Submission: Mini Project

Collaborators: None

1 Introduction

In theory, full gradient descent will always find the global minimum of a convex loss function. The scale of real-world sets of observations and feature spaces, however, often make full gradient descent (FG) infeasible to compute with reasonable efficiency. Instead, we commonly use stochastic gradient descent (SGD), an optimization that approximates the gradient using a batch of the observations in each iteration as opposed to the entire dataset. In this paper, we show how the behavior of SGD with last-half iterate averaging (SGD-IA) and SGD with polynomial decay (SGD-PD) depends on the initial state by fitting a regression model on two different datasets and varying the initial state. To facilitate the exploration, SGD-IA and SGD-PD were implemented in python, and the graphs of the squared norm of the error over time and the error convergence rate for each configuration of hyperparameters were compared. As a baseline, regular SGD was found to have a fast convergence, but a large amount of variance for all initial states. The results of our experiments showed that the SGD-IA algorithm had a slower convergence rate than regular SGD for all initial states, but almost negligible noise once it converged. The SGD-PD algorithm had a slower convergence rate than the other two algorithms for all initial states, and a variance that was greater than SGD-IA but still an improvement over regular SGD. Furthermore, it was found that increasing the rate of polynomial decay did not allow SGD-IA to properly converge for worse initial states.

2 Background

We can generically define the total loss function of the parameter vector x as

$$L(x) := \sum_{n=1}^{N} \ell_n(x) + r(x)$$

where $\ell_n(x)$ is the loss of the nth observation and $r(x)$ is the regularization term. We can define the FG update for positive learning rate η_k as

$$x_k := x_{k-1} - \eta_k \nabla L(x_{k-1}).$$

Computing the total loss is computationally expensive because it requires processing all N observations, so we can instead uniformly and randomly sample a batch of B observations and calculate the batch loss,

$$L_k(x) := \frac{1}{B} \sum_{b=1}^{B} \ell_{(n,(k,b))}(x) + r(x).$$

On average, the batch loss will approximately equal the total loss because $\mathbb{E}[L_k(x)] = L(x)$, so we can use the SGD update,

$$x_k := x_{k-1} - \eta_k \nabla L_k(x_{k-1}).$$

Since SGD requires fewer observations per update, we can increase the number of updates per N observations. By allowing for more updates, SGD lets us find an approximation of x_*, the parameter vector that minimizes the loss function, faster than FG. Recall, however, that SGD uniformly samples the batch of observations at each iteration to approximate the true gradient. The randomness of the sampling process introduces an added variance to the algorithm, causing a noisy and unstable convergence to the minimum.

One way to reduce the variance is to consider an average of recent parameter values instead of the latest update. This naturally leads to the notion of the iterate average, which is defined over the most recent W iterations as

$$\bar{x}_k := \frac{1}{W} \sum_{\ell=k-W+1}^{k} x_\ell.$$

By averaging the iterates, we can smooth out the noise and come to a relatively stable convergence because each individual parameter update has less influence. By further restricting the window to the most recent $\frac{k}{2}$ iterates, we can take advantage of the variance reduction while maintaining a relatively fast convergence by discarding the earlier iterates that are far from x_*.

Another approach to variance reduction can be observed by noticing that during regular SGD, the error continues to oscillate rather than converge tightly. Ideally, we want x_k to be updated by a smaller magnitude as k grows larger and x_k approaches x_*. A simple way to reduce the size of the update is to decrease the learning rate over time. This leads to the idea of polynomial decay, where we define the learning rate η at iteration k as $\eta_k := \eta k^{-\alpha}$, where α is the power. Under this formulation, if $\alpha = 0$, we will simply have regular SGD with constant learning rate η. Since k ranges from 1 to K, it will follow that as $\alpha \to \infty$, $k^{-\alpha}$ will continue to grow smaller and the rate of decay will increase. This leads to a quicker reduction in variance, but if the decay rate is too large, the learning rate will decay too soon and the algorithm will fail to properly converge.

3 Methods

In our experiments, we considered the data $\{(y_n, z_n)\}_{n=1}^N$, where $y_n \in \mathbb{R}$ is the response and $z_n \in \mathbb{R}^D$ are features. For $\beta \in \mathbb{R}^D$, we have the linear regression model

$$y_n = z_n^\top \beta + \varepsilon_n$$

such that ε is t-distributed with scale e^ψ and $\nu > 0$ degrees of freedom. Treating ν as fixed, we have have the model parameter $x = (\psi, \beta_1, \ldots, \beta_D) \in \mathbb{R}^{D+1}$. The conditional distribution of y_n is

$$y_n | z_n, x \sim T(z_n^\top \beta, e^\psi, \nu)$$

where $T(\hat{y}, s, \nu)$ is a t-distribution with location $\hat{y}$, scale s, and ν degrees of freedom. The density of T is

$$p(y \mid \hat{y}, s, \nu) = \frac{c(\nu)}{s}\left\{1 + \frac{1}{\nu}\left(\frac{y - \hat{y}}{s}\right)^2\right\}^{-\frac{\nu+1}{2}}.$$

Assuming that $\nu < \infty$ and letting $\hat{y} := z_n^\top \beta$, it follows that the log loss of the nth observation is

$$\begin{aligned} \ell_n(x) &= -\log p(y_n \mid \hat{y}, \psi, \nu) \\ &= \frac{\nu+1}{2} \log\left\{1 + \frac{e^{-2\psi}}{\nu}(y - \hat{y})\right\} + \psi \qquad (3.1). \end{aligned}$$

With a regularization term of $r(x) = \frac{1}{2}\|x\|^2$, the normalized overall loss function is

$$L(x) = \frac{1}{N}\sum_{n=1}^N \ell_n(x) + \frac{1}{2N}\|x\|^2 \qquad (3.2).$$

Next, by treating ψ and ν as constants, we can determine the range of $\hat{y}$ for which $\ell(x)$ is convex by looking at when $\ell''(x) \geq 0$. From derivation A.3.1, we have that

$$\ell''(x) = \frac{(\nu+1)(e^{-2\psi})(\nu - e^{-2\psi}(y-\hat{y})^2)}{(\nu + e^{-2\psi}(y-\hat{y})^2)^2}.$$

It follows from A.3.2 that $\ell''(x) \geq 0$ for $\hat{y} \in [y - \sqrt{\nu e^{2\psi}}, y + \sqrt{\nu e^{2\psi}}]$. Since $\hat{y}_n = z_n^\top \beta$, we have that $\ell(x)$ is convex for $\beta \in \{\beta : |y_n - z_n^T \beta| < \sqrt{\nu e^{2\psi}}\}$. Within the convex region, the loss function will behave like squared error and penalize large residuals. For outliers in the non-convex region, the gradient will shrink with larger residuals, naturally reducing their influence on parameter estimation. This makes the loss function suitable for robust regression.

4 Experiments

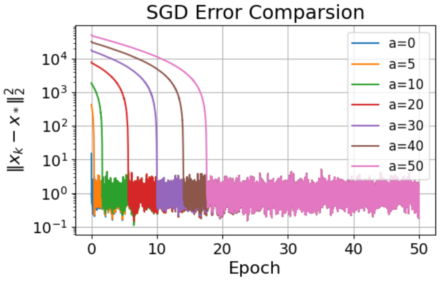
First, we tested the algorithms on a simulated dataset consisting of 10,000 observations and 20 features, sampled from a multivariate t distribution with 10 degrees of freedom and covariance matrix

$$\text{Cov}(i,j) = \frac{\nu-2}{\nu} exp\left(- \frac{(i-j)^2}{4} \right).$$

Each parameter value β_i is calculated as $\frac{\log(1+D)}{1+i}$, and each response y_n is equal to $z_n^\top \beta$ plus an additional t-distributed noise.

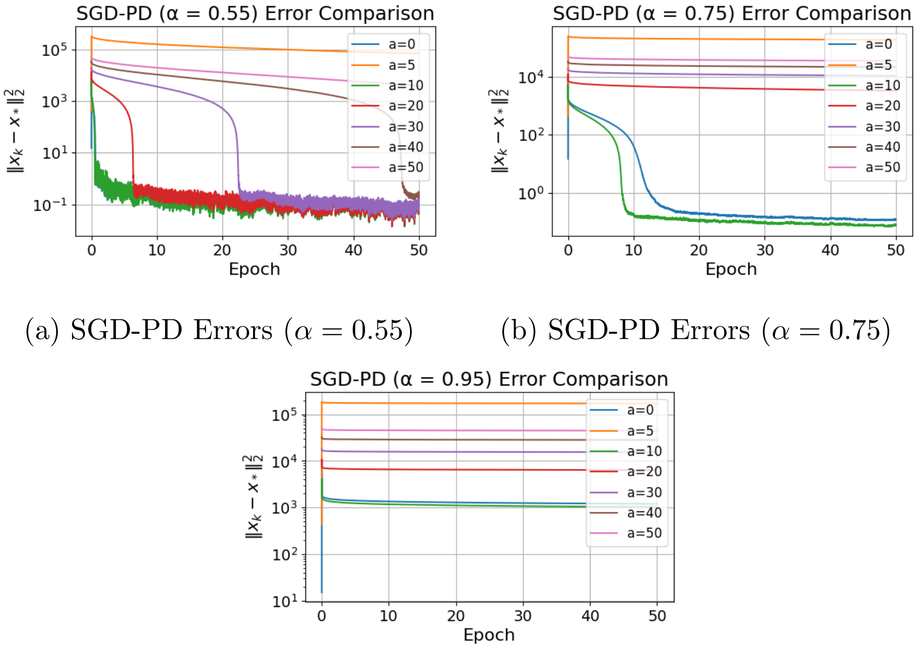
Initially, we ran regular SGD and SGD-IA and plotted the error convergences in figure 1. For these two cases, the behavior exactly matched theoretical expectations. Regular SGD converged faster with more noise, while SGD-IA converged slower but with minimal variance at convergence. Additionally, it appears that although a worse initial state leads to a later convergence, the initial state has no impact on the variance. This makes sense because variance is introduced from the stochastic sampling in the algorithms, which is unrelated to the initial state.

Next, we ran SGD-PD on the simulated dataset with varying power parameters and plotted the error convergences in figure 2. For $\alpha \in (0.5, 1)$, the



(a) Base SGD Errors (b) SGD-IA Errors

Figure 1: Comparison of base and iterate average regression errors on simulated dataset.



(a) SGD-PD Errors ($\alpha = 0.55$) (b) SGD-PD Errors ($\alpha = 0.75$)

(c) SGD-PD Errors ($\alpha = 0.95$)

Figure 2: Comparison of polynomial decay regression errors on simulated dataset.

polynomial decay regression error is bounded by the equation

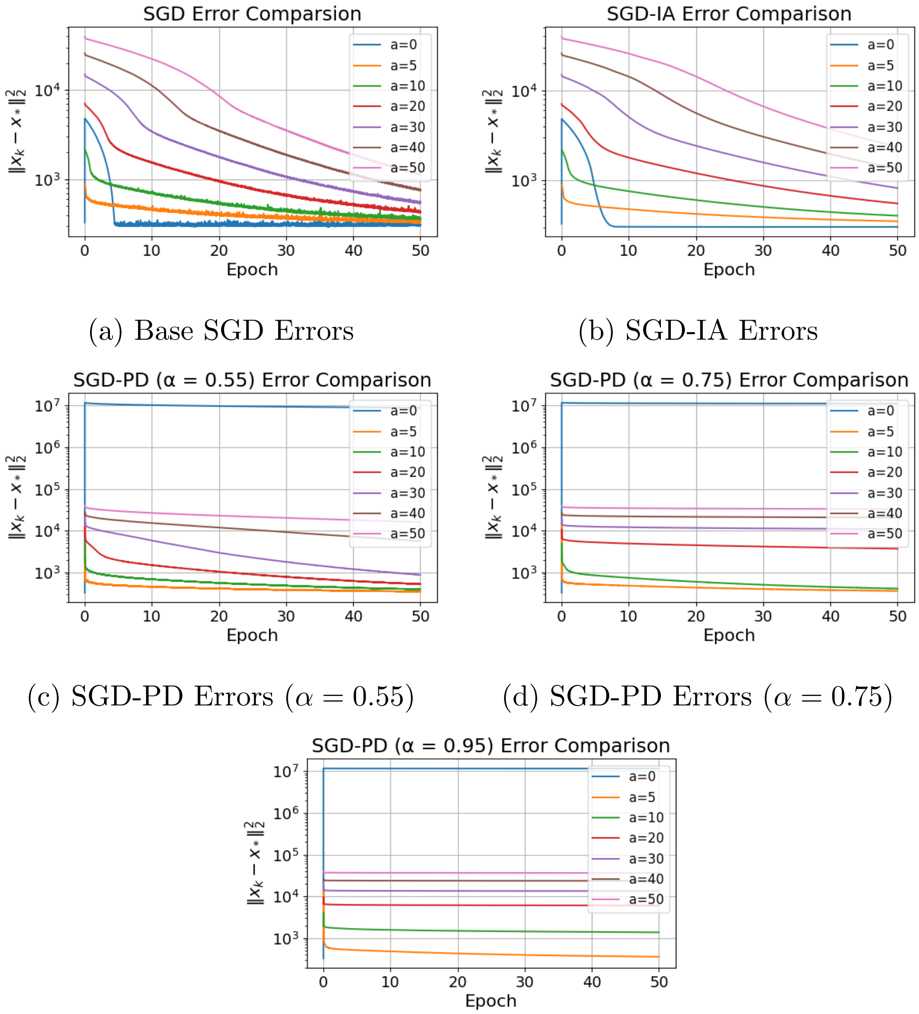
$$\mathbb{E}_{k-1}(\|x_k - x_*\|_2^2) \leq c\|x_{k-1} - x_*\|_2^2 + d,$$

where c and d are constants that have been abstracted out to emphasize the dependence on the initial state, x_*. From the error bound equation, we can conclude that at early iterations the error is significantly impacted by the initial state, but this impact diminishes over time. The SGD-PD experiments match this expected behavior; when the initial state was less optimal, it took longer for the algorithm to converge. But once the convergence occurred, the oscillation of the error was largely the same regardless of the initial state. It is worth mentioning that at larger rates of decay, worse initial states have a greater chance of failing to converge at all. This is not explicitly explained in the error bound equation, but makes sense because the learning rate is approaching 0.

Therefore, we can conclude that SGD-PD has a large tradeoff between variance reduction and convergence rate. If we want to increase the rate of decay to decrease noise, we must ensure that our initial state is relatively close to the optimal state, or the algorithm will fail to converge.

Our second dataset came from the UCI Machine Learning Repository. It consists of 14 features, each representing the state of a parameter of the SGEMM GPU kernel. There are 4 response variables representing 4 separate GPU runs in milliseconds. For the purposes of this experiment, these 4 response variables were consolidated into a single average runtime response, and we worked with the logarithm of the runtimes at the recommendation of the authors. There were 241,600 observations in the dataset, matching the feasible GPU parameter configurations, but we only worked with a random sample of 50,000 observations. Again, we ran the three SGD algorithms on the dataset and plotted the error convergences in figure 3.

In all of the experiments with the GPU dataset, the variance reduction pattern was consistent with both the theory and the observations from the simulated dataset. More interestingly, however, none of the algorithms were able to properly converge, including regular SGD. Surprisingly, SGD-PD failed to converge despite starting with an initial learning rate of 25, as opposed to 0.5 in regular SGD. Initial states with error of magnitude 10^5 were able to converge for the previous dataset, whereas in this dataset even errors of magnitude 10^3 failed to converge. It does remain true that a worse initial state leads to a slower convergence, but there does not seem to be any theory explaining why an initial state with the same magnitude of error converges



(a) Base SGD Errors

(b) SGD-IA Errors

(c) SGD-PD Errors ($\alpha = 0.55$)

(d) SGD-PD Errors ($\alpha = 0.75$)

(e) SGD-PD Errors ($\alpha = 0.95$)

Figure 3: Comparison of regression errors on GPU dataset.

for the for the simulated dataset but not for the GPU dataset.

5 Conclusion

Through our experimentation, we found that although SGD-IA had a slower convergence, it was the best algorithm to minimize the noise that comes with regular SGD. A worse initial state consistently led to a slower convergence, but had no effect on the variance reduction. On the other hand, SGD-PD had both a slower convergence rate and lesser variance reduction than SGD-IA, so it did not seem useful. There was also a more significant trade off between the decay rate and the initial state performance: a large decay rate required a great initial state to have any chance of converging.

Still, we cannot speak too confidently of our results with the GPU dataset. As opposed to the simulated dataset, for which we were given a learning rate, initial states, and batch size, there was no guidance on setting these hyperparameters for the GPU dataset. We used the same learning rate and initial states, but increased the batch size so that the total number of epochs and iterations would stay the same with the larger set of observations. It seems plausible that misconfiguring these hyperparameters would have an adverse effect on the results. In particular, the choice of initial learning rate would have a significant impact on the behavior of SGD-PD. Further experimentation should be conducted with the same dataset and varying the learning rate parameter. Similarly, we have shown from the first dataset that the initial states affect convergence rates, so it would be worthwhile to systematically try more initial states and monitor for better behavior. Additional hyperparameters that could be explored and experimented with include the batch size and the number of epochs and/or iterations. For SGD-PD in particular it would be worth running the algorithm for longer to see if there would be an eventual convergence, or if the learning rate would always decay too quickly.

A Appendix

A.3.1

Treating ψ and $\hat{y}$ as constants, we can calculate $\ell'(x)$ as

$$\begin{aligned}\ell'(x) &= \frac{\partial}{\partial \hat{y}}\left(\frac{\nu+1}{2}\log\left(1+\frac{e^{-2\psi}}{\nu}(y-\hat{y})^2\right)+\psi\right)\\ &= \frac{\nu+1}{2}\frac{\partial}{\partial \hat{y}}\log\left(1+\frac{e^{-2\psi}}{\nu}(y-\hat{y})^2\right)\end{aligned}$$

$$
\begin{aligned}
&= \frac{\nu+1}{2} \times \frac{1}{1+\frac{e^{-2\psi}}{\nu}(y-\hat{y})^2} \frac{\partial}{\partial \hat{y}} \left(1 + \frac{e^{-2\psi}}{\nu}(y-\hat{y})^2 \right) \\
&= \frac{\nu+1}{2\left(1+\frac{e^{-2\psi}}{\nu}(y-\hat{y})^2\right)} \frac{e^{-2\psi}}{\nu} \frac{\partial}{\partial \hat{y}} (y^2 - 2y\hat{y} + \hat{y}^2) \\
&= \frac{(\nu+1)e^{-2\psi}}{2\nu\left(1+\frac{e^{-2\psi}}{\nu}(y-\hat{y})^2\right)} (-2y + 2\hat{y}) \\
&= -\frac{(\nu+1)(e^{-2\psi})(y-\hat{y})}{\nu + e^{-2\psi}(y-\hat{y})^2}.
\end{aligned}
$$

Using $\ell'(x)$ we can calculate $\ell''(x)$ as

$$
\begin{aligned}
\ell''(x) &= \frac{\partial}{\partial \hat{y}} \left(-\frac{(\nu+1)(e^{-2\psi})(y-\hat{y})}{\nu + e^{-2\psi}(y-\hat{y})^2} \right) \\
&= -(\nu+1)(e^{-2\psi}) \frac{\partial}{\partial \hat{y}} \frac{(y-\hat{y})}{\nu + e^{-2\psi}(y-\hat{y})^2} \\
&= -(\nu+1)(e^{-2\psi}) \frac{(\frac{\partial}{\partial \hat{y}}(y-\hat{y}))(\nu + e^{-2\psi}(y-\hat{y})^2) - (y-\hat{y})(\frac{\partial}{\partial \hat{y}}(\nu + e^{-2\psi}(y-\hat{y})^2))}{(\nu + e^{-2\psi}(y-\hat{y})^2)^2} \\
&= -(\nu+1)(e^{-2\psi}) \frac{-(\nu + e^{-2\psi}(y-\hat{y})^2) - (y-\hat{y})(e^{-2\psi})(-2y+2\hat{y})}{(\nu + e^{-2\psi}(y-\hat{y})^2)^2} \\
&= (\nu+1)(e^{-2\psi}) \frac{\nu + e^{-2\psi}(y-\hat{y})^2 - 2(e^{-2\psi})(y-\hat{y})^2}{(\nu + e^{-2\psi}(y-\hat{y})^2)^2}) \\
&= \frac{(\nu+1)(e^{-2\psi})(\nu - e^{-2\psi}(y-\hat{y})^2)}{(\nu + e^{-2\psi}(y-\hat{y})^2)^2}.
\end{aligned}
$$

A.3.2

$\ell''(x) \geq 0$ only if $\nu - e^{-2\psi}(y-\hat{y})^2 \geq 0$. Solving for $\hat{y}$, we have that

$$
\begin{aligned}
\nu - e^{-2\psi}(y-\hat{y})^2 &\geq 0 \\
\nu e^{2\psi} &\geq (y-\hat{y})^2 \\
y - \hat{y} &\in [-\sqrt{\nu e^{2\psi}}, \sqrt{\nu e^{2\psi}}]
\end{aligned}
$$

$$\hat{y} \in [y - \sqrt{\nu e^{2\psi}}, y + \sqrt{\nu e^{2\psi}}].$$

GAIA Disclosure: N/A