Exp-4

Implementation of Stochastic GD

Stochastic GD

Code:

```
import numpy as np


def stochastic_gradient_descent(X, y, learning_rate, num_epochs):
    num_samples, num_features = X.shape
    theta = np.zeros(num_features)

    for epoch in range(num_epochs):
        for i in range(num_samples):
            random_index = np.random.randint(num_samples)
            xi = X[random_index:random_index+1]
            yi = y[random_index:random_index+1]
            gradient = np.dot(xi.T, (np.dot(xi, theta) - yi))
            theta -= learning_rate * gradient

    return theta


# Example usage
# Generate some sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 3)
y = 4 + np.dot(X, np.array([3, 2, 1])) + np.random.randn(100)
```

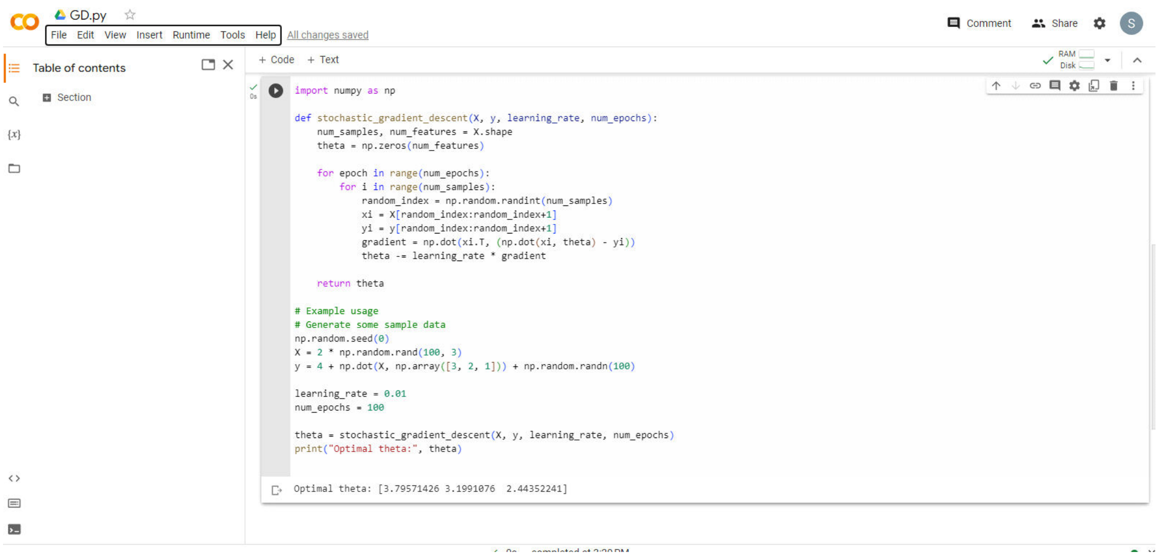
learning_rate = 0.01

num_epochs = 100

theta = stochastic_gradient_descent(X, y, learning_rate, num_epochs)

print("Optimal theta:", theta)

Output:



```
import numpy as np

def stochastic_gradient_descent(X, y, learning_rate, num_epochs):
    num_samples, num_features = X.shape
    theta = np.zeros(num_features)

    for epoch in range(num_epochs):
        for i in range(num_samples):
            random_index = np.random.randint(num_samples)
            xi = X[random_index:random_index+1]
            yi = y[random_index:random_index+1]
            gradient = np.dot(xi.T, (np.dot(xi, theta) - yi))
            theta -= learning_rate * gradient

    return theta

# Example usage
# Generate some sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 3)
y = 4 + np.dot(X, np.array([3, 2, 1])) + np.random.randn(100)

learning_rate = 0.01
num_epochs = 100

theta = stochastic_gradient_descent(X, y, learning_rate, num_epochs)
print("Optimal theta:", theta)
```

```
Optimal theta: [3.79571426 3.1991076  2.44352241]
```

Momentum GD:

code:

import numpy as np

def gradient_descent_momentum(X, y, learning_rate, num_iterations, momentum_factor):

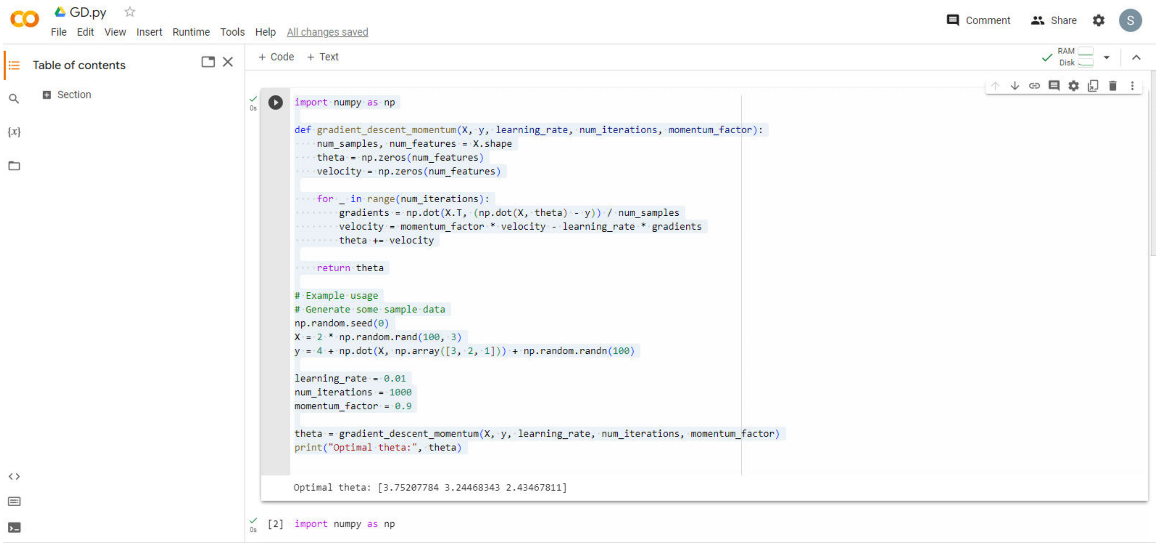
    num_samples, num_features = X.shape

    theta = np.zeros(num_features)

    velocity = np.zeros(num_features)

```
    for _ in range(num_iterations):
        gradients = np.dot(X.T, (np.dot(X, theta) - y)) / num_samples
        velocity = momentum_factor * velocity - learning_rate * gradients
        theta += velocity

    return theta

# Example usage
# Generate some sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 3)
y = 4 + np.dot(X, np.array([3, 2, 1])) + np.random.randn(100)

learning_rate = 0.01
num_iterations = 1000
momentum_factor = 0.9

theta = gradient_descent_momentum(X, y, learning_rate, num_iterations, momentum_factor)
print("Optimal theta:", theta)
```

output:



```python
import numpy as np

def gradient_descent_momentum(X, y, learning_rate, num_iterations, momentum_factor):
    num_samples, num_features = X.shape
    theta = np.zeros(num_features)
    velocity = np.zeros(num_features)

    for _ in range(num_iterations):
        gradients = np.dot(X.T, (np.dot(X, theta) - y)) / num_samples
        velocity = momentum_factor * velocity - learning_rate * gradients
        theta += velocity

    return theta

# Example usage
# Generate some sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 3)
y = 4 + np.dot(X, np.array([3, 2, 1])) + np.random.randn(100)

learning_rate = 0.01
num_iterations = 1000
momentum_factor = 0.9

theta = gradient_descent_momentum(X, y, learning_rate, num_iterations, momentum_factor)
print("Optimal theta:", theta)
```

```
Optimal theta: [3.75207784 3.24468343 2.43467811]
```

```python
[2] import numpy as np
```