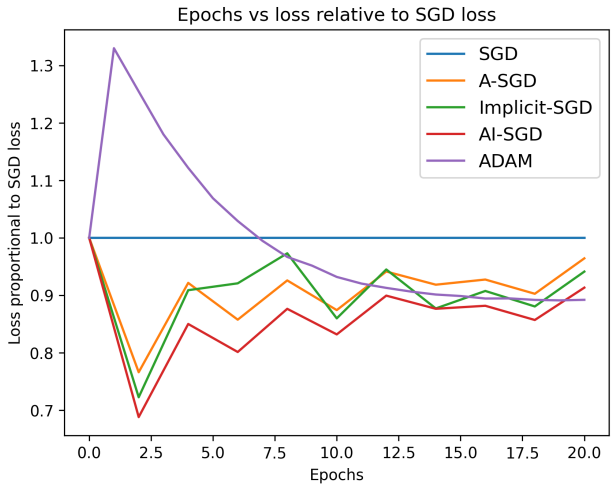

Figure 3

We can see that AI-SGD begins significantly more accurate than SGD, while ADAM lags behind, but by the end of the test, the two algorithms showed similar improvements over their respective SGD implementations. From Figure 2, we can see that at around 10 seconds, both SGD and AI-SGD had begun to plateau to their converging values. This translates to the 7.5 epoch mark in Figure 3. This means that between epochs 7.5 and 20, AI-SGD

makes little progress in relation as seen in the graph. However, since SGD and ADAM were compared in python, it is necessary to check if SGD showed similar behavior in that implementation. In the following graph, we show the loss vs time graph for ADAM and SGD:

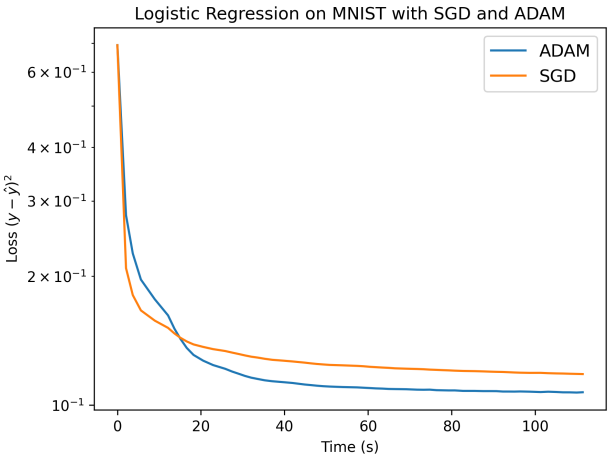

Figure 4

From Figure 4, we find that while SGD doesn't oscillate as much as it does in the R implementation, the rate of loss decline is very slow after 20 second. In contrast, ADAM only begins to plateau at around 40 seconds. Thus, we find that SGD's behavior is close enough for us to compare AI-SGD and ADAM in Figure 3. Looking at Figure 3, we see that the loss reached by ADAM at 20 epochs had been reached by AI-SGD in less than 12 epochs. This confirms our previous analysis that AI-SGD converges to the same loss threshold in fewer epochs. Overall, we see that while AI-SGD takes more time per epoch, it is able to reach the same loss as ADAM in fewer epochs and in fewer seconds.

4.2 CoverType Dataset

Next, to verify our findings, the experiments were repeated on the much larger CoverType dataset. Like before, we begin our analysis by first considering the time taken by each algorithm to run the same number of epochs. Once again, we use SGD as a baseline and compare the other algorithms to it:

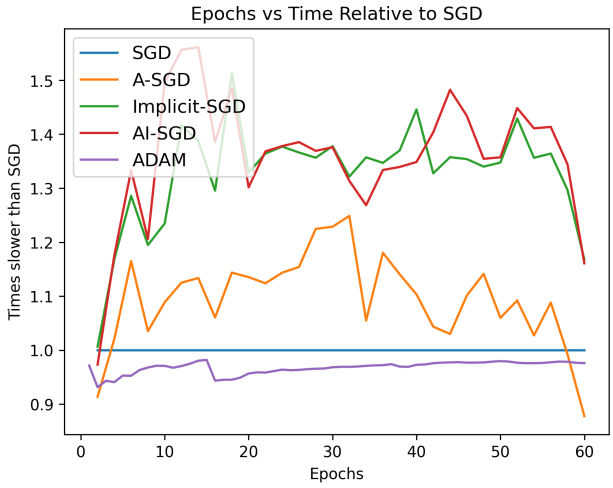

Figure 5

Much like before, we see that AI-SGD, I-SGD and A-SGD take longer than SGD per epoch. Further, AI-SGD took around 40% more time per epoch on this dataset. This once again affirms our hypothesis that AI-SGD's computational cost is quite significant compared to SGD. Another similarity we see is that I-SGD took significantly more than A-SGD per epoch. Since AI-SGD is a composition of I-SGD and A-SGD, we see that AI-SGD's computational costs are more attributed to the implicit step calculation common between it and I-SGD, rather than the averaging.

A surprising observation is the time taken by ADAM. While ADAM ran 10% slower than SGD on the MNIST dataset, it was slightly faster than SGD on the CoverType dataset. Unlike last time, here ADAM offers a very competitive advantage compared to the other algorithms. This result also highlights the importance of testing optimizers on different datasets of different complexities and sizes as results can vary depending on the problem specification.

In addition to measuring time per epoch, it is also important to measure the epochs taken to converge. In the following graph, we compare the number of epochs to the loss of the algorithms:

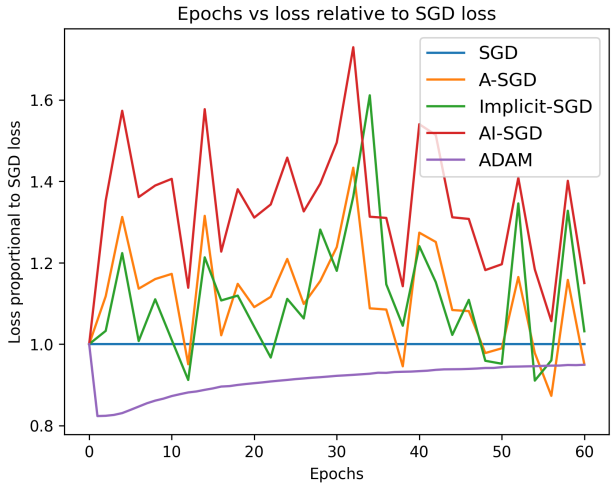

Figure 6

In figure 6, we see a trend opposite to that seen in the MNIST dataset. On almost every epoch, AI-SGD had the highest loss of the five algorithms. This is in stark contrast to the previous experiments wherein AI-SGD had performed the best in both the absolute loss as well as the iterations required to reach a loss threshold. The best performing algorithm was ADAM, followed by SGD, A-SGD, I-SGD and then finally by AI-SGD. Comparing with the MNIST dataset, ADAM appears far more dependable as it produces low results more uniformly, unlike AI-SGD which has shown large variation. Thus, in this example, it is clear that AI-SGD is least suited to the task as it had both the highest loss and the highest computational cost.

Looking at both datasets as a whole, while AI-SGD had appeared much better on the first dataset, its improvement is not guaranteed. In real life applications, an optimizer with more uniform results is ideal. Since ADAM was consistent and had time complexities similar to SGD, it stands as the better choice.

5 Conclusion

AI-SGD is a powerful algorithm that has remedied the issues of numerical instability and statistical efficiency that affect classic SGD. In this paper, our goal was to experimentally estimate the computational cost of AI-SGD

against other contemporary algorithms such as A-SGD, I-SGD and ADAM. Through a series of experiments, we found that AI-SGD takes around 30–40% longer than SGD to run each epoch on the MNIST and CoverType datasets. Nonetheless on the MNIST dataset, we found that AI-SGD converged far more precisely in each epoch which allowed it to run for fewer epochs and fewer seconds to reach the same loss as alternatives—making it the best candidate. Contrarily, it was the least convergent on the CoverType dataset—making it the worst candidate. In both cases, we found ADAM to be well performing and dependable. Thus, given only the results from our experiments, we conclude that AI-SGD has a significant computational cost over SGD and that its loss minimization properties are far more variant than that of ADAM.

In this light, we suggest future experimentalists to conduct experiments on a range of datasets with more complex problems to see if AI-SGD continues to show inconsistency in its loss properties as well as in its computational cost compared to ADAM. In addition, we also encourage the implementation of ADAM and AI-SGD in the same computing environment to allow for more accurate comparisons of absolute computational run-times between the two.

References

[1] Panos Toulis, Dustin Tran, and Edoardo M. Airoldi. *Towards stability and optimality in stochastic gradient descent*. 2016. arXiv: 1505.02417 [stat.ME].