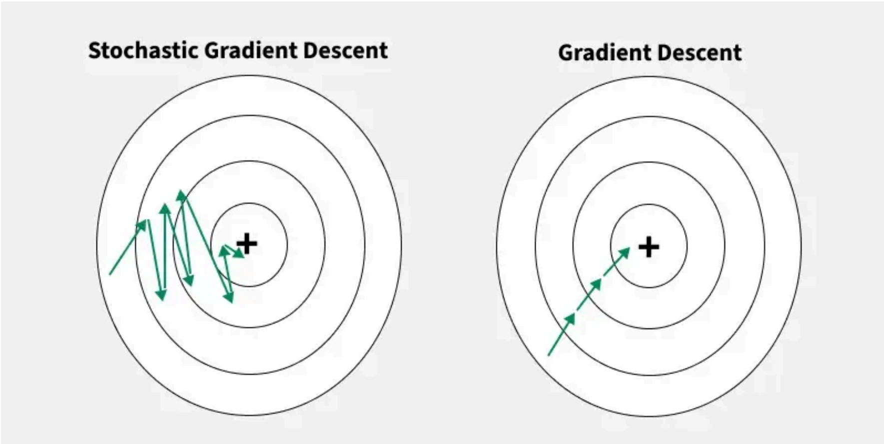
ML - Stochastic Gradient Descent (SGD)

Last Updated : 30 Sep, 2025

Stochastic Gradient Descent (SGD) is an optimization algorithm in machine learning, particularly when dealing with large datasets. It is a variant of the traditional gradient descent algorithm but offers several advantages in terms of efficiency and scalability making it the go-to method for many deep-learning tasks.

Working of Stochastic Gradient Descent



Path followed by batch gradient descent vs. path followed by SGD

- In traditional gradient descent, the gradients are computed based on the entire dataset which can be computationally expensive for large datasets.
- In Stochastic Gradient Descent, the gradient is calculated for each training example (or a small subset of training examples) rather than the entire

Stochastic Gradient Descent update rule is:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x_i, y_i)$$

Where:

- x_i and y_i represent the features and target of the i-th training example.
- The gradient $\nabla_\theta J(\theta; x_i, y_i)$ is now calculated for a single data point or a small batch.

The key difference from traditional gradient descent is that, in SGD, the parameter updates are made based on a single data point, not the entire dataset. The random selection of data points introduces stochasticity which can be both an advantage and a challenge.

Implementing Stochastic Gradient Descent from Scratch

1. Generating the Data

In this step, we generate synthetic data for the linear regression problem. The data consists of feature X and the target y where the relationship is linear, i.e., y = 4 + 3 * X + noise.

- X is a random array of 100 samples between 0 and 2.
- y is the target, calculated using a linear equation with a little random noise to make it more realistic.

```python
import numpy as np

np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

For a linear regression with one feature, the model is described by the equation:

$$y = \theta_0 + (\theta_1) \cdot X$$

Where:

- θ_0 is the intercept (the bias term),
- θ_1 is the slope or coefficient associated with the input feature X.

2. Defining the SGD Function

Here we define the core function for Stochastic Gradient Descent (SGD). The function takes the input data X and y. It initializes the model parameters, performs stochastic updates for a specified number of epochs and records the cost at each step.

- theta (θ) is the parameter vector (intercept and slope) initialized randomly.
- X_bias is the augmented $X$with a column of ones added for the bias term (intercept).

In each epoch, the data is shuffled and for each mini-batch (or single sample), the gradient is calculated and the parameters are updated. The cost is calculated as the mean squared error and the history of the cost is recorded to monitor convergence.

```
def sgd(X, y, learning_rate=0.1, epochs=1000, batch_size=1):
    m = len(X)
```

```
    theta = np.random.randn(2, 1)

    X_bias = np.c_[np.ones((m, 1)), X]

    cost_history = []

    for epoch in range(epochs):
        indices = np.random.permutation(m)
        X_shuffled = X_bias[indices]
        y_shuffled = y[indices]

        for i in range(0, m, batch_size):
            X_batch = X_shuffled[i:i +
batch_size]
            y_batch = y_shuffled[i:i +
batch_size]

            gradients = 2 / batch_size * \
                X_batch.T.dot(X_batch.dot(theta)
- y_batch)
            theta -= learning_rate * gradients

        predictions = X_bias.dot(theta)
        cost = np.mean((predictions - y) ** 2)
        cost_history.append(cost)

        if epoch % 100 == 0:
            print(f"Epoch {epoch}, Cost:
{cost}")

    return theta, cost_history
```

3: Train the Model Using SGD

In this step, we call the sgd() function to train the model. We specify the learning rate, number of epochs and batch size for SGD.

```
theta_final, cost_history = sgd(
batch_size=1)
```

Output:

```
Epoch 0, Cost: 1.581821686856939
Epoch 100, Cost: 1.5664692700155733
Epoch 200, Cost: 1.4445422391173144
Epoch 300, Cost: 1.7037674963102662
Epoch 400, Cost: 0.9101999515899212
Epoch 500, Cost: 0.8184497904316664
Epoch 600, Cost: 0.8352333304446237
Epoch 700, Cost: 0.8542729530055074
Epoch 800, Cost: 1.0508310318687628
Epoch 900, Cost: 0.8261971232182218
```

Train the Model Using SGD

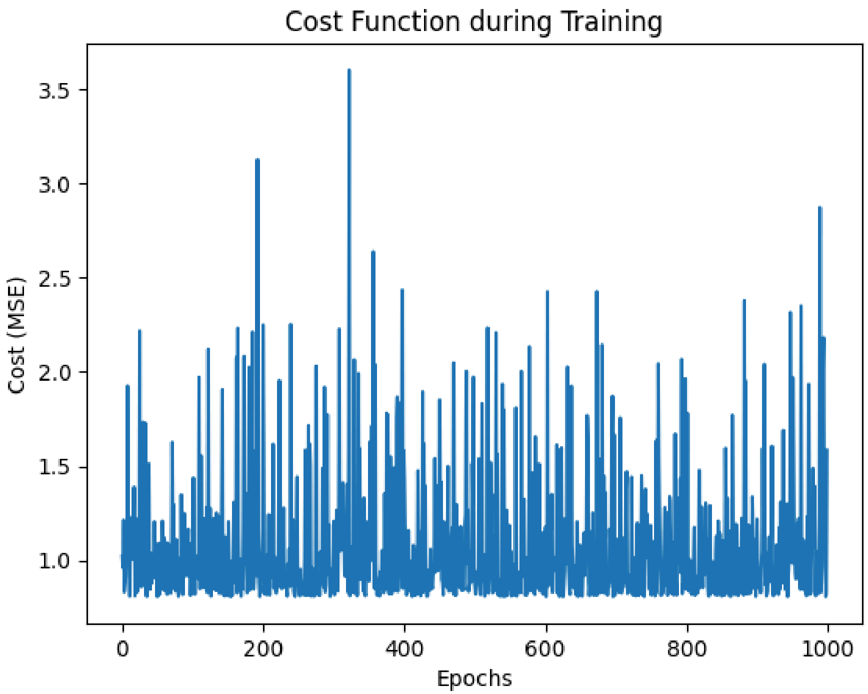
4. Visualizing the Cost Function

After training, we visualize how the cost function evolves over epochs. This helps us understand if the algorithm is converging properly.

```
import matplotlib.pyplot as plt

plt.plot(cost_history)
plt.xlabel('Epochs')
plt.ylabel('Cost (MSE)')
plt.title('Cost Function during Training')
plt.show()
```

Output:



Visualize the Cost Function

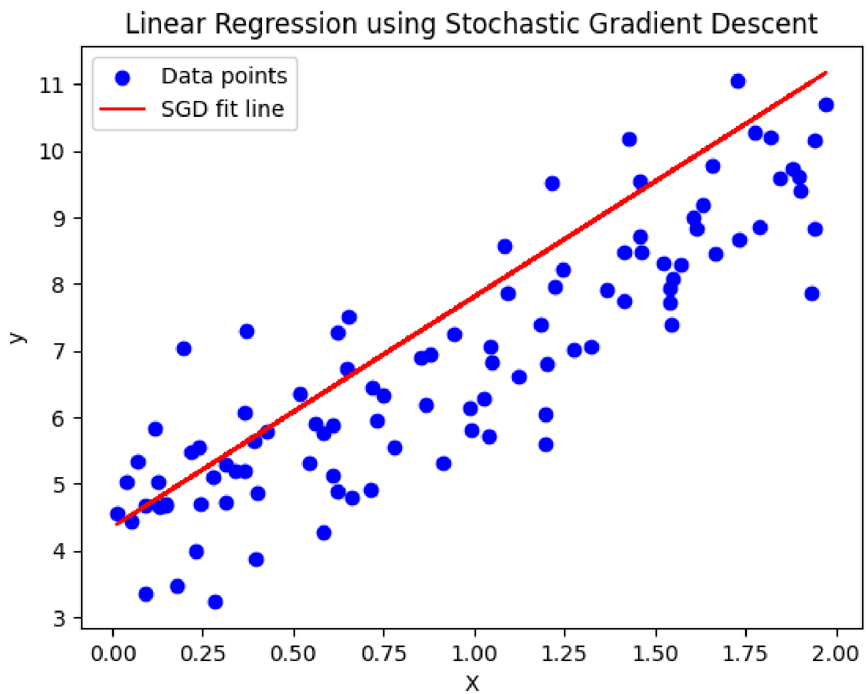
5. Plotting the Data and Regression Line

We will visualize the data points and the fitted regression line after training. We plot the data points as blue dots and the predicted line (from the final θ) as a red line.

```
plt.scatter(X, y, color='blue', label='Data
points')
plt.plot(X, np.c_[np.ones((X.shape[0], 1)),
X].dot(
    theta_final), color='red', label='SGD fit
line')
```

```
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression using Stochastic Gradient Descent')
plt.legend()
plt.show()
```

Output:



Plot the Data and Regression Line

6. Printing the Final Model Parameters

After training, we print the final parameters of the model which include the slope and intercept. These values are the result of optimizing the model using SGD.

```
print(f"Final parameters: {theta_final}")
```

Output:

Final parameters: [[4.35097872]
[3.45754277]]

The final parameters returned by the model are:

$$\theta_0 = 4.35, \quad \theta_1 = 3.45$$

Then the fitted linear regression model will be:

$$y = 4.35 + (3.45) \cdot X$$

This means:

- When X=0, y=4.3(the intercept or bias term).
- For each unit increase in X, y will increase by 3.4 units (the slope or coefficient).

Applications

SGD and its variants are widely used across various domains of machine learning:

- **Deep Learning**: In training deep neural networks, SGD is the default optimizer due to its efficiency with large datasets and its ability to work with large models.
- **Natural Language Processing (NLP)**: Models like Word2Vec and transformers are trained using SGD variants to optimize large models on vast text corpora.
- **Computer Vision**: For tasks such as image classification, object detection and segmentation, SGD has been fundamental in training convolutional neural networks (CNNs).
- **Reinforcement Learning**: SGD is also used to optimize the parameters of models used in reinforcement learning, such as deep Q-networks (DQNs) and policy gradient methods.

Advantages

- **Efficiency**: Because it uses only one or a few data points to calculate the gradient, SGD can be much faster, especially for large datasets. Each step requires fewer computations, leading to quicker convergence.

- **Memory Efficiency**: Since it does not require storing the entire dataset in memory for each iteration, SGD can handle much larger datasets than traditional gradient descent.
- **Escaping Local Minima**: The noisy updates in SGD, caused by the stochastic nature of the algorithm, can help the model escape local minima or saddle points, potentially leading to better solutions in non-convex optimization problems.
- **Online Learning**: SGD is well-suited for online learning where the model is trained incrementally as new data comes in, rather than on a static dataset.

Challenges

- **Noisy Convergence**: Since the gradient is estimated based on a single data point (or a small batch), the updates can be noisy, causing the cost function to fluctuate rather than steadily decrease. This makes convergence slower and more erratic than in batch gradient descent.

- **Learning Rate Tuning**: SGD is highly sensitive to the choice of learning rate. A learning rate that is too large may cause the algorithm to diverge while one that is too small can slow down convergence. Adaptive methods like Adam and RMSprop address this by adjusting the learning rate dynamically during training.
- **Long Training Times**: While each individual update is fast, the convergence might take a longer time overall since the steps are more erratic compared to batch gradient descent.

Stochastic Gradient Descent (SGD) in Machine Learning

Suggested Quiz 5 Questions

Which of the following best describes the key advantage of Stochastic Gradient Descent (SGD) over traditional Gradient Descent?

- (A) It computes gradients using the entire dataset
- (B) It reduces computational cost by updating parameters with one data point at a time
- (C) It guarantees faster convergence in all cases
- (D) It does not require a learning rate

Login to View Explanation 1/5 < Previous Next >

Comment R Rahul_... + Follow 46

Article Tags: Machine Learning python

AI-ML-DS With Python

Explore

Machine Learning Basics

Python for Machine Learning

Feature Engineering

Supervised Learning