```
[ ] Start coding or generate with AI.
```

What is Gradient Descent?

Gradient descent is an optimization algorithm used in machine learning to minimize the cost function by iteratively adjusting parameters in the direction of the negative gradient, aiming to find the optimal set of parameters.

The cost function represents the discrepancy between the predicted output of the model and the actual output. The goal of gradient descent is to find the set of parameters that minimizes this discrepancy and improves the model's performance.

The algorithm operates by calculating the gradient of the cost function, which indicates the direction and magnitude of steepest ascent. However, since the objective is to minimize the cost function, gradient descent moves in the opposite direction of the gradient, known as the negative gradient direction.

By iteratively updating the model's parameters in the negative gradient direction, gradient descent gradually converges towards the optimal set of parameters that yields the lowest cost. The learning rate, a hyperparameter, determines the step size taken in each iteration, influencing the speed and stability of convergence.

Gradient descent can be applied to various machine learning algorithms, including linear regression, logistic regression, neural networks, and support vector machines. It provides a general framework for optimizing models by iteratively refining their parameters based on the cost function.

Example of Gradient Descent

Let's say you are playing a game where the players are at the top of a mountain, and they are asked to reach the lowest point of the mountain. Additionally, they are blindfolded. So, what approach do you think would make you reach the lake?

Take a moment to think about this before you read on.

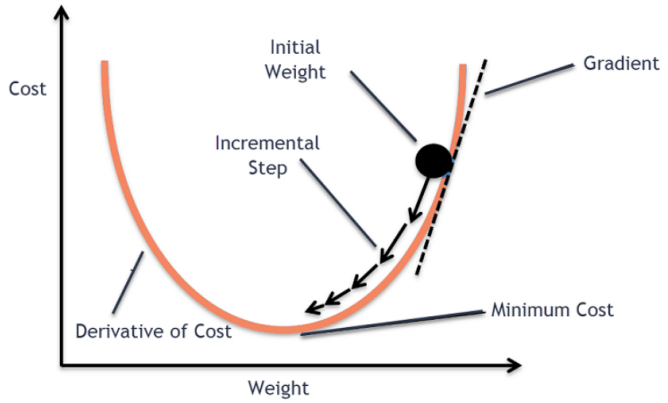
The best way is to observe the ground and find where the land descends. From that position, take a step in the descending direction and iterate this process until we reach the lowest point.

Gradient descent is an iterative optimization algorithm for finding the local minimum of a function.

To find the local minimum of a function using gradient descent, we must take steps proportional to the negative of the gradient (move away from the gradient) of the function at the current point. If we take steps proportional to the positive of the gradient (moving towards the gradient), we will approach a local maximum of the function, and the procedure is called Gradient Ascent.

Gradient descent was originally proposed by CAUCHY in 1847. It is also known as

steepest descent.



The goal of the gradient descent algorithm is to minimize the given function (say cost function). To achieve this goal, it performs two steps iteratively:

1. Compute the gradient (slope), the first order derivative of the function at that point

2. Make a step (move) in the direction opposite to the gradient, opposite direction of slope increase from the current point by alpha times the gradient at that point

Gradient descent algorithm

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$(\text{for } j = 1 \text{ and } j = 0)$$
$$\}$$

Alpha is called Learning rate – a tuning parameter in the optimization process. It decides the length of the steps.

How Does Gradient Descent Work?

1 .It is is an optimization algorithm used to minimize the cost function of a model.

2. The cost function measures how well the model fits the training data and is defined based on the difference between the predicted and actual values.

3. The gradient of the cost function is the derivative with respect to the model's parameters and points in the direction of the steepest ascent.

4. The algorithm starts with an initial set of parameters and updates them in small steps to minimize the cost function.

5. In each iteration of the algorithm, the gradient of the cost function with respect to each parameter is computed.

6. The gradient tells us the direction of the steepest ascent, and by moving in the opposite direction, we can find the direction of the steepest descent.

7. The size of the step is controlled by the learning rate, which determines how quickly the algorithm moves towards the minimum.

8. The process is repeated until the cost function converges to a minimum, indicating that the model has reached the optimal set of parameters.

9. There are different variations of gradient descent, including batch gradient descent, stochastic gradient descent, and mini-batch gradient descent, each with its own advantages and limitations.

10 .Efficient implementation of gradient descent is essential for achieving good performance in machine learning/ deep learning tasks. The choice of the learning rate and the number of iterations can significantly impact the performance of the algorithm.

Types of Gradient Descent

The choice of gradient descent algorithm depends on the problem at hand and the size of the dataset. Batch gradient descent is suitable for small datasets, while stochastic gradient descent algorithm is more suitable for large datasets. Mini-batch is a good compromise between the two and is often used in practice.

Batch Gradient Descent

Batch gradient descent updates the model's parameters using the gradient of the entire training set. It calculates the average gradient of the cost function for all the training examples and updates the parameters in the opposite direction. Batch gradient descent guarantees convergence to the global minimum, but can be computationally expensive and slow for large datasets.

Stochastic Gradient Descent

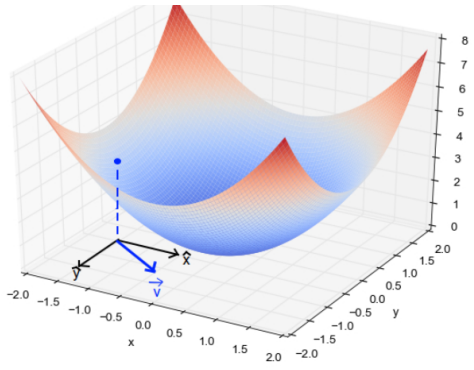
Stochastic gradient descent updates the model's parameters using the gradient of one training example at a time. It randomly selects a training example, computes the gradient of the cost function for that example, and updates the parameters in the opposite direction. Stochastic gradient descent is computationally efficient and can converge faster than batch gradient descent. However, it can be noisy and may not converge to the global minimum.

Mini-Batch Gradient Descent

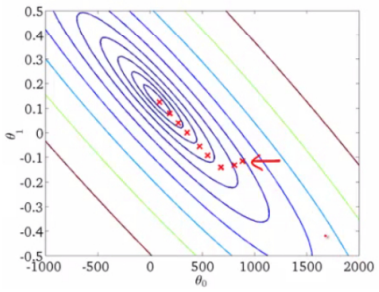
Mini-batch gradient descent updates the model's parameters using the gradient of a small subset of the training set, known as a mini-batch. It calculates the average gradient of the cost function for the mini-batch and updates the parameters in the opposite direction. Mini-batch gradient descent algorithm combines the advantages of both batch and stochastic gradient descent, and is the most commonly used method in practice. It is computationally efficient and less noisy than stochastic gradient descent, while still being able to converge to a good solution.

Plotting the Gradient Descent Algorithm

When we have a single parameter (theta), we can plot the dependent variable cost on the y-axis and theta on the x-axis. If there are two parameters, we can go with a 3-D plot, with cost on one axis and the two parameters (thetas) along the other two axes.



- It can also be visualized by using Contours. This shows a 3-D plot in two dimensions with parameters along both axes and the response as a contour. The value of the response increases away from the center and has the same value along with the rings. The response is directly proportional to the distance of a point from the center (along a direction).

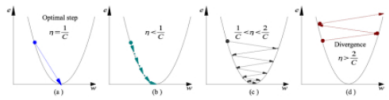


Alpha – The Learning Rate

We have the direction we want to move in, now we must decide the size of the step we must take.

- It must be chosen carefully to end up with local minima.

If the learning rate is too high, we might OVERSHOOT the minima and keep bouncing, without reaching the minima If the learning rate is too small, the training might turn out to be too long



Learning rate is optimal, model converges to the minimum

Learning rate is too small, it takes more time but converges to the minimum

Learning rate is higher than the optimal value, it overshoots but converges (1/C < η <2/C)

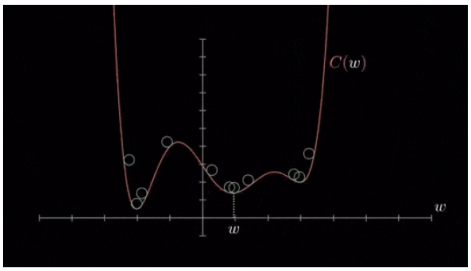
Learning rate is very large, it overshoots and diverges, moves away from the minima, performance decreases on learning

Note: As the gradient decreases while moving towards the local minima, the size of the step decreases. So, the learning rate (alpha) can be constant over the optimization and need not be varied iteratively.

- Local Minima

The cost function may consist of many minimum points. The gradient may settle on any

The cost function may consist of many minimum points. The gradient may settle on any one of the minima, which depends on the initial point (i.e initial parameters(theta)) and the learning rate. Therefore, the optimization may converge to different points with different starting points and learning rate.



Pros:

Flexibility: Gradient descent can be used with different types of models and loss functions, making it a versatile optimization algorithm.

Efficiency: Gradient descent is computationally efficient and can handle large datasets with numerous features.

Convergence: Gradient descent guarantees convergence to a minimum, given a sufficiently small learning rate and enough iterations.

Scalability: Gradient descent can be parallelized across multiple processors or nodes, enabling faster training times.

Cons:

Sensitivity to Learning Rate: The performance of gradient descent is highly sensitive to the choice of the learning rate, which can be difficult to tune.

Local Minima: Gradient descent can get trapped in local minima, which may not be the global minimum.

Overfitting: Gradient descent can overfit the training data if the regularization is not applied or if the model is too complex.

Scaling: Gradient descent may require feature scaling to ensure that each feature contributes equally to the gradient, which can be a time-consuming preprocessing step.

Implementation Steps:

1. **Choose a model and a cost function:**
2. **Initialize the parameters:**
3. **Compute the gradient:**
4. **Update the parameters:**
5. **Repeat until convergence:**
6. **Evaluate the model:**

1.Choose a model and a cost function:

Select a model that you want to optimize, such as linear regression, logistic regression, or a neural network.

Choose a cost function that measures the difference between the predicted output and the actual output, such as mean squared error, cross-entropy loss, or binary log loss.

2. Initialize the parameters:

2. Initialize the parameters:

Set the initial values for the parameters that you want to optimize, such as the weights and biases of the model.

3.Compute the gradient:

Calculate the gradient of the cost function with respect to each parameter by taking the partial derivative of the cost function with respect to each parameter.

4. Update the parameters:

Adjust the parameters in the direction of the negative gradient by multiplying it with a learning rate, which controls the size of the update.

5.Repeat until convergence:

Iterate the above three steps until the cost function converges to a minimum or a satisfactory threshold, such as a small change in the cost function between iterations.

6 .Evaluate the model:

Test the trained model on a separate set of data to evaluate its performance, such as the accuracy, precision, recall, or F1 score

Implementing Gradient Descent in Python

1 .Importing libraries

[2]
```
import numpy as np
import matplotlib.pyplot as plt
```

We then define the function f(x) and its derivative f'(x) −

[3]
```
def f(x):
    return x**2 - 4*x + 6

def df(x):
    return 2*x - 4
```

F(x) is the function that has to be decreased, and df is its derivative (x). The gradient descent approach makes use of the derivative to guide itself toward the minimum by revealing the function's slope along the way.

The gradient descent function is then defined.

[4]
```
def gradient_descent(initial_x, learning_rate, num_iterations):
    x = initial_x
    x_history = [x]

    for i in range(num_iterations):
        gradient = df(x)
        x = x - learning_rate * gradient
        x_history.append(x)

    return x, x_history
```

The starting value of x, the learning rate, and the desired number of iterations are sent to the gradient descent function. In order to save the values of x after each iteration, it initializes x to its original value and generates an empty list. The method then executes the gradient descent for the provided number of iterations, changing x every

iteration in accordance with the equation x = x - learning rate * gradient. The function produces a list of every iteration's x values together with the final value of x.

The gradient descent function may now be used to locate the local minimum of f(x) −

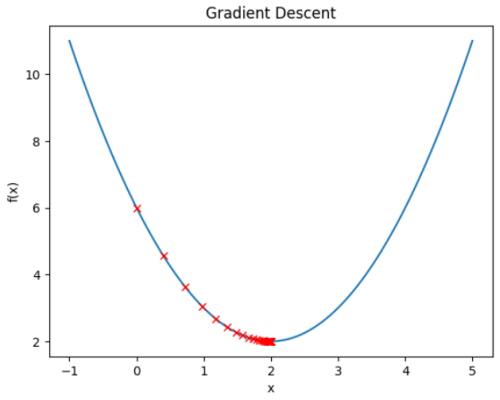
```
[5] initial_x = 0
    learning_rate = 0.1
    num_iterations = 50

    x, x_history = gradient_descent(initial_x, learning_rate, num_iterations)

    print("Local minimum: {:.2f}".format(x))
```

```
Local minimum: 2.00
```

In this illustration, x is set to 0 at the beginning, with a learning rate of 0.1, and 50 iterations are run. Finally, we publish the value of x, which ought to be close to the local minimum at x=2.

Plotting the function f(x) and the values of x for each iteration allows us to see the gradient descent process in action −

```
# Create a range of x values to plot
x_vals = np.linspace(-1, 5, 100)

# Plot the function f(x)
plt.plot(x_vals, f(x_vals))

# Plot the values of x at each iteration
plt.plot(x_history, f(np.array(x_history)), 'rx')

# Label the axes and add a title
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent')

# Show the plot
plt.show()
```



Conclusion

In conclusion, to find the local minimum of a function, Python makes use of the effective optimization process known as gradient descent.

Gradient descent updates the input value repeatedly in the direction of the steepest fall until it achieves the lowest by computing the derivative of the function at each step.

Implementing gradient descent in Python entails specifying the function to optimize and its derivative, initializing the input value, and determining the algorithm's learning rate and the number of iterations.

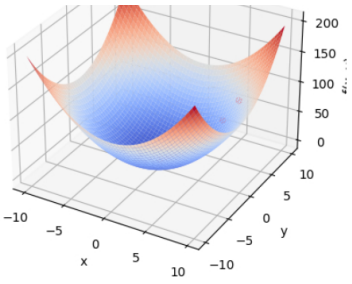
When the optimization is finished, the method can be assessed by tracing its steps to

the minimum and seeing how it reached it.

Gradient descent can be a useful technique in machine learning and optimization applications since Python can handle big datasets and sophisticated functions.

3D Implementation:

[1]
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function to be minimized (a simple quadratic function)
def f(x, y):
    return x**2 + y**2

# Define the partial derivatives of the function with respect to x and y
def df_dx(x, y):
    return 2 * x

def df_dy(x, y):
    return 2 * y

# Define the gradient descent algorithm
def gradient_descent(start_x, start_y, learning_rate, num_iterations):
    # Initialize the parameters
    x = start_x
    y = start_y
    history = []

    # Perform the gradient descent iterations
    for i in range(num_iterations):
        # Calculate the gradients
        grad_x = df_dx(x, y)
        grad_y = df_dy(x, y)

        # Update the parameters
        x = x - learning_rate * grad_x
        y = y - learning_rate * grad_y

        # Save the history of the parameters
        history.append((x, y, f(x, y)))

    return x, y, f(x, y), history

# Define the meshgrid for plotting the function
x_range = np.arange(-10, 10, 0.1)
y_range = np.arange(-10, 10, 0.1)
X, Y = np.meshgrid(x_range, y_range)
Z = f(X, Y)

# Perform gradient descent and plot the results
start_x, start_y = 8, 8
learning_rate = 0.1
num_iterations = 20
x_opt, y_opt, f_opt, history = gradient_descent(start_x, start_y, learning_rate, num_iterations)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='coolwarm')
ax.scatter(*zip(*history), c='r', marker='o')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
plt.show()
```



This implementation defines a simple quadratic function f(x, y) = x^2 + y^2 and its partial derivatives df_dx(x, y) = 2x and df_dy(x, y) = 2y.

It then defines the gradient_descent() function, which takes the starting point (start_x, start_y), learning rate learning_rate, and the number of iterations num_iterations as inputs and returns the optimal point (x_opt, y_opt) and the minimum value f_opt of the function, as well as the history of the parameter values history during the iterations.

The mesh grid is defined for plotting the function and the results of the gradient descent algorithm are plotted in a 3D graph using matplotlib

Double-click (or enter) to edit