Gradient Descent is an iterative optimization algorithm used to find the minimum of a function, typically a cost or objective function $J(\theta)$ in machine learning. It works by repeatedly adjusting the model's parameters (θ) in the direction opposite to the gradient of the function.

Here's a detailed explanation:

Purpose: Gradient Descent aims to minimize an objective function, $J(\theta)$, by iteratively updating the parameters θ. For instance, in linear regression, it minimizes the cost function J to find the optimal parameters (θ■, θ■) that best fit the training data.

The Gradient: The gradient of the function $J(\theta)$ (denoted as $\nabla\theta J(\theta)$) is a vector of its partial derivatives with respect to each parameter θj. It points in the direction of the steepest ascent of the function.

The Update Rule: To minimize the function, Gradient Descent moves in the opposite direction of the gradient. The general update rule for parameters θ at step (t+1) is:

$\theta(t+1) = \theta(t) - \alpha \nabla\theta J(\theta(t))$

Where:

$\theta(t+1)$ are the updated parameters.

$\theta(t)$ are the current parameters.

α (alpha) is the **learning rate**, a hyperparameter that determines the size of the step taken in each iteration.

$\nabla\theta J(\theta(t))$ is the gradient of the objective function J with respect to θ, evaluated at the current parameters $\theta(t)$.

Batch Gradient Descent:

Mechanism: This specific variant of Gradient Descent calculates the gradient using *every example in the entire training set* on every single step.

Gradient Calculation: For a cost function J, the quantity in the summation of its update rule is equivalent to $\partial J(\theta)/\partial\theta j$. For linear regression with a loss function $J(\theta) = (1/2)\ \Sigma(h\theta(x(i)) - y(i))^2$, the update rule for each parameter θj is:

$\theta j(t+1) = \theta j(t) - \alpha\ \Sigma(i=1 \text{ to } n)\ (h\theta(x(i)) - y(i)) * x(i)j$

Where 'n' is the total number of training examples.

Accuracy: Batch Gradient Descent provides more accurate gradients because it considers all training examples.

Disadvantage: It can be computationally expensive and difficult to compute all activations for all examples in a single forward or backward propagation phase, especially with very large datasets.

Properties and Convergence:

Local Minima: While Gradient Descent can generally be susceptible to getting stuck in local minima, for certain problems like linear regression with a squared error cost function, the optimization problem has only one global optimum and no other local optima.

Convexity: For linear regression, the cost function J is a convex quadratic function. This property ensures that Gradient Descent, assuming the learning rate α is not too large, will always converge to the global minimum.

Example: The context illustrates Gradient Descent minimizing a quadratic function, showing its trajectory towards the minimum.

Variations (as alternatives to Batch Gradient Descent):

Stochastic Gradient Descent (SGD): Instead of using all examples, SGD picks *one data point* (x(i)) at a time to compute the gradient and update the parameters. Its update rule for linear regression is:

$\theta(t+1) = \theta(t) - \alpha\ (h\theta(x(i)) - y(i)) * x(i)$

SGD's gradients can be noisy, but it attempts to approximate the gradient from full gradient descent.

Mini-Batch Gradient Descent: This is a compromise between Batch Gradient Descent and Stochastic Gradient Descent. It computes the gradient using a small subset of the training data, called a "mini-batch" (B examples), at each step. The cost function Jmb for a mini-batch is defined as Jmb = $(1/B)\ \Sigma(i=1 \text{ to } B)\ L(i)$, where L(i) is the loss for a single example in the mini-batch. This method is widely used in practice due to its balance of computational efficiency and gradient accuracy.

In summary, Gradient Descent is a fundamental optimization algorithm that iteratively adjusts parameters by moving in the direction opposite to the gradient of the cost function. Its batch variant uses the entire dataset for each update, ensuring accurate gradients but potentially high computational cost, while variations like SGD and Mini-Batch Gradient Descent offer compromises for efficiency.