

Санкт-Петербургский политехнический университет Петра Великого
Институт металлургии, машиностроения и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Программирование на языках высокого уровня
Тема: Метод градиентного пуска

Выполнил студент гр. 3331506/70401

Архипов А.Е.

Преподаватель

Ананьевский М.С.

« ____ » _____ 2020 г.

Санкт-Петербург
2020

Содержание

1. Формулировка задачи, которую решает алгоритм	3
2. Словесное описание алгоритма	3
3. Реализация алгоритма	4
4. Анализ алгоритма	6
5. Применение алгоритма	6
6. Заключение	6

1. Формулировка задачи, которую решает алгоритм

Градиентный спуск — один из наиболее популярных методов оптимизации в машинном обучении. Оптимизация — это процесс нахождения точек максимума/минимума некоторой функции. Задача оптимизации может быть представлена следующим образом:

Дано:

- множество $X = \{x_0, x_1, \dots, x_m\} \subset R^n$
- функция $f : X \rightarrow \mathbb{R}$

Тогда решить задачу оптимизации означает одно из:

- Показать, что $X = \emptyset$
- Найти такие $x_0, x_1, \dots, x_m \in X$, что $f(x_0, x_1, \dots, x_m) = \min f(x)$

В данной работе будет рассмотрен не совсем классический метод градиентного спуска, а его модернизированный вариант, под названием метод наискорейшего спуска. Программа позволяет оптимизировать функции с любым количеством переменных.

2. Словесное описание алгоритма

1. Задать начальное приближение x_0 , и точность ε
2. Рассчитать градиент $\nabla f(\bar{x}^j)$, частные производные берутся с помощью численного дифференцирования центрально-разностной формуле второго порядка.

$$\frac{\partial f(x_1, \dots, x_i, \dots, x_n)}{\partial x_i} = \frac{f(x_1, \dots, x_i + \Delta, \dots, x_n) - f(x_1, \dots, x_i - \Delta, \dots, x_n)}{2\Delta}$$

3. Рассчитать единичный вектор направления функции \bar{S}^j по формуле

$$\bar{S}^j = \frac{\nabla f(\bar{x}^j)}{\|\nabla f(\bar{x}^j)\|}$$

4. Итерационное нахождение экстремума функции

$$\bar{x}^{j+1} = \bar{x}^j \pm \lambda^j \cdot \bar{S}^j$$

5. Шаг λ^j проверяется и при необходимости корректируется в каждой итерации:

$$\text{если } f(\bar{x}^j) \geq \bar{S}^j, \text{ то } \lambda^{j+1} = \lambda^j / 2$$

6. Условие останова:

$$\lambda^j < \varepsilon$$

3. Реализация алгоритма

Листинг программы с реализацией алгоритма градиентного спуска и текстового пользовательского интерфейсом приведен ниже.

```
#include <iostream>
#include <cmath>

using namespace std;

typedef double D;

class Model
{
public:
    D *variable;

    Model();
    D Function();
};

Model :: Model()
{
    variable = new D[3];
    // initial guess
    variable[0]=1.3;
    variable[1]=1.;
    variable[2]=2.;
}

//Enter a function
D Model :: Function()
{
    return variable[0] * variable[1] + variable[0] * variable[0] +
variable[1] * variable[1];
}

class Optimization : public Model
{
public:
    void implementation_descent();
    void calculate_gradient(const int number_variables, D *gradient, D
delta);
};

void Optimization :: calculate_gradient(const int number_variables, D
*gradient, D delta)
{
    int variable_counter;
    D function_value_right, function_value_left, module_gradient = 0;

    for(variable_counter = 0; variable_counter < number_variables;
variable_counter++)
    {
        variable[variable_counter] += delta;
        function_value_right = Function();
        variable[variable_counter] -= delta;

        variable[variable_counter] -= delta;
        function_value_left = Function();
        variable[variable_counter] += delta;
```

```

        gradient[variable_counter] = (function_value_right -
function_value_left) / (2 * delta);
        module_gradient += gradient[variable_counter] *
gradient[variable_counter];
    }
    module_gradient = sqrt(module_gradient);

    for(variable_counter = 0; variable_counter < number_variables;
variable_counter++)
        gradient[variable_counter] /= module_gradient;

    gradient[number_variables] = Function();
}

void Optimization :: implementation_descent()
{
    int variable_counter, number_iterations= 0;
    D function_value, step, eps;
    const int number_variables = 2;
    D *gradient = new D[number_variables + 1];

    step = 0.1;
    eps = 0.000001;

    while(step > eps)
    {
        calculate_gradient(number_variables, gradient, 0.0001);
        for(variable_counter = 0; variable_counter < number_variables;
variable_counter++)
            variable[variable_counter] -= step *
gradient[variable_counter];

        function_value = Function();
        if(function_value >= gradient[number_variables])
        {
            step /= 2.;
            for(variable_counter = 0; variable_counter <
number_variables; variable_counter++)
                variable[variable_counter] += step *
gradient[variable_counter];
            number_iterations++;
            cout << number_iterations << " " << variable[0] << " " <<
variable[1] << " " << variable[2]<< endl;
        }
    }

    int main()
    {
        Optimization fun;
        fun.implementation_descent();
    }
}

```

4. Анализ алгоритма

При анализе оптимизационных методов стоит обращать внимание не только на сложность каждой итерации, но и на скорость сходимости. И, как правило, об эффективности методе судят именно по скорости сходимости. В данной работе будет проведено сравнение времени работы и количества итераций у «классического» метода градиентного спуска и метода наискорейшего спуска.

Функция	Классический метод		Метод наискорейшего спуска	
	Кол-во итераций	Время (мс)	Кол-во итераций	Время (мс)
$f(x, y) = x^2 + x \cdot y + y^2$	140	946	42	830
$f(x, y) = -\cos(x) \cdot \cos(y) \cdot e^{x+y}$	93	707	43	680
$f(x, y) = x^3 + y^2 - 6xy - 39x + 18y + 20$	979	3556	222	824

5. Применение алгоритма

Градиентный спуск — метод численной оптимизации, который может быть использован во многих алгоритмах, где требуется найти экстремум функции — нейронные сети, SVM, k-средних, регрессии.

6. Заключение

Метод градиента вместе с его многочисленными модификациями является распространенным и эффективным методом поиска оптимума исследуемых объектов. Недостатком градиентного поиска (так же и рассмотренных выше методов) является то, что при его использовании можно обнаружить только локальный экстремум функции. Для отыскания других локальных экстремумов необходимо производить поиск из других начальных точек. Так же скорость сходимости градиентных методов существенно зависит также от точности вычислений градиента. Потеря точности, а это обычно происходит в окрестности точек минимума или в овражной ситуации, может вообще нарушить сходимость процесса градиентного спуска.