Gradient Descent: An Optimization Algorithm

Introduction

- **Objective**: Minimize a cost function $J(w, b)$ by systematically adjusting parameters w and b.
- Gradient descent is widely used in machine learning for training models, including deep learning models.
- Applicable not only to linear regression but also to any function minimization problem.

Overview of Gradient Descent

- **Starting Point**:
  - Begin with an initial guess for parameters w and b (often set to zero in linear regression).
- **Iterative Process**:
  - Modify w and b step by step to reduce the cost function $J(w, b)$.
  - Continue updating parameters until reaching a minimum value of J.

Generalization to Multiple Parameters

- Gradient descent extends beyond two parameters:
  - For a cost function $J(w_1, w_2, ..., w_n, b)$, the goal is to minimize J over all parameters.
  - The method systematically updates each w_i and b to find the optimal values.

Intuition Behind Gradient Descent

- **Visualizing the Cost Function**:
  - Imagine the cost function as a surface plot where different values of w and b correspond to different heights.
  - High points represent higher cost values, and valleys represent lower cost values.
- **Descending to the Minimum**:
  - Start at an initial point on the cost surface.
  - Look for the steepest downward direction and take a small step.
  - Repeat this process iteratively to reach a local minimum.

Properties of Gradient Descent

- **Steepest Descent**:
  - At each step, move in the direction where the function decreases the most.
  - Ensures the fastest descent to a minimum.
- **Local Minima**:
  - Some functions have multiple minima.
  - The final minimum reached depends on the initial starting position.
  - If started in a different location, gradient descent might settle in a different valley (local minimum).

Implementing Gradient Descent

- **Update Rule**:
  - Gradient descent updates parameters w and b using the formulas:

$$w := w - \alpha \frac{d}{dw} J(w, b)$$

$$b := b - \alpha \frac{d}{db} J(w, b)$$

  - Here, α (learning rate) determines the step size.
  - The derivative terms indicate the direction and magnitude of the adjustment.

Effect of Learning Rate α

- **Too Small** α: Very slow convergence.
- **Too Large** α: Can overshoot and diverge.
- **At the Minimum**: Gradient is zero, so updates stop.

Feature Scaling for Faster Convergence

- Features with large ranges slow convergence.
- **Methods**:
  - **Min-Max Scaling**: $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - **Z-score Normalization**: $x' = \frac{x - \mu}{\sigma}$

Recognizing Gradient Descent Convergence

- **Learning Curve**: Plot cost vs. iterations.
- **Flat Cost Function**: Indicates convergence.

- **Threshold** ϵ: Stop if cost change is very small.

Feature Engineering and Polynomial Regression

Feature Engineering

- **What is Feature Engineering?**
  - Creating new features by transforming or combining existing ones.
  - Helps machine learning models make better predictions.
- **Example: House Price Prediction**
  - Given two features:
    - x_1 = width (frontage) of a land plot
    - x_2 = depth of a land plot
  - Basic model:

$$f(x) = w_1x_1 + w_2x_2 + b$$

  - **Better Approach**: Create a new feature x_3, where:

$$x_3 = x_1 \times x_2$$

  - New model:

$$f(x) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

  - x_3 represents the **area** of the land, which is more predictive.

Polynomial Regression

- **Extends Linear Regression**:
  - Instead of fitting straight lines, fits curves to the data.
- **Example**: Predicting house prices using **square footage** (x).
  - A linear model may not fit the data well.
  - **Quadratic Model**:

$$f(x) = w_1x + w_2x^2 + b$$

  - **Cubic Model**:

$$f(x) = w_1x + w_2x^2 + w_3x^3 + b$$

  - Higher-degree polynomials allow more flexibility.

Feature Scaling in Polynomial Regression

- **Why Important?**
  - Squaring/cubing features creates vastly different ranges.
  - Example:
    - x ranges from 1 to 1,000
    - x^2 ranges from 1 to 1,000,000
    - x^3 ranges from 1 to 1,000,000,000
  - Without scaling, gradient descent struggles.
- **Alternative Polynomial Features**
  - Instead of x^2 or x^3, use:
    - **Square Root**: $w_1 x + w_2 \sqrt{x} + b$
    - **Logarithm**: $w_1 x + w_2 \log(x) + b$
- **Choosing Features**
  - Use cross-validation to test different feature sets.
  - More features ≠ better model (avoid overfitting).

Tools for Polynomial Regression

- **Scikit-Learn**:
  - Implements polynomial regression with **few lines of code**.
  - Helps avoid reimplementing algorithms manually.
- **Why Learn Implementation?**
  - Understanding fundamentals helps in debugging models.
  - Avoid reliance on "black-box" libraries.

Conclusion

- **Gradient Descent**:
  - Optimizes cost function iteratively.
  - Choice of **learning rate** is critical.
  - Feature scaling speeds up convergence.
- **Feature Engineering**:
  - Creating new features improves model accuracy.
  - Example: Using **area** in house price prediction.

- **Polynomial Regression**:
  - Extends linear regression to model **curved** data.
  - Feature scaling is crucial for stability.
- **Machine Learning Practice**:
  - Implement manually to understand the math.
  - Use tools like **Scikit-Learn** for efficiency.

This version includes all the key points on **Feature Engineering and Polynomial Regression**. Let me know if you need any refinements! 🚀