BGD

Batch Gradient Descent (BGD) is a fundamental algorithm in numerical optimization, particularly useful for training machine learning models. It aims to minimize an objective function (often representing error) by iteratively adjusting the model's parameters in the direction that leads to the steepest descent.

Steps :

1. Initialization:

- The algorithm starts with initial values for all the model parameters (weights and biases) involved. These initial values can be random or chosen based on prior knowledge.

2. Forward Pass:

- The entire training dataset is fed through the model once using the current parameter values.
- For each data point, the model makes a prediction.

3. Error Calculation:

- The difference between the model's predictions and the actual target values (ground truth) in the training data is calculated. This difference is often quantified using an error function, such as mean squared error or cross-entropy.

4. **Gradient Calculation**:

- Compute the gradient of the cost function for the entire training dataset. The gradient is a vector that contains the partial derivatives of the cost function with respect to each parameter.
- The gradient essentially points in the direction of steepest ascent (for maximization) or steepest descent (for minimization) of the error function.

5. **Parameter Update**:

- Update the parameters in the direction that reduces the cost function. This is done by subtracting a fraction of the gradient from the current parameters. The fraction is determined by the learning rate, a hyperparameter that you set.

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \cdot \nabla_\theta J(\theta)$$

6. Iteration:

- Steps 2-5 are repeated for a specified number of epochs (complete passes through the entire training dataset).
- With each iteration, the model parameters are adjusted to minimize the error, gradually leading the model towards a better fit with the training data.

Here are some of the key problems associated with BGD:

1. Computational Cost:

- BGD requires processing the entire training dataset in each iteration to calculate the gradient. This can be very computationally expensive for large datasets, especially when dealing with complex models that have many parameters.
- The memory requirement to store the entire dataset and perform calculations can also be significant for very large datasets.

2. Slow Convergence:

- Although BGD provides a comprehensive view of the error landscape, it can sometimes be slow to converge, especially for problems with non-convex objective functions (functions with multiple minima).
- Since it updates parameters based on the entire dataset, it might take many iterations to navigate through shallow gradients or flat regions in the error surface.

3. Limited Applicability to Big Data:

- With the increasing size and complexity of datasets (Big Data), BGD becomes impractical due to computational limitations. The sheer volume of data can make processing the entire dataset in each iteration infeasible.