Types of Gradient Descent

Gradient descent is a fundamental optimization algorithm in machine learning, used to minimize functions by iteratively moving towards the minimum. It's crucial for training models by fine-tuning parameters to reduce prediction errors.

Batch Gradient Descent

Batch Gradient Descent: Computes the gradient using the entire dataset before updating model parameters. It ensures stable convergence but can be slow for large datasets.

$\theta = \theta - \eta \nabla J(\theta)$

Advantages of Batch Gradient Descent

- **Stable Convergence:** Since the gradient is averaged over all training examples, the updates are less noisy and more stable.
- **Global View:** It considers the entire dataset for each update, providing a global perspective of the loss landscape.

Disadvantages of Batch Gradient Descent

- **Computationally Expensive:** Processing the entire dataset in each iteration can be slow and resource-intensive, especially for large datasets.
- **Memory Intensive:** Requires storing and processing the entire dataset in memory, which can be impractical for very large datasets.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD): Updates model parameters after computing the gradient from a single random sample. It is faster but introduces high variance, making convergence noisy.

$$\theta = \theta - \eta \nabla J(\theta; x^{(i)}, y^{(i)})$$

Advantages

- **Faster Convergence:** Frequent updates can lead to faster convergence, especially in large datasets.
- **Less Memory Intensive:** Since it processes one training example at a time, it requires less memory compared to batch gradient descent.
- **Better for Online Learning:** Suitable for scenarios where data comes in a stream, allowing the model to be updated continuously.

Disadvantages

- **Noisy Updates:** Updates can be noisy, leading to a more erratic convergence path.
- **Potential for Overshooting:** The frequent updates can cause the algorithm to overshoot the minimum, especially with a high learning rate.
- **Hyperparameter Sensitivity:** Requires careful tuning of the learning rate to ensure stable and efficient convergence.

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent: Balances BGD and SGD by computing the gradient using a small batch of samples. It improves efficiency while maintaining some stability.

$$\theta = \theta - \eta \nabla J(\theta; \{x^{(i)}, y^{(i)}\}_{i=1}^{m})$$

Advantages of Mini-Batch Gradient Descent

- **Faster Convergence:** By using mini-batches, it achieves a balance between the noisy updates of SGD and the stable updates of Batch Gradient Descent, often leading to faster convergence.
- **Reduced Memory Usage:** Requires less memory than Batch Gradient Descent as it only needs to store a mini-batch at a time.
- **Efficient Computation:** Allows for efficient use of hardware optimizations and parallel processing, making it suitable for large datasets.

Disadvantages of Mini-Batch Gradient Descent

- **Complexity in Tuning:** Requires careful tuning of the mini-batch size and learning rate to ensure optimal performance.
- **Less Stable than Batch GD:** While more stable than SGD, it can still be less stable than Batch Gradient Descent, especially if the mini-batch size is too small.
- **Potential for Suboptimal Mini-Batch Sizes:** Selecting an inappropriate mini-batch size can lead to suboptimal performance and convergence issues.

Momentum-Based Gradient Descent

Momentum-Based Gradient Descent: Introduces a velocity term to smooth updates, accelerating learning in relevant directions and reducing oscillations.

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta_t) \qquad \theta_{t+1} = \theta_t - v_t$$

Advantages

- **Accelerated Convergence:** Helps in faster convergence, especially in scenarios with small but consistent gradients.
- **Smoother Updates:** Reduces the oscillations in the gradient updates, leading to a smoother and more stable convergence path.
- **Effective in Ravines:** Particularly effective in dealing with ravines or regions of steep curvature, common in deep learning loss landscapes.

Disadvantages

- **Additional Hyperparameter:** Introduces an additional hyperparameter (momentum term) that needs to be tuned.
- **Complex Implementation:** Slightly more complex to implement compared to standard gradient descent.
- **Potential Overcorrection:** If not properly tuned, the momentum can lead to overcorrection and instability in the updates.

AdaGrad (Adaptive Gradient Algorithm)

AdaGrad: Adjusts the learning rate for each parameter based on past gradients, improving convergence for sparse data but sometimes causing excessively small learning rates.

$$g_t = \nabla J(\theta_t) \qquad G_t = G_{t-1} + g_t^2 \qquad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

Advantages

- **Per-Parameter Learning Rate:** Adjusts the learning rate for each parameter based on its historical gradient, leading to more efficient convergence.
- **Effective for Sparse Data:** Particularly useful for sparse data, where some parameters receive updates infrequently.

Disadvantages

- **Learning Rate Decay:** The cumulative sum of squared gradients can grow without bound, causing the learning rate to decay and potentially stopping learning prematurely.
- **Memory Intensive:** Requires storing additional memory for each parameter's accumulated gradient, which can be intensive for large models.