03-gradient_descent

October 17, 2024

Gradient Descent

0.1 Action:

0.1.1 Understanding Gradient Descent Algorithm

In this section, we will first run the gradient descent algorithm and then explain why it works the way it does.

0.2 Main Goal: Understand gradient descent

0.2.1 What it's gradient descent?

It's an algorithm that helps us eventually find the optimal point. Visually, what it does is take small steps in the direction of the steepest descent for a particular objective function. In machine learning, objective functions are more commonly known as loss or cost functions.

If we focus on the mathematical function, we can define it in two cases:

1. Gradient for a function of two variables:

$$\nabla f(x, y) = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j}$$

2. Gradient for a function of three variables:

$$\nabla f(x, y, z) = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j} + \frac{\partial f}{\partial z}\mathbf{k}$$

0.3 Let's Code this Up!

[2]:
```
"""
We're going to use sympy python library to solve the equations
"""
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
```

[3]:
```
# Step 1: Define the variables
x, y = sp.symbols('x y')

# Step 2: Define the function. Example function: f(x,y) = 5y - x³y²
```

```
f = 5*y - x**3 * y**2

# Step 3: Compute the partial derivatives
partial_x = sp.diff(f, x)
partial_y = sp.diff(f, y)

# Step 4: Let's get our gradient as a vector
gradient_2d = (partial_x, partial_y)
print("Symbolic gradient in vector notation:")
gradient_vector = f"({sp.latex(partial_x)})i + ({sp.latex(partial_y)})j"
print(gradient_vector)
```

```
Symbolic gradient in vector notation:
(- 3 x^{2} y^{2})i + (- 2 x^{3} y + 5)j
```

Nice, we've implemented the mathematical concept! But how can we unify this with the algorithm everyone talks about in Machine Learning?

[4]:
```
"""
Let's generate some random data to work with
"""
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 + 3 * X + np.random.randn(100, 1) * 0.1

# Initialize parameters
b0 = 0  # y-intercept
b1 = 0  # slope
learning_rate = 0.1  # learning rate
epochs = 1000  # number of iterations
```

[5]:
```
"""
This is our hypothesis function, representing a straight line where b0 is the␣
 ↪y-intercept
and b1 is the slope.
"""
def h(X, b0, b1):
    return b0 + b1 * X
```

[6]:
```
"""
Let's define our cost function
"""
def cost_function(X, y, b0, b1):
    return np.mean((y - h(X, b0, b1)) ** 2)
```

[7]:
```
# Finally, let's implement the gradient descent algorithm!
def gradient_descent(X, y, b0, b1, learning_rate, epochs):
    m = len(y)
```

```
    cost_history = []

    for _ in range(epochs):
        # Compute predictions
        y_pred = h(X, b0, b1)

        # Compute gradients
        grad_b0 = (-2/m) * np.sum(y - y_pred)
        grad_b1 = (-2/m) * np.sum((y - y_pred) * X)

        # Update parameters
        b0 = b0 - learning_rate * grad_b0
        b1 = b1 - learning_rate * grad_b1

        # Compute and store the cost
        cost = cost_function(X, y, b0, b1)
        cost_history.append(cost)

    return b0, b1, cost_history
```

[8]:
```
b0_final, b1_final, cost_history = gradient_descent(X, y, b0, b1, eta, epochs)
```

[9]:
```
print(f"Final b0: {b0_final:.4f}")
print(f"Final b1: {b1_final:.4f}")
print(f"Final cost: {cost_history[-1]:.4f}")
```

```
Final b0: 2.0222
Final b1: 2.9937
Final cost: 0.0099
```

Perfect! So let's recap what we did:

1. Starts with initial guesses for the line's slope and intercept.
2. Repeats the following steps for a set number of times.
3. Makes predictions using the current slope and intercept.
4. Calculates how wrong these predictions are.
5. Adjusts the slope and intercept to make the predictions a little better.
6. Keeps track of how well it's doing.

Finally, it returns the best slope and intercept it found, along with a record of its progress.

0.4 Batch Gradient Descent

0.4.1 What's the difference?

The key difference from what we looked up before is that in batch, we take into account ALL training data to compute the gradient and update the model parameters after every pass (epoch) through the entire dataset.