Optimization for Learning - FRTN50
Assignment 2

Introduction

This assignment will examine the effect of different problem properties on the convergence of gradient descent. For which problem classes we can design globally convergent algorithms and how those properties affect the rate of convergence will be the main focus.

ProximalOperators.jl In Assignment 1 you implemented functions that calculate gradients and proximal operators of some objective functions. However, this should be avoided whenever possible due to the ease of making mistakes. In this assignment we will therefore use a software package that contain implementations of gradients and proximal operators for a wide range of functions. The package is called `ProximalOperators.jl` and includes also basic operations such as scaling and conjugation. Following is a short example on how to use `ProximalOperators.jl` to compute the gradient and prox for the squared Euclidean norm. For more information we refer to the package documentation `http://kul-forbes.github.io/ProximalOperators.jl/stable/`. It contains simple examples and a list of supported functions and operations that can be performed on them.

```
f = SqrNormL2() # Create the function (1/2)‖·‖₂²
val = f([1.0, 1.0]) # val = 1.0
df, _ = gradient(f, [1.0, 1.0]) # df = [1.0, 1.0]
pf, _ = prox(f, [1.0, 1.0], 0.5) # pf = [0.6666..., 0.6666...]
```

Remark Another way to compute gradients is to automate it using *backpropagation/automatic differentiation*. We will use this in the next assignment.

Preliminaries

You will be asked to solve a number of problems and answer questions regarding the behavior of different algorithms. Before that, you will need to classify the problems in terms of what properties the objective functions have.

Written by: Martin Morin
Latest update: September 21, 2020

Task 1 Consider the following problems:

$$\min_{x \in \mathbf{R}^n} \tfrac{1}{4}\|x\|_2^2 + \tfrac{1}{2}\|x\|_2 \tag{1}$$

$$\min_{x \in \mathbf{R}^n} \tfrac{1}{3}\|x\|_2^3 \tag{2}$$

$$\min_{x \in \mathbf{R}^n} \tfrac{1}{2}\|x\|_2^2 \tag{3}$$

$$\min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} \tfrac{1}{2} \log(1 + e^{-l_i a_i^T x}) \tag{4}$$

$$\min_{x \in \mathbf{R}^n} \sum_{i=1}^{N} \tfrac{1}{2}(a_i^T x - l_i)^2 \tag{5}$$

The problem parameters $a_i \in R^n$ and $l_i \in \{-1, 1\}$ are defined in `Julia`-functions in the file `problems.jl`. Which of the problem objectives are L-smooth? Which are σ-strongly convexity? Motivate your answers and specify the property parameters.

Task 2 Problems (1), (2) and (3) are particularly simple. Prove that all three have the unique solution 0.

Convergence with Constant Step-Size

We'll start by examining the first three problems and will try to solve them with gradient descent, $x^{k+1} = x^k - \gamma \nabla f(x^k)$, with a constant step-size. The problems are scaled so they have the same gradient at $x = 1$ in order to make fair comparisons between them. Note that they are not all differentiable and so technically we can not use gradient descent for all of them. However, `ProximalOperators.jl` will return a subgradient when trying to calculate the gradient of a non-differentiable function, allowing us to perform a subgradient descent instead. The distinction between gradient and subgradients are important but not one we will comment on further in this assignment.

Since the solutions to problems (1), (2) and (3) are known we will judge convergence by plotting the distance to the solution at each iteration. To understand the behavior it might also be beneficial to plot the objective function itself together with the iterates and their function value. In order to make this visualization possible we will only consider the one dimensional case, $n = 1$.

Remark The dimension of these problems does not make any real difference. The problems are all rotationally symmetric and the iterates of gradient descent will therefore only move along a single line.

Task 3 Solve (1) with (sub-)gradient descent. The file `problems.jl` contains functions for generating the objective function. Start at $x = 1$ and try the step-sizes $\gamma \in \{0.3, 0.9, 1.8, 2.1\}$. Does the distance to the solution converge? Do the iterates appear to converge to the solution? Does increasing or decreasing the step-size affect how close to the solution you can come? Does increasing or decreasing the step-size affect how fast you can reach a given distance to the solution? Find a step-size in the interval $(0, 3)$ that makes the iterates converge to the solution. Try changing the initial point to $x = 1.1$, do you still converge?

Hint: The functions in ProximalOperators.jl require the input to be an array, even when the dimension is one.

Task 4 Solve (2) with gradient descent. The file `problems.jl` contains functions for generating the objective function. Use the step-size $\gamma = 1$ and try the initial points $x \in \{0.3, 0.9, 1.8, 2.1\}$. Does the distance to the solution converge? Do the iterates appear to converge to the solution? Does initial point affect how close to the solution you can come? **Hint**: In this case, if the iterates converge, they do so very slowly. It might it therefore not be possible to run until machine precision is achieved, use your best judgment for when to stop.

Task 5 Solve (3) with gradient descent. The file `problems.jl` contains functions for generating the objective function. Use the step-size $\gamma = 1.5$ and try the initial points $x \in \{0.2, 2.0, 20\}$. Do the iterates appear to converge to the solution? Does initial point affect how close to the solution you can come? Start at $x = 3.14$ and search in $(0, 3)$ for the maximal step-size that still guarantee convergence. Try this step-size on the previous three initial points. Does the step-size required for convergence appear to depend on the initial point?

Task 6 When designing algorithms or choosing algorithm parameters, all decisions need to be based on known information. For instance, the solution can't be used in any way since it, of course, is not known. This limits the available information to global properties such as the ones in Task 1. Based on the previous three tasks, which property, apart from convexity, seems necessary in order to construct a globally convergent gradient descent algorithm?

Task 7 If the problem objective is cheaply proximable these restrictions can be loosen by using the proximal point method, $x^{k+1} = \text{prox}_{\gamma f}(x^k)$, instead of gradient descent. Of the first three problems, (2) and (3) are proximable solve them with the proximal point method. Start at $x = 1$ and try several step-sizes in the interval $(0, \infty)$. Does it appear to exist an upper bound on the step-size in order to achieve convergence? Does increasing the step-size affect how fast you converge? Give intuition into why/why not it affects the speed of convergence. **Hint**: What would $\text{prox}_{\gamma f}$ be if one could set $\gamma = \infty$?

Remark The proximal point method as presented here is not the most useful algorithm since it requires the entire minimization objective to be proximable. Proximable functions are usually minimized equally easily, negating the need to solve them iteratively. The proximal operator is mainly useful when the objective can be broken down into simpler parts and combined with either gradient or other proximal steps.

Convergence Rates

From the previous task you probably noticed the significant difference in convergence speed. The rate of which an algorithm converge is of course of great interest. Even if one knows that you eventually get within the desired precision, it does not matter if it takes until the heat death of the universe.

In general, first order methods are usually classified as either linearly, sublinearly och superlinearly convergent for a problem class. A sequence $\{\alpha^k\}_{k\in\mathbf{N}}$ converge to A linearly if

$$\lim_{k\to\infty} \frac{|\alpha^{k+1} - A|}{|\alpha^k - A|} = \rho \in (0, 1) \tag{6}$$

where ρ is called the *linear rate*. Sub-, and superlinear convergence simply means that a sequence converge slower or faster than linearly. Assuming the quotient in (6) is equal to the linear rate ρ for all k we can write the convergence as

$$|\alpha^{k+1} - A| = \rho|\alpha^k - A| = \rho^{k+1}|\alpha^0 - A|$$

and it's clear that linear convergence means that the distance to the limit decreased by a constant proportion each iteration. The reason this is called linear convergence is clear if one takes the log of this expression,

$$\log(|\alpha^k - A|) = k\log(\rho) + \log(|\alpha^0 - A|). \tag{7}$$

The log of the distance to A is clearly a linear expression in k.

The goal here is to examine the convergence rate of gradient descent when applied to the logistic regression and least squares problems, (4) and (5). We will look at both which objective properties give sub-, super-, and linear convergence as well how the property parameters affect the rate. Different to the previous tasks, we are not interested in if we converge but rather how fast. In fact, the step-sizes used in these tasks guarantee convergence, given a correct implementation of gradient decent. Also different from previous tasks, we do not know the distance to the solution and we will instead look at the convergence of the gradient to zero.

Task 8 Solve problems (4) and (5) with gradient decent. The problem parameters $a_i \in r^n$ and $l_i \in \{-1, 1\}$ are defined in `Julia`-functions in the file `problems.jl`. Start at $x = 0$, run for 1000 iterations and use the step-size $\gamma = 1.2/L$ where L is the smoothness constant of the objective function. For each problem, specify if the gradient converge to zero sub-, super-, or linearly. Which objective property seem responsible achieving fast convergence? **Hint**: $\sum_{i=1}^N (a_i^T x - l_i)^2 = \|Ax - l\|_2^2$ where the ith row of A and l is given by a_i and l_i respectively.

Prescaling Prescaling is the practice of transforming the problem data in some way in hopes of improving the convergence speed of the optimization/training. There are many ways of doing this, but here we consider linear transformations. These can be seen as a variable change in the original optimization problem, i.e., setting $x = H\hat{x}$ where is H is a full rank square matrix yields

$$\min_{x\in\mathbf{R}^n} f(Ax) \iff \min_{\hat{x}\in\mathbf{R}^n} f(AH\hat{x}) \iff \min_{\hat{x}\in\mathbf{R}^n} f(\hat{A}\hat{x})$$

where $\hat{A} = AH$. The rightmost problem can then be solved and the solution to the original problem can then be recovered by $x^\star = H\hat{x}^\star$.

The result of this variable change is clearly a problem on the same form but with a different data matrix $\hat{A}$. This change the smoothness and strong-convexity parameters, L and σ, and therefore the convergence rate. It turns out that the convergence rate is not really affected by both L and σ simultaneously but only by the ratio $\kappa = \frac{L}{\sigma}$. This ratio is called the condition number of the problem and the goal when designing a prescaling H is to get a more advantageous condition number.

A good prescaling H should of course yield a good condition number but also not be to expensive to compute. The design of prescaling H is a non-trivial task which usually require domain and problem specific knowledge. For this reason we have provided a function `prescale(A,r)` in `problems.jl` that computes a matrix H suitable for use in the next task. The function takes a data matrix A and a number $r \in [0, 1]$ and returns H. The parameter r sets the trade-off between good prescaling and low computational cost. Setting $r = 1$ yields the best H but is also the most computationally intensive while $r = 0$ gives the H that is cheapest to compute but worst performing. You are encouraged to read the source of `prescale(A,r)` and try to understand what it does and why.

Task 9 Solve the least squares problem (5) with a prescaled gradient descent. Start at $x = 0$, use a step-size of $\gamma = 1.2/L$, and try the prescaling parameters $r \in \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. For each prescaling parameter, calculate the condition number of the scaled problem. What happens with the convergence speed and condition number when r is increased?

Remark There are other methods than prescaling for achieving lower condition numbers. For instance could Newton's method be seen as a form of post-scaling to do the same thing. The main difference is that the post-scaling needs to be applied at each iteration while the prescaling is done only once. This makes Newton-like methods much more costly and are in general deemed to expensive to use in modern learning problems.

Submission

Your submission should contain the following.

- All your code.
- A single pdf where you answer and/or discuss the questions raised in each task.

Use plots, figures and tables to motivate your answers.