Gradient Decent

Gradient Descent is an optimization algorithm used in deep learning (DL) to minimize the **loss function**, which measures how far off a model's predictions are from the actual values.

Key Idea:

Think of the loss function as a hill or valley in a landscape. Gradient descent helps you find the **lowest point (minimum loss)** by taking steps downhill in the direction that reduces the loss most quickly.

How It Works (Step-by-Step):

1. **Initialize Weights**: Start with random weights in the neural network.
2. **Compute Loss**: Calculate how far the current predictions are from the correct outputs using a loss function (e.g., Mean Squared Error or Cross Entropy).
3. **Compute Gradients**: Use backpropagation to compute the **gradient** (partial derivative) of the loss with respect to each weight — this tells us the slope or direction of steepest ascent.
4. **Update Weights**: Move in the **opposite direction** of the gradient (steepest descent) by a small amount (called the **learning rate**):

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

Where:

- w: weight
- η: learning rate (e.g., 0.01)
- $\frac{\partial L}{\partial w}$: gradient of the loss function

5. **Repeat**: Iterate this process for many batches/epochs until the loss is minimized.

Types of Gradient Descent:

- **Batch Gradient Descent**: Uses the entire training dataset for each step.

- **Stochastic Gradient Descent (SGD)**: Uses one training sample at a time (faster but noisier).
- **Mini-Batch Gradient Descent**: Uses a small group of samples — a compromise between speed and stability.

✅ Pros of (Pure) Gradient Descent

1. **Deterministic Updates**
   - Since it uses the entire dataset to compute the gradient, each step is consistent and stable.
2. **Converges Smoothly (if learning rate is good)**
   - No noisy updates; follows a smooth path toward the minimum.
3. **Mathematically Rigorous**
   - The update rule is well-founded and aligns with theoretical optimization frameworks.

❌ Cons of (Pure) Gradient Descent

1. **Computationally Expensive**
   - Evaluating gradients over the **entire dataset** at each step can be slow, especially for large datasets.
2. **Memory-Intensive**
   - Requires loading the full dataset into memory to compute each update.
3. **Slow Convergence**
   - Because it updates weights only after processing the full dataset, progress can be very slow.
4. **Inflexible to Online Learning**
   - Cannot update the model incrementally with new data without reprocessing the whole dataset.
5. **Sensitive to Learning Rate**

    - Needs careful tuning; otherwise, it may converge very slowly or even diverge.

6. **May Get Stuck in Local Minima or Saddle Points**
    - Especially problematic in non-convex functions like those in deep learning.