Akash Adarkar CSE(DS) RollNo: 68 DL Exp 4

Momentum Gradient Descent:

```
def momentum_gradient_descent(gradient_func,
initial_position,learning_rate=0.01, momentum=0.9,
num_iterations=100):
    position =
    initial_position
    velocity = 0

    for _ in range(num_iterations):
        gradient =
        gradient_func(position)
        velocity = momentum * velocity - learning_rate *
        gradientposition += velocity

    return position

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


initial_position = 0 # Initial position of the optimization
process final_position_momentum =
momentum_gradient_descent(quadratic_function,initial_position)
print("Optimal solution using Momentum:", final_position_momentum)
```

Code:

Output:

```
Optimal solution using Momentum: 1.9915437725637428
```

Stochastic Gradient Descent:

```
import random


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 (in this case, only one data
        point)random_data_point = random.uniform(-10, 10)
        gradient =
        gradient_func(random_data_point)
        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 = 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)
```

Code:

Output:

```
Optimal solution using Stochastic Gradient Descent: 3.0643092926851896
```

Nesterov Gradient Descent:

Code:

```
def nesterov_gradient_descent(gradient_func,
initial_position,learning_rate=0.01, momentum=0.9,
num_iterations=100):
    position =
    initial_position
    velocity = 0

    for _ in range(num_iterations):
        # Compute the gradient at the intermediate
        position intermediate_position = position +
```

```
        momentum * velocitygradient =
        gradient_func(intermediate_position)

        # Update the velocity and position using the Nesterov
        update rulevelocity = momentum * velocity - learning_rate
        * gradient position += velocity

    return position

# Example usage:
def quadratic_function(x):
    return 2 * x - 4  # Gradient of the function 2x^2 - 4x
initial_position = 0  # Initial position of the optimization
process final_position_nesterov =
nesterov_gradient_descent(quadratic_function,initial_position)
print("Optimal solution using Nesterov Gradient
Descent:",final_position_nesterov)
```

Output:

```
Optimal solution using Nesterov Gradient Descent: 1.9960756416676375
```