A Gentle Introduction to Gradient Descent and Its Variants

Rishabh Iyer

March 19, 2024

1 Introduction to Gradient Descent

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. It is the cornerstone of optimization and one of the key pillars of machine learning. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point.

1.1 The Gradient Descent Algorithm

The algorithm can be summarized as follows:

1. Choose an initial guess x_0 and a learning rate (also called *step size*) α.
2. Update x to a new value x_{n+1} using the rule $x_{n+1} = x_n - \alpha \nabla f(x_n)$.
3. Repeat step 2 until x_n converges to a minimum or more practically, a stopping criteria (elaborated below) is met.

The algorithm iterates until a stopping criterion is met. Common stopping criteria include:

1. **Fixed Number of Epochs:** The algorithm stops after a predetermined number of iterations, or epochs, has been reached: N_{epochs}.

   $$\text{Stop if } n \geq N_{\text{epochs}}$$

   where n is the current epoch.

2. **Gradient Norm Threshold:** The algorithm stops when the norm of the gradient of the loss function falls below a small threshold ϵ, indicating that a minimum has been approached.

   $$\text{Stop if } ||\nabla f(x_n)|| < \epsilon$$

3. **Difference Between Parameter Vectors:** The algorithm stops when the difference between parameter vectors in subsequent iterations is less than a threshold ϵ, suggesting convergence.

$$\text{Stop if } ||x_{n+1} - x_n|| < \epsilon$$

4. **Difference Between Function Values:** The algorithm stops when the absolute difference between the loss function values of subsequent steps is less than a threshold ϵ, indicating little improvement.

$$\text{Stop if } |f(x_{n+1}) - f(x_n)| < \epsilon$$

These criteria aim to balance computational efficiency with the accuracy of the optimization, ensuring that the algorithm stops when further iterations are unlikely to significantly improve the model.

Convergence of Gradient Descent: Gradient descent is guaranteed to converge to the global minimum for convex functions and to a local minimum for non-convex functions, under appropriate conditions on the learning rate α.

1.2 Importance of Convexity

A function is convex if the line segment between any two points on the graph of the function lies above or on the graph. Convexity is important because it ensures that any local minimum is also a global minimum.

Definition: A function $f : \mathbb{R} \to \mathbb{R}$ is convex if, for any two points $x_1, x_2 \in \mathbb{R}$ and any $\lambda \in [0, 1]$, the following inequality holds:

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$$

This property implies that the line segment between any two points on the function's graph does not fall below the graph itself. Figure 1 illustrates convexity: the red straight line between the points *always lies above* the blue function. If this is satisfied, the function is convex!

1.3 Choosing the Right Step Size

The step size, or learning rate, is crucial in the gradient descent algorithm. It determines how big a step is taken in each iteration towards the minimum.

Too Small Step Size: If the step size is too small, the algorithm will take a very long time to converge, making it inefficient.

Too Large Step Size: Conversely, if the step size is too large, the algorithm might overshoot the minimum and diverge.

Figure 2 shows three choices of learning rates: 0.25 (which is just right), 0.05 (which is too small), and 2 which is too large!

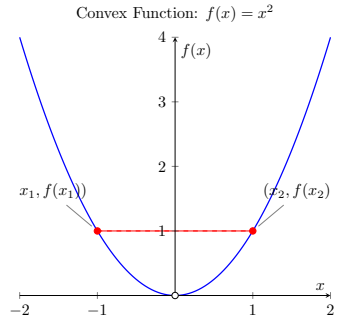

Figure 1: Illustration of a convex function and the convexity property.

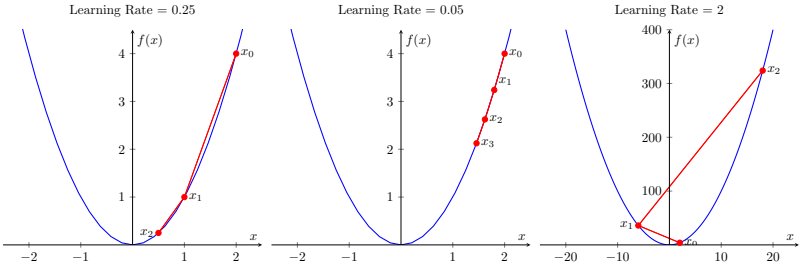

Figure 2: Gradient Descent on $f(x) = x^2$ with Different Learning Rates

Learning Rate = 0.25: This case shows an ideal scenario where the learning rate perfectly matches the scale of the function's gradient, allowing the algorithm to quickly converge to the minimum at $x = 0$ in just a few steps. Suppose we start with $x_0 = 2$. Note that the gradient is 4, and we see that $x_1 = 2 - 0.25 * 4 = 1$. And $x_2 = 1 - 2 * 1 * 0.25 = 0.5$ and so on. So, as we can see, we are making good progress!

Learning Rate = 0.05: With a smaller learning rate of 0.05, the convergence is guaranteed but occurs more slowly. This demonstrates the effect of a too-small learning rate, leading to slow progress towards the minimum. As shown in the middle plot, the progress is slower.

Learning Rate = 2: A too-large learning rate of 2 causes the algorithm to overshoot the minimum and diverge. The values of x rapidly grow in magnitude, moving away from the minimum, illustrating the importance of choosing an appropriate learning rate for convergence. As shown in the right hand plot in Figure 2, gradient descent diverges.

2 Subgradient Descent

Subgradient descent is a generalization of the gradient descent method for optimizing convex functions that may not be differentiable at some points. This method is particularly useful for handling functions with sharp corners or kinks, where traditional gradients do not exist.

TODO: Reference the subgradients defined in the blog article on gradients and derivatives

2.1 Subgradient Descent Update Rule

Subgradient descent utilizes subgradients to update the variables in the optimization problem. The update rule in subgradient descent is similar to that in gradient descent but employs a subgradient g at each step:

$$x_{\text{new}} = x - \alpha g$$

where α is the learning rate. Unlike the gradient in gradient descent, which points in the direction of the steepest decrease of the function, a subgradient does not necessarily point towards the global minimum. However, it guarantees that moving in the opposite direction of the subgradient will not increase the function's value, making it a viable direction for iterative updates.

2.2 Choice of Subgradient

At non-differentiable points, multiple subgradients may exist. The choice of subgradient (g) at each iteration can influence the convergence path of the algorithm. While any subgradient will theoretically guide the optimization process towards minimizing the function, in practice, the selection strategy can affect the speed of convergence and the solution's quality, especially in finite iterations.

2.3 Learning Rate (α) and Convergence

The learning rate (α) plays a crucial role in the convergence of subgradient descent. Unlike gradient descent, where adaptive learning rates and techniques like momentum can be straightforwardly applied, subgradient descent requires more careful consideration of α due to the nature of subgradients. Common strategies for setting the learning rate in subgradient descent include:

- **Constant Learning Rate:** A fixed α can be used, but it may require manual tuning to balance the speed of convergence with the stability of the optimization process.
- **Diminishing Learning Rate:** A learning rate that decreases over time (e.g., $\alpha_t = \frac{\alpha_0}{t}$, where t is the iteration number) ensures convergence under certain conditions for convex functions. This approach helps mitigate the oscillations that can occur due to the non-specific direction of subgradients.

- **Adaptive Learning Rate:** Techniques that adjust α based on the progress of the optimization (such as AdaGrad or RMSProp in gradient descent) can be adapted for subgradient methods, taking into account the variance in subgradient magnitudes to adjust the step size dynamically.

2.4 Ensuring Convergence:

For convex optimization problems, a properly chosen diminishing learning rate can guarantee convergence to the global minimum. The key is to balance the rate of decrease in α to ensure that the algorithm continues to make progress towards the minimum while reducing the step size over time to allow for finer adjustments as it approaches the solution.

3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a pivotal optimization technique in machine learning, especially effective for large-scale data. It modifies the traditional gradient descent approach by approximating the gradient on smaller subsets of data, thereby significantly reducing computational costs.

SGD works well in practice because the variance introduced by the random subset selection averages out over iterations, and the algorithm still progresses towards the minimum.

3.1 The Loss Function in Machine Learning

In machine learning, the objective often involves minimizing a loss function that quantifies the error between the predicted outputs and the actual targets. This loss function is typically represented as a sum over the dataset:

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} l_i(\mathbf{w})$$

where $L(\mathbf{w})$ denotes the total loss, N is the number of samples, $l_i(\mathbf{w})$ is the loss for the i-th sample, and $\mathbf{w}$ are the model parameters.

3.2 Complexity of Gradient Descent

Gradient descent requires computing the gradient of the loss function with respect to the parameters across the entire dataset at each iteration:

$$\nabla L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \nabla l_i(\mathbf{w})$$

This computation becomes exceedingly burdensome for large datasets, making the algorithm slow and computationally expensive.

3.3 Stochastic Gradient Descent (SGD)

SGD addresses these challenges by approximating the gradient using a randomly selected subset of the data (a mini-batch) at each step:

$$\nabla L_{\text{approx}}(\mathbf{w}) = \frac{1}{M} \sum_{i=1}^{M} \nabla l_i(\mathbf{w})$$

where M is the mini-batch size, significantly smaller than N. This approximation allows for much faster computational iterations.

3.4 Advantages of SGD

SGD offers several benefits over traditional gradient descent, including:

- **Efficiency:** Reduced computational cost per iteration, making it feasible for large datasets.
- **Faster Convergence:** More frequent updates can lead to quicker convergence on a solution.
- **Better Generalization:** The inherent noise in the stochastic process can help prevent overfitting, potentially leading to better generalization on unseen data.

3.5 Implementation Considerations

Key considerations for implementing SGD include the selection of mini-batches and the adjustment of the learning rate over time. Mini-batches can be chosen randomly or sequentially, with random selection helping to ensure that each batch is a representative sample of the dataset. The learning rate may also need to be adapted as training progresses, often decreasing to allow the algorithm to fine-tune the model parameters.

Stochastic Gradient Descent provides a practical and efficient means for optimizing loss functions in machine learning, particularly for applications involving large datasets. By leveraging mini-batch approximations, SGD enables faster convergence and improved scalability, making it a cornerstone of modern machine learning optimization.