Steepest Descent Optimization

Optimization Techniques Project

Ragheed Almasry - 412117664
Anas Alarbied - 411116054
Mustafa Sallat - 412117552

Instructor: Dr. Ali Mustafa

11:11PM

Agenda

Topics Covered

Start

- Introduction
- Objective
- Implementation
- Examples

Introduction

Steepest descent optimization is a versatile technique used to find the minimum of a function. The steepest descent aims to converge to the optimal point where the function reaches its minimum value. In this presentation, we explore the implementation of steepest descent optimization, focusing on dynamically computing the optimal point by defining only the objective function and the initial point.

9 May, 2024 4:44PM

objective

The objective is to demonstrate the effectiveness of steepest descent optimization in dynamically computing optimal points without the need for manual intervention. We aim to showcase the simplicity and versatility of the approach by allowing users to define the objective function and the initial point, enabling seamless optimization across various scenarios and problem domains.

9 May, 2024 4:44PM

Implementation

We implement steepest descent optimization using Python and the Autograd library. The implementation involves several key steps:

1. Defining The Objective Function and The Initial Point

```python
def f(x):
    return (x[0] - 4) ** 2 + (x[1] + 5) ** 2 + 1.8 * (x[0] - 4) * (x[1] - 5)
```

Figure 1: Objective Function f(x)

```python
# Initial point
initial_point = np.array([4.0, -7.0], dtype=float)
```

Figure 2: Initial Point

2. Computing The Gradient and Hessian Matrix Using Autograd

```python
# Compute the gradient of the objective function
gradient_f = grad(f)
```

Figure 3: Gradient of The Objective Function ∇f(x)

```python
def steepest_descent(initial_point):
    gradient = gradient_f(initial_point)
    hessian_func = hessian(f)
    hessian_matrix = hessian_func(initial_point)
```

Figure 4: Gradient of Initial Point ∇f(x0) and The Hessian

3. Calculating c and k for the standard form of the quadratic function

```
# Calculate c and k
c = gradient_f(np.zeros_like(initial_point))
k = f(np.zeros_like(initial_point))
```

Figure 5: Calculating The Linear Part and The Constant

4. Computing The Gradient Length, Learning Rate

```python
# Calculate the length of the gradient
gradient_length = np.linalg.norm(gradient)
```

Figure 6: Gradient Length

```python
def calculate_learning_rate(grad, hessian):
    learning_rate = np.dot(grad, grad) / np.dot(grad, np.dot(hessian, grad))
    return learning_rate
```

Figure 7: Learning Rate Function

```python
learning_rate = calculate_learning_rate(gradient, hessian_matrix)
```

Figure 8: Learning Rate

5. Checking Function Convexity

```
# Check if the function is convex
is_convex = np.all(np.linalg.eigvals(hessian_matrix) >= 0)
```

Figure 9: Check Function Convexity

6. Finding Critical Point Using The Hessian Matrix

```
# Find the critical point using the Hessian
critical_point = -np.linalg.inv(hessian).dot(c)
```

Figure 10: Critical Point Using The Hessian

7. Computing The optimal Points Using Steepest Descent

```
# Steepest Descent Optimization Method
x_next = initial_point - learning_rate * gradient
```

Figure 11: Enter Caption

This comprehensive approach enables us to efficiently optimize the objective function without manual intervention, making it a powerful tool for various optimization tasks

Back to Agenda Page

$$f\left(x\right)=\left(2x_1-4x_2\right)^2+\left(x_2+6\right)^2-1.75\left(x_1-3x_2\right)\left(x_2+8\right)$$

```python
def f(x):
    return (2*x[0] - 4*x[1]) ** 2 + (x[1] + 6) ** 2 - 1.75 * (x[0] - 3*x[1]) * (x[1] + 8)
```

Figure 14: Objective Function

```python
# Initial point
initial_point = np.array([4.0, -7.0], dtype=float)

```

Figure 15: Initial Point

results

```
Constant k: 36.0
Linear part c: [-14.  54.]
Hessian matrix:
[[  8.   -17.75]
 [-17.75  44.5 ]]
The function is convex.
Critical point using Hessian: [-8.19541985 -4.48244275]
Gradient at initial point: [ 142.25 -328.5 ]
Gradient length: 357.97669267705123
Learning rate: 0.01934924201406845
Optimal point using Steepest descent: [ 1.24757032 -0.643774  ]
Minimum value: 13.473318492520413
```

codesnap.dev

$$f(x) = (x_1 + 1)^2 + (x_2 + 6)^2 - 1.75(x_1 - 2)(x_2 + 4)$$

```python
def f(x):
    return (x[0] + 1) ** 2 + (x[1] + 6) ** 2 - 1.75 * (x[0] - 2) * (x[1] + 4)

```

Figure 12: Objective Function

```python
# Initial point
initial_point = np.array([5, 7], dtype=float)
```

Figure 13: Initial Point

results

```
Constant k: 51.0
Linear part c: [-5.  15.5]
Hessian matrix:
[[ 2.   -1.75]
 [-1.75  2.  ]]
The function is convex.
Critical point using the Hessian: [-18.26666667 -23.73333333]
Gradient at initial point: [-7.25 20.75]
Gradient length: 21.980104640333266
Learning rate: 0.32364085494776945
Optimal point using Steepest Descent: [7.3463962  0.28445226]
Minimum value: 69.07050597667944
```

codesnap.dev

conclusion

I like that It so cool

We conduct experiments with different objective functions and initial points to evaluate the performance of the dynamically computed optimal points. The results demonstrate the versatility and effectiveness of steepest descent optimization in dynamically computing optimal points across various problem domains, providing valuable insights for practical applications in optimization tasks.

9 May, 2024 4:44PM

Thank you!

Ragheed Almasry - Anas Alarbied - Mustafa Sallat

Instructor: Dr. Ali Mustafa