Gradient Descent –Differentiation Algorithms

Gradient descent is the backbone of the learning process for various algorithms, including linear regression, logistic regression, support vector machines, and neural networks which serves as a fundamental optimization technique to minimize the cost function of a model by **iteratively adjusting the model parameters to reduce the difference between predicted and actual values, improving the model's performance**. Let's see it's role in machine learning:

Prerequisites: Understand the working and math of gradient descent.

1. Training Machine Learning Models

Neural networks are trained using Gradient Descent (or its variants) in combination with backpropagation. Backpropagation computes the gradients of the **loss function with respect to each parameter (weights and biases) in the network by applying the chain rule.** The process involves:

- **Forward Propagation**: Computes the output for a given input by passing data through the layers.
- **Backward Propagation**: Uses the chain rule to calculate gradients of the loss with respect to each parameter (weights and biases) across all layers.

Gradients are then used by Gradient Descent to update the parameters layer-by-layer, moving toward minimizing the loss function.

Neural networks often use advanced variants of Gradient Descent. If you want to read more about variants, please refer : Gradient Descent Variants.

2. Minimizing the Cost Function

The algorithm minimizes a cost function, which quantifies the error or loss of the model's predictions compared to the true labels for:

1. Linear Regression

Gradient descent minimizes the Mean Squared Error (MSE) which serves as the loss function to find the best-fit line. Gradient Descent is used to iteratively update the weights (coefficients) and bias by computing the gradient of the MSE with respect to these parameters.

Since MSE is a convex function **gradient descent guarantees convergence to the global minimum if the learning rate is appropriately chosen.** For each iteration:

The algorithm computes the gradient of the MSE with respect to the weights and biases.

It updates the weights (w) and bias (b) using the formula:

- Calculating the gradient of the log-loss with respect to the weights.

- Updating weights and biases iteratively to maximize the likelihood of the correct classification:

$w=w-\alpha\cdot\partial J(w,b)\partial w, b=b-\alpha\cdot\partial J(w,b)\partial b$ $w=w-\alpha\cdot\frac{\partial}{\partial w}J(w,b), b=b-\alpha\cdot\frac{\partial}{\partial b}J(w,b)$

The formula is the **parameter update rule for gradient descent**, which adjusts the weights w and biases b to minimize a cost function. This process iteratively adjusts the line's slope and intercept to minimize the error.

2. Logistic Regression

In logistic regression, gradient descent minimizes the **Log Loss (Cross-Entropy Loss)** to optimize the decision boundary for binary classification. Since the output is probabilistic (between 0 and 1), the sigmoid function is applied. The process involves:

- Calculating the gradient of the log-loss with respect to the weights.
- Updating weights and biases iteratively to maximize the likelihood of the correct classification:

$w=w-\alpha\cdot\partial J(w)\partial w$ $w=w-\alpha\cdot\frac{\partial}{\partial w}J(w)$

This adjustment shifts the decision boundary to separate classes more effectively.

3. Support Vector Machines (SVMs)

For SVMs, gradient descent optimizes the **hinge loss**, which ensures a maximum-margin hyperplane. The algorithm:

- Calculates gradients for the hinge loss and the regularization term (if used, such as L2 regularization).
- Updates the weights to maximize the margin between classes while minimizing misclassification penalties with same formula provided above.

Gradient descent ensures the **optimal placement of the hyperplane to separate classes with the largest possible margin.**