Gradient Descent: Explanation, Application, and Example Code

# 1. Explanation of Gradient Descent

Gradient Descent (GD) is an optimization algorithm used to minimize a loss function J(θ) by iteratively moving in the opposite direction of the gradient. It is widely used in machine learning for training models.

## Step 1: Initialization

Choose an initial parameter vector θ (e.g., all zeros or small random values).

## Step 2: Forward Pass (Prediction and Loss Calculation)

Compute predictions ŷ = f(X; θ). Then calculate the loss function, e.g. Mean Squared Error (MSE): J(θ) = (1/(2m)) Σ (ŷᵢ - yᵢ)².

## Step 3: Gradient Computation

Compute the gradient vector of the loss with respect to θ. For linear regression: ∇θ J(θ) = (1/m) Xᵀ(Xθ - y).

## Step 4: Parameter Update

Update the parameters using: θ := θ - α ∇θ J(θ), where α is the learning rate.

## Step 5: Iteration

Repeat steps 2–4 for many epochs until convergence.

## Step 6: Stopping Criteria

Stop when one of these holds:  
- Max epochs reached  
- Change in loss is below a tolerance  
- Gradient magnitude is small  
- Early stopping on validation set

# 2. Application in Linear Regression

Gradient Descent is commonly used in Linear Regression to find the optimal parameters w and b that minimize the Mean Squared Error. It is especially useful for large datasets where closed-form solutions are computationally expensive.

# 3. Example Code in Python (NumPy)

Below is an implementation of Gradient Descent for Linear Regression:

import numpy as np  
  
def gradient\_descent\_linear\_regression(X, y, lr=0.1, epochs=1000, tol=1e-7, verbose=False):  
 m, n = X.shape  
 # Add bias column  
 X\_b = np.hstack([np.ones((m, 1)), X])   
 theta = np.zeros(n + 1)  
 loss\_history = []  
  
 for epoch in range(epochs):  
 preds = X\_b.dot(theta)  
 errors = preds - y  
 loss = (1.0 / (2 \* m)) \* np.sum(errors \*\* 2)  
 loss\_history.append(loss)  
  
 grad = (1.0 / m) \* (X\_b.T.dot(errors))  
 theta = theta - lr \* grad  
  
 if epoch > 0 and abs(loss\_history[-2] - loss\_history[-1]) < tol:  
 break  
 return theta, loss\_history  
  
# Example usage  
np.random.seed(42)  
m = 100  
X = 2 \* np.random.rand(m, 1)  
true\_w, true\_b = 3.0, 4.0  
y = true\_b + true\_w \* X[:, 0] + np.random.randn(m) \* 0.5  
  
theta\_est, losses = gradient\_descent\_linear\_regression(X, y, lr=0.1, epochs=5000, tol=1e-9)  
print("Estimated theta (b, w):", theta\_est)

# 4. Explanation of the Code

- Add bias column: Adds a column of ones for intercept.  
- Initialize theta: Start with zeros.  
- Predictions: preds = X\_b.dot(theta).  
- Loss: Mean Squared Error (with 1/2 factor).  
- Gradient: grad = (1/m) X\_bᵀ (X\_bθ - y).  
- Update rule: θ = θ - lr \* grad.  
- Early stopping: Stop when loss improvement < tol.