Allen Minch
allenminch@brandeis.edu
December 7, 2022
MATH 124A
Instructor: Tyler Maunu

Final Project

My chosen method to implement - Adam - in comparison to stochastic gradient descent

For my final project, the adaptive stochastic gradient descent method I have chosen to implement of the five given to us is Adam. As a starting point, one way in which Adam is similar to regular stochastic gradient descent is that, if you are trying to optimize an average of functions $f(x) = \frac{\sum_{i=1}^{n} f_i(x)}{n}$, Adam computes an approximate gradient of $f(x)$ at a given point by averaging the gradients of a random minibatch of the $f_i(x)$. For simplicity, I will refer to such a stochastic gradient as just a gradient so that I don't have to keep using the word stochastic or approximate. One of the most important ways that Adam is different from regular stochastic gradient descent is that instead of just using the negative of the computed gradient as a descent direction, Adam chooses a descent direction that is based on an exponentially weighted moving average of all negative gradients computed so far. I use the language "based on" here intentionally - there is another important feature of Adam that I will discuss later that means that the descent direction is not exactly parallel to the average I talk about above, but this average is computed in an implementation of Adam, which makes it appropriate to use the words "based on." The moving average above will not necessarily be the same as the direction of the particular negative gradient you compute at a given time point, but this is an intentional feature of the algorithm that is meant to denoise stochastic gradient descent. By finding a moving average of all gradients computed so far and basing the descent direction on this average, Adam provides a descent direction that is hopefully more averaged out, less noisy, and thus is likely to get you convergence somewhat faster by preventing bounces back and forth due to noise in the gradient evaluation. The exponential weighting in Adam means that the weight given to a certain gradient declines exponentially as you move back in time. This is a sensible thing to do, because it takes advantage of a lot of prior information to obtain a less noisy descent direction, but it also reflects the fact that more recent gradients will provide a better reflection of the direction in which you ought to descend now than gradients further in the past.

In practice, when Adam is implemented, it would not make very much sense to compute the exponentially weighted average at each iteration by adding up all of its parts down to the very first computed gradient. Instead, it makes more sense to store this average in a vector m, where m_k is evaluated iteratively using m_{k-1} (the previous exponentially weighted average) and g_k, where g_k is the computed gradient at the kth iteration. It turns out that if we call the factor (less than 1) by which weights on the computed gradients multiply as you move back in time β_1, as the authors of the paper do, then we can more conveniently for computational purposes express m_k in the following recursive way:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)g_k$$

If we think about expanding this out, we get that

$$\begin{aligned}
m_k &= \beta_1(\beta_1 m_{k-2} + (1-\beta_1)g_{k-1}) + (1-\beta_1)g_k \\
&= \beta_1^2 m_{k-2} + \beta_1(1-\beta_1)g_{k-1} + (1-\beta_1)g_k \\
&= \beta_1^2(\beta_1 m_{k-3} + (1-\beta_1)g_{k-2}) + \beta_1(1-\beta_1)g_{k-1} + (1-\beta)g_k \\
&= \beta_1^3 m_{k-3} + \beta_1^2(1-\beta_1)g_{k-2} + \beta_1(1-\beta_1)g_{k-1} + (1-\beta_1)g_k \\
&= \dots \\
&= \beta_1^i m_{k-i} + \beta_1^{i-1}(1-\beta_1)g_{k-(i-1)} + \dots + \beta_1(1-\beta_1)g_{k-1} + (1-\beta_1)g_k \\
&= \beta_1^{k-1} m_1 + \beta_1^{k-2}(1-\beta_1)g_2 + \dots + \beta_1(1-\beta_1)g_{k-1} + (1-\beta_1)g_k
\end{aligned}$$

This is starting to look like some kind of exponentially weighted average, since the weights above form a geometric sequence with common ratio β_1. What the implementation of Adam described in the paper does is to take $m_1 = \beta_1 m_0 + (1-\beta_1)g_1$ and to set $m_0 = 0$. Thus, in complete expanded out form, m_k looks as follows:

$$\begin{aligned}
m_k &= \beta_1^{k-1}(\beta_1 m_0 + (1-\beta_1)g_1) + \beta_1^{k-2}(1-\beta_1)g_2 + \dots + \beta_1(1-\beta_1)g_{k-1} + (1-\beta_1)g_k \\
&= \beta_1^{k-1}(1-\beta_1)g_1 + \beta_1^{k-2}(1-\beta_1)g_2 + \dots + \beta_1(1-\beta_1)g_{k-1} + (1-\beta_1)g_k \\
&= (1-\beta_1)\left(g_k + \beta_1 g_{k-1} + \beta_1^2 g_{k-2} + \dots + \beta_1^{k-2} g_2 + \beta_1^{k-1} g_1\right)
\end{aligned}$$

As we can see, the above m_k is indeed characterized by taking a linear combination of all of the gradients computed so far, with the weight being multiplied by a factor of β_1 for every gradient one time step further in the past. However, as it stands, the above weights do not actually really provide a weighted average of the gradients, because the weights do not sum up to 1. To get a true weighted average, it is necessary to multiply by a certain factor. We can note that the sum of the weights as it stands is

$$(1-\beta_1)+\beta_1(1-\beta_1)+\ldots+\beta_1^{k-1}(1-\beta_1)$$

This is a geometric series with common ratio β_1 and first term $1-\beta_1$, and there are k terms, so its sum is

$$\frac{(1-\beta_1)\left(1-\beta_1^k\right)}{1-\beta_1}=1-\beta_1^k.$$

Thus, if we were to divide the expression for m_k by $1-\beta_1^k$, we would actually obtain a linear combination whose coefficients sum to 1, which would give us a result that is actually a true exponentially weighted average of all gradients computed so far. This makes it advantageous when working with the Adam method to compute $\hat{m_k}$, which is defined by

$$\begin{aligned}\hat{m_k}&=\frac{m_k}{1-\beta_1^k}\\&=\frac{(1-\beta_1)\left(\sum_{i=0}^{k-1}\beta_1^i g_{k-i}\right)}{1-\beta_1^k}\end{aligned}$$

In the paper, this $\hat{m_k}$ is referred to as an initialization bias correction, which is a reflection of the biased initialization $m_0=0$. This $\hat{m_k}$ provides one important building block of the final descent direction chosen in Adam; I will now discuss the other.

One downside of ordinary stochastic gradient descent is that different components of the computed gradient may have different variances which, in particular, might mean that some of the components have large variance, subjecting stochastic gradient descent to a lot of noise. The other denoising improvement Adam makes over ordinary stochastic gradient descent, in addition to taking an exponentially weighted moving average of all gradients computed so far, is that it does a step of standardization which equalizes the variance of the different components of the descent direction used. To accomplish this, Adam also computes an exponentially weighted moving average $\hat{v_k}$ of the component-wise squares of the computed gradient vectors. The rate of exponential decrease of the weights is called β_2 in this case. This is very similar to the $\hat{m_k}$ exponentially weighted moving average above, except that instead of weighting the vectors g_i, the vectors that are weighted are g_i^2, where the jth component of g_i^2 is the square of the jth component of g_i. In the paper, there is also a v_k defined much like the m_k that is useful for computational purposes. The following formulas hold for v_k and $\hat{v_k}$ and quite precisely parallel the formulas involving m_k and $\hat{m_k}$:

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$$

$$\hat{v_k} = \frac{v_k}{1 - \beta_2^k} = \frac{(1 - \beta_2)\left(\sum_{i=0}^{k-1} \beta_2^i g_{k-i}^2\right)}{1 - \beta_2^k}$$

According to the paper, what $\hat{v_k}$ represents an estimate of the second moment - or uncentered variance - of the stochastic gradient. To do standardization of the variance of the components of the gradient, what Adam does is to start by computing the vector $\sqrt{\hat{v_k}}$, where each of the components of this vector are the square root of the corresponding component of $\hat{v_k}$. Adam then takes $\hat{m_k}$, the exponentially weighted moving average of the gradients computed so far, and does a component-wise division of $\hat{m_k}$ by $\sqrt{\hat{v_k}}$. A small ϵ is added to each component of this square root vector in order to prevent division by zero. This standardized result that comes from doing a component-wise division of $\hat{m_k}$ by $\sqrt{\hat{v_k}}$ (plus an ϵ for regularization) is the negative of the final descent direction used by Adam. Note that this direction is not exactly parallel to $\hat{m_k}$, since the components of $\hat{m_k}$ are divided by different scalars. Adam then uses some step size α that is multiplied to this final descent direction, much like a step size is chosen in any other optimization algorithm. In summary, what Adam really seeks to do with this descent direction is to reduce the noise of stochastic gradient descent in two ways:

1. Computing an exponentially weighted moving average of gradients computed so far, but with a heavier weight on more recent gradients since they are closer to the desired descent direction.

2. Standardizing the above average to equalize the variance of the different components of the descent direction used, by dividing out by the component-wise square root of an exponentially weighted moving average of the component-wise squares of gradients computed so far.

Experimental Setup

I did three different experiments for this project. To start, I will talk about the setup that is common to all three experiments. All three of my experiments were done on an unconstrained least squares problem of the following form, which is a convex optimization problem:

$$\text{minimize } \frac{\sum_{i=1}^{n}(x_i^T\theta - y_i)^2}{n},$$

where θ is the variable that is being minimized over, x_i is a given vector for each i and y_i is a given scalar for each i. If we let $X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix}$ and let $y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$, we can write the problem as

$$\text{minimize } \frac{||X\theta - y||_2^2}{n}.$$

In all of my experiments, I had $n = 100$, with $y \in \mathbb{R}^{100}$. Moreover, for each i, I had $x_i \in \mathbb{R}^{10}$, so it was also the case that $\theta \in \mathbb{R}^{10}$.Each time I carried out an experiment, I generated a random matrix X and vector y. In particular, I sampled each component of each x_i (and thus each entry of X) from a normal distribution with mean 0 and some standard deviation σ. Whenever in the rest of this report I refer to the parameter σ, this is what I am referring to. Furthermore, I generated a random $\epsilon \in \mathbb{R}^{100}$, where each component of ϵ is drawn from a standard normal distribution. I then generated y from X and ϵ, taking $y = X\beta + \epsilon$, where $\beta \in \mathbb{R}^{10}$ is a vector of all 1's.

One can determine analytically with some multivariable calculus and linear algebra that the optimal θ for the above least squares problem, call it $\hat{\theta}$, is given by the following expression:

$$\hat{\theta} = (X^T X)^{-1} X^T y. \tag{1}$$

As part of my experiments, I computed $\hat{\theta}$ directly using numpy commands in order to gauge the performance of my algorithms.

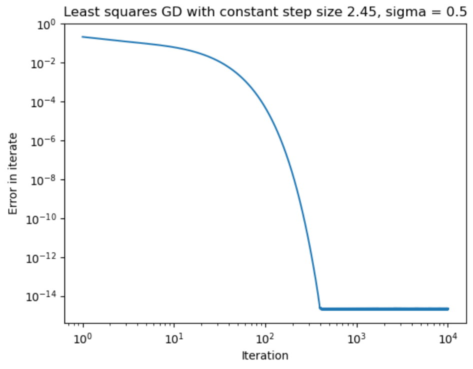
In every experiment I did, I ran whatever algorithms I ran for a total of 10000 iterations, and I always used an initial solution guess of $\theta_0 = \beta$, the vector of all 1's. I wrote functions in Python for each algorithm - namely full gradient descent, stochastic gradient descent, and Adam - that returned a list of the integers from 0 to the total number of iterations, as well as a list consisting of the corresponding values of the iterates. To see how well a given method was performing in a given setting over 10000 iterations, I went through each iterate and computed the norm of the difference between the optimal solution $\hat{\theta}$ and the iterate as a measure of error after each number of iterations. Finally, for each and every run of an algorithm, I made a log-log plot of the error versus iteration. Using a log scale on the y-axis helped spread out points on a given plot, and the reason I used a log scale on the x-axis was to get a good picture of the sublinear convergence of stochastic gradient descent and Adam. Additionally, I used Python's time package to keep track of how long any given run of any algorithm took.

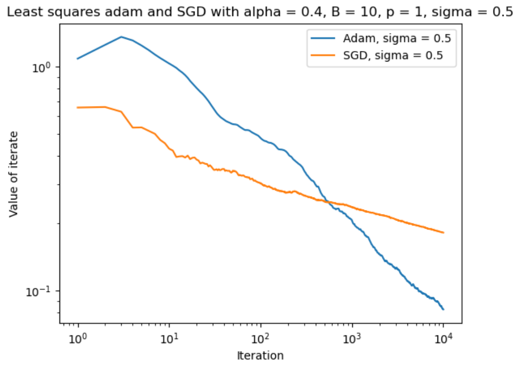
As a final note on the common setup of my experiments, there were certain sets of parameters I passed to my functions for carrying out gradient descent, stochastic gradient descent, and Adam. For gradient descent I always used a constant step size, and I passed as parameters to my gradient descent function this constant step size and the initial guess. For stochastic gradient descent and Adam, I instead had the step size decline according to a power law. In my functions for carrying out stochastic gradient descent and Adam, I had four parameters, namely the initial guess, the initial step size α, the minibatch size B, and the exponent p associated with the power law. It is useful to keep these variable names in mind as I discuss my experimental results.

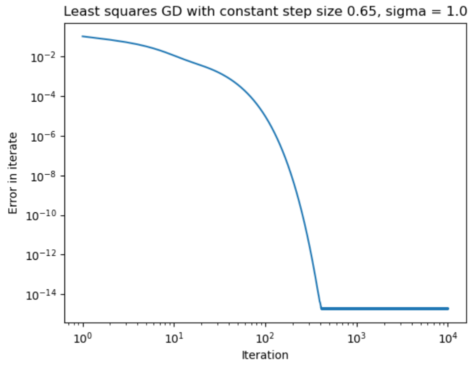
Experiment 1: Varying the standard deviation of the components of X

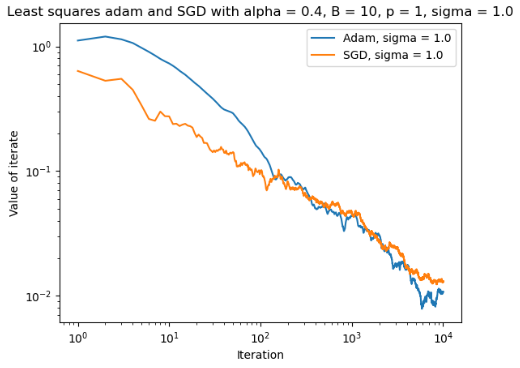
In this experiment, I saw how the various algorithms performed when the standard deviation σ of the components of X was varied. I used $\sigma = 0.5, \sigma = 1.0$, and $\sigma = 2.0$ in my experiment. For each case, I played around to approximate the maximum constant step size for which gradient descent would converge for the given value of σ and used this as my constant step size for gradient descent. In this experiment, for stochastic gradient descent and Adam, I kept the initial step size fixed at $\alpha = 0.4$, the minibatch size fixed at $B = 10$, and the exponent on the power law fixed at $p = 1$. For each value of σ, I plotted the result of gradient descent on one plot and plotted the results of stochastic gradient descent and Adam on a separate plot. I also kept track of the time each algorithm took to run with each value of σ. For gradient descent, I used a step size of 2.45 for $\sigma = 0.5$, a step size of 0.65 for $\sigma = 1.0$, and a step size of 0.15 for $\sigma = 2.0$. Interestingly, I found with some exploration that the maximum step size for which gradient descent converges appears to be roughly related to σ by an inverse square law; for instance, quadrupling σ from 0.5 to 2.0 decreases the step size I used from 2.45 to 0.15, which is a factor of about 16.3 of decrease in the step size used. Note that $16.3 \approx 4^2 = 16$. Moreover, doubling σ from 0.5 to 1.0 decreases the step size I used from 2.45 to 0.65 by a factor of about 3.77, which is not too far from $2^2 = 4$. All this being said, I thought this inverse square relationship was an interesting extra finding along the way.

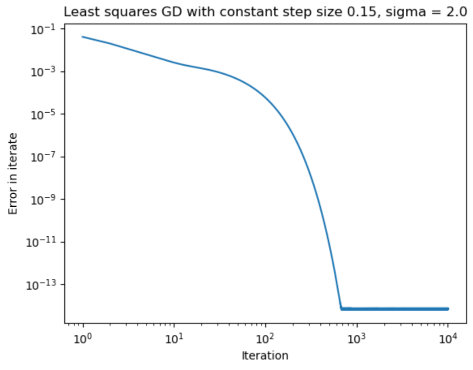
Having described the setup specific to my first experiment, the results that I obtained are on the next page. Note: The y-axis label on the plots on the right should be "Error in iterate"; I must have either typed it wrong accidentally or have accidentally copied "value of iterate" from somewhere else. That being said, it is totally my intention that the plots on the right have y-axis label "Error in iterate."

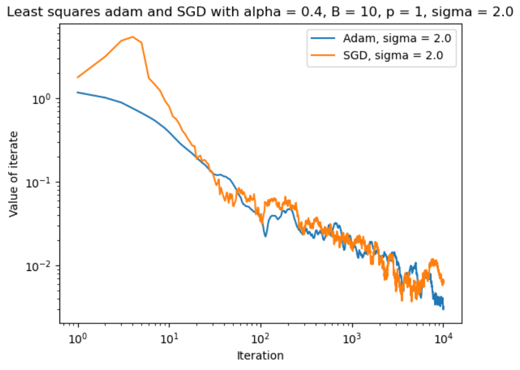












Time required for 10000 iterations of the different algorithms for varying σ

σ	GD	SGD	Adam
0.5	0.179	0.557	0.716
1.0	0.182	0.568	0.745
2.0	0.202	0.646	0.897

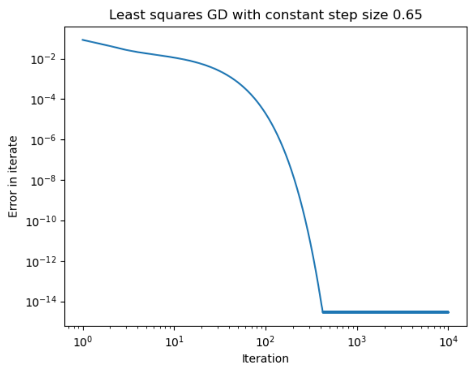
In all of these cases, we can see is that gradient descent is much more accurate after 10000 iterations than either stochastic gradient descent or Adam. In particular, after 10000 iterations, gradient descent has an error on the order of 10^{-15} to 10^{-13}, whereas the error with both Adam and SGD is always at least 10^{-3}. The theoretical reason behind this is that gradient descent, when it converges, exhibits linear convergence, whereas stochastic gradient descent and Adam both have sublinear convergence. With linear convergence, the error in gradient descent roughly decreases exponentially with the number of iterations, but with the sublinear convergence of stochastic gradient descent and Adam, the error roughly decreases only according to a power law like $\frac{1}{k}$. Indeed, the approximately linear nature of the log log plots shown above for SGD and Adam verifies that they have sublinear convergence, because if they do, a log log plot of iteration versus the error in the iterate should be linear.

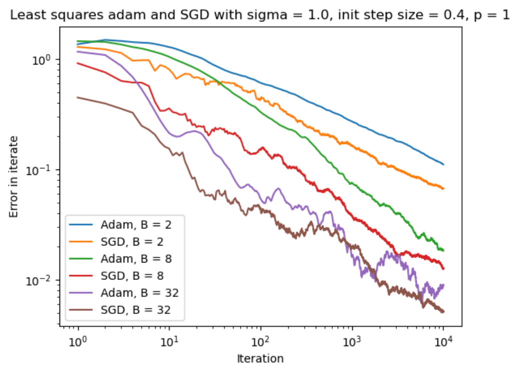
In the plots on the left, it really is hard to tell in which one gradient descent is most accurate after 10000 iterations, because the error is negligible at that point for gradient descent. We can, however, notice some things in the plots on the right. In particular, as σ increases, the accuracy of SGD and Adam after 10000 iterations increases, as can be seen by the way the error decreases from being on the order of 10^{-1}, down to the order of 10^{-2}, and then even lower. Moreover, in all three cases, Adam turns out to be more accurate after 10000 iterations than stochastic gradient descent is, by some factor perhaps between 1 and 10. This comes, however, with Adam appearing to, at least in two of the three cases, have worse accuracy initially, which would seem to suggest that the error in Adam may be declining at a faster rate than the error in SGD. Nonetheless, in the $\sigma = 2.0$ case, the two methods seem to decline in error at a similar rate, so we cannot make too much of a generalization here. But one thing that I feel like is clear is that with both SGD and Adam, the volatility of the error increases as σ increases (becomes more noisy), which would seem kind of to make sense given that the entries of the matrix X have a greater standard deviation. One last remark I will make is that in comparing Adam and SGD in the above plots, it appears that in all three cases, at least initially, Adam's error declines a bit more smoothly than that of SGD, which would make sense given the ways in which Adam tries to smooth things out. However, I would say that later on, after maybe 100 iterations or so, this advantage of Adam does not really seem to be visible. In the $\sigma = 1.0$ and $\sigma = 2.0$ cases, after a certain point, Adam does not really seem to be spared being noisy.

As far as the times 10000 iterations took, we can see that as σ increased, the time 10000 iterations took to run for each method increased slightly, but not by an overwhelming amount. This is not entirely surprising, considering that simply altering the standard deviation of the entries of X does not change the complexity of the problem in terms of size. Moreover, in each case, Adam appears to have taken somewhere between 1 and 1.5 times longer than SGD, and both Adam and SGD took a lot longer than GD to run. The first of these results is not so surprising to me, because Adam, with its computational refinements of computing moving averages and doing standardization, should take somewhat more time to run per iteration than SGD. However, the result of gradient descent being a lot faster than the other methods is surprising to me. It is particularly surprising to me because I took steps to improve the efficiency of the gradient computation for SGD and Adam, by using index slicing and things of that sort. I would have expected that because gradient descent uses the full gradient and does matrix multiplication on bigger matrices, rather than just doing minibatching, gradient descent would take longer to run. My guess is that these surprising results would be explained by some internal Python fluke that I don't really understand, perhaps by index slicing taking a longer time to run than I realize.

Experiment 2: Varying the minibatch size on SGD and Adam

In this experiment, I saw how the various algorithms performed relative to one another with an increasing minibatch size on SGD and Adam. In this experiment, I kept σ constant at 1.0, the initial step size α for SGD and Adam fixed at 0.4, and the exponent on the power law for SGD and Adam fixed at $p = 1$. I also kept the step size for gradient descent fixed at the 0.65 value from Experiment 1, since I fixed σ at 1.0. I varied the minibatch size, using sizes $B = 2, B = 8$, and $B = 32$. The results are shown below, on the next page.





Time required for 10000 iterations of the different algorithms for varying B

B	GD	SGD	Adam
2	0.171	0.502	0.690
8	0.171	0.587	0.699
32	0.171	0.525	0.750

Looking at the above plots, we see once again the contrast between the linear convergence of gradient descent and the sublinear convergence of stochastic gradient descent and Adam, illustrated by the roughly linear nature of the log log plots of error versus iteration for these two methods. Moreover, we see that as the minibatch size increases, both stochastic gradient descent and Adam become more accurate after 10000 iterations. This makes sense, because as the minibatch size increases, more and more $f_i(\theta)$ are being used to compute an approximation to the gradient in these methods, and this should lead to more and more accurate approximations of the gradient. Essentially, as the minibatch size would be increased to $n = 100$, we would expect the results for stochastic gradient descent and Adam to get closer and closer to the result of gradient descent, which we do see happening in at least a small way. However, we also see in the plot on the right what seems to be a disadvantage in that as the batch size increases, the error in both SGD and Adam appears to become more noisy, as seen by the purple and brown curves that seem to zigzag more than any of the other curves after 1000 iterations.

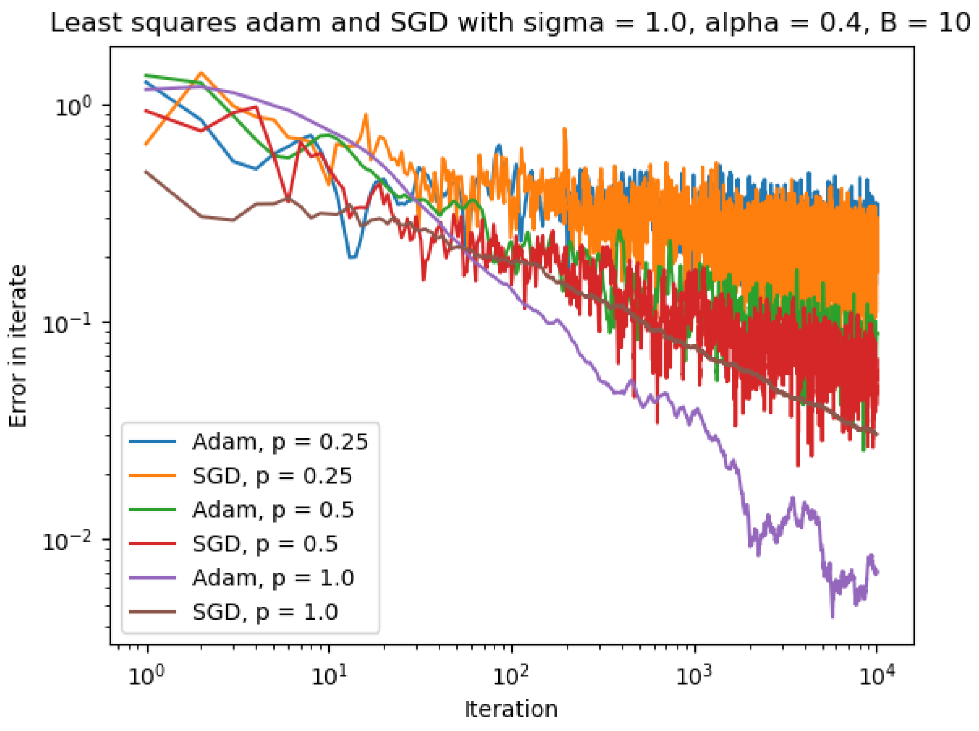
In this particular instance, it appears that for all three cases, Adam is actually less accurate than stochastic gradient descent after 10000 iterations.

This is kind of surprising behavior to me, since I would have thought that Adam would have been an improvement. In fact, as it turns out, I tried running this experiment other times, and from what I remember, it seemed to me that more often than not, Adam was actually better than SGD after 10000 iterations, at least for $B = 8$ and $B = 32$. Despite this particular instance where Adam appears to have worse accuracy after 10000 iterations than SGD, something interesting is worth noting. In particular, near the beginning (between 1 and maybe 5 iterations, or something), the three Adam curves are 3 of the 4 curves with the worst accuracy. Despite this worse accuracy that Adam has compared to SGD initially, we can see that for $B = 8$ and $B = 32$, Adam really appears to nearly catch up with stochastic gradient descent in terms of accuracy after 10000 iterations, and it seems that the error in Adam declines more rapidly than that in SGD. In particular, the purple curve appears to be steeper than the brown one, and the green one appears to be steeper than the red one. Thus, at least in the $B = 8$ and $B = 32$ cases, it appears that the error declines faster with Adam than with SGD, which is an advantage. It is worth noting a caveat to this, however, because for $B = 2$, the blue curve does not really seem to be any steeper than the orange curve, and Adam seems to be clearly worse than SGD in accuracy after 10000 iterations. Indeed, even other times when I tried the experiment and Adam was better after 10000 iterations for $B = 8$ and $B = 32$, if I recall correctly, Adam was consistently worse in accuracy than SGD after 10000 iterations for $B = 2$. Nonetheless, one interesting thing to note in the $B = 2$ case is that the blue curve really appears to be smoother than the orange curve, suggesting Adam's goal of smoothing out variance that can occur in the gradient is successfully achieved. Additionally, it appears that the green curve is somewhat smoother initially than the red curve, so Adam seems to be more smooth near the beginning in the $B = 8$ case as well. However, this advantage seems to not be as noticeable in the $B = 32$ case, and after maybe like 1000 iterations, Adam seems to be as volatile or even more volatile than SGD. In some sense, I would conclude from all of this that Adam appears to have more of an advantage relative to SGD in terms of the rate of decline of the error for larger values of B, whereas Adam seems to have more of an advantage in terms of smoothness of the error relative to SGD at lower values of B.

In terms of the time taken for 10000 iterations to run, I once again see the surprising outcome of GD being a lot faster to run than either stochastic gradient descent or Adam. Again, this is surprising, because it would seem that minibatching would decrease the computational complexity per iteration, and I would expect once again that this finding has something to do with something internal to Python that I am not aware of. It appears to be the case once again that Adam consistently takes between a factor of 1 and 1.5 longer than SGD to run, which is not surprising for reasons I explained in Experiment 1. However, what really surprised me with the time data I got for this minibatching experiment is the lack of noticeable increase in the amount of time it took SGD and Adam to run as the minibatch size increased. One would expect the time to increase because with a larger minibatch size, computations must be done on matrices of larger size. It is thus quite unexpected that the time it took SGD to run paradoxically appears to have decreased when the minibatch size was increased from 8 to 32. I would guess that this is explained once again by flukes going on with the running speed or something of Python for me at the time I ran this particular instance of the experiment; I think I remember trying the experiment other times where the amounts of time for the different minibatch sizes seemed to be reasonable relative to one another.

Experiment 3: Varying the exponent of the power law according to which the step size of SGD and Adam decreases, no gradient descent

In this experiment, I saw how SGD and Adam compare as you increase the exponent of the power law according to which their step size decreases. In this experiment, I kept σ constant at 1.0, the initial step size α for SGD and Adam fixed at 0.4, and the minibatch size fixed at $B = 10$. I varied the exponent on the power law, using $p = 0.25, p = 0.5$, and $p = 1$. The results are shown below, on the next page.



Time required for 10000 iterations of SGD and Adam for varying p

p	SGD	Adam
0.25	0.554	0.773
0.5	0.551	0.744
1	0.529	0.733

Looking at this plot, we can once again see the sublinear convergence of both Adam and stochastic gradient descent, as the log log plot of error versus iteration is roughly linear for both algorithms for each value of p. It appears that for both Adam and stochastic gradient descent, the accuracy after 10000 iterations improves as p is increased, at least increased up to 1. Moreover, the volatility of the error seems to definitely diminish for both SGD and Adam as p is increased. In terms of comparing Adam and SGD, in this particular case, after 100 iterations or so, there seems to be relatively little difference between the performance of stochastic gradient descent and Adam for $p = 0.25$ and $p = 0.5$. Essentially, after 100 iterations, in both cases, the plot for SGD appears to be on top of the plot for Adam, and the two seem to have relatively similar volatility in this region. However, we can see that Adam is a lot more accurate than stochastic gradient descent after 10000 iterations for $p = 1$, and in particular, the error for Adam seems to decline a lot faster than the error for SGD (note how much steeper the purple curve is than the brown one). However, it is interesting to note that from maybe about 500 to 1000 iterations and on, Adam appears in this case to be a lot more volatile than SGD for $p = 1$, which kind of feels surprising considering that Adam is supposed to reduce the variance in the descent direction in general. Nonetheless, considering the same two curves (the purple one and the brown one), the purple one appears to be smoother than the brown one initially. One final remark in this experiment is that like in previous experiments, the accuracy of Adam appears to be worse than SGD at the very beginning. In particular, the three curves that correspond to Adam, namely the green, blue, and purple curves, are the three curves with the worst accuracy after the very first iteration.

It appears that the time required for 10000 iterations of SGD and Adam did not change much at all with p. This is not entirely surprising, considering that all that varies is whether you are dividing by a number raised to the 1st power or a number raised to the 0.5 power; the computational complexity of either method per iteration should not really change much at all with p. Once again, Adam consistently took somewhere between 1 and 1.5 times as long to run as SGD, which is not surprising.

Having discussed the results of the three experiments, I would say that I get the following important takeaways about Adam and in comparing stochastic gradient descent with Adam in the least squares problem:

1. Adam seems to tend to have worse accuracy after the first iteration or few iterations than stochastic gradient descent; I would wonder if this is because of the first moment estimator being initialized to 0. However, under appropriate conditions, it seems like Adam has sublinear convergence with a steeper slope than does stochastic gradient descent, and eventually it can overtake stochastic gradient descent and become more accurate (or almost as accurate) despite having had a bad start.

2. Adam consistently takes longer to run than stochastic gradient descent by a factor roughly in the range of 1 to 1.5.

3. It does not seem to be an absolute given that Adam is more accurate than SGD after 10000 iterations, but the accuracy of Adam relative to SGD after 10000 iterations seems to be better with a larger B and larger p.

4. Adam seems to in some cases have an advantage of being more smooth and less noisy than stochastic gradient descent initially, but after like 100 or 1000 iterations, Adam seems like it can in some cases get more noisy, and the advantage that Adam has relative to stochastic gradient descent in terms of smoothness of the plot of the error may dissipate. In some cases, Adam may have even more noisy error after 1000 or more iterations than stochastic gradient descent, like in the third experiment with $p = 1$.

I have certainly not done all possible experiments I could do. One other kind of experiment that might be insightful would be to see what happens to SGD and Adam when one changes the dimensionality of the least squares problem by changing n or by changing the dimension of the vector θ. Another experiment that might be worthwhile would be to try using a constant step size on SGD and Adam instead and to see under what conditions the methods converge or diverge. A third potential experiment would be to vary the initial step size for SGD and Adam while still having the step size decline according to a power law. Finally, a fourth potentially worthwhile experiment would be to consider the performance of SGD and Adam in some nonconvex optimization problem other than the least squares problem, to see if Adam is a lot better suited than SGD to this type of problem. Despite not having the time to do all of these experiments, I think it was very interesting to learn about Adam as an alternative optimization algorithm and to glean some insight from the experiments I did, especially about how Adam performed in certain contexts, and in particular how it performed relative to gradient descent and stochastic gradient descent.