Implementation of Gradient Descent - Labwork 1

Do Thanh Dat - 2440059

May 2, 2025

1 Introduction

Gradient descent is one of the fundamental concepts in Deep Learning; its purpose is to reduce the value of loss function, which measures how far the model's predictions are from the actual target values. In the first Lab-work, I tried to implement this from scratch to find the minimum value of any given function $f(x)$ (assume that there is no other variable such as y, z, etc.).

2 Method

The basic idea behind this algorithm lies at its core: iteratively update the value of x with the learning rate and the first order derivative of the function $f(x)$.

- Initialize a random value of x
- Update $x \leftarrow x - r \cdot f'(x)$
- Repeat after a countable iterations, or if $f(x)$ is small enough.

Surprisingly, even though the update rule always subtracts the derivative term, the value of x does not always decrease. The direction of change depends on the sign of the derivative $f'(x)$. If $f'(x) > 0$, x moves to the left (decreases); if $f'(x) < 0$, x moves to the right (increases). This behavior allows gradient descent to "slide down" towards the minimum of the function, regardless of which side it starts from.

Example: Let $f(x) = x^2$. Simple Linear Algebra knowledge tells us that the function $f(x) = x^2$ decreases in the interval $(-\infty, 0)$ and increases in $(0, +\infty)$, with a global minimum at $x = 0$. This is confirmed by taking the first derivative $f'(x) = 2x$, which is negative when $x < 0$ and positive when $x > 0$.

Suppose that we initiate $x = 10$ and use a learning rate $r = 0.1$. Then the first update becomes:

$$x_1 = 10 - 0.1 \cdot 2 \cdot 10 = 8$$

Continuing this process will eventually decrease x closer to 0 — the minimum of $f(x)$.

If we start at a negative point like $x = -5$, then $f'(x) = -10$, so the update is:

$$x_1 = -5 - 0.1 \cdot (-10) = -5 + 1 = -4$$

And the value of x starts to increase, until it reaches 0.

3 Code Implementation

```
from sympy import symbols, diff, lambdify
import random
#import numpy as np
import matplotlib.pyplot as plt

"""
sympy is a library that return the derivative of a function, which is essen
"""
"""
Implementation of gradient descent from scratch. this iterative function w
1. Take the input f, learning rate r, iteration iters
2. Initialize x0
3. calculate the first derivative of f as f_deri
4. return the new value of x0 = x0 - r * f_deri(x0), after looping through
5. re-compute f(x), stop when f(x) is small enough
"""

def gradient_descent(f, r: float, iters: int =10):
    x = symbols('x')
    f_deri = diff(f, x)
    f = lambdify(x, f, 'math')
    f_deri = lambdify(x, f_deri, 'math')

    x0 = random.uniform(-10,10)
    print(f"{'Step'} {'x'} {'f(x)'}")

    for i in range(iters):
        x0 = x0 - r * f_deri(x0)
        f_value = f(x0)
        print(f"{i} {x0} {f_value}")
    return x0, f_value

f = x**2
for lr in [0.1,0.001,0.0001]:
    x_final, f_value_final = gradient_descent(f, r=lr)
    print(f"learning rate: {lr}, x_final: {x_final}, f_value_final: {f_value_
```

4 Results

I have tested with 3 different learning rates, and the results are captured from below:

Step	**x**	**f(x)**
0	3.295	10.857
1	2.636	6.948
2	2.109	4.447
3	1.687	2.846
4	1.350	1.821
5	1.080	1.166
6	0.864	0.746
7	0.691	0.477
8	0.553	0.306
9	0.442	0.196

Table 1: Learning rate $r = 0.1$

Step	**x**	**f(x)**
0	9.601	92.188
1	9.582	91.820
2	9.563	91.453
3	9.544	91.087
4	9.525	90.723
5	9.506	90.361
6	9.487	90.000
7	9.468	89.640
8	9.449	89.282
9	9.430	88.925

Table 2: Learning rate $r = 0.001$

Step	**x**	**f(x)**
0	-5.898	34.792
1	-5.897	34.778
2	-5.896	34.764
3	-5.895	34.750
4	-5.894	34.736
5	-5.893	34.722
6	-5.891	34.708
7	-5.890	34.695
8	-5.889	34.681
9	-5.888	34.667

Table 3: Learning rate $r = 0.0001$

5 Comment

In the above experiment, I witnessed some interesting findings:

- The algorithm showed progress: even with different initial value x and learning rate, it steadily approached global minium value of function $f(x) = x^2$.
- The experiment also revealed the impact of learning rate: a small $r = 0.0001$ leaded to extremely slow convergence, while with $r = 0.1$, the algorithm converged and reached global minium faster.

6 Conclusion

I have demonstrated a simple implementation of the Gradient Descent algorithm. I found that the algorithm works well, but it depends also on proper learning rate, and the number of iterations, really affect the final result.