Study Guide: Stochastic Gradient Descent (SGD)

Student Name

October 9, 2024

1 Introduction

Stochastic Gradient Descent (SGD) is an iterative optimization algorithm used for minimizing an objective function, particularly when the objective is expressed as a sum of differentiable functions. SGD is especially effective in machine learning where the data size is large, and processing the entire dataset at once would be computationally expensive. It is widely used in training machine learning models, including linear classifiers, deep neural networks, and support vector machines.

2 Gradient Descent (Section 14.1)

The classic gradient descent algorithm minimizes a convex, differentiable function $f(w)$. The update rule for gradient descent is:

$$w^{(t+1)} = w^{(t)} - \eta \nabla f(w^{(t)}),$$

where $\eta > 0$ is the learning rate, and $\nabla f(w^{(t)})$ is the gradient of the function at $w^{(t)}$.

2.1 Convergence of Gradient Descent (Section 14.1.1)

For convex-Lipschitz functions, we can bound the suboptimality of the solution as follows:

$$f(\bar{w}) - f(w^*) \leq \frac{B\rho}{\sqrt{T}},$$

where $\bar{w} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$, B is an upper bound on $\|w^*\|$, and ρ is the Lipschitz constant of f. This shows that the suboptimality decreases as the number of iterations T increases.

3 Subgradients (Section 14.2)

The gradient descent algorithm can be extended to nondifferentiable convex functions using the concept of subgradients. A **subgradient** of a convex function f at w, denoted $v \in \partial f(w)$, satisfies:

$$f(u) \geq f(w) + \langle u - w, v \rangle \quad \forall u.$$

The subgradient descent update rule is similar to gradient descent, but it uses a subgradient at each step.

4 Stochastic Gradient Descent (Section 14.3)

Stochastic Gradient Descent (SGD) is a variant of gradient descent where the update is based on a noisy estimate of the gradient, derived from a single randomly selected data point (or a small batch of points). This reduces the computational cost of each iteration, making SGD particularly suitable for large-scale problems.

4.1 SGD Algorithm

At each iteration t, a random sample z_t is drawn from the data distribution D, and the update rule becomes:

$$w^{(t+1)} = w^{(t)} - \eta v_t,$$

where v_t is an unbiased estimate of the gradient at $w^{(t)}$, typically $v_t = \nabla \ell(w^{(t)}, z_t)$, where ℓ is the loss function.

4.2 Convergence of SGD for Convex-Lipschitz Functions (Section 14.3.1)

SGD enjoys a convergence rate similar to that of gradient descent, with the expected suboptimality of the solution being bounded as:

$$\mathbb{E}[f(\bar{w})] - f(w^*) \leq \frac{B\rho}{\sqrt{T}},$$

where B and ρ are problem-dependent constants.

5 Variants of SGD (Section 14.4)

Several modifications to SGD can improve its performance or adapt it to specific scenarios.

5.1 Projection Step (Section 14.4.1)

In some cases, it is necessary to restrict the weight vector w to lie in a convex set H. This can be achieved by adding a projection step to the SGD update:

$$w^{(t+1)} = \arg\min_{w \in H} \|w - (w^{(t)} - \eta v_t)\|.$$

5.2 Variable Step Size (Section 14.4.2)

Instead of using a constant step size η, a decreasing step size η_t can be used to improve convergence. A common choice is $\eta_t = \frac{B}{\rho\sqrt{t}}$.

5.3 Strongly Convex Functions (Section 14.4.4)

When the objective function is strongly convex (i.e., it satisfies a stronger version of convexity), the convergence rate of SGD improves. For λ-strongly convex functions, the suboptimality is bounded by:

$$\mathbb{E}[f(\bar{w})] - f(w^*) \leq \frac{\rho^2}{2\lambda T}.$$

6 SGD for Learning (Section 14.5)

In the context of learning, the goal is to minimize the expected risk:

$$L_D(w) = \mathbb{E}_{z \sim D}[\ell(w, z)],$$

where $\ell(w, z)$ is the loss function on the example z. Since we do not know the distribution D, SGD approximates the risk by sampling examples $z_t \sim D$ and updating based on $\nabla \ell(w^{(t)}, z_t)$.

6.1 Sample Complexity of SGD (Section 14.5.1)

For convex-Lipschitz-bounded problems, the sample complexity required to achieve an error ϵ is:

$$T \geq \frac{B^2\rho^2}{\epsilon^2}.$$

This shows that the number of iterations required for SGD to converge is similar to the sample complexity of empirical risk minimization.

7 Summary (Section 14.6)

- **Gradient Descent** minimizes convex functions using full-batch gradients, while **Stochastic Gradient Descent** uses noisy estimates based on random samples.
- SGD enjoys the same convergence rates as gradient descent for convex-Lipschitz functions, but with lower computational cost per iteration.
- Variants of SGD, such as projection steps and variable step sizes, improve convergence or adapt SGD to specific constraints.
- The sample complexity of SGD matches that of empirical risk minimization, making it a practical and efficient learning algorithm.

Gradient Descent: Update Rule and Implementation

Study Notes

1 Calculating Update Rule and Applying Gradient Descent

Gradient descent is a widely-used optimization algorithm for minimizing functions. The process involves iteratively updating parameters by moving in the direction opposite to the gradient of the function with respect to the parameters. This document provides a systematic approach for deriving the update rule and implementing gradient descent to minimize a function.

2 Step-by-Step Procedure

2.1 Step 1: Define the Objective Function

The first step is to define the objective function $f(\mathbf{x})$ that we aim to minimize. This function must be differentiable, and it maps from $\mathbb{R}^n \to \mathbb{R}$.

For instance, consider the quadratic function:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}^\top \mathbf{x} + c,$$

where $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, and c is a scalar.

2.2 Step 2: Compute the Gradient of the Objective Function

The gradient of $f(\mathbf{x})$, denoted as $\nabla f(\mathbf{x})$, is a vector that points in the direction of steepest ascent. To minimize the function, we move in the opposite direction of the gradient at each step.

The gradient is defined as:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right).$$

For the quadratic function defined above, the gradient is given by:

$$\nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b}.$$

This gradient will be essential for constructing the update rule in the next step.

2.3 Step 3: Define the Update Rule

In gradient descent, the update rule for the parameters is derived by taking steps in the opposite direction of the gradient. The general update rule is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t),$$

where:

- $\mathbf{x}_t$ is the current parameter value at iteration t,
- η_t is the learning rate (or step size) at iteration t,
- $\nabla f(\mathbf{x}_t)$ is the gradient of f evaluated at $\mathbf{x}_t$.

2.4 Step 4: Select a Learning Rate η

The learning rate η_t controls how far we move in the direction of the negative gradient. Choosing the right learning rate is crucial for the convergence of the algorithm.

- If η_t is too large, the algorithm may overshoot and fail to converge.
- If η_t is too small, the algorithm will converge slowly.

A common choice is a fixed learning rate $\eta_t = \eta$ for all iterations. Alternatively, a decaying learning rate can be used for better convergence:

$$\eta_t = \frac{\eta_0}{1 + \beta t},$$

where η_0 is the initial learning rate, and $\beta > 0$ is a decay factor. Line search methods can also be used to dynamically adjust the step size based on local function properties.

2.5 Step 5: Iterate the Update Rule

Starting from an initial guess $\mathbf{x}_0$, we iteratively apply the update rule:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t),$$

until the gradient norm becomes sufficiently small, or the change in the function value is below a predefined threshold, indicating convergence.

3 Example: Minimizing a Quadratic Function

Consider minimizing the quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}^\top \mathbf{x} + c$. The process follows these steps:

3.1 1. Gradient Calculation

The gradient of $f(\mathbf{x})$ is:

$$\nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b}.$$

3.2 2. Update Rule

The gradient descent update rule is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t(A\mathbf{x}_t - \mathbf{b}).$$

3.3 3. Choice of Learning Rate

If A is positive definite, a fixed learning rate $\eta_t = \eta$ can be used, where η satisfies:

$$0 < \eta < \frac{2}{\lambda_{\max}(A)},$$

where $\lambda_{\max}(A)$ is the largest eigenvalue of A. This ensures convergence.

3.4 4. Iteration

The update rule is applied iteratively until the gradient $\nabla f(\mathbf{x}_t)$ becomes small, or the iterates $\mathbf{x}_t$ stop changing significantly.

4 General Case for Non-Quadratic Functions

For general nonlinear and non-quadratic functions, the procedure is similar:

- Compute the gradient $\nabla f(\mathbf{x})$.
- Apply the update rule:
$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t).$$
- Choose an appropriate learning rate η_t.
- Iterate until convergence.

5 Special Considerations

5.1 Convexity

If $f(\mathbf{x})$ is convex, gradient descent will converge to the global minimum. If the function is non-convex, gradient descent may converge to a local minimum or a saddle point.

5.2 Step Size

For strongly convex functions, a constant step size guarantees convergence. For non-convex functions, step size schedules (e.g., decaying learning rates) or adaptive methods like momentum or Adam can improve convergence.

5.3 Lipschitz Continuity of Gradient

If the gradient of $f(\mathbf{x})$ is Lipschitz continuous, theoretical results can guarantee convergence with an appropriate choice of η_t.

6 Conclusion

The gradient descent algorithm iteratively updates parameters by moving in the opposite direction of the gradient of the objective function, scaled by a learning rate. To ensure convergence, one must compute the gradient accurately, choose a suitable learning rate, and iterate until the gradient norm approaches zero or the function value stops changing significantly.

Gradient Descent: Update Rule and Implementation

Study Notes

1 Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) introduces key differences from standard (batch) gradient descent, although the general structure of the update rule remains similar. Let's explore how SGD works, how it alters the gradient computation, and the specific distinctions between gradient descent and stochastic gradient descent.

1.1 Gradient Calculation: Batch vs. Stochastic

The main distinction between batch gradient descent (BGD) and stochastic gradient descent (SGD) lies in how the gradient is computed at each iteration.

- **Batch Gradient Descent (BGD)** computes the gradient using the entire dataset at each iteration. If the cost function depends on m data points (e.g., a sum of loss functions for each point), the gradient is computed as:

$$\nabla f(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(\theta),$$

  where $f_i(\theta)$ is the loss function for the i-th data point. This ensures the gradient is an accurate representation of the direction in which the overall cost function decreases.

- **Stochastic Gradient Descent (SGD)** uses a single data point (or a mini-batch) to compute an approximate gradient at each iteration. The gradient in SGD is calculated as:

$$\nabla f_i(\theta),$$

  where $f_i(\theta)$ is the loss for the randomly selected data point i.

  Hence, in each iteration of SGD, we pick a random data point i and use its gradient for the update, introducing randomness into the gradient computation.

1.2 Update Rule in Stochastic Gradient Descent

The update rule in SGD has the same general structure as in standard gradient descent, with the difference being the use of a gradient estimate based on a single data point or mini-batch.

- In batch gradient descent, the update rule is:

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t),$$

  where $\nabla f(\theta_t)$ is the gradient of the objective function computed over the entire dataset.

- In stochastic gradient descent, the update rule becomes:

$$\theta_{t+1} = \theta_t - \eta_t \nabla f_i(\theta_t),$$

where $\nabla f_i(\theta_t)$ is the gradient computed using a single randomly chosen data point i at iteration t.

Thus, the key difference lies in the source of the gradient:

- **BGD**: Uses the gradient of the full dataset.
- **SGD**: Uses the gradient of one sample (or a small mini-batch) at each step.

1.3 Theoretical Differences

1.3.1 Convergence Speed

- **Batch Gradient Descent**: Computes the exact gradient, so it typically requires fewer iterations to converge. However, since each iteration involves the entire dataset, the overall computational cost can be high.
- **Stochastic Gradient Descent**: Per-iteration cost is much lower because it only computes the gradient using a single (or few) data points. Although it takes more iterations to converge due to noisy gradients, SGD can often reach a good solution faster in terms of total computational time, especially with large datasets.

1.3.2 Convergence Behavior

- **Batch Gradient Descent**: Converges smoothly to the global minimum (for convex problems) or a local minimum (for non-convex problems), as the full gradient is always used.
- **Stochastic Gradient Descent**: Tends to oscillate around the minimum due to the noisy nature of updates. However, by decaying the learning rate η_t over time, we can reduce oscillations and ensure convergence to a neighborhood around the minimum.

1.4 Learning Rate Adaptations in SGD

Because SGD uses noisy gradient estimates, adjusting the learning rate η_t is crucial for achieving convergence. Common strategies include:

1.4.1 Fixed Learning Rate

A fixed learning rate is simple but can lead to oscillations around the minimum. If the learning rate is too large, the algorithm may overshoot the minimum.

1.4.2 Decaying Learning Rate

A more common strategy is to use a decaying learning rate:

$$\eta_t = \frac{\eta_0}{1 + \beta t},$$

where η_0 is the initial learning rate, and β controls the decay. This allows larger steps in the beginning and smaller steps as the algorithm approaches the minimum, reducing oscillations and aiding convergence.

1.4.3 Adaptive Methods (e.g., Adam, RMSprop)

Advanced optimization methods like Adam and RMSprop adapt the learning rate dynamically based on past gradients. These methods combine the fast convergence of SGD with mechanisms to stabilize the updates, often improving performance.

1.5 Example: SGD on a Quadratic Function

Let's revisit the quadratic function $f(x) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}^\top \mathbf{x} + c$, but apply stochastic gradient descent (SGD).

1. **Gradient for a Data Point**: Suppose the objective function is a sum of losses across multiple data points, e.g., $f(x) = \sum_{i=1}^{m} f_i(x)$, where $f_i(x) = \frac{1}{2}\mathbf{x}^\top A_i\mathbf{x} - \mathbf{b}_i^\top \mathbf{x} + c_i$. In SGD, we compute the gradient for a randomly chosen data point i:

$$\nabla f_i(\mathbf{x}_t) = A_i\mathbf{x}_t - \mathbf{b}_i.$$

2. **Update Rule in SGD**: Using the SGD update rule:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t(A_i\mathbf{x}_t - \mathbf{b}_i),$$

where i is randomly selected at each iteration t.

3. **Repeat**: At each iteration, a new random data point i is selected, and the process is repeated until convergence.

1.6 Practical Considerations

- **Mini-batches**: Mini-batch gradient descent balances computational efficiency and stability by using a small subset of data points at each iteration.
- **Variance Reduction**: Techniques like momentum or variance-reduced SGD (e.g., SVRG) are often employed to reduce the high variance in SGD updates and improve convergence.

2 Conclusion

Stochastic gradient descent (SGD) modifies the standard gradient descent algorithm by using an approximate gradient based on a single data point (or mini-batch) instead of the entire dataset. This leads to faster but noisier updates. The update rule remains structurally the same, but the gradient computation reflects the randomness introduced by selecting individual data points. SGD is especially effective for large datasets, where computing the full gradient would be computationally expensive, though careful tuning of the learning rate is essential for ensuring convergence.

Gradient Descent: Success vs. Failure

Study Notes

1 Key Factors for Gradient Descent Success or Failure

1.1 1. Convexity of the Objective Function

- **Convex Functions**: Gradient descent will succeed if the function is convex, as there is a single global minimum. With an appropriate step size, the algorithm will converge to this minimum.
- **Non-Convex Functions**: Gradient descent may fail by converging to a local minimum or getting stuck at a saddle point. Multiple runs from different starting points or using methods like momentum can help avoid this.

1.2 2. Step Size (Learning Rate)

- **Small Step Size**: A very small step size will result in slow progress, causing the algorithm to take too long to converge.
- **Moderate Step Size**: A well-chosen step size ensures smooth and efficient convergence to the minimum.
- **Large Step Size**: If the step size is too large, the algorithm may oscillate around the minimum or even diverge, preventing convergence.

1.3 3. Gradient Smoothness (Lipschitz Continuity)

If the gradient is Lipschitz continuous, meaning the gradients change smoothly, there exists a range for the step size that guarantees convergence. Gradient descent is more reliable when this property holds.

1.4 4. Initialization (Starting Point)

- **Convex Functions**: The starting point is less critical in convex functions, as gradient descent will eventually find the global minimum.
- **Non-Convex Functions**: Poor initialization may lead to getting stuck in a local minimum or saddle point, making the choice of the starting point more important.

1.5 5. Challenges with Ill-Conditioned Functions

Ill-conditioned functions (those with very different curvatures in different directions) can slow down convergence. Gradient descent struggles in these cases, but techniques such as momentum or preconditioning can help speed up convergence.

1.6 Example: Minimizing a Simple Quadratic Function

Consider the simple quadratic function:

$$f(x) = (x - 2)^2.$$

This is a convex function with a global minimum at $x = 2$.

1.6.1 1. Step Size and Success

Success with an appropriate step size: If we choose a moderate step size, say $\eta = 0.1$, gradient descent will gradually reduce the distance from x to 2:

$$x_{t+1} = x_t - \eta \nabla f(x_t).$$

Since $\nabla f(x) = 2(x - 2)$, the update rule becomes:

$$x_{t+1} = x_t - 0.1 \cdot 2(x_t - 2).$$

This will lead to steady convergence toward $x = 2$, the global minimum.

1.6.2 2. Failure with a Large Step Size

Failure with a large step size: If we choose a step size that is too large, say $\eta = 1$, gradient descent will overshoot the minimum. For example:

$$x_{t+1} = x_t - 1 \cdot 2(x_t - 2),$$

resulting in oscillations between values above and below 2. This prevents convergence and can lead to divergence, where x_t keeps moving farther away from the minimum.

1.6.3 3. Initialization and Success

Since this is a convex function, the choice of the starting point doesn't affect the success of gradient descent (though starting closer to 2 will reduce the number of iterations needed). Gradient descent will eventually reach the minimum for any initial point if the step size is properly chosen.

1.7 Summary

- **Correct step size**: Smooth and steady progress to the minimum.
- **Too large step size**: Oscillations or divergence.
- **Convex function**: Always converges if the step size is appropriate.
- **In non-convex functions**, gradient descent can fail by:
  - Oscillating around a local minimum or saddle point (large step size).
  - Converging very slowly (small step size).
  - Getting stuck at a saddle point (non-convex functions).
  - Diverging if the step size is too large.

Convexity and Minimums

Study Notes

1 How to Tell if the Function is Convex

A function is convex if its second derivative is non-negative (i.e., $f''(x) \geq 0$) for all x, or if the graph of the function curves "upwards" and has a single global minimum. In simpler terms, for a function $f(x)$ to be convex:

- **Geometrically:** The line segment between any two points on the function lies above or on the graph of the function.
- **Mathematically:** The second derivative test can be used to verify convexity.

1.1 Checking Convexity for $f(x) = \left(\max(0, x) - \frac{1}{2}\right)^2$

To check whether this function is convex, we split it into two cases based on $\max(0, x)$:

Case 1: When $x > 0$

The function becomes:

$$f(x) = \left(x - \frac{1}{2}\right)^2$$

- First derivative:

$$f'(x) = 2\left(x - \frac{1}{2}\right)$$

- Second derivative:

$$f''(x) = 2$$

Since $f''(x) = 2$, which is positive, the function is convex for $x > 0$.

Case 2: When $x \leq 0$

The function becomes:

$$f(x) = \left(0 - \frac{1}{2}\right)^2 = \frac{1}{4}$$

which is a constant. The derivative of a constant is zero:

$$f'(x) = 0 \quad \text{and} \quad f''(x) = 0$$

A constant function is trivially convex since it is flat and has no curvature.

1.2 Conclusion

The function is convex for both cases ($x > 0$ and $x \leq 0$), meaning $f(x) = \left(\max(0, x) - \frac{1}{2}\right)^2$ is convex everywhere.

2 How to Find the Minimum

To find the minimum of a function, you typically check where the derivative equals zero (i.e., where $f'(x) = 0$) and verify that it's a minimum using the second derivative test.

2.1 Finding the Minimum for $f(x) = \left(x - \frac{1}{2}\right)^2$ (when $x > 0$)

- **First Derivative:**

$$f'(x) = 2\left(x - \frac{1}{2}\right)$$

Set the derivative equal to zero to find the critical points:

$$2\left(x - \frac{1}{2}\right) = 0$$

Solving this gives:

$$x = \frac{1}{2}$$

- **Second Derivative:**

$$f''(x) = 2$$

Since $f''(x) = 2$ is positive, this confirms that $x = \frac{1}{2}$ is indeed a local minimum. Because the function is convex, this is also the global minimum.

2.2 Behavior for $x \leq 0$

For $x \leq 0$, the function is constant, with $f(x) = \frac{1}{4}$. The global minimum still occurs at $x = \frac{1}{2}$, as the function increases for $x > 0$ and remains flat for $x \leq 0$.

3 Conclusion

- The function is convex everywhere, which guarantees it has a single global minimum.
- By setting $f'(x) = 0$, we find that the minimum occurs at $x = \frac{1}{2}$, and the second derivative confirms that this is the global minimum.