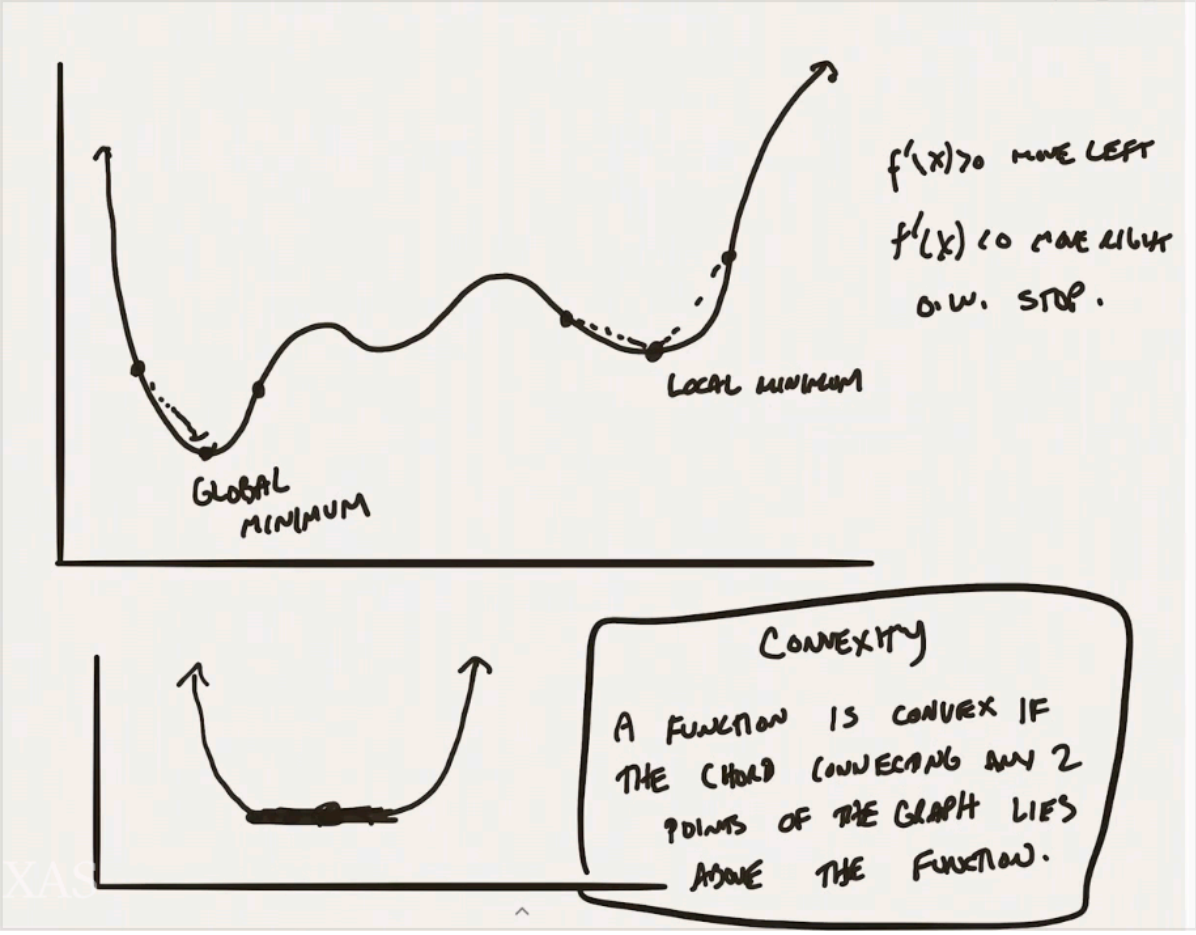
Gradient Descent



Understanding Gradient Descent (Explained Simply with Examples)

This image explains **Gradient Descent**, a method used to **find the minimum of a function**. Let's break it down step by step in a simple way. 🚀

1️⃣ What is Gradient Descent?

Gradient Descent is like **hiking down a hill**:

- If you are at the **top of a hill**, you need to **go downward** to reach the lowest point.
- You take **small steps** in the direction that **reduces the height the most**.
- The **steeper** the slope, the **bigger** your step.

Mathematical Idea

We are given a function $f(x)$, and we want to **find the value of x that gives the lowest $f(x)$ (the minimum)**.

- The **slope** of the function at x is given by the **derivative** $f'(x)$.
- If the slope is **negative** ($f'(x) < 0$), move **right**.
- If the slope is **positive** ($f'(x) > 0$), move **left**.
- If the slope is **zero** ($f'(x) = 0$), **stop** (you reached the lowest point!).

2️⃣ Simple Example: Walking Down a Hill

Imagine you are standing on a **mountain** and want to reach the **valley**.

- If the **ground is sloping downward**, you **keep walking**.
- If the **ground is flat**, you **stop** (you reached the lowest point).

In **Gradient Descent**, we do the same:

1. Compute the **slope** (derivative $f'(x)$).
2. Take a **step** in the direction that **decreases** $f(x)$.
3. Repeat **until we reach the minimum**.

3️⃣ Example: Finding the Minimum of $f(x) = x^2$

Let's apply Gradient Descent to **find the minimum of**:

$$f(x) = x^2$$

Step 1: Compute the Derivative

$$f'(x) = 2x$$

The derivative tells us **the slope at any point** x.

Step 2: Update Rule

The **update rule** for Gradient Descent is:

$$x_{\text{new}} = x_{\text{old}} - \alpha f'(x)$$

where:

- α = **learning rate** (controls step size).
- $f'(x)$ = **slope** (derivative).

Step 3: Start Gradient Descent

Let's start at $x = 5$ and use a **learning rate** $\alpha = 0.1$.

Step	x	$f'(x) = 2x$	$x_{\text{new}} = x - 0.1 \cdot 2x$
1	5	10	$5 - 0.1 \times 10 = 4$
2	4	8	$4 - 0.1 \times 8 = 3.2$
3	3.2	6.4	$3.2 - 0.1 \times 6.4 = 2.56$
4	2.56	5.12	$2.56 - 0.1 \times 5.12 = 2.048$
...	...	...	...
10	0.5	1.0	$0.5 - 0.1 \times 1.0 = 0.4$

As we repeat, x moves **closer to zero**, which is the minimum!

Why Do We Find the Minimum of $f(x) = x^2$ (or Any Function)?

Finding the minimum of a function like $f(x) = x^2$ is important in many real-world applications. **But why?** Let's break it down with simple examples.

1️⃣ Why Do We Minimize Functions?

Many problems in **science, engineering, finance, and AI** involve **optimization**—finding the best solution.

- **In Machine Learning** → We minimize the "error" to make predictions more accurate.
- **In Economics** → We minimize "cost" to maximize profit.
- **In Physics** → We find "equilibrium" by minimizing potential energy.
- **In Engineering** → We optimize designs by minimizing material usage.

🔹 **Minimizing functions helps us make things more efficient, accurate, or cost-effective.**

2️⃣ Example 1: Reducing Error in Machine Learning 🎯

Imagine you are training a **self-driving car** 🚗. The car makes predictions about **when to stop**.

- If the car predicts **wrong distances**, it might **crash**.
- We define an **error function** $f(x)$ (how wrong the car is).
- Our goal is to **minimize** $f(x)$ so the car makes **better predictions**.

$$\text{Error} = (\text{Predicted Distance} - \text{Actual Distance})^2$$

This function is similar to $f(x) = x^2$.

🔹 **Lower error → Better self-driving decisions.**

5️⃣ Why $f(x) = x^2$?

The function $f(x) = x^2$ **is the simplest example** of minimization:

- The minimum is at $x = 0$.
- It teaches us how to **find minima** in complex problems.

In real-world problems, we use the same logic but with more complex functions.

Understanding Local vs. Global Minima & Convexity (Explained Simply with Examples)

This image explains **two key concepts** in optimization:

1. **Local vs. Global Minima**
2. **Convexity**

Let's break it down in a simple way.

1️⃣ Local Minimum vs. Global Minimum

Imagine you are **hiking in the mountains** ⛰️:

- You want to **find the lowest valley** (global minimum).
- Sometimes, you might get **stuck in a small dip** (local minimum) and think it's the lowest point.

Definitions

- **Global Minimum** → The **lowest** point in the entire function.
- **Local Minimum** → A **low point** that is **not necessarily the lowest overall**.

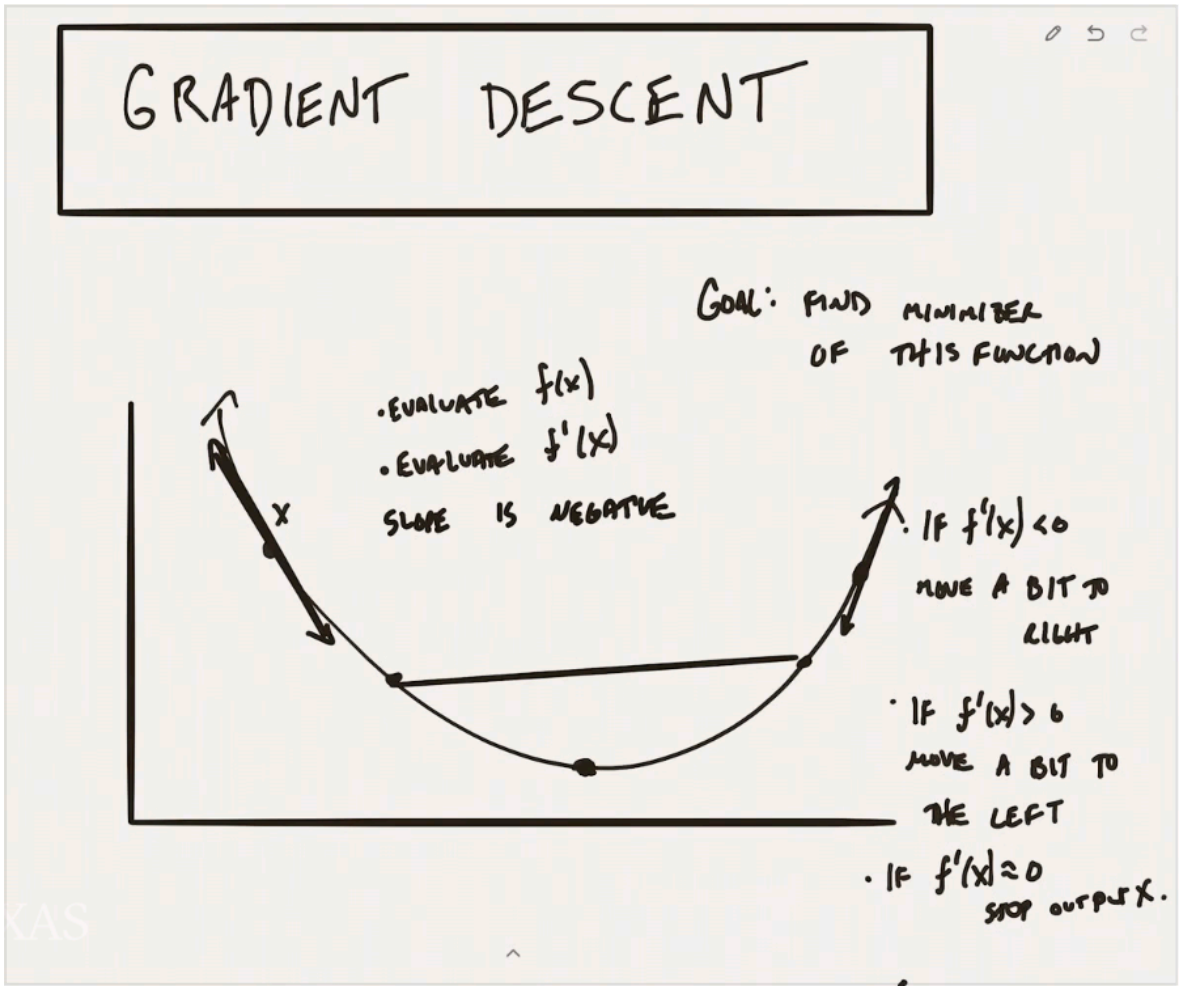
📌 **In Gradient Descent:**

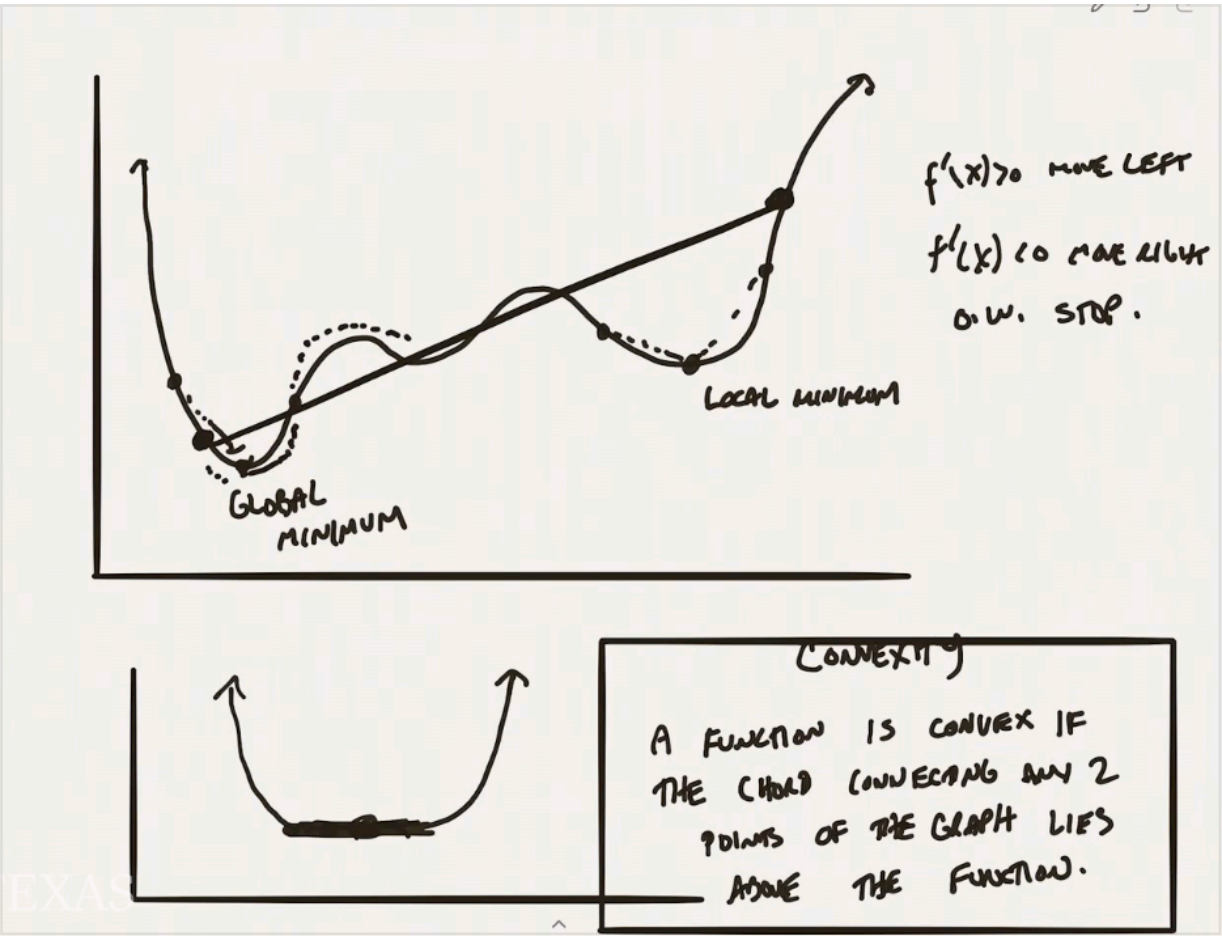
- If the function has **one minimum**, we will find it.
- If the function has **multiple minima**, we might get **stuck in a local minimum**.

Example 1: Finding the Best Temperature for Baking a Cake 🎂

- You want to bake the **perfect cake.**
- You test **different oven temperatures** and measure **how good the cake tastes.**
- A **local minimum** could be **180°C** (decent cake).
- A **global minimum** could be **200°C** (perfect cake).
- If you only test nearby values, you **might miss the best temperature.**

🔷 **Lesson:** You need a **good search strategy** to find the global minimum.





Above image has 2 graphs 1. Is non convex 2. Convex

2️⃣ What is Convexity?

The **bottom diagram** explains **convex functions**.

Definition

A function is **convex** if **a straight line connecting any two points on the graph lies above the curve.**

In simple terms:

- **If a function is convex** → It has **one global minimum** (easy to find).
- **If a function is non-convex** → It has **many minima** (harder to find the global minimum).

📌 **Convex functions are great because Gradient Descent always finds the global minimum!**

Example 2: Convex vs. Non-Convex Optimization

Scenario: Training an AI Model 🤖

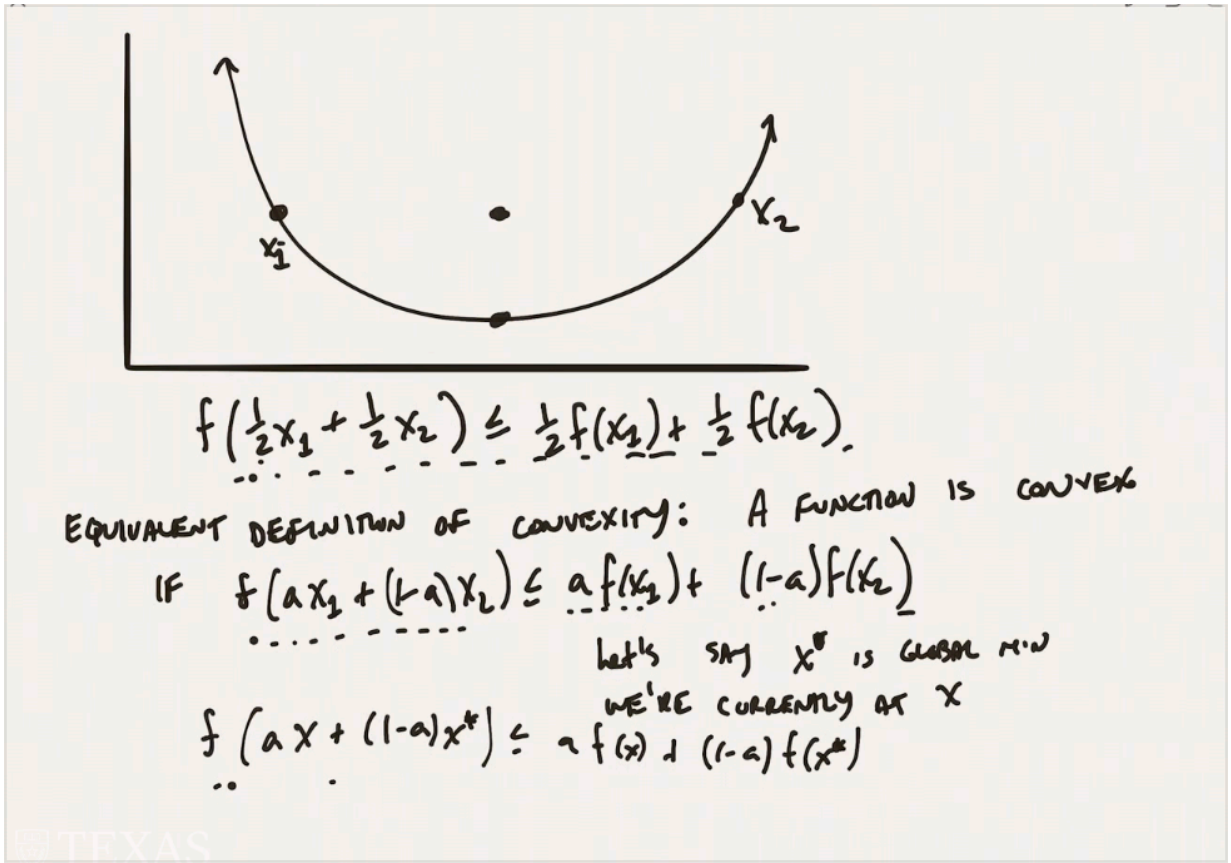
- You are **training an AI** to recognize cats 🐱.
- The goal is to **minimize prediction error**.

🔷 **Case 1: Convex Loss Function (Easy to Optimize)**

- If the **loss function** is convex, Gradient Descent **always finds the best model**.
- Like a **smooth bowl shape**.

🔷 **Case 2: Non-Convex Loss Function (Hard to Optimize)**

- If the **loss function** is non-convex, Gradient Descent **might get stuck** in a local minimum.
- Like a **bumpy mountain**.



Understanding Convexity with a Simple Example

The image explains the **convexity property** of a function. A function $f(x)$ **is convex** if for any two points x_1 and x_2, the function satisfies:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2), \quad \forall \lambda \in [0,1]$$

This means that if we take a weighted average of two points x_1 and x_2, the function value at that average is **less than or equal to** the weighted average of the function values.

🔷 Simple Example:

Consider the function:

$$f(x) = x^2$$

We check if this function satisfies the convexity condition.

Step 1: Pick Two Points

Let's choose:

- $x_1 = -2$, so $f(x_1) = (-2)^2 = 4$
- $x_2 = 2$, so $f(x_2) = (2)^2 = 4$

Step 2: Compute Midpoint Function Value

If we take an average (midpoint) of x_1 and x_2:

$$\frac{x_1 + x_2}{2} = \frac{-2+2}{2} = 0$$

$$f\left(\frac{x_1 + x_2}{2}\right) = f(0) = 0^2 = 0$$

Step 3: Compute Weighted Average of Function Values

$$\frac{f(x_1) + f(x_2)}{2} = \frac{4 + 4}{2} = 4$$

Step 4: Compare Both Sides

$$f\left(\frac{x_1 + x_2}{2}\right) = 0 \leq 4 = \frac{f(x_1) + f(x_2)}{2}$$

✅ **Since the inequality holds, $f(x) = x^2$ is convex.**

🔷 Geometric Intuition

- If a function is convex, the line segment **between any two points on the function** lies **above** the function itself.
- In the graph, you can see a **curved U-shaped function**.
- The point in the middle is **lower** than the average height of the two endpoints, satisfying convexity.

🔷 Why Is Convexity Important?

1. **Optimization**
   - If a function is convex, any **local minimum is also the global minimum**.
   - This is why gradient descent works well for convex functions.
2. **Machine Learning**
   - Many loss functions (e.g., mean squared error) are convex, ensuring efficient optimization.
3. **Logistics & Economics**
   - Convex functions model **cost minimization** problems in business and supply chains.

🔷 Summary

- A function is **convex** if a **chord connecting two points** on the function lies **above or on the function**.
- The inequality:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

  holds for all $\lambda \in [0, 1]$.
- Example: $f(x) = x^2$ is convex because the midpoint is lower than the average function value.

ANOTHER EXAMPLE:

$f(x) = w^T x + b$ (LINEAR FUNCTIONS IN d dimensions)

CURRENTLY AT POINT x. WE WANT TO KNOW WHAT DIRECTION SHOULD WE MOVE IN TO MINIMIZE f?

BY "DIRECTION" WE MEAN UNIT VECTOR.

(u WILL BE UNIT VECTOR)

$f(x+u) = w^T x + w^T u + b$ CORRECT CHOICE OF $u = \frac{-w}{\|w\|}$

IF WE MOVE IN DIRECTION $\frac{-w}{\|w\|}$ THEN f DECREASES BY $\|w\|_2$

Understanding Directional Minimization for a Linear Function

The image explains **how to move in the direction that minimizes a linear function**:

$$f(x) = w^T x + b$$

where:

- w is the **weight vector** (also known as the gradient in optimization).
- x is the **input vector**.
- b is the **bias term**.
- $w^T x$ is the **dot product** of w and x, giving a **scalar output**.

1️⃣ What Is the Goal?

We are at some point x and want to **move in a direction** that minimizes $f(x)$.

- The function $f(x)$ **increases most in the direction of** w.
- To **decrease** $f(x)$, **we should move in the opposite direction** of w.

This means the **best direction to move in** is:

$$u = -\frac{w}{||w||_2}$$

which is the **unit vector in the opposite direction of** w.

2️⃣ What Does $||w||_2$ (Norm of w) Mean?

The **norm** of a vector w, written as $||w||_2$, represents the **length (magnitude) of the vector.**

For a vector $w = (w_1, w_2, ..., w_d)$, the **L2-norm (Euclidean norm)** is:

$$||w||_2 = \sqrt{w_1^2 + w_2^2 + ... + w_d^2}$$

This measures the **distance from the origin to the point** w.

🔹 Example 1: 2D Vector

If $w = (3, 4)$:

$$||w||_2 = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

Thus, the **unit vector** in the opposite direction is:

$$u = -\frac{w}{||w||_2} = -\frac{(3, 4)}{5} = \left(-\frac{3}{5}, -\frac{4}{5}\right) = (-0.6, -0.8)$$

🔹 Example 2: 3D Vector

For $w = (2, -1, 2)$:

$$||w||_2 = \sqrt{2^2 + (-1)^2 + 2^2} = \sqrt{4 + 1 + 4} = \sqrt{9} = 3$$

The unit vector in the opposite direction is:

$$u = -\frac{w}{||w||_2} = -\frac{(2, -1, 2)}{3} = \left(-\frac{2}{3}, \frac{1}{3}, -\frac{2}{3}\right)$$

3️⃣ Why Move in the Direction of $-w/||w||_2$?

- The weight vector w **points in the direction of the steepest increase.**
- Moving **in the opposite direction of** w minimizes $f(x)$.
- $-w/||w||_2$ ensures we move **in a unit step**, so we decrease the function smoothly.

🔷 **How much does $f(x)$ decrease?**

$$f(x+u) = w^T x + w^T u + b$$

Since $u = -w/||w||_2$, we get:

$$w^T u = -||w||_2$$

Thus, $f(x)$ **decreases by** $||w||_2$ **at each step.**

4️⃣ Summary

✅ **Norm** $||w||_2$ measures **the length of the weight vector.**
✅ **To minimize** $f(x)$**, move in the direction** $-w/||w||_2$.
✅ **Each step decreases** $f(x)$ **by** $||w||_2$, ensuring smooth optimization.

Why Do We Want to Minimize a Linear Function?

Minimizing a function is a core concept in **optimization**, especially in **machine learning, economics, and physics**. In this case, we are working with a **linear function**:

$$f(x) = w^T x + b$$

where:

- w is the **weight vector**.
- x is the **input feature vector**.
- b is the **bias**.

In optimization, we often aim to **find the minimum value** of a function to make optimal decisions.

But Does a Linear Function Have a Global Minimum?

- **No!** A linear function does **not** have a single global minimum like a quadratic function (e.g., x^2).
- A linear function either **increases or decreases infinitely**.
- However, in many cases, we still minimize it **within certain constraints or over a given region**.

🔷 Why Move in the Opposite Direction of w?

The weight vector w **points in the direction of the steepest increase** of the function.

- If we want to **maximize** $f(x)$ → move **in the direction of** w.
- If we want to **minimize** $f(x)$ → move **in the opposite direction of** w.

🔷 Geometric Interpretation

- w defines a **hyperplane (flat surface) in higher dimensions.**
- Moving in **the opposite direction of** w moves towards **lower values of** $f(x)$.

Yes! If $f(x)$ **represents profit**, then minimizing $f(x)$ means **reducing profit**.

Example: Profit Function

Let's say the profit function is:

$$f(x) = 5x + 3$$

where:

- x = number of products sold.
- $5x$ = revenue per product.
- 3 = fixed profit.

If we **want to minimize profit**, we **decrease** x, which means moving in the **opposite direction of** $w = 5$.

Key Insight

Minimization **depends on the context**:

- If $f(x)$ represents **loss**, minimizing it is **good** (reducing loss).
- If $f(x)$ represents **profit**, minimizing it is **bad** (reducing profit).

In machine learning, we use **gradient descent** to minimize **loss functions** like:

$$L(w) = \frac{1}{m} \sum (y^j - w^T x^j)^2$$

where:

- $L(w)$ is the **loss function**.
- y^j is the **actual target**.
- $w^T x^j$ is the **predicted output**.

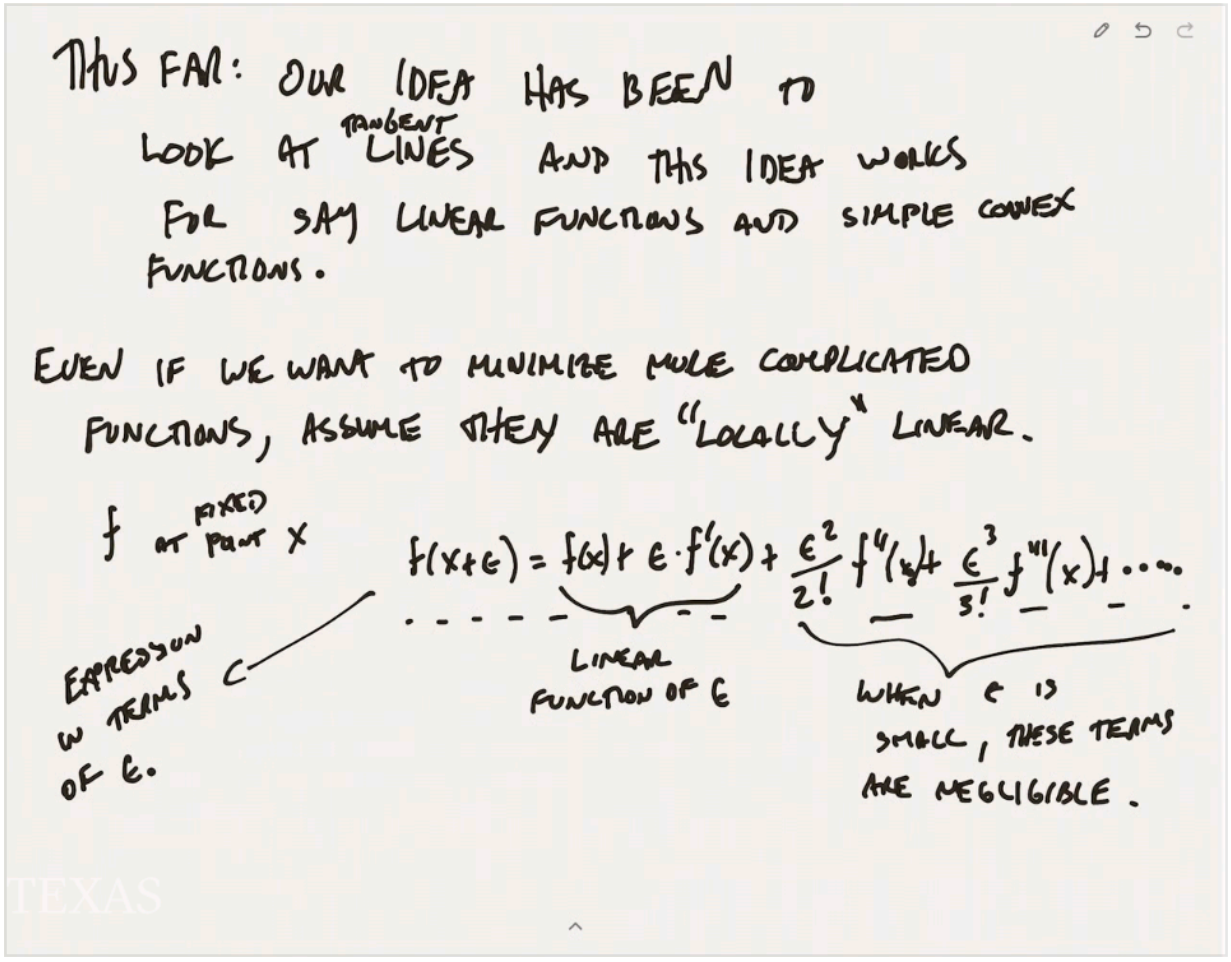
To **minimize the loss function**, we update w using **gradient descent**:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \nabla L(w)$$

where:

- η is the **learning rate**.
- $\nabla L(w)$ is the **gradient (which is w itself for a linear function)**.

Since w **points towards increasing loss**, we **move in the opposite direction** to **reduce loss.**



Understanding Local Linearity and Taylor Series Expansion

The image explains the **idea of approximating complicated functions using local linearity**. This is based on the **Taylor series expansion**, which allows us to approximate a function around a given point.

1️⃣ Key Idea: Any Function is Locally Linear

Even if a function is complex, **zooming in around a small region makes it look like a straight line**. This means we can approximate it using a **linear function** plus small correction terms.

For a function $f(x)$, we approximate it at a nearby point $x + \epsilon$ using the Taylor series:

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + \frac{\epsilon^2}{2!} f''(x) + \frac{\epsilon^3}{3!} f'''(x) + \dots$$

- The **first two terms** ($f(x) + \epsilon f'(x)$) form a **linear approximation**.
- The **higher-order terms** ($\frac{\epsilon^2}{2!} f''(x)$, etc.) account for the curvature of the function.
- When ϵ is **small**, these higher-order terms are **negligible**, making the function behave **almost like a straight line**.

2️⃣ Simple Example: Approximating $f(x) = x^2$

Let's approximate $f(x) = x^2$ around $x = 3$ for small ϵ.

Step 1: Compute Derivatives

- $f(x) = x^2$
- First derivative: $f'(x) = 2x$
- Second derivative: $f''(x) = 2$

Step 2: Taylor Expansion at $x = 3$

Using:

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

Substituting values:

$$(3+\epsilon)^2 \approx 3^2 + \epsilon(2 \times 3) + \frac{\epsilon^2}{2!}(2)$$

$$= 9 + 6\epsilon + \epsilon^2$$

Step 3: When ϵ is Small

If ϵ is very small (e.g., 0.01), the ϵ^2 term becomes **negligible**.

$$(3+\epsilon)^2 \approx 9 + 6\epsilon$$

🔹 **Interpretation:**

- **Linear Approximation:** $9 + 6\epsilon$ behaves like a **straight line**.
- The function $f(x) = x^2$ is **quadratic**, but for small changes ϵ, it behaves almost **like a linear function**.

3️⃣ Why Is This Important?

1. **Simplifies Complex Functions**
   - Instead of working with nonlinear functions, we approximate them with **straight lines** (which are easier to analyze).
2. **Used in Optimization (Gradient Descent)**
   - When minimizing a function, we **move in the direction of the derivative** because higher-order terms are small.
3. **Foundation for Newton's Method & Machine Learning**
   - Many optimization algorithms rely on this **local linearity assumption**.

4️⃣ Summary

- **Complex functions can be locally approximated by linear functions**.
- The **Taylor expansion** helps approximate $f(x)$ using the first derivative $f'(x)$.
- When ϵ is small, **higher-order terms are negligible**, making the function behave **almost like a straight line**.
- This idea is used in **optimization, physics, and machine learning**.

Would you like a Python example to **visualize this approximation**? 🚀

What Are We Trying to Achieve Here?

Great question! The **core idea** behind this concept is to **simplify complex functions by approximating them locally as linear functions**. This makes it **easier to analyze, compute, and optimize**. Let's break it down step by step **without formulas** and explain why this is useful in real-world applications.

1️⃣ What's the Problem?

Many functions in **mathematics, physics, and machine learning** are **complicated**. They involve nonlinear behavior, making them difficult to **work with, optimize, or understand intuitively**.

Imagine you're hiking in the mountains 🌄:

- The terrain is **uneven and complex** (just like a nonlinear function).
- But **locally, any small section of the mountain looks like a straight path**.
- Instead of analyzing the **entire mountain shape**, you can **approximate the slope** at your current position.

This is **exactly** what we do in **calculus and optimization**—we replace the function with a simpler **linear approximation** in a small region.

2️⃣ Why Do We Approximate a Function as Locally Linear?

🔷 Example 1: Walking Blindfolded (Gradient Descent in ML)

Imagine you are **blindfolded** and placed on a hill. Your goal is to **walk downhill** to reach the lowest point (**global minimum**).

- You **can't see the whole landscape** (full function).
- But you **can feel the slope at your feet** (derivative).
- So you take **small steps downhill** in the direction of the steepest descent.

By repeating this process, you **eventually** reach the bottom.

This is **exactly how gradient descent works** in machine learning—it minimizes a function by following the local slope.

🔷 Example 2: Car Speedometer (Rate of Change)

Imagine you're driving a car 🚗:

- The **speedometer** tells you how fast you're going **right now** (first derivative).
- If you maintain this **exact speed**, you can estimate **how far you'll travel in the next second**.
- This estimate is an **approximation** because roads have turns, but it works well for **short time intervals**.

This is how **Taylor approximations work**—for **small changes**, the function behaves **almost like a straight line**.

3️⃣ What Are We Trying to Achieve?

1️⃣ Simplify Complex Problems

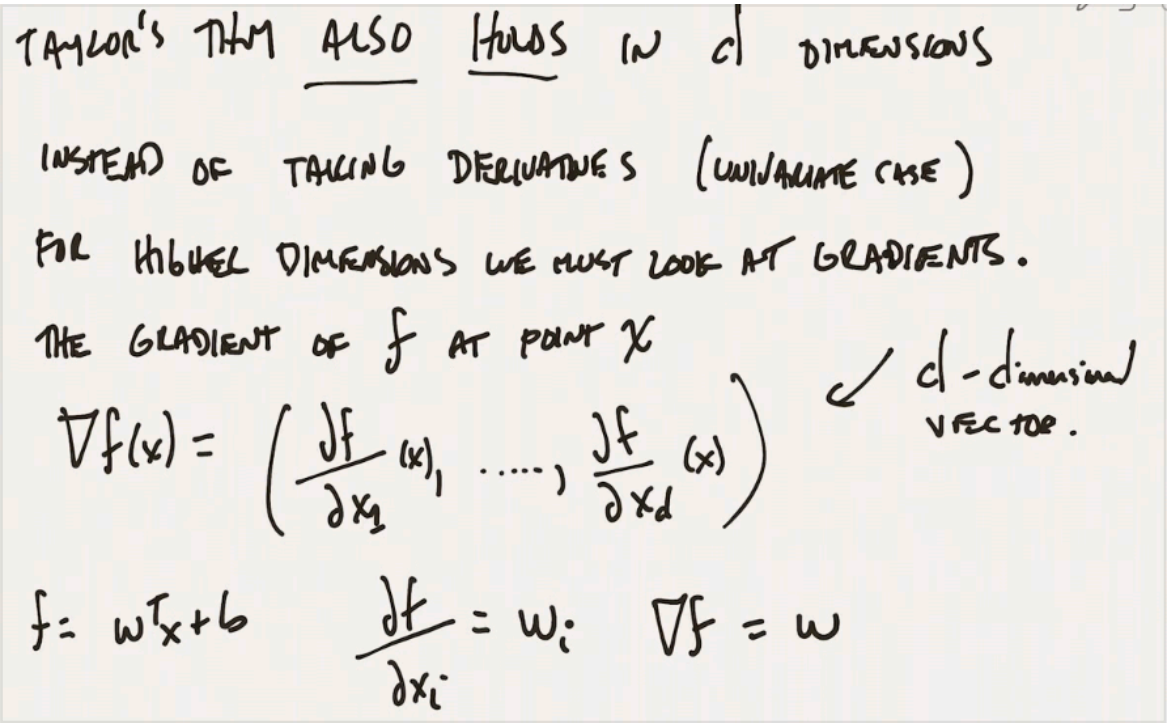
- Instead of working with nonlinear equations, we use a **straight-line approximation**.
- This makes calculations **faster and easier**.

2️⃣ Make Optimization Possible (Machine Learning & AI)

- We use the **local slope (derivative)** to decide **which direction to move** to minimize a function.
- This is how **gradient descent** works in neural networks.

3️⃣ Predict Small Changes Efficiently

- If a function is too **complex to compute exactly**, we approximate it **locally**.
- This is useful in **physics, economics, and engineering** (e.g., estimating how an object moves under gravity).



This image explains how **Taylor's Theorem extends to multiple dimensions** and introduces the concept of **gradients** in higher-dimensional spaces.

Let's break it down **step by step** in simple terms.

1️⃣ What is Taylor's Theorem?

In **one dimension**, Taylor's Theorem lets us **approximate a function** near a given point using derivatives.

For example, if you have a function $f(x)$, its small change can be estimated as:

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

This works well when ϵ is **small**. This idea helps in **optimization and predictions**.

2️⃣ What Happens in Multiple Dimensions?

When a function has **multiple variables** (e.g., $f(x_1, x_2, x_3, \ldots, x_d)$), we cannot just take a single derivative. Instead, we need to compute **partial derivatives** with respect to **each variable**.

The collection of all these partial derivatives is called the **gradient**.

3️⃣ What is a Gradient?

The **gradient** is a **vector (a list of numbers)** that tells us how fast a function changes in each direction.

Mathematically, the gradient of $f(x)$ is:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_d} \right)$$

This means:

- $\frac{\partial f}{\partial x_1}$ tells us how $f(x)$ changes when we **only change** x_1.
- $\frac{\partial f}{\partial x_2}$ tells us how $f(x)$ changes when we **only change** x_2.
- And so on.

So, instead of a **single number (derivative)** like in one dimension, we now have a **vector (list of derivatives)**.

4️⃣ Example: Gradient of a Linear Function

In the image, we have a **linear function**:

$$f(x) = w^T x + b$$

which means:

$$f(x) = w_1 x_1 + w_2 x_2 + ... + w_d x_d + b$$

Taking partial derivatives:

- $\frac{\partial f}{\partial x_1} = w_1$
- $\frac{\partial f}{\partial x_2} = w_2$
- ...
- $\frac{\partial f}{\partial x_d} = w_d$

So, the **gradient of $f(x)$ is just w itself**:

$$\nabla f = (w_1, w_2, ..., w_d) = w$$

$$\nabla f = (w_1, w_2, ..., w_d) = w$$

Key insight:

For a **linear function**, the gradient is just the coefficient vector w. This means:

- The **gradient tells us the direction in which the function increases the fastest**.
- If we want to **minimize** the function, we move in the **opposite direction of the gradient**.

5️⃣ Why is This Important?

🔹 In 1D, we use derivatives to find the slope of a function.
🔹 In higher dimensions, we use gradients to find the slope in every direction.
🔹 In optimization (e.g., machine learning), we move in the opposite direction of the gradient to minimize a function (gradient descent).

6️⃣ Summary in Super Simple Terms

✅ **A derivative** tells you **how a function changes at a single point**.
✅ **A gradient** tells you **how a function changes in every direction** when there are multiple variables.
✅ **For linear functions, the gradient is just the weights w.**
✅ **In machine learning, we minimize loss by moving in the opposite direction of the gradient (gradient descent).**

What's the Difference Between a Derivative and a Gradient?

Both **derivatives** and **gradients** measure how a function changes, but they apply to **different situations**.

Concept	Derivative	Gradient
Applies to	Single-variable functions $f(x)$	Multi-variable functions $f(x_1, x_2, ..., x_d)$
Definition	Measures how $f(x)$ changes with respect to x	Measures how $f(x)$ changes with respect to **each variable**
Representation	A **single number** (scalar)	A **vector** (list of numbers)
Notation	$\frac{df}{dx}$	$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_d}\right)$
Example Function	$f(x) = x^2$	$f(x_1, x_2) = x_1^2 + x_2^2$
Example Computation	$\frac{df}{dx} = 2x$	$\nabla f = (2x_1, 2x_2)$

1️⃣ Why Not Use the Derivative for Multiple Variables?

A **derivative** only tells us **how a function changes with respect to one variable.**

But in **multiple dimensions**, a function can change **in many different directions** at once.

🔹 **Example:**

Imagine a function that depends on two variables:

$$f(x_1, x_2) = x_1^2 + x_2^2$$

If we take the derivative **with respect to x_1 only**, we get:

$$\frac{\partial f}{\partial x_1} = 2x_1$$

But what if x_2 **also affects the function**? We also need:

$$\frac{\partial f}{\partial x_2} = 2x_2$$

So the correct way to **represent all directions at once** is with the **gradient**:

$$\nabla f = (2x_1, 2x_2)$$

✅ **Gradient tells us the direction in which $f(x)$ increases the fastest.**

✅ **Derivative alone only tells us about one direction at a time.**

2️⃣ How Do We Use the Gradient to Create a Linear Function?

We can **approximate any function using its gradient**. This is useful in:

- **Machine learning** (linear regression)
- **Optimization** (gradient descent)
- **Physics & engineering**

🔹 Taylor Expansion for Linear Approximation

A function $f(x)$ can be approximated **near some point** x_0 using the **first-order Taylor series**:

$$f(x) \approx f(x_0) + \nabla f(x_0) \cdot (x - x_0)$$

This looks like:

$$\text{New function} = \text{Constant} + \text{Gradient} \times \text{Change in x}$$

This is a **linear function**!

3️⃣ Example: Using the Gradient to Create a Linear Function

Let's approximate:

$$f(x_1, x_2) = x_1^2 + x_2^2$$

at the point $(1, 2)$.

Step 1: Compute the Gradient

$$\nabla f = (2x_1, 2x_2)$$

At $(1, 2)$:

$$\nabla f(1, 2) = (2(1), 2(2)) = (2, 4)$$

Step 2: Create the Linear Approximation

Using:

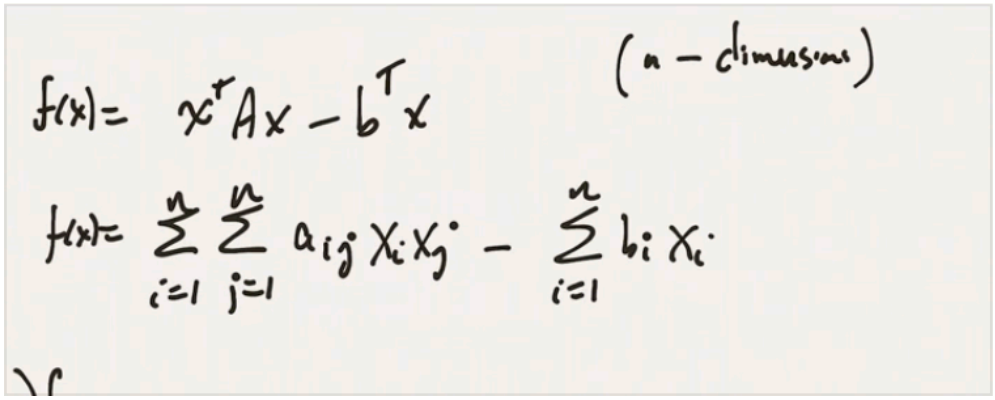
$$f(x) \approx f(x_0) + \nabla f(x_0) \cdot (x - x_0)$$

We get:

$$f(x_1, x_2) \approx 5 + 2(x_1 - 1) + 4(x_2 - 2)$$

This is a linear function!

- The gradient **(2,4)** acts like the **weights** in a linear equation.
- The function behaves **like a straight line** when zoomed in.



This formula is related to **quadratic functions in multiple dimensions**, which are super important in **optimization and gradient descent**. Let's break it down into **simple terms** so it makes sense! 🚀

The formula given is:

$$f(x) = x^T Ax - b^T x$$

Or written as **sums**:

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j - \sum_{i=1}^{n} b_i x_i$$

This formula represents a **quadratic function**, which is like a **multi-dimensional version** of the classic **parabola (U-shape curve) we see in math**.

Why Quadratic?

- $x^T Ax$: This part **introduces curvature** (how the function bends).
- $-b^T x$: This part **shifts the function** and moves the minimum to a different location.

In simple words:
This function describes a bowl-shaped surface in multiple dimensions!

2️⃣ How Does This Connect to Gradient Descent?

Gradient Descent is a way to **find the lowest point (minimum)** of a function.

Imagine you're on top of a **big hill**, and your goal is to **reach the bottom**:

1. You check **which direction slopes downward the most** (this is the **gradient**).
2. You **take small steps in that direction** until you reach the lowest point.

For this function:

- The **gradient** of $f(x)$ tells us how to move **step by step** to reach the minimum.
- We update our position **opposite to the gradient** to find the minimum.

Let's assume we have **two variables** x_1 and x_2, and we want to minimize:

$$f(x_1, x_2) = 2x_1^2 + 3x_2^2 - 4x_1 - 6x_2$$

Step 1: Identify the Matrix Form

Here, we can rewrite the function as:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$$

So,

$$f(x) = x^T Ax - b^T x$$

Step 2: Compute the Gradient

The gradient (derivative in multi-dimensions) is:

$$\nabla f = 2Ax - b$$

Substituting A and b:

$$\nabla f = 2 \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} x - \begin{bmatrix} 4 \\ 6 \end{bmatrix}$$

$$= \begin{bmatrix} 4x_1 \\ 6x_2 \end{bmatrix} - \begin{bmatrix} 4 \\ 6 \end{bmatrix}$$

$$= \begin{bmatrix} 4x_1 - 4 \\ 6x_2 - 6 \end{bmatrix}$$

Step 3: Apply Gradient Descent

To minimize, we move in the **opposite direction of the gradient**:

$$x_{\text{new}} = x_{\text{old}} - \eta \nabla f$$

where η is a small learning rate.

If we start at $(x_1, x_2) = (0, 0)$, we update:

$$x_1 = 0 - \eta(4(0) - 4) = \eta \cdot 4$$

$$x_2 = 0 - \eta(6(0) - 6) = \eta \cdot 6$$

Repeating this step moves us toward the **minimum (1,1)**.

Does Reaching $(1, 1)$ Mean We Found the Global Minimum?

Yes! If the gradient **becomes zero** at $(x_1, x_2) = (1, 1)$, then we have reached a **critical point**—which could be a **global minimum, maximum, or saddle point**.

Step-by-Step Check

We want to confirm if $(1, 1)$ is a **global minimum** of the function:

$$f(x_1, x_2) = 2x_1^2 + 3x_2^2 - 4x_1 - 6x_2$$

Step 1: Compute the Gradient

The gradient of $f(x_1, x_2)$ is:

$$\nabla f = \begin{bmatrix} 4x_1 - 4 \\ 6x_2 - 6 \end{bmatrix}$$

Setting $\nabla f = 0$:

$$4x_1 - 4 = 0 \quad \Rightarrow \quad x_1 = 1$$

$$6x_2 - 6 = 0 \quad \Rightarrow \quad x_2 = 1$$

So, **the function is minimized at** $(1, 1)$.

Step 2: Check If It's a Global Minimum

To confirm this is a **global minimum**, we check the **Hessian matrix**:

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

For our function:

$$H = \begin{bmatrix} 4 & 0 \\ 0 & 6 \end{bmatrix}$$

Since:

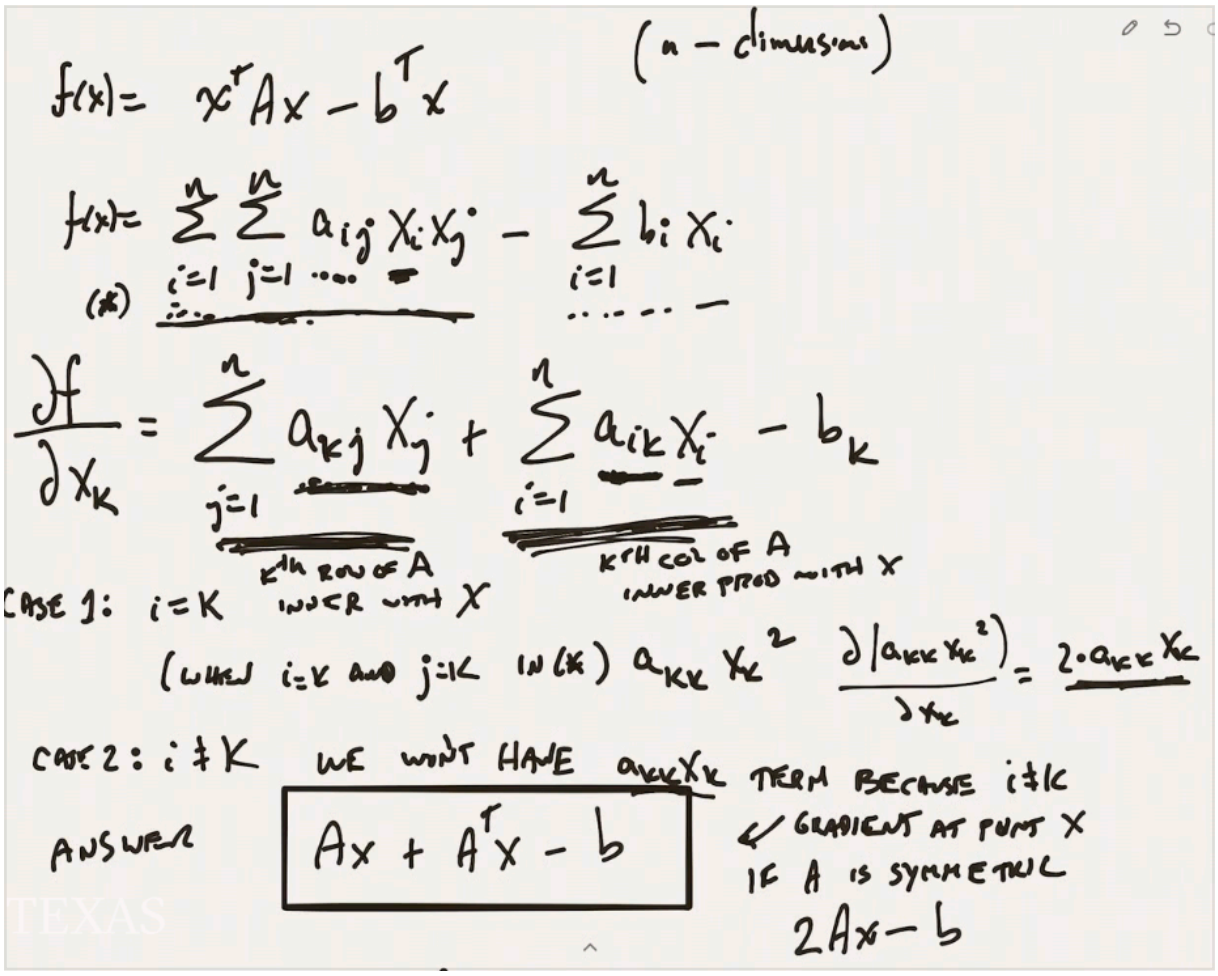
- The **diagonal elements are positive** $(4, 6)$.
- The **matrix is positive definite** (all eigenvalues are positive).

✅ **This confirms $(1, 1)$ is a global minimum!**

Intuition: Why Is This the Minimum?

- The function is **quadratic** (like a bowl).
- A **positive Hessian** means the bowl **curves upwards**.
- The gradient reaching **zero** means we are at the **bottom of the bowl**.

Thus, **we have reached the global minimum at $(1, 1)$.**



We are trying to **find the minimum** of a function that looks like this:

$$f(x) = x^T Ax - b^T x$$

This function is like a **big bowl (parabola in multiple dimensions)**, and we want to find the **lowest point** of the bowl.

- $x^T Ax$ means **the function has a curved shape (quadratic term)**.
- $-b^T x$ **shifts the function** so that the minimum is not necessarily at $(0, 0)$.
- Our goal: **Find the point where the function is lowest**.

Imagine you are **rolling a ball down a hill**.

- The **gradient** (derivative) tells you **which way the ball will roll**.
- We want to **follow the slope down to the lowest point**.

2️⃣ How Do We Find the Lowest Point?

We need to **find where the slope is zero** (where the ball stops rolling):

$$\frac{\partial f}{\partial x_k} = 0$$

Step 1: Compute the Gradient

The gradient (change of function with respect to x_k) is:

$$\frac{\partial f}{\partial x_k} = \sum_{j=1}^{n} a_{kj} x_j + \sum_{i=1}^{n} a_{ik} x_i - b_k$$

This means:

- Each x_k is affected by **all the other x values** through A.
- The function depends on the **sum of weighted terms**.

Final gradient equation:

$$Ax + A^T x - b$$

If A **is symmetric** (which is common in quadratic functions), then:

$$\nabla f = 2Ax - b$$

3️⃣ What Does This Mean in Simple Terms?

- Ax is a **linear transformation** of x.
- $A^T x$ is the **same transformation but flipped**.
- When we **set the gradient to zero**, we solve for x, which gives the **minimum**.

Imagine you're **balancing on a skateboard ramp** 🛹:

- The **steepest part** is where the gradient is **largest**.
- The **lowest point** is where the slope **is zero** (ball stops moving).
- This is **exactly what gradient descent does**—it helps us **roll towards the lowest point** step by step.

We need to **compute the gradient (derivative) of**:

$$f(x) = x^T Ax - b^T x$$

to find the **direction of steepest descent**. The goal is to **minimize** this function using **gradient descent**.

1️⃣ Understanding Each Term

We have two terms:

1. **Quadratic Term**: $x^T Ax$ → This represents a **curved shape** (like a bowl).
2. **Linear Term**: $-b^T x$ → This shifts the function.

To minimize $f(x)$, we compute its **gradient** $\nabla f(x)$, which is the derivative with respect to x.

Step 1: Derivative of $x^T Ax$

Let's expand the term:

$$x^T Ax = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j$$

Now, taking the **partial derivative** with respect to x_k:

$$\frac{\partial}{\partial x_k} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j$$

Each term contributes **twice** because of symmetry ($A_{ij} = A_{ji}$):

$$\sum_{j=1}^{n} a_{kj} x_j + \sum_{i=1}^{n} a_{ik} x_i = 2 \sum_{j=1}^{n} a_{kj} x_j$$

So,

$$\frac{\partial}{\partial x} (x^T Ax) = 2Ax$$

(If A is symmetric, otherwise it is $Ax + A^T x$).

Step 2: Derivative of $-b^T x$

Since:

$$b^T x = \sum_{i=1}^{n} b_i x_i$$

its derivative is simply:

$$\frac{\partial}{\partial x_k}(-b^T x) = -b_k$$

So the gradient of the full function:

$$\nabla f(x) = Ax + A^T x - b$$

If A is symmetric:

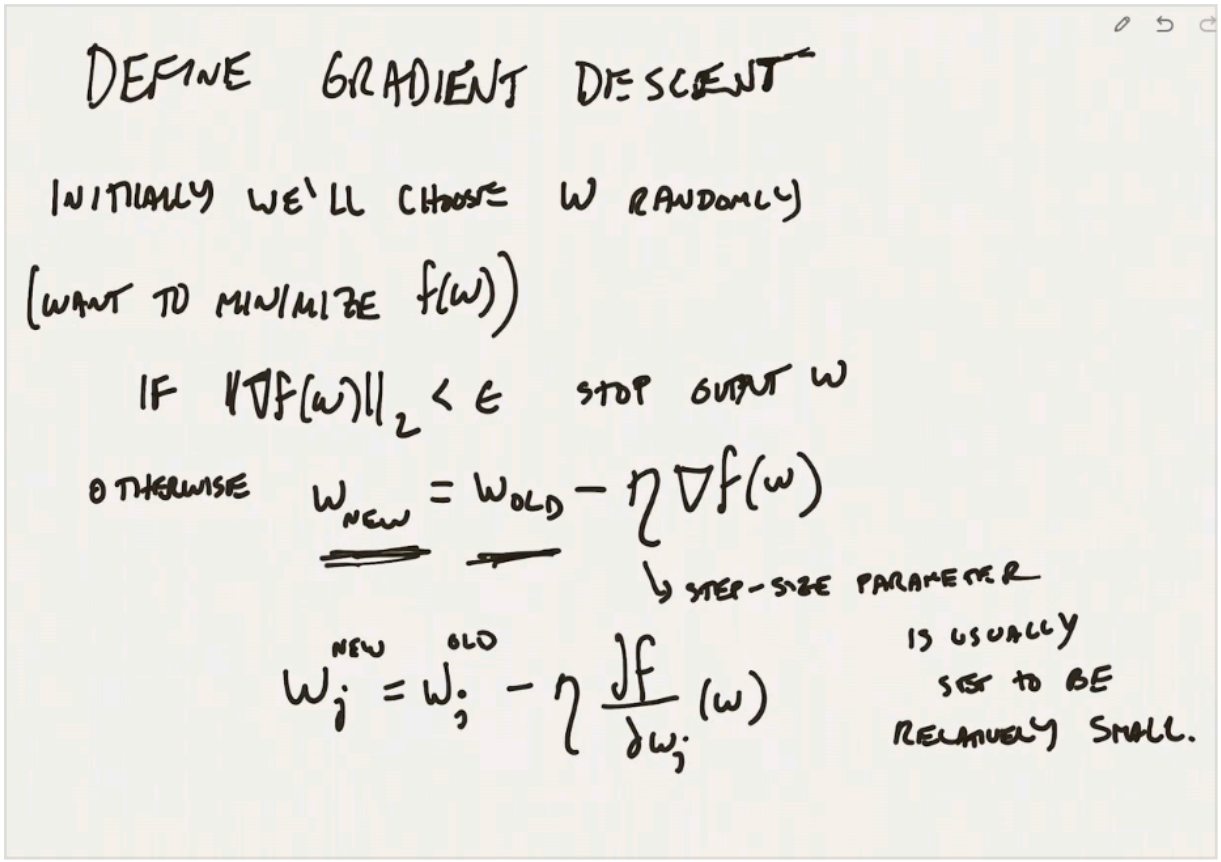
$$\nabla f(x) = 2Ax - b$$

3️⃣ Why Does This Matter?

- The gradient tells us the **direction of steepest increase**.
- To **minimize** $f(x)$, we follow **gradient descent**:

$$x_{\text{new}} = x_{\text{old}} - \eta \nabla f$$

- When $\nabla f(x) = 0$, we have found the **minimum**.



Gradient Descent is an **algorithm used to find the lowest point (minimum) of a function**. It's like finding the **deepest valley** when walking on a hilly surface **with your eyes closed**.

Let's break it down **step by step with simple examples and diagrams**.

1️⃣ What is Gradient Descent?

Gradient Descent helps us **minimize a function**. Imagine you are on a **mountain**, and you want to **reach the lowest point** (global minimum).

- Since you **can't see**, you feel the **steepest downward slope** and take a **small step in that direction**.
- If you repeat this process, you will eventually **reach the lowest point**.

This is exactly how **gradient descent works**:

- It starts at a **random point**.
- It **moves in the direction of the steepest descent** (negative gradient).
- It **stops when it reaches the minimum**.

🟢 **Mathematically, it follows this rule:**

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla f(w)$$

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla f(w)$$

where:

- w is our variable (weight, parameter, or input).
- η **(eta)** is the **step size** (how big our step is).
- $\nabla f(w)$ is the **gradient** (direction of the steepest slope).
- **If the gradient becomes very small**, we stop (we are near the minimum).

2️⃣ Example: Rolling a Ball Down a Hill

Imagine you drop a **ball on a hill**. The ball will:

1. **Roll down in the steepest direction**.
2. **Slow down as it reaches the lowest point**.
3. **Stop when it reaches the bottom**.

This is exactly how gradient descent **optimizes functions** in machine learning and optimization.

🔹 **If steps are too big (η is large)** → The ball jumps too far and may never stop.
🔹 **If steps are too small (η is tiny)** → The ball moves slowly and takes forever.

4️⃣ Example: Minimizing a Simple Function

Let's take a simple function:

$$f(w) = (w - 3)^2$$

This is a **parabola** with a minimum at $w = 3$.

Step 1: Compute the Gradient

$$\frac{df}{dw} = 2(w - 3)$$

Step 2: Gradient Descent Update

If we start at $w = 10$, we update:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot 2(w - 3)$$

If $\eta = 0.1$:

1. **Iteration 1:**

$$w = 10 - 0.1 \times 2(10 - 3) = 10 - 1.4 = 8.6$$

2. **Iteration 2:**

$$w = 8.6 - 0.1 \times 2(8.6 - 3) = 8.6 - 1.12 = 7.48$$

3. **Iteration 3:**

$$w = 7.48 - 0.1 \times 2(7.48 - 3) = 7.48 - 0.896 = 6.584$$

📉 **We keep moving closer to $w = 3$, the minimum!**

5️⃣ When Do We Stop?

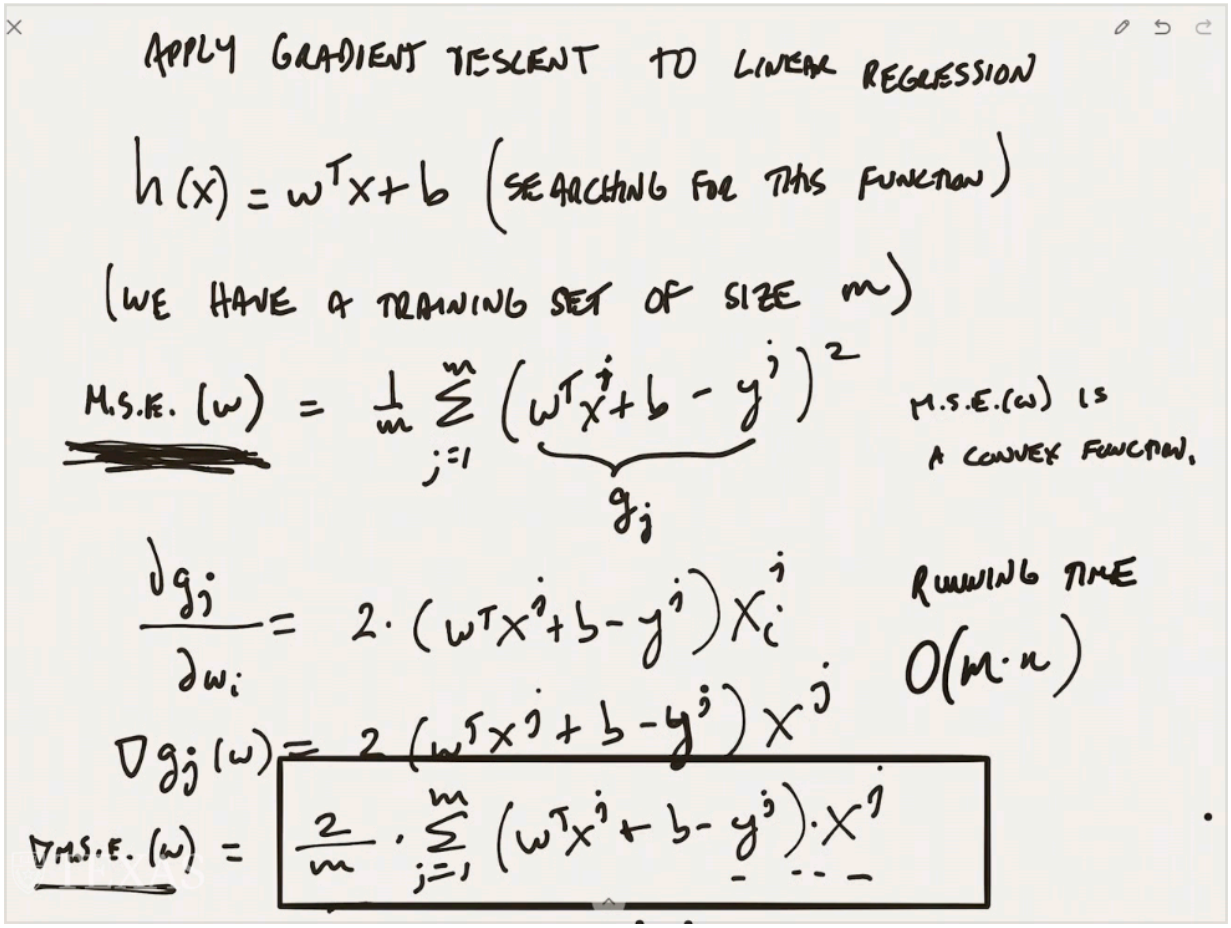
We stop when the **gradient is close to zero**, meaning **we are at the bottom of the valley**.

Stopping Condition:

$$||\nabla f(w)||_2 < \epsilon$$

where:

- $||\nabla f(w)||_2$ is the gradient size.
- ϵ is a small number (like 0.001).
- If the gradient is **smaller than** ϵ, we stop.



Explaining Gradient Descent for Linear Regression in Simple Terms 🚀

This image explains how **gradient descent is applied to train a linear regression model**. Let's break it down **step by step with examples and diagrams**. 🎨

1️⃣ What Are We Trying to Do?

We are **trying to find the best straight-line equation** to **predict** values.

A linear equation looks like:

$$h(x) = w^T x + b$$

Where:

- $h(x)$ is our **predicted value** (like house price).
- x is the **input data** (like house size).
- w are the **weights (slopes)** we are trying to find.
- b is the **bias** (intercept).

🔷 **Goal:** Adjust w and b to minimize the prediction error.

2️⃣ How Do We Measure Error?

We use **Mean Squared Error (MSE)** to measure how far our predictions are from actual values:

$$MSE(w) = \frac{1}{m} \sum_{j=1}^{m} (w^T x^j + b - y^j)^2$$

Where:

- m is the **number of training points**.
- $(w^T x^j + b)$ is the **prediction for point** j.
- y^j is the **actual value**.

🎯 **Smaller MSE = Better Model!**

3️⃣ How Do We Minimize Error?

We use **gradient descent** to adjust w and b step by step.

💡 **Gradient = Slope of Error Function**

- If **gradient is positive** → decrease w.
- If **gradient is negative** → increase w.

To update w, we take the **derivative of MSE** with respect to w:

$$\frac{\partial g_j}{\partial w_i} = 2(w^T x^j + b - y^j)x_i^j$$

This means:

🔷 **Error depends on how far the prediction is from the actual value.**

🔷 **Larger error = bigger adjustment to w.**

5️⃣ Updating w Using Gradient Descent

To update our weights w:

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla MSE(w)$$

Using the gradient:

$$\nabla MSE(w) = \frac{2}{m} \sum_{j=1}^{m} (w^T x^j + b - y^j) x^j$$

where:

- η **is the learning rate** (step size).
- **Summation means we update using all training points.**

6️⃣ Example: Predicting House Prices

Let's say we have:

House Size (sq ft) x	House Price ($1000s) y
1000	250
1500	350
2000	450

We want to fit a line:

$$\text{Price} = w \times \text{Size} + b$$

Step 1: Initialize Randomly

Let's start with **random values** $w = 0.1, b = 0.1$.

Step 2: Compute Gradient

For **each training point**:

1. Compute **prediction** $h(x) = w \times x + b$.
2. Compute **error** $(h(x) - y)$.
3. Compute **gradient** $\nabla MSE(w)$.
4. **Update** w **and** b.

Step 3: Iterate Until the Error is Small

We repeat **1000 times** until the error is **small enough**.

7️⃣ Running Time Complexity

Gradient descent runs in:

$$O(m \cdot n)$$

where:

- m = number of training points.
- n = number of features.

For **large datasets**, we use **stochastic gradient descent (SGD)** to speed things up.

8️⃣ Summary

✅ **We are finding the best line $h(x) = w^T x + b$.**

✅ **We measure error using MSE** (Mean Squared Error).

✅ **We compute the gradient to adjust w and b.**

✅ **We update w using gradient descent.**

✅ **We repeat until error is small.**

Few examples

We have the student scores matrix X:

$$X = \begin{bmatrix} 90 & 80 & 85 \\ 70 & 75 & 80 \\ 60 & 85 & 95 \end{bmatrix}$$

Each **row** represents a **student** (Alice, Bob, Charlie),

Each **column** represents a **subject** (Math, Science, English).

🔹 **Step 1: Compute the Column Sums**

To find the **average score per subject**, we need to **sum each column** and divide by 3:

$$\text{Math Sum} = 90 + 70 + 60 = 220$$

$$\text{Science Sum} = 80 + 75 + 85 = 240$$

$$\text{English Sum} = 85 + 80 + 95 = 260$$

So, the **average per subject** is:

$$\text{Math Avg} = \frac{220}{3} = 73.3$$

$$\text{Science Avg} = \frac{240}{3} = 80$$

$$\text{English Avg} = \frac{260}{3} = 86.6$$

2️⃣ Using Matrix Transpose to Do This Faster 🚀

Instead of summing **each column manually**, let's **transpose** the matrix first.

Step 1: Compute the Transpose X^T

We flip rows into columns:

$$X^T = \begin{bmatrix} 90 & 70 & 60 \\ 80 & 75 & 85 \\ 85 & 80 & 95 \end{bmatrix}$$

Step 2: Multiply by a Summation Vector

We can multiply this by a **summation vector** v to compute the sums efficiently:

$$v = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Now, compute:

$$X^T v = \begin{bmatrix} 90 & 70 & 60 \\ 80 & 75 & 85 \\ 85 & 80 & 95 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

This results in:

$$X^T v = \begin{bmatrix} (90 + 70 + 60) \\ (80 + 75 + 85) \\ (85 + 80 + 95) \end{bmatrix} = \begin{bmatrix} 220 \\ 240 \\ 260 \end{bmatrix}$$

Finally, we divide by the number of students (3) to get the **averages**.

$$\frac{1}{3} X^T v = \begin{bmatrix} \frac{220}{3} \\ \frac{240}{3} \\ \frac{260}{3} \end{bmatrix} = \begin{bmatrix} 73.3 \\ 80 \\ 86.6 \end{bmatrix}$$

✅ **This method does everything in one step!**

In **gradient descent**, we compute:

$$\nabla f(w) = \frac{2}{m} X^T (Xw - y)$$

Step 1: Define Our Matrix Variables

Let's say we are predicting house prices with:

Size x_1	Bedrooms x_2	Floors x_3	Price y
1000	2	1	250
1500	3	2	350
2000	4	2	450

$$X = \begin{bmatrix} 1000 & 2 & 1 \\ 1500 & 3 & 2 \\ 2000 & 4 & 2 \end{bmatrix}$$

$$y = \begin{bmatrix} 250 \\ 350 \\ 450 \end{bmatrix}$$

Assume **initial weights** $w = [0.1, 0.1, 0.1]$.

Compute Xw:

$$Xw = \begin{bmatrix} 1000 & 2 & 1 \\ 1500 & 3 & 2 \\ 2000 & 4 & 2 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 1000(0.1) + 2(0.1) + 1(0.1) \\ 1500(0.1) + 3(0.1) + 2(0.1) \\ 2000(0.1) + 4(0.1) + 2(0.1) \end{bmatrix}$$

$$= \begin{bmatrix} 100 + 0.2 + 0.1 \\ 150 + 0.3 + 0.2 \\ 200 + 0.4 + 0.2 \end{bmatrix} = \begin{bmatrix} 100.3 \\ 150.5 \\ 200.6 \end{bmatrix}$$

Now, compute the **error** $Xw - y$:

$$Xw - y = \begin{bmatrix} 100.3 \\ 150.5 \\ 200.6 \end{bmatrix} - \begin{bmatrix} 250 \\ 350 \\ 450 \end{bmatrix}$$

$$= \begin{bmatrix} -149.7 \\ -199.5 \\ -249.4 \end{bmatrix}$$

Step 3: Compute the Gradient $\nabla f(w)$ Using Transpose

$$X^T(Xw - y)$$

$$\begin{bmatrix} 1000 & 1500 & 2000 \\ 2 & 3 & 4 \\ 1 & 2 & 2 \end{bmatrix} \begin{bmatrix} -149.7 \\ -199.5 \\ -249.4 \end{bmatrix}$$

Each element is computed as:

1. **First row (sum of all size contributions)**:

   $$1000(-149.7) + 1500(-199.5) + 2000(-249.4) = -149700 - 299250 - 498800 = -947750$$

2. **Second row (sum of bedroom contributions)**:

   $$2(-149.7) + 3(-199.5) + 4(-249.4) = -299.4 - 598.5 - 997.6 = -1895.5$$

3. **Third row (sum of floor contributions)**:

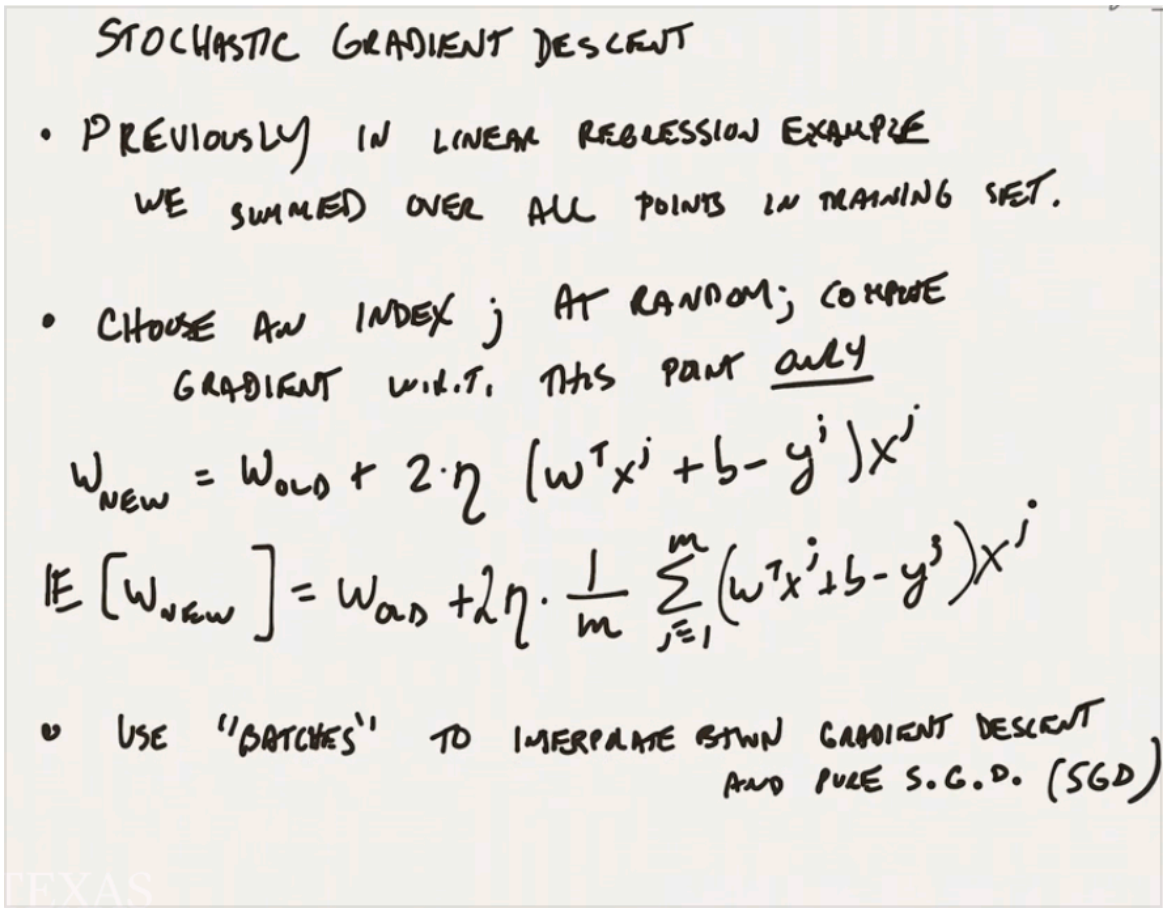
   $$1(-149.7) + 2(-199.5) + 2(-249.4) = -149.7 - 399 - 498.8 = -1047.5$$

$$X^T(Xw - y) = \begin{bmatrix} -947750 \\ -1895.5 \\ -1047.5 \end{bmatrix}$$

Final gradient:

$$\nabla f(w) = \frac{2}{m} X^T(Xw - y)$$

Stochastic Gradient Descent



Explaining Stochastic Gradient Descent (SGD) in Simple Terms 🚀

Gradient Descent is like **climbing down a hill** step by step **until you reach the lowest point**. But **Stochastic Gradient Descent (SGD)** is a **faster, more random way** to do it!

1️⃣ What's the Difference Between Normal Gradient Descent and SGD?

Gradient Descent (Full Batch) 📊

- Looks at **all data points at once** to compute the gradient.
- **Example**: If you are trying to guess the best price for houses, you look at **ALL houses** before adjusting your price prediction.

✅ **Accurate** but **slow**.

Stochastic Gradient Descent (SGD) 🎯

- **Instead of using all data, we pick one random data point** to update our model.
- **Example**: Instead of checking **all houses**, we pick **one house at random**, adjust our price prediction, and repeat.

✅ **Faster** but **more noisy** (random jumps).

2️⃣ What's Happening in the Formula?

Step 1: Regular Gradient Descent Update Rule

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla f(w)$$

- w = weights (parameters we are updating)
- η = learning rate (how big our step is)
- $\nabla f(w)$ = gradient (direction of steepest descent)

In **full gradient descent**, we compute:

$$\nabla f(w) = \frac{2}{m} \sum_{j=1}^{m} (w^T x^j + b - y^j) x^j$$

✅ **We sum over ALL training points (big dataset).**
❌ **Slow when data is large.**

Step 2: SGD Update Rule (Faster Way!)

Instead of summing over **all data points**, we **pick one random point** j:

$$w_{\text{new}} = w_{\text{old}} + 2\eta (w^T x^j + b - y^j) x^j$$

Key Difference: 🚀

- **Instead of computing the average gradient**, we update **using just one training example**.
- **Faster updates** but with **more variation**.

✅ **Much faster, especially for large datasets!**
❌ **More noise because it jumps around.**

Formula is minus

It should be $w_{new} = w_{old} - \ldots$ and $E[w_{new}] = w_{old} - \ldots$

3️⃣ Example: Predicting House Prices with SGD

Dataset:

House Size (sq ft)	Bedrooms	Floors	Price ($1000s)
1000	2	1	250
1500	3	2	350
2000	4	2	450

We are trying to learn:

$$\text{Price} = w_1 \times \text{Size} + w_2 \times \text{Bedrooms} + w_3 \times \text{Floors} + b$$

Step 1: Full Gradient Descent

- Look at **all houses**.
- Compute gradient **using all examples**.
- Adjust weights w_1, w_2, w_3.

🔷 **Slow but accurate.**

Step 2: Stochastic Gradient Descent

- Pick **one house at random**.
- Compute gradient **for that one house only**.
- Adjust weights **immediately**.

🔹 **Much faster!** But weights change a lot each step.

📉 **Visual Example of SGD:**

mathematica Copy Edit

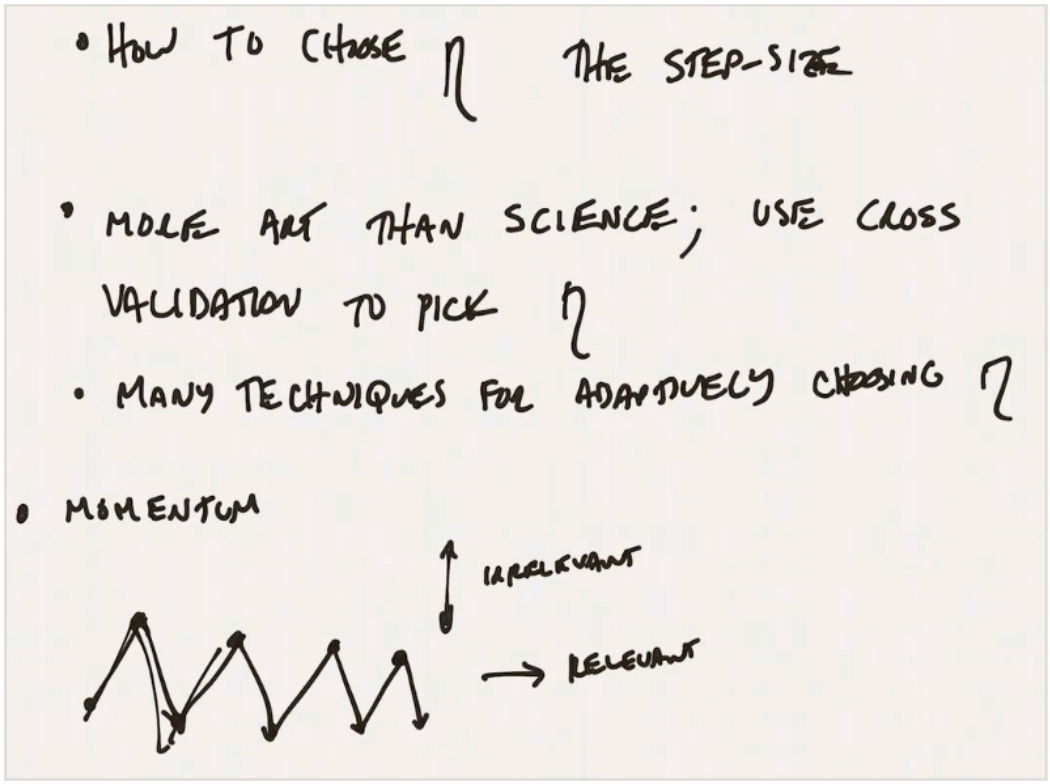
```
Full Gradient Descent:
⬇ Move down a little (smooth)
⬇ Move down a little (smooth)
⬇ Move down a little (smooth)

Stochastic Gradient Descent:
⬇ Big jump down
⬆ Random jump up
⬇ Big jump down
⬇ Small jump
```

🔹 **SGD moves faster but jumps around more.**

4️⃣ Using "Mini-Batches"

- **Full Gradient Descent** → Uses **all data**.
- **SGD** → Uses **one random point**.
- **Mini-Batch Gradient Descent** → Uses **a small group of points**.

✅ **Mini-batches are a balance** between speed and accuracy!



Understanding Learning Rate η and Momentum in Gradient Descent 🚀

This image explains **how to choose the learning rate η (step size)** and introduces **momentum** to improve gradient descent. Let's break it down **step by step with examples**.

1️⃣ What is the Learning Rate η?

The **learning rate** η is the **step size** that determines **how big of a step** we take in gradient descent.

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla f(w)$$

💡 **Choosing η correctly is important!**

Example: Walking Down a Hill ⛰️

Imagine you are **walking down a hill** to find the lowest point.

- **If η is too big** 🚀
  - You take **huge steps** and might **overshoot** or miss the valley.
  - You might **never reach the lowest point**.
- **If η is too small** 🐌
  - You take **tiny steps** and move very slowly.
  - It takes **forever to reach the bottom**.

📉 Effect of Different Learning Rates

```scss
Too Big (η = 1.0) → 🚀 Overshoots
Too Small (η = 0.0001) → 🐌 Slow Learning
Just Right (η = 0.1) → ✅ Smooth Descent
```

✅ The best learning rate helps the model learn fast without overshooting!

2️⃣ How Do We Pick η?

The image says:

> "More art than science; use cross-validation to pick η."

What Does This Mean?

- There is **no exact formula** for choosing η.
- We **test different values** and see which one works best.
- **Cross-validation** is used to try different η values on small data to see which performs best.

3️⃣ Why Momentum is Needed?

In **gradient descent**, the updates can **oscillate (zig-zag) a lot**, especially when the landscape has **steep slopes**.

The diagram in the image shows this:

- **Up and down motion (irrelevant directions) → Wastes time!**
- **Smooth progress toward the goal (relevant direction) → Better!**

Momentum Helps Solve This!

Momentum helps **smooth out** the updates by **keeping track of past gradients**.

Momentum Update Rule:

$$v_{\text{new}} = \beta v_{\text{old}} - \eta \nabla f(w)$$

$$w_{\text{new}} = w_{\text{old}} + v_{\text{new}}$$

Where:

- v is the **velocity** (previous update speed).
- β is the **momentum factor** (usually **0.9**).
- η is the **learning rate**.
- $\nabla f(w)$ is the **gradient**.

4️⃣ Example: Rolling a Ball Down a Slope 🛤️

Imagine you **push a ball down a hill**:

- Without momentum → The ball **stops and starts again**, **zig-zagging** down.
- With momentum → The ball **gains speed** and moves **smoothly to the bottom**.

Effect of Momentum

🟢 **Without Momentum (Normal GD)**

```scss
⬇ Move Left
⬆ Move Right
⬇ Move Left (slow!)
```

🔴 **With Momentum**

```css
→ Moves smoother in one direction! 🚀
```

Copy Edit

✅ **Momentum helps speed up learning by reducing oscillations!**

5️⃣ Summary

- ✓ Learning Rate η controls step size.
- ✓ Too big → Overshoots, Too small → Too slow.
- ✓ Momentum helps reduce zig-zag and speeds up convergence.
- ✓ Best η is found by testing different values (cross-validation).
- ✓ Using both learning rate & momentum gives better results.

MOMENTUM HAS A "VELOCITY" VARIABLE ✓

$V_0 = 0$ $\qquad V_1 = -\eta g_i$

$V_i = \alpha \cdot V_{i-1} - \eta g_i$

THIS TAKES A WEIGHTED MOVING AVERAGE OF $-\eta g_i$'S

EXPONENTIAL " "

$W_{NEW} = W_{OLD} + V_i$

ACCELERATED GRADIENT DESCENT

Explaining Momentum in Gradient Descent with Simple Examples 🚀

This image describes **Momentum-Based Gradient Descent**, which is an **improvement over standard gradient descent**. Let's break it down **step by step with examples** so you understand **why** it's useful!

1️⃣ What is Momentum in Gradient Descent?

Momentum **helps smooth out updates** so that we can move towards the minimum **faster and without oscillations**.

🔹 **Regular Gradient Descent**:

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla f(w)$$

- Updates depend **only on the current gradient**.
- If the gradient keeps changing direction, it **zig-zags and slows down**.

🔹 **Momentum-Based Gradient Descent**:

$$v_i = \alpha v_{i-1} - \eta g_i$$

$$w_{\text{new}} = w_{\text{old}} + v_i$$

- v **is velocity** (tracks past updates).
- α **is momentum factor** (usually 0.9).
- The update considers **past gradients** to move smoothly.

2️⃣ What Do These Equations Mean?

Step 1: Initialize Velocity

We start with **no velocity**:

$$v_0 = 0$$

This means at the **first step**, it behaves just like regular gradient descent.

Step 2: Compute Velocity Update

$$v_i = \alpha v_{i-1} - \eta g_i$$

- v_{i-1} **stores past updates** (momentum).
- α **controls how much of past updates we keep** (e.g., 0.9 means 90% past, 10% new update).
- ηg_i **is the normal gradient step**.

Step 3: Update Weights Using Velocity

$$w_{\text{new}} = w_{\text{old}} + v_i$$

- Instead of moving **only by the gradient**, we move using **velocity**.
- This helps **smooth out jumps and speeds up convergence**.

3️⃣ Example: Rolling a Ball Down a Hill 🎢

Imagine you are rolling a **ball down a hill**.

Without Momentum:

- The ball stops **every step**, checks the slope, and moves.
- If the path is zig-zag, it **keeps changing direction**, making **slow progress**.

With Momentum:

- The ball **remembers past motion** and keeps rolling.
- If the path is **bumpy**, it won't stop at every little hill—it keeps going!

Mathematical Example

Let's say we are minimizing:

$$f(w) = w^2$$

Without Momentum (Regular Gradient Descent)

Let's start at $w = 5$ and set $\eta = 0.1$.

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla f(w)$$

$$w = 5 - 0.1(2 \times 5) = 4$$

Next step:

$$w = 4 - 0.1(2 \times 4) = 3.2$$

It takes **many small steps** to reach 0.

With Momentum

Now, let's set $\alpha = 0.9$ and use momentum.

1️⃣ **Initialize:** $v = 0$

2️⃣ **First step:**

$$v_1 = 0.9(0) - 0.1(2 \times 5) = -1$$

$$w = 5 + (-1) = 4$$

3️⃣ **Second step:**

$$v_2 = 0.9(-1) - 0.1(2 \times 4) = -1.8$$

$$w = 4 + (-1.8) = 2.2$$

4️⃣ **Third step:**

$$v_3 = 0.9(-1.8) - 0.1(2 \times 2.2) = -2.42$$

$$w = 2.2 + (-2.42) = -0.22$$

🎯 **With momentum, we reach the minimum much faster!**

4️⃣ Why Does Momentum Help?

✅ **Faster Convergence** → Reaches the minimum quicker.

✅ **Avoids Zig-Zagging** → Smoothens updates in bumpy areas.

✅ **Helps Escape Local Minima** → Doesn't get stuck in small dips.

📉 **Comparison of Gradient Descent vs. Momentum:**

```sql
Without Momentum: ⬇⬆⬇⬆⬇⬆ (zig-zag, slow progress)
With Momentum: ➡➡➡⬇ (smoother, faster)
```

5️⃣ Summary

✔ Momentum introduces velocity to gradient descent.

✔ Reduces oscillations and speeds up convergence.

✔ Equation: $v_i = \alpha v_{i-1} - \eta g_i$, then update $w_{\text{new}} = w_{\text{old}} + v_i$.

✔ Used in deep learning optimizers (SGD with Momentum, Adam).