Experiment No: 3

29_Dinesh Madhav_CSE(DS)

Batch Gradient Descent :

```
import numpy as np

def batch_gradient_descent(gradient_func, initial_position, learning_rate=0.01, num_iterations=100, batch_size=10):
    position = initial_position
    for _ in range(num_iterations):
        # Generate a batch of random data points
        batch_data = np.random.uniform(-10, 10, size=batch_size)

        # Calculate the gradient using the batch data
        gradient = np.mean([gradient_func(data_point) for data_point in batch_data])

        # Update the position using the gradient and learning rate
        position -= learning_rate * gradient

    return position

# Example usage:
def quadratic_function(x):
    return 2 * x - 4 # Gradient of the function 2x^2 - 4x

initial_position = 5.0 # Initial position of the optimization process
final_position_bgd = batch_gradient_descent(quadratic_function, initial_position)
print("Optimal solution using Batch Gradient Descent:", final_position_bgd)
```

```
Optimal solution using Batch Gradient Descent: 9.248769269784802
```

Stochastic Gradient Descent :

```
import numpy as np

def stochastic_gradient_descent(gradient_func, initial_position, learning_rate=0.01, num_iterations=100):
    position = initial_position
    for _ in range(num_iterations):
        # Randomly select a data point
        random_data_point = np.random.uniform(-10, 10)

        # Calculate the gradient using the selected data point
        gradient = gradient_func(random_data_point)

        # Update the position using the gradient and learning rate
        position -= learning_rate * gradient

    return position

# Example usage:
def quadratic_function(x):
    return 2 * x - 4 # Gradient of the function 2x^2 - 4x

initial_position = 5.0 # Initial position of the optimization process
final_position_sgd = stochastic_gradient_descent(quadratic_function, initial_position)
print("Optimal solution using Stochastic Gradient Descent:", final_position_sgd)
```

```
Optimal solution using Stochastic Gradient Descent: 8.396874450486884
```

Mini-batch Gradient Descent :

```
import numpy as np

def mini_batch_gradient_descent(gradient_func, initial_position, learning_rate=0.01, num_iterations=100, batch_size=10):
    position = initial_position
    for _ in range(num_iterations):
        # Generate a mini-batch of random data points
        mini_batch_data = np.random.uniform(-10, 10, size=batch_size)

        # Calculate the gradient using the mini-batch data
        gradient = np.mean([gradient_func(data_point) for data_point in mini_batch_data])

        # Update the position using the gradient and learning rate
        position -= learning_rate * gradient

    return position

# Example usage:
def quadratic_function(x):
    return 2 * x - 4 # Gradient of the function 2x^2 - 4x

initial_position = 5.0 # Initial position of the optimization process
final_position_mbgd = mini_batch_gradient_descent(quadratic_function, initial_position)
print("Optimal solution using Mini-Batch Gradient Descent:", final_position_mbgd)
```

```
Optimal solution using Mini-Batch Gradient Descent: 8.30670418941336
```

✓ 0s completed at 3:36 PM