

Neural Networks: Assignment 2

Pepijn van Heiningen
pvheinin@liacs.nl

Michiel Vos
msvos@liacs.nl

November 10, 2013

1 Introduction

The second assignment of the Neural Networks course consists of three tasks:

- Task 1: Function Optimization
- Task 2: The XOR Problem
- Task 3: Handwritten digit recognition

For the first task, we were given the Rosenbrock's function, and we were asked to test 5 different algorithms for finding the global minimum of this function. The purpose is to get an insight into the limitations of the classical gradient descent algorithm.

2 Task 1: Function Optimization

2.1 Problem Description

The Rosenbrock function is a function that is used as a performance test for optimization algorithms. The equation can be found in figure 1. It has a global minimum at the point (1,1), where the value of $f = 0$. This is visualized in figure 2.

$$f(x, y) = 100 * (y - x^2)^2 + (1 - x)^2$$

Figure 1: The Rosenbrock's function

We were given the task to optimize the Rosenbrock function using five different algorithms, and subsequently compare their performance, in order to get an insight into the advantages, disadvantages and limitations of the different algorithms.

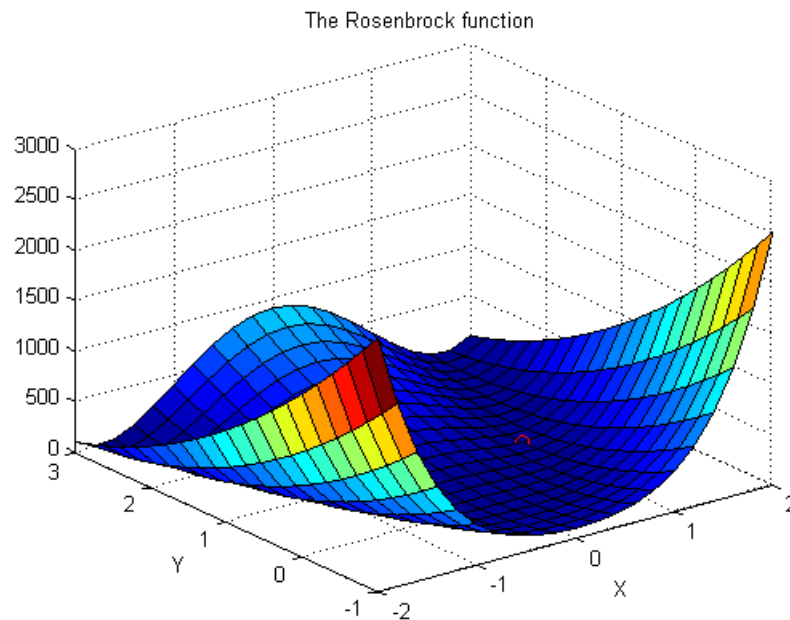


Figure 2: The Rosenbrock function, with minimal point

The 5 different algorithms we tested are:

- Gradient descent
- Gradient descent with line search
- Scaled conjugate gradient
- Conjugate gradient
- Quasi-Newton

To get a good comparison between the algorithms, each algorithm was run 100 times. We first generated 100 random points around (-1,1) as starting points. Each algorithm was started from the same random point.

There were 4 different measures to compare the algorithms with:

- The average number of evaluations of f
- The average number of evaluations of the gradient of f
- The average run time of the algorithm
- The average “success rate”.

Together these measures should provide a decent indication which algorithm performs better. Because some functions might be a lot more computationally expensive to evaluate than the Rosenbrock function, the number of evaluations of both the function and the gradient should be as low as possible. A run obviously shouldn't take too long, and it should have a high success rate.

The success rate is measured as reaching the minimum with an accuracy of 0.0001. This means that when the optimal point found by the algorithm is evaluated, the value of f is smaller than 0.0001. Of course we would like to have an optimizer that gets a 100% accuracy every time the algorithm is ran, but that is not always possible.

2.2 Implementation

In figure 3 you can see the pseudo-code of the implemented algorithm.

```
starting_point = repmat([-1,1],100,1) + 0.5*randn(100,2);
options = foptions;           % Standard options
options(1) = -1;              % Turn off printing completely
options(3) = 1e-8;            % Tolerance in value of function
options(14) = 100;            % Max. 100 iterations of algorithm
options(18) = 0.001;          % Learning rate
default_options = options;
functionList = {@graddesc, @graddesc, @scg, @conjgrad, @quasineu};
for i = 1:100
    for j = 1:5
        options = default_options;
        if(j==1 || j==2)
            options(18) = 0.008;
        end
        if(j==2)
            options(7) = 1;
        end
        tic;
        [dump, options, dump, dump] = functionList{j}('rosen', starting_point(i,:), options, 'roseg');
        time = toc;
        results(i,j,1) = options(10);
        results(i,j,2) = options(11);
        results(i,j,3) = time;
        results(i,j,4) = options(8);
    end
end
```

Figure 3: Pseudo-code of task 1.

As described in the assignment, we run the algorithms using 100 randomly generated starting points from the distribution $[-1, 1] + 0.5 * \text{randn}(1,2)$.

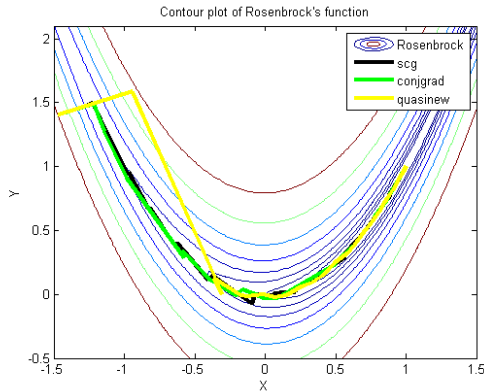
We set the options of the algorithm to the default options, with a few changes, we set the tolerance of the function value to $1 * 10^{-8}$, the iterations to 100, and we use a different learning rate for the two gradient descent algorithms. The reasoning behind the value is described in section 2.3.

2.3 Experiments

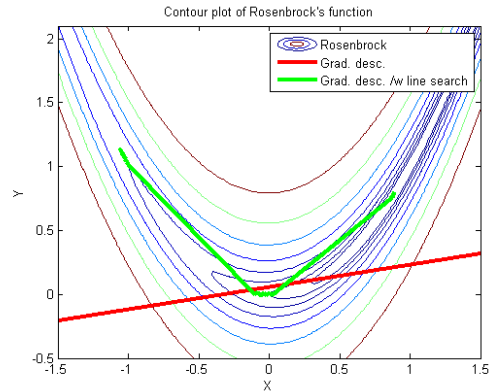
In task 1.3 we were asked to tune both gradient descent algorithms manually. We set the number of iterations to 100.000, and tested different values of the learning rate.

The values are averages over 500 runs. As you can see the optimal error is somewhere around a learning rate of 0.008, which is why we set the learning rate for the experiment to that value.

In order to acquire additional insights into how the algorithms work, we plotted the optimal points found after each iteration in the following plot.



(a) Nadenken



(b) Nadenken

2.4 Conclusions

3 Task 2: The XOR Problem

3.1 Problem Description

3.2 Implementation

3.3 Experiments

3.4 Conclusions

4 Task 3: Handwritten Digit Recognition with MLP

4.1 Problem Description

4.2 Implementation

4.3 Experiments

4.4 Conclusions