Gradient Descent

Note: We were able to completely cover the outcomes of our work on the project in the video. This document adds some more background detail to our presentation.

The goal of gradient descent is to find the minimum of a function and the position vectors that correspond to that minimum. Gradient descent is an iterative process, moving in the direction of steepest descent, which is the direction directly opposite to the gradient vector.

For a scalar function f that can be expressed in the form of k independent variables, the gradient of f is defined as the vector of the partial derivatives of f with respect to each variable. The gradient is essential to gradient descent because its components inform us of the rate of change with respect to any variable, able to be evaluated at any position.

Our formulation of the iterative process is below:

$$\mathbf{r}_{n+1} = \mathbf{r}_n - \gamma \frac{\nabla F(\mathbf{r}_n)}{\left\|\nabla F(\mathbf{r}_n)\right\|}$$

Here, r_n represents the position vector on the nth iteration and r_{n+1} represents the position vector on the next iteration. This expression above tells us how to update our position from one iteration to the next.

The rightmost expression describes the normalized gradient vector evaluated at the current position vector multiplied by scalar gamma. Gamma is also known as the learning rate, which is analogous to the step size used in Euler’s Method. Whereas Euler’s Method is conventionally considered in two dimensions, gradient descent can be executed in any number of dimensions, since dimensions correspond to the number of elements in each position vector and gradient vector.

The learning rate exhibits a Goldilocks zone phenomenon. A learning rate that is too low will cause gradient descent to take many iterations of computation to reach the minimum, which is an inefficient use of a computer’s resources. Especially with data sets in the real world, a low learning rate can costs hours of additional processing time. A learning rate that is too high will quickly reach the vicinity of the minimum, but the position vector will converge slowly, bouncing around the minimum because the learning rate, or step size, is too large.

Only with an appropriately adjusted learning rate will the algorithm be both efficient and precise in determining the minimum.

The Benefits of Gradient Descent

The first benefit of gradient descent is that its iterations are computationally cheap. Estimating the partial derivatives at a position is cheap because the limit definition of a derivative can be used. By setting an arbitrarily small h, we can obtain an accurate value for each partial derivative by adding h to the relevant independent variable and finding (f([x+h, y]) - f([x, y])) / h.

The second benefit of gradient descent is that it is particularly effective on convex problems. A convex problem is one in which the relevant surface has one minimum. This guarantees that the gradient descent algorithm will locate the global minimum. Compared to non-convex problems, where there is no guarantee that the local minimum is optimal, convex problems are much more suitable for gradient descent.

The Faults of Gradient Descent

The largest fault of gradient descent is that it cannot guarantee locating a global minimum. Gradient descent, given that the algorithm is executed one time at one start point, can only locate a local minimum. However, we implement and discuss an approach that counteracts this downside later in the presentation.

Another downside of gradient descent is that it does not mesh well with non-differentiable functions. For example, the absolute value function is used in lasso regression, a type of regression used in L1 regularization for reinforcement learning. Combining gradient descent with such non-differentiable functions may breed issues, as the gradient vector relies on the function being differentiable at all positions.

Problems with Gradient Descent

- cannot handle non-differentiable functions
- absolute value is used in L1 regularization
- cannot guarantee a global minimum
- random sampling used in advanced optimization algorithms
- many problems are not strongly convex or well-conditioned

- stochastic gradient descent (SGD), mini-batch gradient descent, and momentum-based methods are optimized versions that try to solve some of these problems

Easy Application: Paraboloid

Python implementation of gradient descent on the paraboloid z=x^2+y^2 (convex surface).

It's important to note that the gradient can be pre-defined for a differentiable function (for f(x,y) = x^2 + y^2 it would be <2x, 2y>), but estimating the gradient with estimations of partial derivatives generalizes (what we did) the code to non-differentiable surfaces.

Pseudocode is as listed below:

- specify the surface function
- specify inputs

- Find negative gradient function:
- use the difference quotient with a very small h value to estimate partial derivatives
- return the negative gradient since we want to descend not ascend

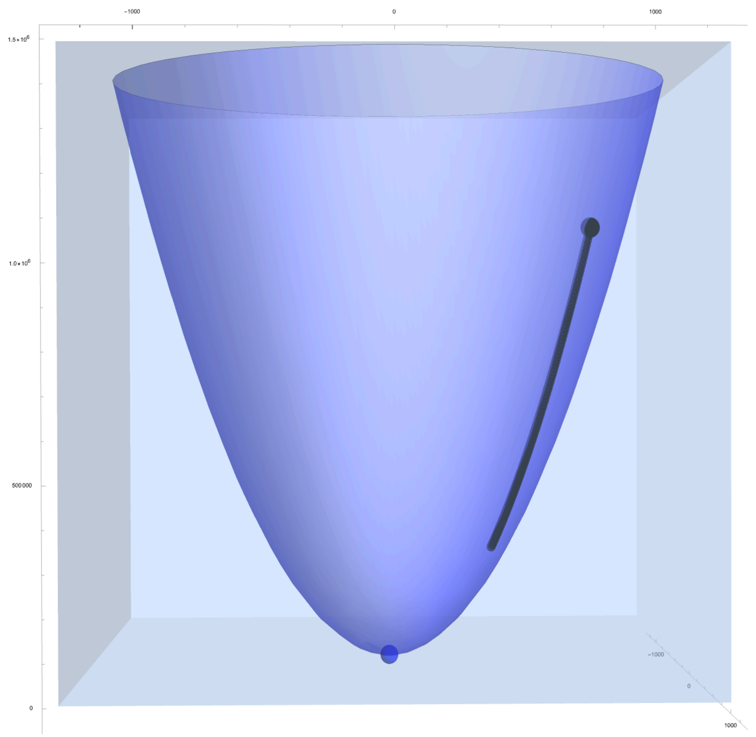
- descend the specified number of steps:
    - find the negative gradient for the current point
    - calculate the vector's magnitude
    - normalize the gradient vector into a unit vector (for it to specify direction)
    - add the direction * learningRate to the final point
- after the descent return the point (which should hopefully be a minimum)

gradient descent is easily performed

Easy Application: Learning Rates

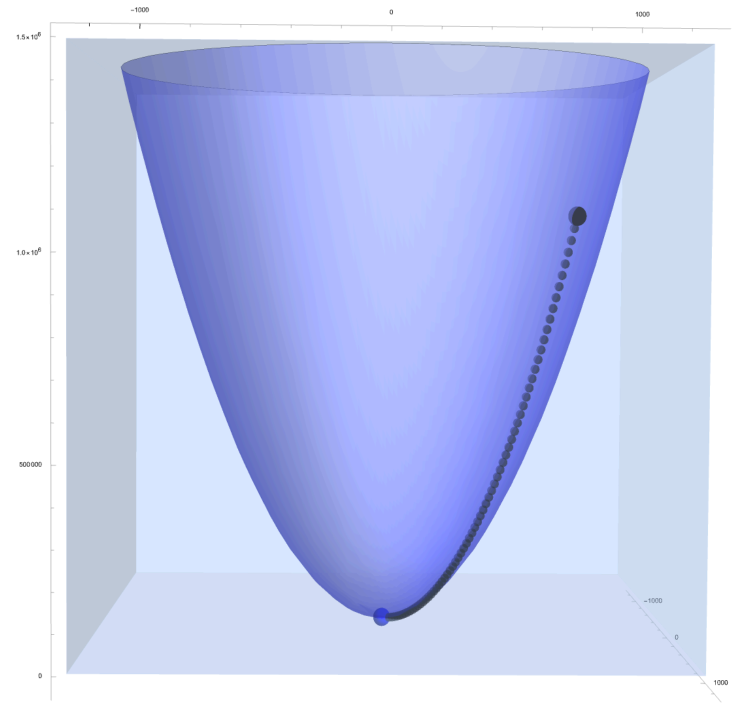
We generated three Mathematica figures to represent three different progressions of gradient descent. Through these three pictures, we varied the learning rate from 5 to 25 and to 100.

Learning rate 5



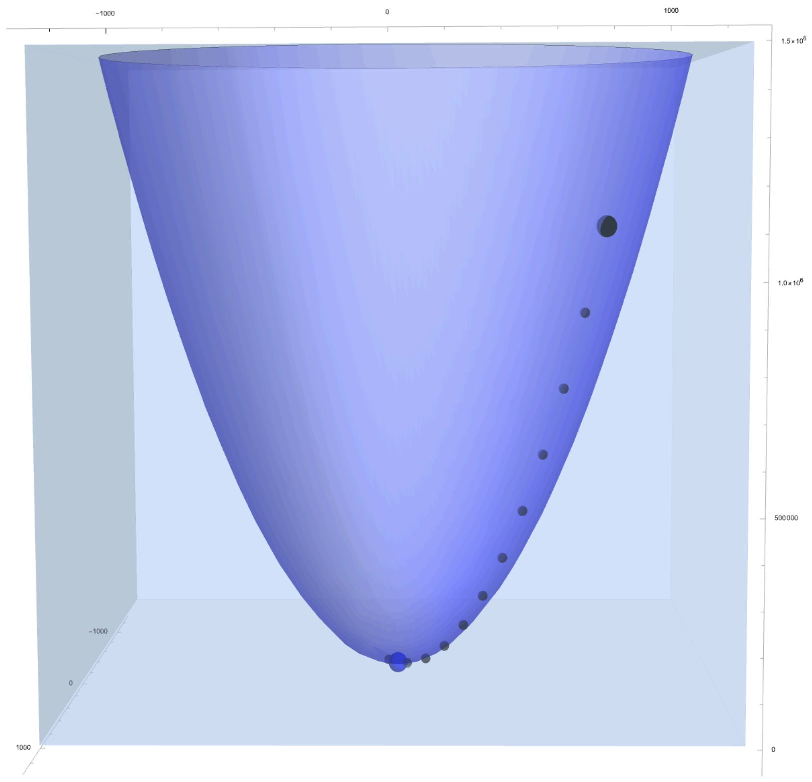
Given 100 iterations, the gradient descent with learning rate = 5 did not make it to the minimum because the rate was too low. This is an example of how a low learning rate leads to high computation time: both other learning rates were able to approach the minimum.

Learning rate 25



After 100 iterations, the gradient descent with learning rate = 25 comfortably approached the minimum at (0, 0, 0). This is an example of a learning rate that falls into the Goldilocks zone: it is negligibly close to the minimum after a moderate number of iterations.

Learning rate 100



After 100 iterations, the gradient descent with learning rate = 100 quickly descended and approached the local minimum. However, it should be noted that the position vector bounced around the minimum (0, 0, 0) iteration after iteration, still remaining moderately far away from the actual optimal point. This is an example of how a learning rate that is too large can cause inaccuracies and suboptimal results.

Non-Trivial Application: More Realistic Surface

describe the surface here. describe where the big dip is supposed to be and how we made it that way. problem: many different mins. iterative starting from one starting point can lead to local min, but not extreme min.

Non-Trivial Application: Random Sampling

Mathematica's FindMinimum function only finds local mins

Our function can find global mins with enough random samples (or if the samples are lucky)

The number of random samples chosen depends on the certainty in which you want the program to find the global min.

Simulation:

The simulation is an interactive demo to play with the inputs for gradient descent.

The pseudocode for the simulation is listed below:

```
● import necessary graphing libraries
● specify the surface function
● specify inputs

● function to find negative gradient (used to find direction of descent)
● use the difference quotient with a very small h value to estimate partial
  derivatives
● return the negative gradient since we want to descend not ascend

● function to find local minimum using gradient descent from a starting point
● initialize data array to store points
    ○ descend the specified number of steps:
    ○ find the negative gradient for the current point
    ○ calculate the vector's magnitude
    ○ normalize the gradient vector into a unit vector (for it to specify direction)
    ○ add the direction * learningRate to the final point
    ○ add the point to the data array
● after the descent return the point array (which should hopefully be a minimum) to
  use for the animation skipping a set amount of frames

● define bounds for the surface
● Initialize an empty array for z values
● Compute z values using the function f
● reshape data to be graphed
● Create a figure and axis
● Create legend with color box
● create 3D axes
```

- Plot the 3D surface
- Label the plot
- Add a color bar
- Add space for the sliders to go in
- Add sliders in these spaces
- define initial points
- graph start point
- graph endpoint
- define list of points to be changed per input update
- define empty list of data to be changed per input update
- fill list with initial descent path

- define update function for the animation
- for each frame, display the next point in the descent
- update the scatter plot
- return a single element tuple of the scatter plot for the animation

- function to update the descent path points per input change
- clear and relabel axes
- re-plot 3D surface
- update data by rerunning the gradient descent algorithm with new inputs
- plot start point
- plot end point
- redefine the plotted point list
- redefine point list

- Define the update functions for the sliders
- Call update functions on changes in the sliders
- Define the animation manager
- Show plot

Sources

L. Vandenberghe, Lecture notes for EE 236C, UCLA, Spring 2011-2012. http://www.seas.ucla.edu/~vandenbe/ee236c.html.

We visited this source (specifically lecture slides 1, gradient method) to find out more about the iterative gradient descent method as well as get visual inspiration for some of our visualizations.

A. Ahmadi, Lecture notes for ORF 523, Princeton University, Spring 2017-2018. https://www.princeton.edu/~aaa/Public/Teaching/ORF523/ORF523_Lec7.pdf.

We visited this source to find out more about what convexity means for a surface.

A. Nagpal, L1 and L2 Regularization Methods, Towards Data Science, October 13 2017. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c.

This article describes using L1 Regularization through lasso regression, which includes an absolute value function in its formulation. I referred to this technique when I was discussing the faults of gradient descent because the absolute value function is nondifferentiable and computing the gradient can be troublesome.

H. John, D. Dale, E. Firing, M. Droettboom, Matplotlib 3.8.4 documentation, Matplotlib, 2024, https://matplotlib.org/stable/index.html.

This is the official documentation page for matplotlib, the python graphing library we used for the simulation. I referred to this page several times for syntax and examples of animation and 3D graphing.