**1. Thuật toán Gradient Descent**

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates.

**Thuật toán: Gradient Descent cho Hồi quy Tuyến tính Đa biến**

1. **Khởi tạo Tham số**:

Bắt đầu bằng cách khởi tạo các tham số mô hình (trọng số và sai số) với các giá trị ngẫu nhiên hoặc bằng không.

Đặt θ để biểu thị các tham số, trong đó θ\_0 tương ứng với thuật ngữ sai số và θ\_1, θ\_2, ..., θ\_n đại diện cho các hệ số cho các biến độc lập (thuộc tính).

1. **Xác định Hàm Chi phí**:

Chọn một hàm chi phí để đo hiệu suất của mô hình hồi quy tuyến tính. Đối với hồi quy tuyến tính đa biến, sai số bình phương trung bình (MSE) thường được sử dụng làm hàm chi phí:

* MSE(θ) = (1/2m) \* Σ(yi - h(xi))^2

Trong đó:

* Θ là vector tham số.
* m là số lượng ví dụ huấn luyện.
* xi là vector giá trị thuộc tính cho ví dụ thứ i.
* yi là giá trị mục tiêu thực tế cho ví dụ thứ i.
* h(xi) là giá trị mục tiêu dự đoán bằng cách sử dụng các tham số mô hình hiện tại.

1. **Cập nhật Tham số**:

* Lặp lại các bước sau cho đến khi hội tụ:
* Tính gradient của hàm chi phí đối với từng tham số:

∂(MSE(θ))/∂θ\_j = (1/m) \* Σ(yi - h(xi)) \* xij

* Cập nhật từng tham số θ\_j bằng cách sử dụng gradient và một tốc độ học (α):

θ\_j = θ\_j - α \* ∂(MSE(θ))/∂θ\_j cho tất cả j

Bước này liên quan đến việc điều chỉnh các tham số theo hướng giảm hàm chi phí.

**Hội tụ**: Kiểm tra một tiêu chí hội tụ, chẳng hạn như đạt đến một số lượng tối đa vòng lặp hoặc quan sát một sự thay đổi nhỏ trong hàm chi phí qua các vòng lặp. Nếu đạt tiêu chí, dừng thuật toán.

**Sử dụng Mô hình**: Sau khi thuật toán hội tụ, bạn có thể sử dụng các giá trị cuối cùng của θ để thực hiện dự đoán cho các điểm dữ liệu mới:

Trong đó, x\_1, x\_2, ..., x\_n là giá trị thuộc tính của điểm dữ liệu mới.

**Hyperparameters**: Thử nghiệm với các tốc độ học (α) và các tiêu chí hội tụ khác nhau để tìm sự kết hợp tốt nhất cho vấn đề cụ thể của bạn.

## Mã Python cho Gradient Descent

## Đã chạy thử

## import numpy as np

## # Khởi tạo dữ liệu mẫu (X: ma trận đặc trưng, y: vector đầu ra)

## X = np.array([[1, 2, 3],

## [4, 5, 6],

## [7, 8, 9]])

## y = np.array([10, 20, 30])

## # Số lượng mẫu và số lượng đặc trưng

## m, n = X.shape

## # Khởi tạo các tham số (theta) ban đầu

## theta = np.zeros(n + 1)

## # Siêu tham số

## alpha = 0.01

## iterations = 1000

## # Thêm cột 1 vào ma trận X cho bias term

## X = np.column\_stack((np.ones(m), X))

## # Gradient Descent

## for \_ in range(iterations):

## # Tính dự đoán

## predictions = np.dot(X, theta)

## # Tính sai số

## errors = predictions - y

## # Cập nhật tham số theta

## theta -= (alpha / m) \* np.dot(X.T, errors)

## # Sử dụng các tham số cuối cùng cho việc dự đoán

## X\_new = np.array([[1, 2, 3]]) # Dữ liệu mới cần dự đoán

## X\_new = np.column\_stack((np.ones(1), X\_new)) # Thêm cột 1 cho bias term

## final\_predictions = np.dot(X\_new, theta)

## print("Tham số theta sau khi học:")

## print(theta)

## print("Dự đoán cho dữ liệu mới:")

## print(final\_predictions)

**3. Các phương pháp để Xác định Tốc độ Học**

Chọn một tốc độ học (α) phù hợp là rất quan trọng để thuật toán Gradient Descent thành công. Dưới đây là một số phương pháp phổ biến để xác định tốc độ học:

* **Grid Search**: Bạn có thể thực hiện tìm kiếm lưới qua một phạm vi của tốc độ học và chọn tốc độ học cho ra mô hình hiệu suất tốt nhất trên tập dữ liệu xác thực.
* **Learning Rate Schedules**: Sử dụng các lịch trình tốc độ học thích ứng tốc độ học trong quá trình huấn luyện. Hai lịch trình phổ biến là:
  + **Fixed Schedule**: Giữ một tốc độ học cố định trong suốt quá trình huấn luyện.
  + **Step Decay Schedule**: Giảm tốc độ học bằng một hệ số (ví dụ: 0.1) sau một số vòng lặp hoặc số lượng epoch cố định.
* **Learning Rate Annealing**: Bắt đầu với một tốc độ học tương đối lớn và dần giảm nó khi quá trình huấn luyện diễn ra. Điều này có thể giúp thuật toán hội tụ nhanh ban đầu và điều chỉnh sau này.
* **Adaptive Learning Rate Methods**: Các thuật toán như Adam và RMSprop tự động điều chỉnh tốc độ học dựa trên gradient quá khứ, giảm cần thiết cho việc điều chỉnh thủ công.
* **Validation Set**: Sử dụng tập dữ liệu xác thực để theo dõi hiệu suất của mô hình với các tốc độ học khác nhau và chọn tốc độ học mà giảm thiểu sai số trên tập dữ liệu xác thực.