Gradient Descent

A notebook covering the theory, mathematics, and Python implementation of gradient descent optimization.

1. Brief Idea

Gradient Descent is an iterative optimization algorithm used to find the minimum of a function. By repeatedly moving in the direction of the negative gradient, it updates parameters to reduce the loss. It is the backbone of many machine learning algorithms, especially for training neural networks and regression models.

2. Context and Backstory

First introduced in the 19th century, gradient descent became central to machine learning with the rise of large-scale optimization problems. It is used in linear regression, logistic regression, deep learning, and more. Variants like stochastic and mini-batch gradient descent address scalability and convergence issues.

3. Mathematical Derivations and Formulae

Given a function $f(\theta)$, the update rule for gradient descent is:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$$

where η is the learning rate. For example, in linear regression with loss $J(\theta)$, the gradient is computed with respect to θ and used to update the parameters iteratively.

4. Diagrams and Visual Aids

Figure: Gradient descent steps on a convex loss surface.

(Insert a plot or image here if desired.)

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Quadratic function: f(theta) = (theta-3)^2
f = lambda theta: (theta-3)**2
f_grad = lambda theta: 2*(theta-3)

# Gradient Descent
theta = 0
eta = 0.1
history = [theta]
for _ in range(20):
    theta -= eta * f_grad(theta)
    history.append(theta)

# Plot
thetas = np.linspace(-1, 5, 100)
plt.plot(thetas, f(thetas), label='f(θ)')
```

```
plt.plot(history, f(np.array(history)), 'o-', label='GD steps')
plt.xlabel('θ')
plt.ylabel('f(θ)')
plt.title('Gradient Descent on a Quadratic Function')
plt.legend()
plt.show()

print('Final θ:', theta)
```

6. Frequently Asked Questions (FAQ)

1. **What is the learning rate?** The step size for each update; too large can diverge, too small can be slow.
2. **What are variants of gradient descent?** Stochastic, mini-batch, momentum, Adam, RMSprop, etc.
3. **Does gradient descent always find the global minimum?** Only for convex functions; otherwise, it may get stuck in local minima.
4. **How to choose the number of iterations?** Use convergence criteria or validation performance.

7. Higher-Order Thinking Questions (HOTS)

- Derive the convergence rate of gradient descent for strongly convex functions.
- Compare batch, stochastic, and mini-batch gradient descent in terms of speed and accuracy.
- How would you adapt gradient descent for non-differentiable functions?

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define quadratic function and its gradient
f = lambda theta: (theta - 3) ** 2
f_grad = lambda theta: 2 * (theta - 3)

# Gradient Descent from scratch
theta = 0.0
eta = 0.1
history = [theta]
for _ in range(20):
    theta -= eta * f_grad(theta)
    history.append(theta)

# Plot
x_vals = np.linspace(-1, 5, 100)
plt.plot(x_vals, f(x_vals), label='f(θ)')
plt.plot(history, [f(t) for t in history], 'o-', label='GD steps')
plt.xlabel('θ')
plt.ylabel('f(θ)')
plt.title('Gradient Descent (from scratch)')
plt.legend()
plt.show()

print('Final θ:', theta)
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic linear data
def generate_data(n=100):
    np.random.seed(0)
    X = 2 * np.random.rand(n, 1)
    y = 4 + 3 * X[:, 0] + np.random.randn(n)
```

```
    return X, y

X, y = generate_data()

# Add bias term
X_b = np.c_[np.ones((X.shape[0], 1)), X]

# Gradient Descent for linear regression
eta = 0.1
n_iter = 1000
m = X_b.shape[0]
theta = np.zeros(X_b.shape[1])
for i in range(n_iter):
    gradients = 2/m * X_b.T @ (X_b @ theta - y)
    theta -= eta * gradients
    if i % 200 == 0:
        mse = np.mean((X_b @ theta - y) ** 2)
        print(f"Iter {i}, MSE: {mse:.2f}")

print('Estimated coefficients:', theta)

# Plot
plt.scatter(X, y, label='Data')
plt.plot(X, X_b @ theta, color='red', label='GD Fit')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression via Gradient Descent (from scratch)')
plt.legend()
plt.show()
```