Lab 4 - Tutorial Writing

Ratun Rahman, 170042011

August 2021

Gradient Descent

1 Introduction

Gradient descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent is simply used in machine learning to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible. You start by defining the initial parameter's values and from there gradient descent uses calculus to iteratively adjust the values so they minimize the given cost-function. To understand this concept full, it's important to know about gradients.

Gradient descent is generally attributed to Cauchy, who first suggested it in 1847. Hadamard

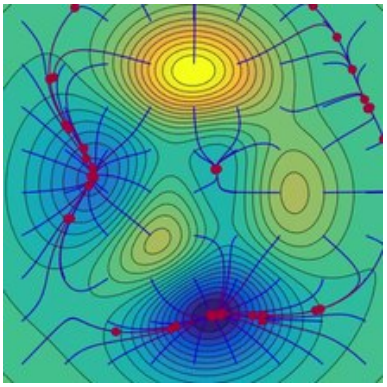

Figure 1: Gradient Descent in 2D

independently proposed a similar method in 1907. Its convergence properties for non-linear optimization problems were first studied by **Haskell Curry** in 1944, with the method becoming increasingly well-studied and used in the following decades, also often called steepest descent.

2 Problem Formulation

Gradient descent is based on the observation that if the multi-variable function F(x) is defined and differentiable in a neighborhood of a point a, then F(x) a in the direction of the negative gradient of F at a, -∇ F(a). It follows that, if

$$a_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

for a $\gamma \in R_+$ small enough, then $F(a_n) \geq F(a_{n+1})$. In other words, the term $\gamma \nabla F(a)$ is subtracted from a because we want to move against the gradient, toward the local minimum. With this observation in mind, one starts with a guess x_0 for a local minimum of F, and considers the sequence $x_0, x_1, x_2, \ldots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \ n \geq 0$$

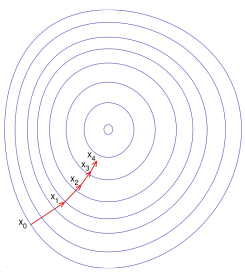

Figure 2: Illustration of gradient descent on a series of level sets

We have a monotonic sequence,

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots$$

So, hopefully, the sequence $\mathbf{x}_n$ converges to the desired local minimum. Note that the value of the step size γ is allowed to change at every iteration. With certain assumptions on the function F (for example, F convex and ∇F Lipschitz) and particular choices of γ (e.g., chosen either via a line search that satisfies the Wolfe conditions, or the Barzilai–Borwein method shown as following),

$$\gamma_n = \frac{\left|(\mathbf{x}_n - \mathbf{x}_{n-1})^T \left[\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})\right]\right|}{\left\|\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})\right\|^2}$$

convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

This process is illustrated in the adjacent picture. Here, F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient descent leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal.

3 General Solution

Since using a step size γ that is too small would slow convergence, and a γ too large would lead to divergence, finding a good setting of γ is an important practical problem. Philip Wolfe also advocated for using "clever choices of the [descent] direction" in practice. Whilst using a direction that deviates from the steepest descent direction may seem counter-intuitive, the idea is that the smaller slope may be compensated for by being sustained over a much longer distance.

To reason about this mathematically, let's use a direction $\mathbf{p}_n$ and step size γ_n and consider the more general update:

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma_n \mathbf{p}_n$$

.

Finding good settings of $\mathbf{p}_n$ and γ_n requires a little thought. First of all, we would like the update direction to point downhill. Mathematically, letting θ_n denote the angle between $\nabla F(\mathbf{a_n})$ and $\mathbf{p}_n$, this requires that $\cos\theta_n > 0$. To say more, we need more information about the objective function that we are optimising. Under the fairly weak assumption that F is continuously differentiable, we may prove that:

$$F(\mathbf{a}_{n+1}) \leq F(\mathbf{a}_n) - \gamma_n \|\nabla F(\mathbf{a}_n)\|_2 \|\mathbf{p}_n\|_2 \left[\cos\theta_n - \max_{t\in[0,1]} \frac{\|\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n) - \nabla F(\mathbf{a}_n)\|_2}{\|\nabla F(\mathbf{a}_n)\|_2}\right] \quad (1)$$

This inequality implies that the amount by which we can be sure the function F is decreased depends on a trade off between the two terms in square brackets. The first term in square brackets measures the angle between the descent direction and the negative gradient. The second term measures how quickly the gradient changes along the descent direction. In principle inequality (1) could be optimized over $\mathbf{p}_n$ and γ_n to choose an optimal step size and direction. The problem is that evaluating the second term in square brackets requires evaluating

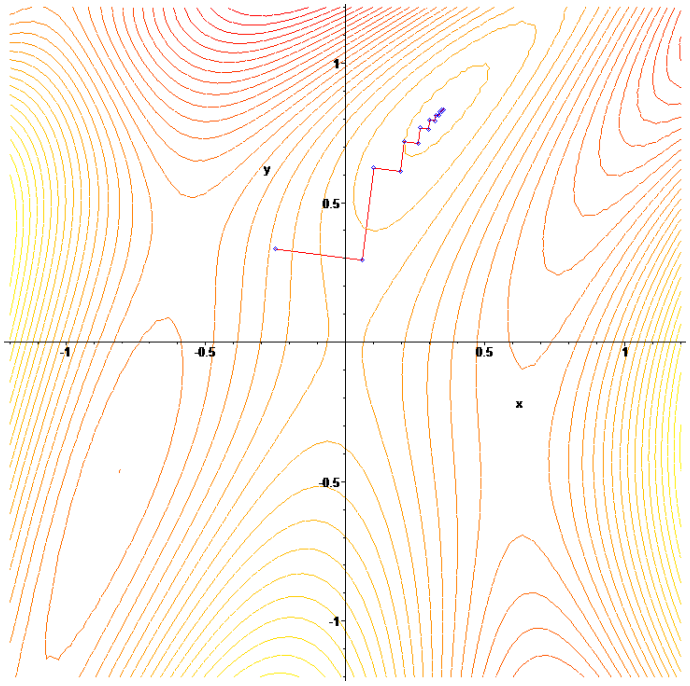

Figure 3: The gradient descent algorithm in action (contour)

$\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n)$, and extra gradient evaluations are generally expensive and undesirable. Some ways around this problem are:

1. Forgo the benefits of a clever descent direction by setting $\mathbf{p}_n = \nabla F(\mathbf{a_n})$, and use line search to find a suitable step-size γ_n, such as one that satisfies the Wolfe conditions.
2. Assuming that F is twice-differentiable, use its Hessian $\nabla^2 F$ to estimate

$$\|\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n) - \nabla F(\mathbf{a}_n)\|_2 \approx \|t\gamma_n \nabla^2 F(\mathbf{a}_n)\mathbf{p}_n\|.$$

Then choose $\mathbf{p}_n$ and γ_n by optimising inequality (1).

3. Assuming that ∇F is Lipschitz, use its Lipschitz constant L to bound $\|\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n) - \nabla F(\mathbf{a}_n)\|_2 \leq Lt\gamma_n \|\mathbf{p}_n\|$. Then choose $\mathbf{p}_n$ and γ_n by optimising inequality (1).
4. Build a custom model of

$$\max_{t \in [0,1]} \frac{\|\nabla F(\mathbf{a}_n - t\gamma_n \mathbf{p}_n) - \nabla F(\mathbf{a}_n)\|_2}{\|\nabla F(\mathbf{a}_n)\|_2}$$

for F. Then choose $\mathbf{p}_n$ and γ_n by optimising inequality (1).
5. Under stronger assumptions on the function F such as convexity, more advanced techniques may be possible.

Usually by following one of the recipes above, convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

4 Example

Consider the nonlinear system of equations

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{cases}$$

Let us introduce the associated function

$$G(\mathbf{x}) = \begin{bmatrix} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} \end{bmatrix}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

One might now define the objective function

$$F(\mathbf{x}) = \frac{1}{2} G^{\mathrm{T}}(\mathbf{x}) G(\mathbf{x})$$

$$= \frac{1}{2}\left[\left(3x_1 - \cos(x_2 x_3) - \frac{3}{2}\right)^2 + \left(4x_1^2 - 625x_2^2 + 2x_2 - 1\right)^2 + \left(\exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3}\right)^2\right],$$

which we will attempt to minimize. As an initial guess, let us use

$$\mathbf{x}^{(0)} = \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We know that

$$\mathbf{x}^{(1)} = \mathbf{0} - \gamma_0 \nabla F(\mathbf{0}) = \mathbf{0} - \gamma_0 J_G(\mathbf{0})^{\mathrm{T}} G(\mathbf{0})$$

where the Jacobian matrix J_G is given by

$$J_G(\mathbf{x}) = \begin{bmatrix} 3 & \sin(x_2 x_3) x_3 & \sin(x_2 x_3) x_2 \\ 8x_1 & -1250x_2 + 2 & 0 \\ -x_2 \exp(-x_1 x_2) & -x_1 \exp(-x_1 x_2) & 20 \end{bmatrix}.$$

We calculate:

$$J_G(\mathbf{0}) = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 20 \end{bmatrix}, \qquad G(\mathbf{0}) = \begin{bmatrix} -2.5 \\ -1 \\ 10.472 \end{bmatrix}$$

Thus

$$\mathbf{x}^{(1)} = \mathbf{0} - \gamma_0 \begin{bmatrix} -7.5 \\ -2 \\ 209.44 \end{bmatrix}$$

and

$$F(\mathbf{0}) = 0.5\left((-2.5)^2 + (-1)^2 + (10.472)^2\right) = 58.456$$

Now, a suitable γ_0 must be found such that

$$F\left(\mathbf{x}^{(1)}\right) \leq F\left(\mathbf{x}^{(0)}\right) = F(\mathbf{0})$$

This can be done with any of a variety of line search algorithms. One might also simply guess $\gamma_0 = 0.001$, which gives

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0.0075 \\ 0.002 \\ -0.20944 \end{bmatrix}$$

Evaluating the objective function at this value, yields

$$F\left(\mathbf{x}^{(1)}\right) = 0.5\left((-2.48)^2 + (-1.00)^2 + (6.28)^2\right) = 23.306$$

The decrease from F(0) = 58.456 to the next step's value of

$$F\left(\mathbf{x}^{(1)}\right) = 23.306$$

is a sizable decrease in the objective function. Further steps would reduce its value further, until an approximate solution to the system was found.

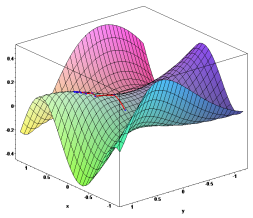

Figure 4: The gradient descent algorithm in action

5 Complexity Analysis

---

according to the Machine Learning course by Stanford University, the complexity of gradient descent is $O(kn^2)$, where

k= Number of iteration

n= Number of variables

When n is very large, it is recommended to use Gradient Descent algorithm.