Module1-AML

Dr. Linu Pinto
Assistant Professor

Department of Mathematics, CUSAT

August 7, 2024

Basics of Optimization

- An optimization problem has an objective function.
- Optimization Problem is defined in terms of a set of variables, referred to as optimization variables.
- The goal of the optimization problem is to compute the values of the variables at which the objective function is either maximized or minimized.
- It is common to use a minimization form of the objective function in machine learning.
- Corresponding objective function is often referred to as a loss function.
- The term “objective function” is a more general concept than the term “loss function.”
- A loss function is always associated with a minimization objective function, and it is often interpreted as a cost with a nonnegative value.
- Most objective functions in machine learning are multivariate loss functions over many variables.

Example: Predicting House Prices

- We're training a machine learning model to predict house prices based on square footage (X_1) and number of bedrooms (X_2).
- Model:

$$\text{Predicted Price} = \theta_0 + \theta_1 \times X_1 + \theta_2 \times X_2$$

Example: Data and Initial Predictions

- **Data:**
  - House 1: $X_1 = 1000$, $X_2 = 2$, Actual Price = \$300,000
  - House 2: $X_1 = 1500$, $X_2 = 3$, Actual Price = \$400,000
  - House 3: $X_1 = 1200$, $X_2 = 2$, Actual Price = \$350,000
- **Initial Predictions:**
  - House 1: Predicted Price = \$250,000
  - House 2: Predicted Price = \$380,000
  - House 3: Predicted Price = \$320,000

Example: Mean Squared Error (MSE)

- **MSE Formula:**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\hat{y}_i)^2)$$

- **Calculations:**

$$MSE = \frac{1}{3}[(300,000 - 250,000)^2 + (400,000 - 380,000)^2 + (350,000 - 320,000)^2]$$

$$MSE = \frac{1}{3}[(50,000)^2 + (20,000)^2 + (30,000)^2]$$

$$MSE = \frac{1}{3}[2,500,000,000 + 400,000,000 + 900,000,000]$$

$$MSE = \frac{1}{3}[3,800,000,000]$$

$$MSE = 1,266,666,666.67$$

Aggarwal, C. C., Aggarwal, L. F., & Lagerstrom-Fife. (2020). Linear algebra and optimization for machine learning (Vol. 156). Springer International Publishing.

- This value of MSE represents how 'wrong' our initial model's predictions are on average across the dataset.
- The objective during training is to adjust the parameters $\theta_0, \theta_1,$ and θ_2 to minimize this MSE, thereby improving the model's predictive accuracy.
- Here the objective is to minimize the MSE.
- Hence the objective function here is the MSE

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\theta_0 + \theta_1(X_1)_i + \theta_2(X_2)_i))^2 = f(x)$$

which is defined over three variables $\theta_0, \theta_1, \theta_2$. Hence this is a multivariate objective function.

- Consider a single-variable objective function $f(x)$ as follows: $f(x) = x^2 - 2x + 3$
- This objective function is an upright parabola, it clearly takes on its minimum value at $x = 1$
- Note that at the minimum value, the rate of change of $f(x)$ with respect to x is zero, as the tangent to the plot at that point is horizontal.
- One can also find this optimal value by computing the first derivative.
- Intuitively, the function f(x) changes at zero rate on slightly perturbing the value of x from $x = 1$, which suggests that it is an optimal point. However, this analysis alone is not sufficient to conclude that the point is a minimum.
- Consider the inverted parabola $-f(x)$.
- Here $-f(x)$ also changes at zero rate on slightly perturbing the value of x from $x = 1$, which also suggests that it is an optimal point.

- In this case the solution $x = 1$ is a maximum rather than a minimum.
- Furthermore, the point $x = 0$ is an inflection point or saddle point of the function $f(x) = x^3$ even though the derivative is 0 at $x = 0$. Such a point is neither a maximum nor a minimum.
- All points for which the first derivative is zero are referred to as critical points of the optimization problem.
- A critical point might be a maximum, minimum, or saddle point.
- How does one distinguish between the different cases for critical points?
- One observation is that a function looks like an upright bowl at a minimum point, which implies that its first derivative increases at minima.
- In other words, the second derivative (i.e., derivative of the derivative) will be positive for minima (although there are a few exceptions to this rule).

- The case where the second derivative is zero is somewhat ambiguous, because such a point could be a minimum, maximum, or an inflection point. Such a critical point is referred to as degenerate.
- Therefore, for a single-variable optimization function $f(x)$ in minimization form, satisfying both $f'(x) = 0$ and $f''(x) > 0$ is sufficient to ensure that the point is a minimum with respect to its immediate locality. Such a point is referred to as a local minimum. This does not, however, mean that the point x is a global minimum across the entire range of values of x.

Lemma (Lemma 4.2.1)

(Optimality Conditions in Unconstrained Optimization) A univariate function $f(x)$ is a minimum value at $x = x_0$ with respect to its immediate locality if it satisfies both $f'(x_0) = 0$ and $f''(x_0) > 0$.

- These conditions are referred to as first-order and second-order conditions for minimization. The conditions $f'(x_0) = 0$ and $f''(x_0) > 0$ are sufficient for a point to be minimum with respect to its infinitesimal locality, and they are "almost" necessary for the point to be a minimum with respect to its locality.
- We use the word "almost" in order to address the degenerate case where a point x_0 might satisfy $f'(x_0) = 0$ and $f''(x_0) = 0$.
- As an example of this ambiguity consider the functions $F(x) = x^3$ and $G(x) = x^4$.
- Analyze these functions using taylor series expansion within a small locality $x_0 + \Delta$

$$f(x_0 + \Delta) \approx f(x_0) + \Delta f'(x_0) + \frac{\Delta^2}{2} f''(x_0)$$

- Note that Δ might be either positive or negative, although Δ^2 will always be positive.
- The value of $|\Delta|$ is assumed to be extremely small, and successive terms rapidly drop off in magnitude. Therefore, it makes sense to keep only the first non-zero term in the above expansion in order to meaningfully compare $f(x_0)$ with $f(x_0 + \Delta)$. Since $f'(x_0)$ is zero, the first non-zero term is the second-order term containing $f''(x_0)$.
- Furthermore, since Δ^2 and $f''(x_0)$ are positive, it follows that $f(x_0 + \Delta) = f(x_0) + \epsilon$, where ϵ is some positive quantity.
- This means that $f(x_0)$ is less than $f(x_0 + \Delta)$ for any small value of Δ, whether it is positive or negative. In other words, x_0 is a minimum with respect to its immediate locality.
- The Taylor expansion also provides insights as to why the degenerate case $f'(x_0) = f''(x_0) = 0$ is problematic.

- In the event that $f''(x)$ is zero, one would need to keep expanding the Taylor series until one reaches the first non-zero term. If the first non-zero term is positive, then one can show that $f(x_0 + \Delta) < f(x_0)$.
- An example of such a function is $f(x) = x^4$ at $x_0 = 0$. In such a case, x_0 is indeed a minimum with respect to its immediate locality. However, if the first non-zero term is negative or it depends on the sign of Δ, it could be a maximum or saddle point.
- A quadratic function is a rather simple case in which a single minimum or maximum exists, depending on the sign of the quadratic term.
- However, other functions have multiple turning points. For example, the function $\sin(x)$ is periodic, and has an infinite number of minima/maxima over $x \in (-\infty, +\infty)$.

- Optimality conditions of Lemma 4.2.1 only focus on defining a minimum in a local sense. In other words, the point is a minimum with respect to its infinitesimal locality.
- A point that is a minimum only with respect to its immediate locality is referred to as a local minimum.
- Intuitively, the word “local” refers to the fact that the point is a minimum only within its neighborhood of (potentially) infinitesimal size.
- The minimum across the entire domain of values of the optimization variable is the global minimum. It is noteworthy that the conditions of above lemma do not tell us with certainty whether or not a point is a global minimum.
- However, these conditions are sufficient for a point to be at least a local minimum and “almost” necessary to be a local minimum (i.e., necessary with the exception of the degenerate case discussed earlier with a zero second derivative).

- Consider an objective function:

$$F(x) = \frac{x^4}{4} - \frac{x^3}{3} - x^2 + 2$$

- Consider an objective function:

$$F(x) = \frac{x^4}{4} - \frac{x^3}{3} - x^2 + 2$$

On differentiating $F(x)$ with respect to x and setting it to zero, we obtain the following condition:

$$x^3 - x^2 - 2x = x(x+1)(x-2) = 0$$

The roots are $x \in \{-1, 0, 2\}$. The second derivative is $3x^2 - 2x - 2$, which is positive at -1 and 2 (minima), and negative at $x = 0$ (maximum). The value of the function at the two minima are as follows:

$$F(-1) = \frac{1}{4} + \frac{1}{3} - 1 + 2 = \frac{19}{12}$$

$$F(2) = 4 - \frac{8}{3} - 4 + 2 = -\frac{2}{3}$$

- Local optima pose a challenge for opti mization problems, because there is often no way of knowing whether a solution satisfying the optimality conditions is the global optimum or not.
- Certain types of optimization functions, referred to as convex functions, are guaranteed to have a single local minimum which is a global minimum.
- It is often difficult to exactly solve the equation $f(x) = 0$ because this derivative might itself be a complex function of x.
- In other words, a closed form solution (like the example above) typically does not exist.
- For example, consider the following function that needs to be minimized:

  $$f(x) = x^2 \cdot \log_e(x) - x$$

  Setting the first derivative of this function to 0 yields the following condition:

  $$f(x) = 2x \cdot \log_e(x) + x - 1 = 0$$

  Unfortunately, it is not always possible to compute such analytical solutions in closed form.

- This equation is somewhat hard to solve, although iterative methods exist for solving it. By trial and error, one might get lucky and find out that $x = 1$ is indeed a solution to the first derivative equation. Solving the equation $f(x) = 0$ for x provides an analytical solution for a critical point.
- $$f'(1) = 2\log_e(1) + 1 - 1 = 0.$$
- Furthermore, the second derivative $f''(x)$ can be shown to be positive at $x = 1$, and therefore this point is at least a local minimum.
- However, solving an equation like this numerically causes all types of numerical and computational challenges; these types of challenges increase when we move from univariate optimization to multivariate optimization.

- A very popular approach for optimizing objective functions (irrespective of their functional form) is to use the method of gradient descent. In gradient descent, one starts at an initial point $x = x_0$ and successively updates x using the steepest descent direction:

$$x \leftarrow x - \alpha f'(x)$$

- Here, $\alpha > 0$ regulates the step size, and is also referred to as the learning rate.
- In the univariate case, the notion of "steepest" is hard to appreciate, as there are only two directions of movement (i.e., increase x or decrease x).
- One of these directions causes ascent, whereas the other causes descent.
- However, in multivariate problems, there can be an infinite number of possible directions of descent, and the generalization of the notion of univariate derivative leads to the steepest descent direction.

- Consider the taylor series expansion within a small locality x

$$f(x+\delta x) \approx f(x) + \delta x\ f'(x) + \frac{\delta x^2}{2} f''(x)$$

- After Truncating and consider only the linear approximation

$$f(x+\delta x) \approx f(x) + \delta x\ f'(x)$$

- The value of x changes in each iteration by $\delta x = -\alpha f'(x)$. Note that at infinitesimally small values of the learning rate $\alpha > 0$, the update rule becomes:

$$x \leftarrow x - \alpha f'(x)$$

- The above updates will always reduce $f(x)$. This is because for very small α, we can use the first-order Taylor expansion to obtain the following:

$$f(x+\delta x) \approx f(x) + \delta x f'(x) = f(x) - \alpha [f'(x)]^2 < f(x)$$

- Consider the function

$$f(x) = x^2 \cdot \log_e(x) - x$$

- Consider the case where we start at $x_0 = 2$, which is larger than the optimal value of $x = 1$.

- Consider the function

$$f(x) = x^2 \cdot \log_e(x) - x$$

- Consider the case where we start at $x_0 = 2$, which is larger than the optimal value of $x = 1$.
- At this point, the value of $f'(x)$ can be shown to be $2\log_e(2) + 1 \approx 2.4$. If we use $\alpha = 0.2$, then the value of x gets updated from x_0 as follows:

$$x_1 \leftarrow x_0 - 0.2 \cdot 2.4 = 2 - 0.48 = 1.52$$

- This new value of x is closer to the optimal solution. One can then recompute the derivative at $x_1 = 1.52$ and perform the update $x \leftarrow 1.52 - 0.2 \cdot f'(1.52)$.
- Performing this update again and again to construct the sequence $x_0, x_1, x_2, \ldots, x_t$ will eventually converge to the optimal value of $x_t = 1$ for large values of t.

- Note that the choice of α does matter. For example, if we choose $\alpha = 0.8$, then it results in the following update:

$$x_1 \leftarrow x_0 - \alpha f'(x_0) = 2 - 2.4 \cdot 0.8 = 0.08$$

- In this case, the solution has overshot the optimal value of $x = 1$, although it is still closer to the optimal solution than the initial point of $x_0 = 2$. The solution can still be shown to converge to an optimal value, but after a longer time.
- The execution of gradient-descent updates will generally result in a sequence of values $x_0, x_1, \ldots, x_t$ of the optimization variable, which become successively closer to an optimum solution.
- As the value of x_t nears the optimum value, the derivative $f'(x_t)$ also tends to be closer and closer to zero (thereby satisfying the first-order optimality conditions of Lemma 4.2.1).
- In other words, the absolute step size will tend to reduce over the execution of the algorithm.

- As gradient descent nears an optimal solution, the objective function will also improve at a slower rate.
- This observation provides some natural ideas on making decisions regarding the termination of the algorithm (when the current solution is sufficiently close to an optimal value).
- The idea is to plot the current value of $f(x_t)$ with iteration index t as the algorithm progresses.
- The objective function value need not be monotonically decreasing over the course of the algorithm, but it will tend to show small noisy changes (without significant long-term direction) after some point.
- This situation can be treated as a good termination point for the algorithm. However, in some cases, the update steps can be shown to diverge from an optimal solution, if the step size is not chosen properly.

- Choosing a very large learning rate α can cause overshooting from the optimal solution, and even divergence in some cases.
- consider the quadratic function $f(x)$ of Figure 4.2(a), which takes on its optimal value at $x = 1$:

$$f(x) = x^2 - 2x + 3$$

- Now imagine a situation where the starting point is $x_0 = 2$, and one chooses a large learning rate $\alpha = 10$.

- Choosing a very large learning rate α can cause overshooting from the optimal solution, and even divergence in some cases.
- consider the quadratic function $f(x)$ of Figure 4.2(a), which takes on its optimal value at $x = 1$:

$$f(x) = x^2 - 2x + 3$$

- Now imagine a situation where the starting point is $x_0 = 2$, and one chooses a large learning rate $\alpha = 10$.
- The derivative of $f(x) = 2x - 2$ evaluates to $f'(x_0) = f'(2) = 2$. Then, the update from the first step yields the following:

$$x_1 \leftarrow x_0 - 10 \cdot 2 = 2 - 20 = -18$$

- Note that the new point x_1 is much further away from the optimal value of $x = 1$, which is caused by the overshooting problem.
- Even worse, the absolute gradient is very large at this point, and it evaluates to $f'(-18) = -38$. If we keep the learning rate fixed, it will cause the solution to move at an even faster rate in the opposite direction:

$$x_2 \leftarrow x_1 - 10 \cdot (-38) = -18 + 380 = 362$$

- In this case, the solution has overshot back in the original direction but is even further away from the optimal solution. Further updates cause back-and-forth movements at increasingly large amplitudes:

$$x_3 \leftarrow x_2 - 10 \cdot 722 = 362 - 7220 = -6858, \quad x_4 \leftarrow x_3 + 10 \cdot 13718 = 130322$$

- Note that each iteration flips the sign of the current solution and increases its magnitude by a factor of about 20. In other words, the solution moves away faster and faster from an optimal solution until it leads to a numerical overflow.
- It is common to reduce the learning rate over the course of the algorithm, and one of the many purposes served by such an approach is to arrest divergence; however, in some cases, such an approach might not prevent divergence, especially if the initial learning rate is large.
- Therefore, when an analyst encounters a situation in gradient descent, where the size of the parameter vector seems to increase rapidly (and the optimization objective worsens),it is a tell-tale sign of divergence.

- The first adjustment should be to experiment with a lower initial learning rate. However, choosing a learning rate that is too small might lead to unnecessarily slow progress, which causes the entire procedure to take too much time.
- There is a considerable literature in finding the correct step size or adjusting it over the course of the algorithm.

- The univariate optimization is rather unrealistic, and most optimization problems in real-world settings have multiple variables.
- we will first consider the case of an optimization function containing two variables called bivariate optimization
- Let us consider bivariate generalizations of the univariate optimization functions
- We construct bivariate functions by adding two instances of the univariate function

  $$f(x) = x^2 - 2x + 3$$

  and

  $$g(x) = \frac{x^4}{4} - \frac{x^3}{3} - x^2 + 2$$

  as follows:

  $$g(x, y) = f(x) + f(y) = x^2 + y^2 - 2x - 2y + 6$$

  $$G(x, y) = g(x) + g(y) = \frac{x^4 + y^4}{4} - \frac{x^3 + y^3}{3} - x^2 - y^2 + 4$$
- Note that these functions are simplified and have very special structure

- They are additively separable.
  (Additively Separable Functions) A function $F(x_1, x_2, \ldots, x_d)$ in d variables is said to be additively separable, if it can be expressed in the following form for appropriately chosen univariate functions $f_1(\cdot), f_2(\cdot), \ldots, f_d(\cdot)$:

$$F(x_1, x_2, \ldots, x_d) = \sum_{i=1}^{d} f_i(x_i)$$

- Additively separable functions are those in which univariate terms are added, and they do not interact with one another. In other words, an additively separable function might contain terms like $\sin(x^2)$ and $\sin(y^2)$, but not $\sin(xy)$.
- Consider the quadratic function $f(x) = x^T Ax + b^T x + c$ (Any multivariable quadratic function can be expressed in this form).
- Unless the symmetric matrix A is diagonal, the resulting function contains terms of the form $x_i x_j$. Such terms are referred to as interacting terms. Most real-world quadratic functions contain such terms.

- It is noteworthy that any multivariate quadratic function can be transformed to an additively separable function (without interacting terms) by basis transformation of the input variables of the function.**(Assignment 1)**

- It is noteworthy that any multivariate quadratic function can be transformed to an additively separable function (without interacting terms) by basis transformation of the input variables of the function.**(Assignment 1)**
- **Hint:**
  - In matrix notation, any multivariate quadratic function can be compactly written as:

$$Q(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$$

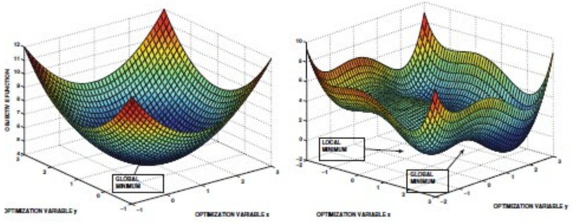
    where:
    $\mathbf{x}$ is the vector of variables $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T$.
    $\mathbf{A}$ is an $n \times n$ symmetric matrix of coefficients $[a_{ij}]$.
    $\mathbf{b}$ is the vector of linear coefficients $[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]^T$.
    c is the scalar constant term.

- Using Basis transformation is it possible to remove the interacting terms.
- Is the representation of coordinates unique for a particular basis transformation (Change of Basis theorem).
- Only if A is diagonalizable that interacting terms can be removed in new basis representation.
- Since A is a symmetric matrices, is all symmetric matrices diagonalizable.

- Nevertheless, these simplified polynomial functions are adequate for demonstrating the complexities associated with multivariable optimization.
- Can all quadratic functions can be represented in additively separable form (although this is not true for non-quadratic functions).
- Why additively separable functions are important in Optimization!!
- Additively separable functions are much easier to optimize, because one can decompose the optimization problem into smaller optimization problems on individual variables.
- For example,a multivariate quadratic function would appear as a simple sum of univariate quadratic functions (each of which is extremely simple to optimize). How? **(Assignment 1)**

Visual Implication of additively separable property of multivariate quadratic functions

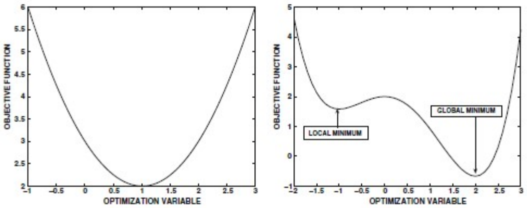
- The two bivariate functions $g(x, y)$ and $G(x, y)$ are shown in Figure 4.4(a) and (b) respectively. It is evident that the single-variable cross-sections of the objective functions in Figure 4.4(a) and (b) are similar to the 1-dimensional functions in Figure 4.2(a) and (b).
- Analyze local maxima,minima and global maxima and minima.



(a) Single global minimum
$g(x, y) = x^2 + y^2 - 2x - 2y + 6$

(b) Global and local minimum
$G(x, y) = ([x^4 + y^4]/4)$
$-([x^3 + y^3]/3) - x^2 - y^2 + 4$

Figure: Fig: 4.4



(a) Single global minimum
$f(x) = x^2 - 2x + 3$

(b) Global and local minimum
$F(x) = (x^4/4) - (x^3/3) - x^2 + 2$

Figure: Fig: 4.2

- In this case, one can compute the partial derivative of the objective functions $g(x, y)$ and $G(x, y)$ in order to perform gradient descent.
- One can compute the gradient of the function $g(x, y)$ in Figure 4.4(a) as follows:

$$\nabla g(x, y) = \begin{pmatrix} \frac{\partial g(x,y)}{\partial x} \\ \frac{\partial g(x,y)}{\partial y} \end{pmatrix}^T = \begin{pmatrix} 2x - 2 \\ 2y - 2 \end{pmatrix}$$

- The notation $\nabla_{x,y} g(x, y)$ clarifies the choice of variables with respect to which the gradient is computed.
- In this case, the gradient is a column vector with two components, because we have two optimization variables x and y.
- Each component of the 2-dimensional vector is a partial derivative of the objective function with respect to one of the two variables.

- The simplest approach for Solving the optimization problem is to set the gradient $\nabla g(x, y)$ to zero, which leads to the solution $[x, y] = [1, 1]$.
- This might not always lead to a system of equations with a closed-form solution.
- Closed form solution is a solution that provides a direct way to compute the solution without the need for iterative or numerical methods.
- The common solution is to use gradient-descent updates with respect to the optimization variables $[x, y]$ as follows:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} \leftarrow \begin{bmatrix} x_t \\ y_t \end{bmatrix} - \alpha \nabla g(x_t, y_t) = \begin{bmatrix} x_t - \alpha(2x_t - 2) \\ y_t - \alpha(2y_t - 2) \end{bmatrix}$$

- So far, we have only examined additively separable functions with simple structure.
- Now let us consider a somewhat more complicated function:

$$H(x, y) = x^2 - \sin(xy) + y^2 - 2x$$

- This has an interacting term

$$sin(xy)$$

  and also H is not a quadratic function
- The gradient of the function can be shown to be the following:

$$\nabla H(x, y) = \begin{bmatrix} \frac{\partial H(x,y)}{\partial x} \\ \frac{\partial H(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x - y\cos(xy) - 2 \\ 2y - x\cos(xy) \end{bmatrix}$$

- Although the partial derivative components are no longer expressed in terms of individual variables, gradient descent updates can be performed in a similar manner to the previous case.

- In the case of the function $G(x, y)$ shown in Figure 4.4(b), local optima are clearly visible. All critical points can be found by setting the gradient $\nabla G(x, y)$ to 0:
- $$\nabla G(x, y) = \begin{bmatrix} \frac{\partial G(x,y)}{\partial x} \\ \frac{\partial G(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} x^3 - x^2 - 2x \\ y^3 - y^2 - 2y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
- This optimization problem has an interesting structure, because any of the nine pairs $(x, y) \in \{-1, 0, 2\} \times \{-1, 0, 2\}$ satisfies the first order optimality conditions, and are therefore critical points.
- Among these, there is a single global minimum, three local minima, and a single local maximum at $(0, 0)$.
- The other four can be shown to be saddle points. The classification of points as minima, maxima, or saddle points can only be accomplished with the use of multivariate second-order conditions, which are direct generalizations of the univariate optimality conditions.

- In optimization problems where the objective function is expressed as the sum of functions, each depending on a single variable, the number of local optima (solutions where the gradient is zero but not necessarily global optima) increases very rapidly as the number of variables increases.
- **Sum of Univariate Functions**: When the objective function can be written as a sum of functions, each depending on only one variable, like $f(x_1, x_2, \ldots, x_n) = g_1(x_1) + g_2(x_2) + \cdots + g_n(x_n)$.
- **Local Optima**: These are points in the domain of the objective function where the function value is lower (for a minimum) or higher (for a maximum) than in the immediate surrounding points, but not necessarily the lowest or highest overall.
- **Proliferate Exponentially**: The number of these local optima grows very rapidly, following an exponential function. If each univariate function $g_i(x_i)$ has k local optima, then for n variables, the total number of local optima can be k^n.

- **Example**:
  Consider an optimization problem with 3 variables x, y, z, where the objective function is $f(x, y, z) = g_1(x) + g_2(y) + g_3(z)$.
  - If each function g_1, g_2, g_3 has 3 local optima, the total number of local optima for $f(x, y, z)$ is $3 \times 3 \times 3 = 3^3 = 27$.
  - For n variables, if each g_i has k local optima, the total number of local optima is k^n.
- This rapid increase in the number of local optima makes finding the global optimum more challenging as the number of variables increases.

Problem

Consider a univariate function $f(x)$, which has k values of x satisfying the optimality condition $f'(x) = 0$. Let $G(x, y) = f(x) + f(y)$ be a bivariate objective function. Show that there are k^2 pairs (x, y) satisfying $\nabla G(x, y) = 0$. How many tuples $[x_1, \ldots, x_d]^T$ would satisfy the first-order optimality condition for the d-dimensional function $H(x_1, \ldots, x_d) = \sum_{i=1}^{d} f(x_i)$?

Hint

- Given the bivariate function $G(x, y) = f(x) + f(y)$, the gradient $\nabla G(x, y)$ is given by:

$$\nabla G(x, y) = \left(\frac{\partial G(x, y)}{\partial x}, \frac{\partial G(x, y)}{\partial y} \right) = (f'(x), f'(y))$$

  For $\nabla G(x, y) = 0$, we require:

$$f'(x) = 0 \quad \text{and} \quad f'(y) = 0$$

- $f(x)$ has k values of x that satisfy $f'(x) = 0$ and $f(y)$ has k values of y that satisfy $f'(y) = 0$
- Therefore, the number of pairs (x, y) that satisfy $\nabla G(x, y) = 0$ is:

$$k \times k = k^2$$

- Similar case for the function H.

- At higher learning rates, it is possible for the gradient descent to overshoot a local/global optimum and move to a different bowl (or even behave in an unpredictable way with numerical overflows).
- Therefore, the final resting point of gradient descent depends on small details of the computational procedure, such as the starting point or the learning rate.

Multivariate Optimization

- Most machine learning problems are defined on a large parameter space containing multiple optimization variables.
- For example, in a linear regression problem, the optimization variables $w_1, w_2, \ldots, w_d$ are used to predict the dependent variable y from the independent variables $x_1, \ldots, x_d$ as follows:

$$y = \sum_{i=1}^{d} w_i x_i$$

- Starting from this section, we assume that only the notations $w_1, \ldots, w_d$ represent optimization variables, whereas the other "variables" like x_i and y are really observed values from the data set at hand (which are constants from the optimization perspective). This notation is typical for machine learning problems.

- For example, if we have many observed tuples of the form $[x_1, x_2, \ldots, x_d, y]$, one can sum up the values of $(y - \sum_{i=1}^{d} w_i x_i)^2$ over all the observed tuples. Such objective functions are often referred to as loss functions in machine learning.
- we will assume that the loss function $J(w)$ is a function of a vector of multiple optimization variables $w = [w_1, \ldots, w_d]^T$.
- we will use the notations $w_1, \ldots, w_d$ for optimization variables, because the notations $\overline{X}, x_i, \overline{y}$, and y_i will be reserved for the attributes in the data (whose values are observed).
- The value of d corresponds to the number of optimization variables in the problem, and the parameter vector $w = [w_1, \ldots, w_d]^T$ is assumed to be a column vector.

- The computation of the gradient of an objective function of d variables is similar to the bivariate case discussed above.
- The main difference is that a d-dimensional vector of partial derivatives is computed instead of a 2-dimensional vector.
- The i-th component of the d-dimensional gradient vector is the partial derivative of J with respect to the i-th parameter w_i.
- The simplest approach to solve the optimization problem directly (without gradient descent) is to set the gradient vector to zero, which leads to the following set of d conditions:

$$\frac{\partial J(\overline{w})}{\partial w_i} = 0, \quad \forall i \in \{1, \ldots, d\}$$

- These conditions lead to a system of d equations, which can be solved to determine the parameters $w_1, \ldots, w_d$.
- Then we characterize whether a critical point (i.e., zero-gradient point) is a maximum, minimum, or inflection point.

- In single-variable optimization, the condition for $f''(w)$ to be a minimum is $f''(w) > 0$.
- In multivariate optimization, this principle is generalized with the use of the Hessian matrix.
- Instead of a scalar second derivative, we have a $d \times d$ matrix of second derivatives, which includes pairwise derivatives of J with respect to different pairs of variables.
- The Hessian of the loss function $J(w)$ with respect to the optimization variables $w_1, \ldots, w_d$ is given by a $d \times d$ symmetric matrix H, in which the (i, j)-th entry H_{ij} is defined as follows:

$$H_{ij} = \frac{\partial^2 J(\mathbf{w})}{\partial w_i \partial w_j} \tag{4.7}$$

- $$H_{ij} = \frac{\partial^2 J(w)}{\partial w_i \partial w_j}$$
- Here note that the (i, j)-th entry of the Hessian is equal to the (j, i)-th entry because partial derivatives are commutative.

Theorem (Schwarz's Theorem (Clairaut's Theorem))

Let f be a real-valued function of two variables, $f(x, y)$. If the second mixed partial derivatives of f are continuous at a point (x_0, y_0), then the mixed partial derivatives are equal at that point. That is,

$$\frac{\partial^2 f}{\partial x \partial y}(x_0, y_0) = \frac{\partial^2 f}{\partial y \partial x}(x_0, y_0).$$

- Hence by Schwarz's Theorem Hessian matrix is a symmetric matrix and is helpful in many computational algorithms that require eigendecomposition of the matrix.

- The Hessian matrix is a direct generalization of the univariate second derivative $f''(w)$. For a univariate function, the Hessian is a 1×1 matrix containing $f''(w)$ as its only entry.
- Hessian is a function of $\mathbf{w}$, and should be denoted by $H(\mathbf{w})$.
- In the event that the function $J(\mathbf{w})$ is quadratic, the entries in the Hessian matrix do not depend on the parameter vector $\mathbf{w} = [w_1, \ldots, w_d]^T$.
- This is similar to the univariate case, where the second derivative $f''(w)$ is a constant when the function $f(w)$ is quadratic.
- In general, however, the Hessian matrix depends on the value of the parameter vector $\mathbf{w}$ at which it is computed.
- For a parameter vector $\mathbf{w}$ at which the gradient is zero (i.e., critical point), one needs to test the Hessian matrix H in the same way we test $f''(w)$ in univariate functions.
- Just as $f''(w)$ needs to be positive for a point w to be a minimum, the Hessian matrix H needs to be positive-definite for a point to be guaranteed to be a minimum.

- In order to understand this point, we consider the second-order, multivariate Taylor expansion of $J(\mathbf{w})$ in the immediate locality of $\mathbf{w}_0$ along the direction $\mathbf{v}$ and small radius $\epsilon > 0$:

$$J(\mathbf{w}_0 + \epsilon\mathbf{v}) \approx J(\mathbf{w}_0) + \epsilon\mathbf{v}^T[\nabla J(\mathbf{w}_0)] + \frac{\epsilon^2}{2}\mathbf{v}^T H\mathbf{v}$$

- The Hessian matrix H, which depends on the parameter vector, is computed at $\mathbf{w} = \mathbf{w}_0$.
- It is evident that the objective function $J(\mathbf{w}_0)$ will be less than $J(\mathbf{w}_0 + \epsilon \mathbf{v})$ when we have $\mathbf{v}^T H \mathbf{v} > 0$.
- If we can find even a single direction $\mathbf{v}$ where we have $\mathbf{v}^T H \mathbf{v} < 0$, then $\mathbf{w}$ is clearly not a minimum with respect to its immediate locality.
- A matrix H that satisfies $\mathbf{v}^T H \mathbf{v} > 0$ is positive definite.
- The notion of positive definiteness of the Hessian is the direct generalization of the second-derivative condition $f''(w) > 0$ for univariate functions.
- After all, the Hessian of a univariate function is a 1×1 matrix containing the second derivative. The single entry in this matrix needs to be positive for this 1×1 matrix to be positive-definite.

- Assuming that the gradient is zero at the critical point **w**, we can summarize the following second-order optimality conditions:
  1. If the Hessian is positive definite at $\mathbf{w} = [w_1, \ldots, w_d]^T$, then **w** is a local minimum.
  2. If the Hessian is negative definite at $\mathbf{w} = [w_1, \ldots, w_d]^T$, then **w** is a local maximum.
  3. If the Hessian is indefinite at **w**, then **w** is a saddle point.
  4. If the Hessian is positive- or negative semi-definite, then the test is inconclusive, because the point could either be a local optimum or a saddle point.

  Consider the following optimization objective function $g(w_1, w_2) = w_1^2 - w_2$. The Hessian of this quadratic function is independent of the parameter vector $[w_1, w_2]^T$, and is defined as follows:

$$\begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$$

- This Hessian turns out to be a diagonal matrix, which is clearly indefinite because one of the two diagonal entries is negative.
- The point $[0, 0]$ is a critical point because the gradient is zero at that point. However, this point is a saddle point because of the indefinite nature of the Hessian matrix.

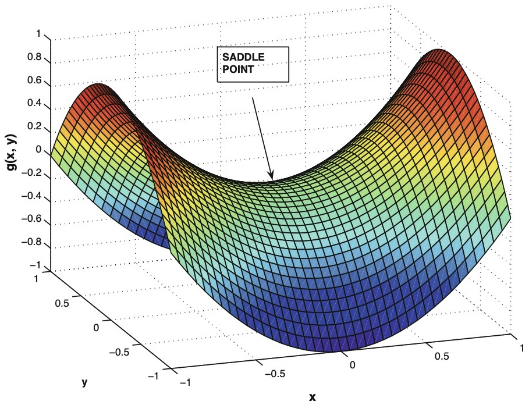


Figure: Fig: 4.5

- Setting the gradient of the objective function to 0 and then solving the resulting system of equations is usually computationally difficult. Therefore, gradient-descent is used.

$$\begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix} \Leftarrow \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial J(\overline{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\overline{w})}{\partial w_d} \end{bmatrix}$$

- One can also write the above expression in terms of the gradient of the objective function with respect to $\overline{w}$:

$$\overline{w} \Leftarrow \overline{w} - \alpha \nabla J(\overline{w})$$

  Here, $\nabla J(\overline{w})$ is a column vector containing the partial derivatives of $J(\overline{w})$ with respect to the different parameters in the column vector $\overline{w}$. Although the learning rate α is shown as a constant here, it usually varies over the course of the algorithm.

Convex Objective Functions

- The presence of local minima creates uncertainty about the effectiveness of gradient-descent algorithms. Ideally, one would like to have an objective function without local minima. A specific type of objective function with this property is the class of convex functions.

Definition

Definition 4.3.1 (Convex Set) A set S is convex, if for every pair of points $\overline{w}_1, \overline{w}_2 \in S$, the point $\lambda\overline{w}_1 + (1-\lambda)\overline{w}_2$ must also be in S for all $\lambda \in (0, 1)$.

Definition

A convex function $F(\overline{w})$ is defined as a function with a convex domain that satisfies the following condition for any $\lambda \in (0, 1)$:

$$F(\lambda\overline{w}_1 + (1-\lambda)\overline{w}_2) \leq \lambda F(\overline{w}_1) + (1-\lambda)F(\overline{w}_2)$$

Problem

Problem 4.3.1 *For a convex function $F(\cdot)$, and k parameter vectors $w_1, \ldots, w_k$, show that the following is true for any $\lambda_1, \ldots, \lambda_k \geq 0$ and satisfying $\sum_{i=1}^{k} \lambda_i = 1$:*

$$F\left(\sum_{i=1}^{k} \lambda_i w_i\right) \leq \sum_{i=1}^{k} \lambda_i F(w_i)$$

Hint:

1. By Induction
2. $\lambda_{n+1} = \alpha$
3. Let $\mu_i = \frac{\lambda_i}{1-\alpha}$

Some Properties of Convex functions

Lemma

1. *A linear function of the vector* $\mathbf{w}$ *is always convex.*
2. *The sum of convex functions is always convex.*
3. *The maximum of convex functions is convex.*
4. *The square of a nonnegative convex function is convex.*
5. *If* $F(\cdot)$ *is a convex function with a single argument and* $G(\mathbf{w})$ *is a linear function with a scalar output, then* $F(G(\mathbf{w}))$ *is convex. (**Composition of a convex univariate function with a linear multivariate function preserves convexity**.)*
6. *If* $F(\cdot)$ *is a convex non-increasing univariate function and* $G(\mathbf{w})$ *is a concave multivariate function with a scalar output, then* $F(G(\mathbf{w}))$ *is convex.*
7. *If* $F(\cdot)$ *is a convex non-decreasing univariate function and* $G(\mathbf{w})$ *is a convex multivariate function with a scalar output, then* $F(G(\mathbf{w}))$ *is convex.*

Proof Hints to the Lemma

- A linear function of a vector $\mathbf{w}$ often takes the form of a dot product or a linear transformation. For example, if $\mathbf{w} \in \mathbb{R}^n$ and $\mathbf{a} \in \mathbb{R}^n$, a typical linear function $G(\mathbf{w})$ could be represented as:

$$G(\mathbf{w}) = \mathbf{a}^T\mathbf{w} = \sum_{i=1}^{n} a_i w_i$$

- The function f is concave if the Hessian matrix $H_f(\mathbf{x})$ is negative semi-definite for all $\mathbf{x}$ in the domain. This means that for any vector $\mathbf{z}$, the quadratic form $\mathbf{z}^T H_f(\mathbf{x})\mathbf{z} \leq 0$.
- The function f is convex if the Hessian matrix $H_f(\mathbf{x})$ is positive semi-definite for all $\mathbf{x}$ in the domain. This means that for any vector $\mathbf{z}$, the quadratic form $\mathbf{z}^T H_f(\mathbf{x})\mathbf{z} \geq 0$.
- The function f is non-decreasing if for any $x_1 \leq x_2$, we have $f(x_1) \leq f(x_2)$.
- The function f is non-decreasing if for any $x_1 \leq x_2$, we have $f(x_1) \geq f(x_2)$.

Definition

(First-Derivative Characterization of Convexity)
A differentiable function $F(\mathbf{w})$ is a convex function if and only if the following is true for any pair $\mathbf{w}_0$ and $\mathbf{w}$:

$$F(\mathbf{w}) \geq F(\mathbf{w}_0) + \nabla F(\mathbf{w}_0) \cdot (\mathbf{w} - \mathbf{w}_0)$$

- Note that if the gradient of $F(\mathbf{w})$ is zero at $\mathbf{w} = \mathbf{w}_0$, it would imply that $F(\mathbf{w}) \geq F(\mathbf{w}_0)$ for any $\mathbf{w}$. In other words, $\mathbf{w}_0$ is a global minimum.
- Therefore, any critical point that satisfies the first-derivative condition is a global minimum.
- The main disadvantage of the first-derivative condition (with respect to the direct definition of convexity) is that it applies only to differentiable functions.
  Interestingly, there is a third characterization of convexity in terms of the second derivative:

Definition

(Second-Derivative Characterization of Convexity)
The twice differentiable function $F(\mathbf{w})$ is convex, if and only if it has a positive semidefinite Hessian at every value of the parameter $\mathbf{w}$ in the domain of $F(\cdot)$.

The second derivative condition has the disadvantage of requiring the function $F(\mathbf{w})$ to be twice differentiable. Therefore, the following convexity definitions are equivalent for twice-differentiable functions defined over $\mathbb{R}^d$:

1. **Direct:** The convexity condition $F(\lambda \mathbf{w}_1 + [1-\lambda]\mathbf{w}_2) \leq \lambda F(\mathbf{w}_1) + (1-\lambda)F(\mathbf{w}_2)$ is satisfied for all $\mathbf{w}_1, \mathbf{w}_2$ and $\lambda \in (0,1)$.
2. **First-derivative:** The first-derivative condition $F(\mathbf{w}) \geq F(\mathbf{w}_0) + \nabla F(\mathbf{w}_0) \cdot (\mathbf{w} - \mathbf{w}_0)$ is satisfied for all $\mathbf{w}$ and $\mathbf{w}_0$.
3. **Second-derivative:** The Hessian of $F(\mathbf{w})$ is positive semidefinite for all $\mathbf{w}$.

- Many machine learning objective functions are of the form $F(G(\mathbf{w}))$, where $G(\mathbf{w})$ is the linear function $\mathbf{w} \cdot \mathbf{X}^T$ for a row vector containing a d-dimensional data point $\mathbf{X}$, and $F(\cdot)$ is a univariate function.
- In such a case, one only needs to prove that the univariate function $F(\cdot)$ is convex.
- It is particularly easy to use the second-order condition $F''(\cdot) \geq 0$ for univariate functions.

Problem

Use the second derivative condition to show that the univariate function

$$F(x) = \log_e(1 + \exp(-x))$$

is convex.

Problem

Use the second derivative condition to show that if the univariate function $F(x)$ is convex, then the function

$$G(x) = F(-x)$$

must be convex as well.

- A slightly stronger condition than convexity is strict convexity in which the convexity condition is modified to strict inequality.
- A strictly convex function $F(w)$ is defined as a function that satisfies the following condition for any $\lambda \in (0, 1)$:

$$F(\lambda w_1 + (1 - \lambda)w_2) < \lambda F(w_1) + (1 - \lambda)F(w_2)$$

- A bowl with a flat bottom is convex, but it is not strictly convex. A strictly convex function will have a unique global minimum.
- One can also adapt the first-order conditions to strictly convex functions. A function $F(\cdot)$ can be shown to be strictly convex if and only if the following condition holds for all w and w_0:

$$F(w) > F(w_0) + [\nabla F(w_0)] \cdot (w - w_0)$$

- The second-derivative condition cannot, however, be fully generalized to strict convexity.
- If a function has a positive definite Hessian everywhere, then it is guaranteed to be strictly convex.

- However, the converse does not necessarily hold. For example, the function $f(x) = x^4$ is strictly convex, but its second derivative is 0 at $x = 0$. An important property of strictly convex functions is the following:
  Note: A strictly convex function can contain at most one critical point. If such a point exists, it will be the global minimum of the strictly convex function.
- One often constructs objective functions in machine learning by adding convex and strictly convex functions. In such cases, the sum of these functions is strictly convex.

Lemma

The sum of a convex function and a strictly convex function is strictly convex.

- Many objective functions in machine learning are convex, and they can often be made strictly convex by adding a strictly convex regularizer.

- A special case of convex functions is that of quadratic convex functions, which can be directly expressed in terms of the positive semidefinite Hessian. Although the Hessian of a function depends on the value of the parameter vector at a specific point, it is a constant matrix in the case of quadratic functions.
- A convex objective function is an ideal setting for a gradient-descent algorithm; the approach will never get stuck in a local minimum.
- Although the objective functions in complex machine learning models (like neural networks) are not convex, they are often close to convex. As a result, gradient-descent methods work quite well in spite of the presence of local optima.

- An improper choice of the learning rate can cause divergence of gradient descent, rather than convergence.
- Many machine learning algorithms use complex objective functions over millions of parameters.
- The gradients are computed either analytically and then hand-coded into the algorithm, or they are computed using automatic differentiation methods in applications like neural networks.

Properties of Optimization in Machine Learning

Typical Objective Functions and additive separability

- Most objective functions in machine learning penalize the deviation of a predicted value from an observed value in one form or another.
- For example, the objective function of least-squares regression is as follows:

$$\sum_{i=1}^{n} \left\| \vec{w} \cdot \vec{X}_i^T - y_i \right\|^2 \tag{1}$$

Here, $\vec{X}_i$ is a d-dimensional row vector containing the ith of n training points, $\mathbf{w}$ is a d-dimensional column vector of optimization variables, and y_i contains the real-valued observation of the ith training point.

- Another form of penalization is the negative log-likelihood objective function.
- This form of the objective function uses the probability that the model's prediction of a dependent variable matches the observed value in the data.
- Clearly, higher values of the probability are desirable, and therefore the model should learn parameters that maximize these probabilities (or likelihoods).
- It is desired to maximize the probability of the true class.
- For the *i*th training point, this probability is denoted by $P(\mathbf{X}_i, y_i, \mathbf{w})$, which depends on the parameter vector **w** and training pair $(\mathbf{X}_i, y_i)$.
- The probability of correct prediction over all training points is given by the product of the probabilities because each prediction is considered independent.
- When each event is independent, the joint probability of these events occurring together is the product of their individual probabilities.

- For independent events, outcome of one event does not influence the outcome of other events.
- The joint probability of multiple independent events occurring simultaneously is the product of their individual probabilities. Mathematically, if $A_1, A_2, \ldots, A_n$ are n independent events, then the joint probability $P(A_1 \cap A_2 \cap \ldots \cap A_n)$ is given by:

$$P(A_1 \cap A_2 \cap \ldots \cap A_n) = P(A_1) \times P(A_2) \times \cdots \times P(A_n) \tag{2}$$

  In other words, for n independent events $A_1, A_2, \ldots, A_n$, the probability of all events occurring is the product of their individual probabilities.
- The joint probability of multiple independent events occurring simultaneously is the product of their individual probabilities.
- Mathematically, if $A_1, A_2, \ldots, A_n$ are n independent events, then the joint probability $P(A_1 \cap A_2 \cap \ldots \cap A_n)$ is given by:

$$P(A_1 \cap A_2 \cap \ldots \cap A_n) = P(A_1) \times P(A_2) \times \cdots \times P(A_n) \tag{3}$$

- Hence the probability of correct prediction over all training points is given by the product of probabilities $P(\mathbf{X}_i, y_i, \mathbf{w})$ over all $(\mathbf{X}_i, y_i)$.
- Logarithm is applied to transform products into sums. Because dealing with sums is often easier than dealing with products, especially when optimizing functions.

$$\log\left(\prod_{i=1}^{n} P(\mathbf{X}_i, y_i, \mathbf{w})\right) = \sum_{i=1}^{n} \log\left(P(\mathbf{X}_i, y_i, \mathbf{w})\right)$$

- Multiplying many small probabilities can result in numerical underflow, where the product becomes so small that it cannot be represented accurately by the computer. Taking the logarithm of these probabilities helps to solve this issue because it prevents the product from becoming too small.

- Machine learning algorithms often involve minimization problems. By taking the negative logarithm, we convert a maximization problem into a minimization one. Maximizing the likelihood (product of probabilities) is equivalent to minimizing the negative log-likelihood.

$$J(\mathbf{w}) = -\log_e \left(\prod_{i=1}^{n} P(\mathbf{X}_i, y_i, \mathbf{w}) \right) = -\sum_{i=1}^{n} \log_e (P(\mathbf{X}_i, y_i, \mathbf{w})) \quad (4)$$

- Using the logarithm also makes the objective function appear as an additively separable sum over the training points.
- Many machine learning problems use additively separable data-centric objective functions, whether squared loss or log-likelihood loss is used.
- This means that each individual data point creates a small (additive) component of the objective function.

- In each case, the objective function contains n additively separable terms, and each point-specific error [such as $J_i = (y_i - \mathbf{w} \cdot \mathbf{X}_i^T)^2$ in least-squares regression] can be viewed as a point-specific loss.
- Therefore, the overall objective function can be expressed as the sum of these point-specific losses:

$$J(\mathbf{w}) = \sum_{i=1}^{n} J_i(\mathbf{w}) \tag{5}$$

- This type of linear separability is useful, because it enables the use of fast optimization methods like stochastic gradient descent and mini-batch stochastic gradient descent, where one can replace the objective function with a sampled approximation.

Stochastic Gradient Descent

- Consider a sample S of the n data points $\mathbf{X}_1, \ldots, \mathbf{X}_n$. The set S of data points is referred to as a mini-batch.
- One can set up a sample-centric objective function $J(S)$ as follows:

$$J(S) = \frac{1}{2} \sum_{i \in S} (y_i - \mathbf{w} \cdot \mathbf{X}_i^T)^2 \tag{6}$$

- The key idea in mini-batch stochastic gradient descent is that the gradient of $J(S)$ with respect to the parameter vector $\mathbf{w}$ is an excellent approximation of the gradient of the full objective function J.

- Therefore, the gradient-descent update of is modified to mini-batch stochastic gradient descent as follows:

$$[w_1, \ldots, w_d]^T \Leftarrow [w_1, \ldots, w_d]^T - \alpha \left[\frac{\partial J(S)}{\partial w_1}, \ldots, \frac{\partial J(S)}{\partial w_d} \right]^T$$

  This approach is referred to as mini-batch stochastic gradient descent.
- Computing the gradient of $J(S)$ is far less computationally intensive compared to computing the gradient of the full objective function.
- A special case of mini-batch stochastic gradient descent is one in which the set S contains a single randomly chosen data point. This approach is referred to as stochastic gradient descent.

- The use of stochastic gradient descent is rare. Typical mini-batch sizes are powers of 2, such as 64, 128, 256, and so on. The reason for this is purely practical rather than mathematical; using powers of 2 for mini-batch sizes often results in the most efficient use of resources such as Graphics Processor Units (GPUs).
- The data points are shuffled randomly to ensure that the model does not learn any specific order of the data.
- After shuffling, data points are divided into smaller groups called mini-batches. These mini-batches are used for updating the model's parameters.
- An epoch is a full pass through the entire training dataset. This means every data point has been used once to update the model.
- If the mini-batch size is 1, each update involves only one data point.
- Thus, for a dataset with n data points, there will be n updates in one epoch.

- If the mini-batch size is k, each update involves k data points.
- For a dataset with n data points, there will be $\lceil n/k \rceil$ updates in one epoch. Here, $\lceil \cdot \rceil$ denotes the ceiling function, which rounds up to the nearest integer.
- An epoch means the model has processed every data point in the training set exactly once, regardless of the mini-batch size.
- Stochastic gradient-descent methods have much lower memory requirements than pure gradient-descent, because one is processing only a small sample of the data in each step.
- Although each update is more noisy (less accurate), the sampled gradient can be computed much faster. Therefore, even though more updates are required, the overall process is much faster.
- **Why does stochastic gradient descent work so well in machine learning?**

- **Random Sampling**: Mini-batch methods are random sampling methods. One is trying to estimate the gradient of a loss function using a random subset of the data.
- **Initial Steps**: At the very beginning of the gradient descent, the parameter vector **w** is grossly incorrect. Therefore, using only a small subset of the data is often sufficient to estimate the direction of descent very well.
- The updates of mini-batch stochastic gradient descent are almost as good as those obtained using the full data, but with a tiny fraction of the computational effort. This significantly improves running time.
- **Near Optimal Value**: When the parameter vector **w** nears the optimal value during descent, the effect of sampling error is more significant.
- Interestingly, this type of error is actually beneficial in machine learning applications because of an effect referred to as **regularization.**

- After building the model on the training data, one evaluates the performance of the model on the test data, which was never seen during the training phase.
- This is a key difference from traditional optimization, because the model is constructed using a particular data set; yet, a different (but similar) data set is used to evaluate performance of the optimization algorithm.
- This difference is crucial because models that perform very well on the training data might not perform very well on the test data. In other words, the model needs to generalize well to unseen test data.
- When a model performs very well on the training data, but does not perform very well on the unseen test data, the phenomenon is referred to as overfitting.
- All machine learning applications are used on unseen test data in real settings; therefore, it is unacceptable to have models that perform well on training data but perform poorly on test data.

Properties of Optimization in Machine learning

Traditional Optimization

- Focuses on improving an objective function by iteratively adjusting parameters.
- Continues until no further improvements are detected or other stopping criteria are met.

Machine Learning Optimization

- Aims to create a model that generalizes well to unseen data.
- Overfitting is a key challenge: a model may perform well on training data but poorly on new data.

Held-Out Data (Validation Set)

- A portion of the data is set aside to evaluate the model's performance on unseen data.
- Helps assess generalization ability and detect overfitting.

Aggarwal, C. C., Aggarwal, L. F., & Lagerstrom-Fife. (2020). Linear algebra and optimization for machine learning (Vol. 156). Springer International Publishing.

Monitoring and Termination

- During training, both training loss and performance on held-out data are monitored.
- Training is typically stopped when the accuracy on the held-out data peaks and starts to decline, even if training loss continues to decrease, to prevent overfitting.
- While stochastic gradient-descent methods have lower accuracy than gradient-descent methods on training data (because of a sampling approximation), they often perform comparably (or even better) on the test data.
- This is because the random sampling of training instances during optimization reduces overfitting.
- The **objective function** is a mathematical expression that measures how well your model is performing. It's what the training process tries to minimize (for example, the error between the model's predictions and the actual data).

- Sometimes, this objective function is **modified** by adding a penalty term that involves the **squared norms of weight vectors**. This means that we are adding an extra cost if the weights (parameters) of the model become too large.
- The **unmodified objective function** focuses solely on minimizing the error on the training data.
- The **penalized objective function** includes an additional term that discourages the model from having very large weights. This penalty term is often the sum of the squared values of the weights.
-
- While the unmodified objective function might make the model fit the training data very well, it can lead to **overfitting.** Overfitting happens when the model learns not only the underlying pattern but also the noise in the training data, resulting in poor performance on new, unseen data (test data).

- The penalized objective function helps the model to generalize better. By keeping the weight vectors smaller, the model becomes simpler and less likely to overfit, thus performing better on out-of-sample test data.
- **Concise parameter vectors** with smaller squared norms lead to simpler models. Simpler models are less prone to overfitting because they are less likely to fit the noise in the training data.
- This practice of adding a penalty term to the objective function to keep the model parameters small is known as **regularization.** Regularization is a technique used to improve the generalization of a model by adding a penalty to the objective function that discourages large weights. This helps to reduce overfitting and results in better performance on new, unseen data.

Hyperparameters

- **Hyperparameters** are settings like the learning rate and regularization weight that we choose before training a model. They are different from the model's weights, which are learned from the training data. We use a separate validation set to tune hyperparameters to avoid overfitting to the training data.
- The challenge is to find the best combination of hyperparameters for a model. This requires testing different combinations to see which ones work best.
- Grid search is a method where all possible combinations of selected hyperparameter values are tested. This helps in finding the optimal choice.
- As the number of hyperparameters increases, the number of combinations to test grows exponentially. For example, with 5 hyperparameters and 10 values for each, you would need to test $10^5 = 100,000$ combinations.

Grid Search Example for Hyperparameter tuning

Suppose you are optimizing two hyperparameters: learning rate and regularization weight. You choose the following values:

- Learning rate: [0.01, 0.1, 1]
- Regularization weight: [0.001, 0.01, 0.1]

Grid search will test all combinations:

- (0.01, 0.001)
- (0.01, 0.01)
- (0.01, 0.1)
- (0.1, 0.001)
- (0.1, 0.01)
- (0.1, 0.1)
- (1, 0.001)
- (1, 0.01)
- (1, 0.1)

Each of these 9 combinations will be evaluated to find the best-performing set.

Importance of Feature prepossessing

- The scale (range) of the features (inputs) affects this sensitivity. For instance, consider a model that predicts a person's wealth based on age (x_1) and years of college education (x_2).
    - Age (x_1) ranges from 0 to 100.
    - Years of college (x_2) ranges from 0 to 10.
- The model could be:

$$y = w_1 x_1 + w_2 x_2$$

- The change in output y with respect to w_1 is influenced by x_1.
- The change in output y with respect to w_2 is influenced by x_2.
- Small steps in w_2 can lead to large steps in w_1, causing overshooting and back-and-forth bouncing along the w_1 direction, while w_2 makes slow but steady progress.
- This results in slow convergence (slow learning).

- **Mean-Centering** Subtracting the mean of each feature from the data to remove bias.
- **Min-Max Normalization** Scaling features to a range of [0, 1] using the formula:

$$x_{ij} \leftarrow \frac{x_{ij} - \text{min}_j}{\text{max}_j - \text{min}_j}$$

- Feature normalization helps avoid issues with learning and ensures smoother convergence of gradient descent methods.
- Hence to ensure efficient learning, it's important to preprocess features so that they have similar scales. This can be done through mean-centering, min-max normalization etc.
- These techniques help prevent the learning process from being dominated by features with larger scales and lead to smoother and faster convergence.

Computing Derivatives with respect to Vectors

- In machine learning, optimization models often involve finding the best parameters for a given objective function, like minimizing a loss function. This objective function $J(\mathbf{w})$ depends on a vector of parameters **w**.
- To avoid writing out numerous partial derivatives for each component of the vector, we use matrix calculus notation. This allows us to compute derivatives in a compact form, making calculations more efficient and less tedious.
- Matrix calculus can handle derivatives between different types of mathematical objects:
    - Scalars
    - Vectors
    - Matrices
    - Tensors

- For gradient descent in multivariate optimization, the update rule for the parameters **w** is:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla J$$

- Using matrix calculus notation, the gradient ∇J can be written as:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$$

- This notation returns a vector:

$$\nabla J = \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \left[\frac{\partial J(\mathbf{w})}{\partial w_1}, \ldots\ldots, \frac{\partial J(\mathbf{w})}{\partial w_d} \right]^T$$

- **Jacobian Matrix**: When dealing with derivatives of vectors with respect to vectors, the result is a matrix called the Jacobian.
- For example, if we have a vector **h** with respect to a vector **w**, the Jacobian is:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{w}}$$

- The (i, j)-th entry of the Jacobian is the derivative of h_j with respect to w_i:

$$\left(\frac{\partial \mathbf{h}}{\partial \mathbf{w}}\right)_{ij} = \frac{\partial h_j}{\partial w_i}$$

- When differentiating a scalar function J with respect to a matrix **W**, the result matches the shape of the matrix. The (i, j)-th entry of the derivative is:

$$\left(\frac{\partial J}{\partial \mathbf{W}}\right)_{ij} = \frac{\partial J}{\partial W_{ij}}$$

- In machine learning, we often use matrix calculus identities to simplify and compute derivatives efficiently.
- **Quadratic Form**

$$F(\mathbf{w}) = \mathbf{w}^T A\mathbf{w}$$

  - A is a symmetric $d \times d$ matrix with constant values.
  - $\mathbf{w}$ is a d-dimensional column vector of variables.
- This type of function appears frequently in problems like least-squares regression and support vector machines. The gradient (or derivative) of $F(\mathbf{w})$ with respect to $\mathbf{w}$ is:

$$\nabla F(\mathbf{w}) = \frac{\partial F(\mathbf{w})}{\partial \mathbf{w}} = 2A\mathbf{w}$$

- **Verify that the result is a column vector?**

- **Linear Form**

$$G(\mathbf{w}) = \mathbf{b}^T B\mathbf{w} = \mathbf{w}^T B^T \mathbf{b}$$

  - B is an $n \times d$ matrix with constant values.
  - **w** is a d-dimensional column vector of variables.
  - **b** is an n-dimensional constant vector, independent of **w**.

- This function is linear in **w**, meaning the gradient components are constants.
- The values $\mathbf{b}^T B\mathbf{w}$ and $\mathbf{w}^T B^T \mathbf{b}$ are the same because transposing a scalar doesn't change its value.
- The gradient of $G(\mathbf{w})$ is:

$$\nabla G(\mathbf{w}) = \frac{\partial G(\mathbf{w})}{\partial \mathbf{w}} = B^T \mathbf{b}$$

Verify..?

Problem

Let $A = [a_{ij}]$ be a symmetric $d \times d$ matrix of constant values, $B = [b_{ij}]$ be an $n \times d$ matrix of constant values, $\mathbf{w}$ be a d-dimensional column vector of optimization variables, and $\mathbf{b}$ be an n-dimensional column vector of constants. Let $F(\mathbf{w}) = \mathbf{w}^T A\mathbf{w}$ and let $G(\mathbf{w}) = \mathbf{b}^T B\mathbf{w}$. Show using component-wise partial derivatives that $\nabla F(\mathbf{w}) = 2A\mathbf{w}$ and $\nabla G(\mathbf{w}) = B^T\mathbf{b}$.