

Prinshi Jha

CSE(DS)/23

Experiment no 4

Implementation of stochastic gradient descent

```
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)
```

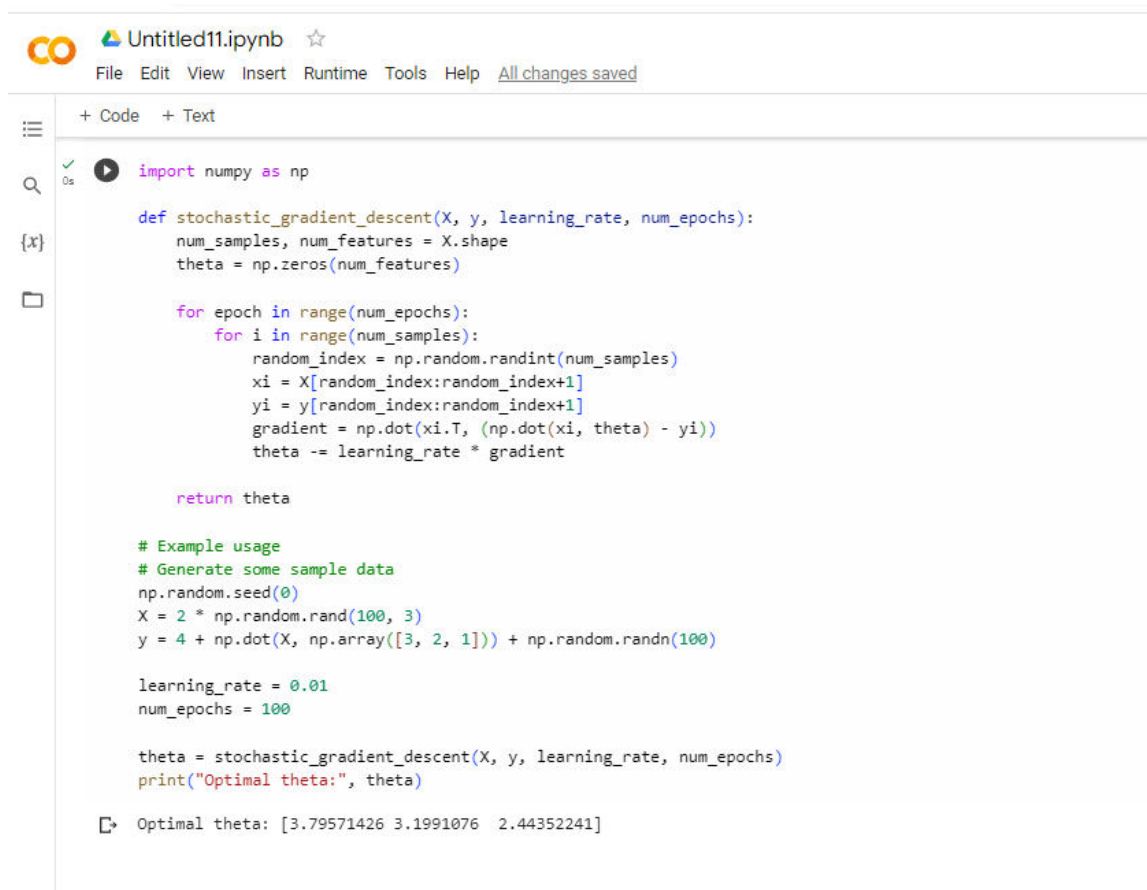
```
y = 4 + np.dot(X, np.array([3, 2, 1])) + np.random.randn(100)
```

```
learning_rate = 0.01
```

```
num_epochs = 100
```

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

```
print("Optimal theta:", theta)
```



The screenshot shows a Jupyter Notebook titled "Untitled11.ipynb". The interface includes a top menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", "Help", and "All changes saved". Below the menu is a toolbar with icons for running code, saving, and other functions. The main area displays a Python script that defines a function for stochastic gradient descent and uses it to find the optimal theta. The script includes comments for example usage and data generation. The output of the script is displayed at the bottom.

```
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]

Implementation of mini batch gradient descent

```
import numpy as np
```

```
# Define the function to be minimized and its gradient
```

```
def f(x):
```

```
    return x**2
```

```
def df(x):
```

```
    return 2*x
```

```
# Define the mini-batch gradient descent function
```

```
def minibatch_gradient_descent(x_init, learning_rate, batch_size, num_iterations):
```

```
    x = x_init
```

```
    for i in range(num_iterations):
```

```
        # Generate random mini-batch samples
```

```
        indices = np.random.choice(len(x), size=batch_size)
```

```
        batch = x[indices]
```

```
        # Compute the gradient of the mini-batch loss
```

```
        gradient = np.mean(df(batch))
```

```
        # Update the parameter using the learning rate and mini-batch gradient
```

```
        x -= learning_rate * gradient
```

```
return x
```

```
# Test the mini-batch gradient descent function
```

```
x_init = np.array([1.0, 2.0, 3.0, 4.0])
```

```
learning_rate = 0.1
```

```
batch_size = 2
```

```
num_iterations = 100
```

```
result = minibatch_gradient_descent(x_init, learning_rate, batch_size,  
num_iterations)
```

```
print("Result:", result)
```

```

import numpy as np

# Define the function to be minimized and its gradient
def f(x):
    return x**2

def df(x):
    return 2*x

# Define the mini-batch gradient descent function
def minibatch_gradient_descent(x_init, learning_rate, batch_size, num_iterations):
    x = x_init
    for i in range(num_iterations):
        # Generate random mini-batch samples
        indices = np.random.choice(len(x), size=batch_size)
        batch = x[indices]

        # Compute the gradient of the mini-batch loss
        gradient = np.mean(df(batch))

        # Update the parameter using the learning rate and mini-batch gradient
        x -= learning_rate * gradient

    return x

# Test the mini-batch gradient descent function
x_init = np.array([1.0, 2.0, 3.0, 4.0])
learning_rate = 0.1
batch_size = 2
num_iterations = 100

result = minibatch_gradient_descent(x_init, learning_rate, batch_size, num_iterations)
print("Result:", result)

```

Result: [-1.4231927 -0.4231927 0.5768073 1.5768073]