CSCI 0410: AI Foundations **Fall 2025**
Professor Eric Ewing[1] **Continuous Optimization Unit**

Introduction to Continuous Optimization

1 Introduction

Although CS has traditionally focused on discrete problems—after all, computers are digital not analog systems—modern AI, specifically deep learning, largely concerns continuous problems. Likewise, our course thus far has primarily focused on the discrete: combinatorial optimization, constraint satisfaction, automated planning via symbolic reasoning, and classical search. But today we switch gears, and begin our study of continuous optimization, which arises in robotics and control, computer vision, and machine learning.

To begin, consider the following **Router Optimization Problem**: You have been tasked with placing a WiFi router in an office. There are k computers in the office, all of which require WiFi access. The farther away a computer is from the router, the worse the signal is, and the slower the network connection is. The computer locations are given by $\{\boldsymbol{c}^1, \ldots, \boldsymbol{c}^k\}$, with $\boldsymbol{c}^i \in \mathbb{R}^2$ for all $i \in [k]$, and the speed of the network connection for computer i with router placement $\boldsymbol{x} \in \mathbb{R}^2$ is given by $q(\boldsymbol{x}, \boldsymbol{c})$, where the function q defines the *quality* of a signal connection. Your goal is to find a router placement $\boldsymbol{x} \in \mathbb{R}^2$, such that the total speed of all computers in the office is maximized, i.e., you must solve:

$$\max_{\boldsymbol{x}} \sum_{i=1}^{k} q(\boldsymbol{x}, \boldsymbol{c}^i)$$

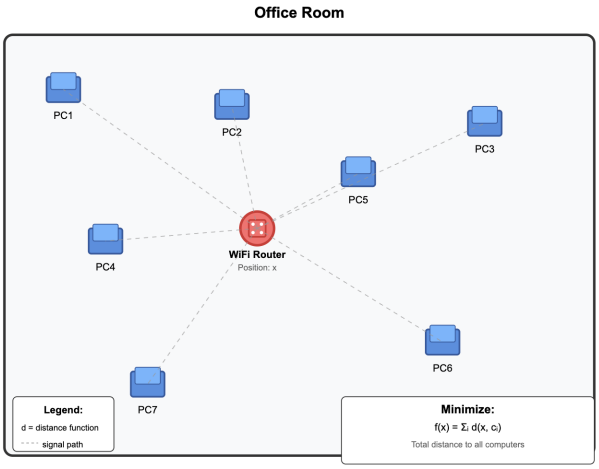

Figure 1: Router Optimization Problem. Where should the router be placed to maximize network connectivity across all computers at their given locations? Figure generated by Claude Sonnet 4.0.

The Router Optimization Problem is continuous optimization problem because the variable in your control, namely the router's placement $\boldsymbol{x}$, is a pair $(x_1, x_2) \in \mathbb{R}^2$, i.e., a vector of real numbers. In the next four lectures, we will develop a number of techniques for solving such problems, and introduce you to various tools that employ these techniques. Why so many lectures? How hard is this problem, anyway? Well, that

[1] These notes were compiled in conjunction with Professor Amy Greenwald.

depends on a number of factors. If q is a "nice" function, then it is quite easy to solve. If it is not "nice",[2] or if there are constraints on where you can place the router, then it can be quite challenging.

2 Applications

Continuous optimization is one of the most important topics in Artificial Intelligence.

- Robot Motion Planning and Manipulation: Earlier this semester, we looked at search problems and algorithms, like A*, that work in discrete space. A*was developed to plan paths for Shakey the Robot. However, robots exist in the real world and the real world is not discrete. Why should we limit our robots to transitioning between discrete states (i.e., move left, right, up, or down)? Motion Planning for robots seeks to find continuous paths through space that optimize some objective, avoid obstacles, and accomplish other tasks. In order to work in continuous space, motion planning relies heavily on continuous optimization techniques.
- Mathematical Programming: Mathematical Programming is one of the secret engines that makes our modern world work. At its core, Mathematical Programming is the practice of formulating and solving optimization problems in a consistent way. Every time packages get routed across the country, ride-sharing apps match drivers to riders, or airlines schedule flights, an optimization problem is being solved behind the scenes. At its heart, mathematical programming is about making the best possible decision when there are many choices and limited resources — whether that means minimizing cost, maximizing efficiency, or balancing fairness. As you can probably imagine, optimizing for revenue and profit while minimizing costs is one of the most important tools companies have in a capitalist society.
- Machine Learning and Deep Learning: These topics are at the forefront of everyone's minds at the moment, whether they are working on AI or not. In general, machine learning seeks to approximate real-world functions with simpler functions (like a linear model to approximate some real-world data). These simpler models are "fit" to real-world data, using some form of optimization technique. Most machine learning models use continuous parameters and therefore use continuous optimization techniques. The entire premise of Deep Learning is optimizing the (continuous) parameters of a model so that it minimizes some objective. The reason LLMs are able to function at all is because of techniques from continuous optimization, like stochastic gradient descent (SGD).

3 Continuous Optimization

This unit is concerned with solving continuous optimization problems of the form: find $\boldsymbol{x}^* \in \mathbb{R}^d$ such that

$$f(\boldsymbol{x}^*) = \min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x})$$

Here, $\mathbb{R}^d$ is a finite-dimensional vector space ($d \in \mathbb{N}$) with elements $\boldsymbol{x} = (x_1, \ldots, x_d)$ and $f : \mathbb{R}^d \to \mathbb{R}$.

In words, we are looking for a point $\boldsymbol{x}^*$ at which the function f is minimal, as well as an associated solution value $f(\boldsymbol{x}^*)$. Such a point is called a **global minimum**, as $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$, for all $\boldsymbol{x} \in \mathbb{R}^d$.

More generally, a point $\tilde{x}$ is called a **local minimum** if $f(\tilde{x}) \leq f(\boldsymbol{x})$, for all $\boldsymbol{x} \in \mathbb{R}^d$ within some small distance (i.e., neighborhood) of $\tilde{x}$. Just as we saw in our study of discrete optimization, it is often not

[2] WiFi signal strength is definitely not a nice function.

possible to find a global optimum, so we instead settle for a local optimum. And again, that local optimum is sometimes in fact a global optimum, but there is no (efficient) way to confirm it as such.

Optimization problems are typically expressed in terms of minimization, but fear not. We can easily convert maximization problems (like our WiFi-router example) to minimization problems:

$$\max_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x})$$

is equivalent to:

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} -f(\boldsymbol{x})$$

In the context of optimization, the function f is called an **objective function**.

4 Solution Methods

There are two primary classes of methods for solving continuous optimization problems: **analytical methods**, which you should be familiar with from calculus (they involve solving for **critical points**) and **gradient methods**, which navigate a function's landscape by following its **gradient**. We discuss each of these methods in turn, after defining "gradient," which pertains to both methods.

Recall the meaning of a **derivative**, as you were taught in calculus. Given a function $f : \mathbb{R} \to \mathbb{R}$ of one variable, say x, the derivative of f at x indicates how fast the value of f is increasing at x. The **gradient** of a function generalizes this concept to real-valued functions $f : \mathbb{R}^d \to \mathbb{R}$ defined on vectors in $\mathbb{R}^d$.

Given such a function $f : \mathbb{R}^d \to \mathbb{R}$, the **gradient** of f, denoted ∇f, is the vector of its **partial derivatives**: i.e.,

$$\nabla f = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_d} \right]^T = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_d} \end{bmatrix}$$

Generalizing the notion of derivative to multiple dimensions, the gradient points in the direction of steepest increase. Moreover, each component $\frac{\partial f(\boldsymbol{x})}{\partial x_i}$ in ∇f indicates the rate of increase in f's value that would result from a small move in the direction of x_i, holding all other variables (i.e., all x_j, for $j \neq i$) constant. In other words, each component is the magnitude of the slope along its particular axis.

For example, let $f(x, y) = x - 2y$. The gradient of f is $[1, -2]$. The first component, 1, is the partial derivative of $f(\boldsymbol{x})$ with respect to x_1; likewise, the second component, -2 is the partial derivative with respect to x_2. The meaning of this gradient is as follows: if we were to increase x by 1 unit, f would likewise increase by 1; but if we were to increase y by 1 unit, f would decrease by 2.

As another example, consider $g(x) = 2x^3$ so that $\nabla g = [6x]$, i.e., the derivative[3] of $g(x)$ with respect to x. Unlike for the function $f(x, y)$, where the gradient was constant (i.e., it did not vary with x and y), the slope of $g(x)$ varies with x. When $x = 1$, if we were to increase x by 1 unit, g would likewise increase by 6; but when $x = 2$, if we were to increase x by 1 unit, g would increase by 12.

We are now ready to describe the two aforementioned solution methods, and use them to solve the Router Optimization Problem. Note that because optimization problems are most often stated as minimization problems, gradient methods are typically *descent* methods, that move in the direction *opposite* of that which is indicated by the gradient (as the gradient points in the direction of steepest increase).

[3]Recall that the gradient and the derivative coincide on functions of just 1 variable.

4.1 Method 1: Critical Points

You should already be familiar with solving continuous optimization problems analytically from high school calculus. The method proceeds as follows:

1. Finding the **critical points** (i.e., points that satisfy the **first-order condition for optimality**)
2. Apply the **second-derivative test** (i.e., test the second-order condition for optimality)

We define all of the highlighted terms presently.

A **critical point** of a function is a point at which the gradient is zero (or does not exist). The critical points may be local minima or they may be local maxima.[4]

When the gradient at a point is zero, the point is said to satisfy the **first-order condition for optimality**. This name derives from the fact that the gradient, which relies on first derivatives, contains what is called "first-order information". While necessary for optimality, satisfying the first-order condition is not sufficient.

To ascertain whether a critical point is a minimum or a maximum requires a second-derivative test, i.e., using second-order information. Whereas first-order information concerns a function's slope, second-order information concerns its **curvature**. Curvature pertains to how a function "bends". Where a function has positive curvature, it bends upward—it is shaped like a bowl; and where it has negative curvature, it bends downward—it is shaped like a hill.

With these definitions in hand, we can define the second-derivative test: a critical point is a local minimum when the second derivative is strictly positive; and it is a local maximum when the second derivative is strictly negative. (When the curvature is zero, the second-derivative test is inconclusive.)

Consider the function $f = 3x^2$ with $\nabla f = [6x]$ and $\nabla^2 f = 6$. This function has positive curvature everywhere; thus, the critical point 0, where $\nabla f = 0$, is a local minimum. (In fact, it is a global minimum.)

By way of comparison, the function $g = 2x^3$ with $\nabla g = [6x^2]$ and $\nabla^2 g = [12x]$ has positive curvature when x is positive, and negative curvature when x is negative. As a result, the second-derivative test is inconclusive. (You may recall from high school calculus that such a point is called an **inflection point**.)

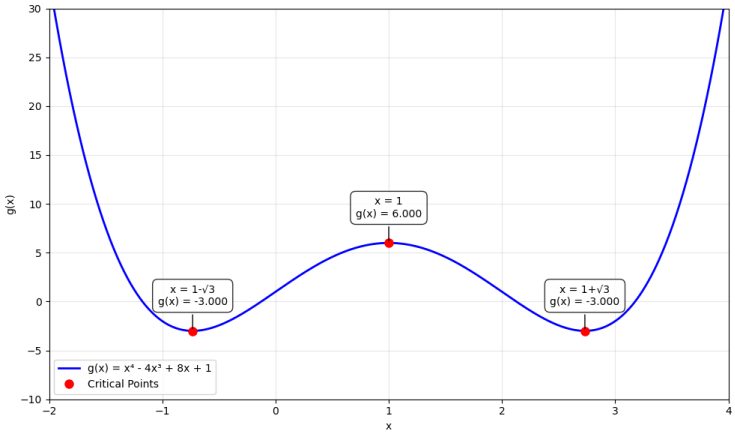

Figure 2: The critical points and values for $g(x)$

[4] A critical point may also indicate the presence of a **saddle point**, namely a point that is neither a minimum nor a maximum, as from such a point, the function increases in one direction and decreases in another.

Example The following function is depicted in Figure 2:

$$g(x) = x^4 - 4x^3 + 8x + 1$$

We analytically derive its global minima and local maxima by solving for its critical points (i.e., finding points that satisfy the first-order optimality condition) and then applying the second-derivative test.

The derivative of $g(x)$ is

$$\frac{\partial g}{\partial x} = 4x^3 - 12x^2 + 8$$

We set this derivative equal to zero and solve for x. As the derivative is cubic, it yields three roots, i.e., three critical points: $x \in \{1 - \sqrt{3}, 1, 1 + \sqrt{3}\}$.

To determine which critical points correspond to minima versus maxima, we evaluate $g(x)$ at each critical point:

$$\begin{aligned} g(1 - \sqrt{3}) &= -3 \\ g(1) &= 6 \\ g(1 + \sqrt{3}) &= -3 \end{aligned}$$

From these values, we can see that $x = 1$ yields a larger function value than the other two critical points, suggesting it is a local maximum, while $x = 1 \pm \sqrt{3}$ appear to be minima.

To verify that $x \in \{1 \pm \sqrt{3}\}$ are indeed *global* minima (not just local), we must consider the behavior of g as $x \to \pm\infty$. Since the x^4 term dominates for large $|x|$, we have $g(x) \to \infty$ as $x \to \pm\infty$. This confirms that the function values at the critical points $x = 1 \pm \sqrt{3}$ are the smallest values that g attains anywhere.

Therefore, the function g has two global minima at $x \in \{1 \pm \sqrt{3}\}$, both with function value -3. In addition, $x = 1$ is a local maximum with function value 6.

Router Optimization Problem ℓ^2 Distance: Let's return to the Router Optimization Problem, and assume the strength of the WiFi signal for a computer $\boldsymbol{c} \in \mathbb{R}^d$ is proportional to the negative of the squared Euclidean distance from $\boldsymbol{c}$ to the router $\boldsymbol{x} \in \mathbb{R}^d$.[5] The squared Euclidean distance between two vectors is written as $\|\boldsymbol{x} - \boldsymbol{c}\|_2^2$ and defined as the square root of the sum of the squared differences between components, i.e.,

$$\|\boldsymbol{x} - \boldsymbol{c}\|_2^2 = \sum_{j=1}^{d} (\boldsymbol{x}_j - c_j^i)^2$$

N.B. The $\|\cdot\|_2$ notation stems from the definition of the L_2-norm (or simply, the 2 norm), of a vector $\boldsymbol{y} \in \mathbb{R}^d$, which is defined as $\|\boldsymbol{y}\|_2 = \sqrt{\sum_{i=1}^{d} y_i^2}$.

Thus, to maximize signal strength, we aim to solve the following optimization problem:

$$\max_{\boldsymbol{x} \in \mathbb{R}^2} \sum_{i=1}^{k} -\|\boldsymbol{x} - \boldsymbol{c}^i\|_2^2$$

[5] We use negative distance here because high signal strength occurs at short distances and low signal strength, at far distances. As distance is always positive (and negative distance, negative), negative distance is greatest at $d(\boldsymbol{x}, \boldsymbol{c}) = 0$, when the router and computer are in the same location, and is lower at all greater distances.

Or, equivalently:

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} \sum_{i=1}^{k} \|\boldsymbol{x} - \boldsymbol{c}^i\|_2^2$$

We are interested in finding the critical points of this function. We start by computing the gradient of the objective function $\sum_{i=1}^{k} \|\boldsymbol{x} - \boldsymbol{c}^i\|_2^2$, with respect to $\boldsymbol{x}$, the variable in this function. (The $\boldsymbol{c}^i$ are constants.)

To compute this gradient, we apply the sum rule, namely the derivative of a sum is the sum of the derivatives (i.e., the sum and derivative operators are interchangeable, as they are both linear):

$$\begin{aligned}\nabla_{\boldsymbol{x}} f &= \nabla_{\boldsymbol{x}} \sum_{i=1}^{k} \|\boldsymbol{x} - \boldsymbol{c}^i\|_2^2 \\ &= \sum_{i=1}^{k} \nabla_{\boldsymbol{x}} \|\boldsymbol{x} - \boldsymbol{c}^i\|_2^2\end{aligned}$$

Now the gradient with respect to $\boldsymbol{x}$ for a single computer $\boldsymbol{c}^i$ is as follows:

$$\nabla_{\boldsymbol{x}} \|\boldsymbol{x} - \boldsymbol{c}^i\|_2^2 = \begin{bmatrix} 2(x_1 - c_1^i) \\ 2(x_2 - c_2^i) \\ \vdots \\ 2(x_d - c_d^i) \end{bmatrix}$$

Thus, if there were only one computer, the gradient would be $\mathbf{0}$ iff the router and the computer shared the same location, as each component of the gradient would evaluate to 0.

For the more interesting case of multiple computers, we continue as follows:

$$\begin{aligned}\nabla_{\boldsymbol{x}} f &= \sum_{i=1}^{k} \nabla_{\boldsymbol{x}} \|\boldsymbol{x} - \boldsymbol{c}^i\|_2^2 \\ &= \sum_{i=1}^{k} \begin{bmatrix} 2(x_1 - c_1^i) \\ 2(x_2 - c_2^i) \\ \vdots \\ 2(x_d - c_d^i) \end{bmatrix} \\ &= \begin{bmatrix} 2(kx_1 - \sum_{i=1}^{k} c_1^i) \\ 2(kx_2 - \sum_{i=1}^{k} c_2^i) \\ \vdots \\ 2(kx_d - \sum_{i=1}^{k} c_d^i) \end{bmatrix}\end{aligned}$$

Setting this gradient equal to 0 and solving for $\boldsymbol{x}$, we find:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} \frac{1}{k}\sum_{i=1}^{k} c_1^i \\ \frac{1}{k}\sum_{i=1}^{k} c_2^i \\ \vdots \\ \frac{1}{k}\sum_{i=1}^{k} c_n^i \end{bmatrix}$$

Wow! *The optimal placement of the WiFi router is at the average location of all the computers!* This solution is both convenient and intuitive!

But what would happen if we hadn't used the squared Euclidean distance as indicative of speed, but rather, the actual distance, instead? Finding the critical points without squaring the distance would be non-trivial, because the square roots would make it so that the solution requires solving a system of nonlinear equations.

We can solve for the optimal solutions to simple problems analytically, by finding critical points. When this method yields an explicit formula that characterizes an optimal value exactly, as it did for $g(x) = x^4 - 4x^3 + 8x + 1$ and the Router Optimization Problem, we say we have derived a solution in **closed-form**.

Closed-form solutions, however, are not easy to come by in more complex and realistic problems, such as portfolio optimization (find an asset portfolio that maximizes returns) or protein folding (find a protein structure that minimizes free energy). Moreover, even if we could derive such solutions in closed form, there may be exponentially many local optima (e.g., in SAT or other CSPs).[6] Finally, solving problems in closed form may also require too many computational resources (e.g., inverting a very large matrix, an $O(n^3)$ operation).[7] Thus, for many real-world problems of interest, the preferred method is gradient descent.

4.2 Method 2: Gradient Descent

Gradient descent is a form of local search, used to optimize differentiable functions in continuous space. It is an iterative method, which begins at some initial point and then takes steps in a direction that improves that objective value, until a termination condition is met (e.g., the gradient is sufficiently small).

Consider the function $f(\boldsymbol{x}) = {x_1}^2 + 2{x_2}^4$. The gradient $\nabla f(\boldsymbol{x}) = \begin{bmatrix} 2x_1 \\ 8{x_2}^3 \end{bmatrix}$. Thus, at the point $\boldsymbol{x}_0 = (3, -1)$, for example, the gradient vector is $\begin{bmatrix} 6 \\ -8 \end{bmatrix}$. So, if we were to initalize gradient descent at $\boldsymbol{x}_0$ on a graph and travel in the direction of the gradient vector, namely $\begin{bmatrix} 6 \\ -8 \end{bmatrix}$, the value of $f(\boldsymbol{x})$ would *increase*.

Recall that the gradient points in the direction in which a function's value *increases* fastest. More often than not, we are solving minimization problems; that is, we seek to *decrease* a function's value. We can achieve this goal by moving in the *opposite* direction of the gradient, i.e., by following the *negative* of the gradient. Hence, the name **gradient descent**.

Remember, gradients provide only *local* information about a function's behavior, meaning their value at a point only holds in small neighborhoods around that point. As a result, to guarantee that a function's value decreases, gradient descent can only take small steps; that way, it remains in the neighborhood of the point at which the gradient was evaluated. We keep steps small in gradient descent by introducing a so-called **learning rate** $\alpha \in (0, 1]$, which we multiply by the value of the gradient to control the step size: i.e, the rate of movement around the solution space.

N.B. Learning rate is an appropriate name for α, as gradient descent is commonly used in machine learning applications (e.g., in neural networks). A learning rate is an example of a **hyperparameter**. The values of an algorithm's "parameters" (i.e., variables) are learned by the algorithm. Hyperparameters, in contrast, are inputs to the algorithm that are assigned by the programmer, and thus outside the scope of what is learned.

Gradient descent (Algorithm 1) starts at an initial point $\boldsymbol{x}^{(0)}$. This point can be any point in the function's domain, and is often initialized randomly. Then, at each step t, the gradient $\nabla_{\boldsymbol{x}} f$ is evaluated at $\boldsymbol{x}^{(t)}$. This evaluation is expressed mathematically as $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(t)})$. The current point $\boldsymbol{x}^{(t)}$ is then updated to a new point $\boldsymbol{x}^{(t+1)}$ by moving in the direction of the negative gradient at $\boldsymbol{x}^{(t)}$: i.e.,

$$\boldsymbol{x}^{(t+1)} \leftarrow \boldsymbol{x}^{(t)} - \alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(t)})$$

[6] As a simple example of a function with exponentially-many solutions, consider minimizing the XOR function on n bits. This function returns 1 iff the number of bits with value 1 is odd. There are 2^n inputs, half (i.e., 2^{n-1}) of which are optima.

[7] Strassen's algorithm can invert matrices in $O(n^{2.7})$, but it may run into numerical precision issues in practice.

Gradient descent runs until convergence to an optimum; or until a point is reached that is deemed sufficiently close to an optimum; or, as in local search for discrete optimization, until our patience is exhausted.

Algorithm 1 Gradient Descent

Require: objective function $f : \mathbb{R}^d \to \mathbb{R}$, initial point $\boldsymbol{x}^{(0)} \in \mathbb{R}^d$, learning rate $\alpha \in (0, 1]$
Ensure: a local optimum $\boldsymbol{x}^{(t)}$

```
t = 0
repeat
    Evaluate ∇_x f at x^(t)          ▷ This evaluation is written as ∇_x f(x^(t))
    x^(t+1) ← x^(t) − α ∇_x f(x^(t))
    t ← t + 1
until convergence
return x^(t)
```

Example We now demonstrate the inner workings of gradient descent by running through two examples, namely $f(x) = x^2$ and $f(x) = x^4$. For both functions, the unique global minimum is 0. We initialize our starting point to $\boldsymbol{x}^{(0)} = 1$, and try two hyperparameter settings $\alpha = 0.1$ (Table 1) and $\alpha = 0.6$ (Table 2).

For $f(x) = x^2$, when $\alpha = 0.1$, the candidate solution gradually descends from 1 down to 2.5×10^{-10} after 100 iterations. When $\alpha = 0.6$, the step sizes at each iteration are (correspondingly) much larger. Indeed, in the former case, the first iteration descends from 1 to a value of 0.8, while in the latter, the first iteration leads to a much smaller value, namely -0.2. Moreover, when $\alpha = 0.6$, the candidate solutions oscillate around 0 (sometimes $\boldsymbol{x}^{(t)}$ is positive, and sometimes it is negative). Nonetheless, gradient descent still appears to be converging, and at a faster rate than when $\alpha = 0.1$.

	$x^{(t)}$	$f(x^{(t)})$	$\frac{\partial f}{\partial x^t}$	$-\alpha\frac{\partial f}{\partial x^t}$
$x^{(0)}$	1	1	2	-0.2
$x^{(1)}$	0.8	0.64	1.6	-0.16
$x^{(2)}$	.64	0.41	1.28	-0.128
$x^{(3)}$	.512	0.26	1.24	-0.124
$\vdots$				
$x^{(20)}$	0.009	8×10^{-5}	0.018	-0.0018
$\vdots$				
$x^{(100)}$	2.5×10^{-10}	6×10^{-20}	5.0×10^{-10}	-5.0×10^{-11}

Table 1: Gradient descent on $f(x) = x^2$ with $\alpha = 0.1$

	$x^{(t)}$	$f(x^{(t)})$	$\frac{\partial f}{\partial x^t}$	$-\alpha \cdot \frac{\partial f}{\partial x^t}$
$x^{(0)}$	1	1	2	-1.2
$x^{(1)}$	-0.2	0.04	-0.4	0.24
$x^{(2)}$	0.04	0.0016	0.08	-0.048
$x^{(3)}$	-0.008	$6.4 \times 10^{-}5$	-0.016	0.01

Table 2: Gradient descent on $f(x) = x^2$ with $\alpha = 0.6$

Next, we use the same hyperparameters ($\boldsymbol{x}^{(0)} = 1$ and $\alpha = 0.6$), and we apply gradient descent to $f(x) = x^4$. The result is very different (Table 3). Once again, the algorithm oscillates between positive and negative

solutions, but this time, those solutions are moving farther and farther away from one another with each consecutive step. Too large of a learning rate can actually cause gradient descent to *diverge*!

The upshot of this discussion is: we must be cautious when choosing a learning rate. Too large of a choice may preclude us from finding a solution, even when one exists. On the other hand, too small of a choice may eventually lead to convergence, but progress may be painfully slow.

	$x^{(t)}$	$f(x^{(t)})$	$\frac{\partial f}{\partial x^t}$	$-\alpha \cdot \frac{\partial f}{\partial x^t}$
$x^{(0)}$	1	1	4	-2.4
$x^{(1)}$	-1.4	3.84	-10.98	6.58
$x^{(2)}$	5.18	723	557	-334
$x^{(3)}$	-329	1.1×10^{10}	-1.4×10^{8}	-8.6×10^{7}
$\vdots$				

Table 3: Gradient Descent on $f(x) = x^4$ with $\alpha = 0.6$

	$x^{(t)}$	$f(x^{(t)})$	$\frac{\partial f}{\partial x^t}$	$-\alpha \cdot \frac{\partial f}{\partial x^t}$
$x^{(0)}$	1	1	4	-0.4
$x^{(1)}$	0.6	0.1296	0.864	-0.0864
$x^{(2)}$	0.513	0.692	0.54	-0.054
$x^{(3)}$	0.459	0.060	0.387	-0.0387
$\vdots$				
$x^{(100)}$	0.109	0.0001	0.005	-0.0005

Table 4: Gradient Descent on $f(x) = x^4$ with $\alpha = 0.1$

5 Summary

	Analytical Methods	Gradient Descent
Pros	◦ Find closed-form solutions ◦ Can identify global optima by comparing all critical points (when feasible)	◦ Applies to any differentiable function ◦ Scales to large problems ◦ Memory efficient ◦ Flexible & general (and consequently, used extensively in machine learning, especially when training neural networks)
Cons	◦ Limited to specific (usually, simple) problems ◦ May require computationally expensive operations (e.g., inverting large matrices) ◦ Can involve exponentially many equations to solve and check ◦ Many problems do not have straightforward analytical solutions	◦ May only find local solutions ◦ Requires hyperparameter tuning (e.g., a learning rate α) ◦ Iterative process, which may require many iterations

Table 5: Closed-Form Solutions vs. Gradient Descent