École polytechnique fédérale de Lausanne

Semester project

Bachelor or Master in Mathematics or Minor heading such as course title

Title of your document
Second line of your title

Author:
First Name Last name

Supervisor:
First Name Last name

Contents

1 Introduction

In numerical optimisation, one aims to find the minimizer

$$\min_x l(x)$$

of a differentiable scalar field (also known as "loss function") $l : \mathbb{R}^n \to \mathbb{R}$. The quintessential algorithm to solve this problem is gradient descent (GD) which given a step size $s \in \mathbb{R}$ computes a sequence of iterates

$$x_{k+1} = x_k + s\nabla l(x_k)$$

whose convergence properties are suitable for a wide range of applications and have been extensively studied by thae optimization commmunity.

Nowadays, there are problems in which we would like instead to calculate equilibria or "saddle points" of our scalar field. We change the problem in the following way: the loss $l : \mathbb{R}^n \times \mathbb{R}^m \to R$ now takes two arguments x and y and we would like to know which are the values so that

$$\min_x \max_y l(x, y)$$

is reached.

One of the most natural approaches we could try to solve this new problem would be to adapt gradient descent in the following way

$$\begin{aligned} x_{n+1} &= x_n - s\nabla_x l(x_n, y_n) \\ y_{n+1} &= y_n + s\nabla_y l(x_n, y_n) \end{aligned}$$

to obtain gradient descent-ascent (GDA). However we can show that even on a simple example GDA fails to find a saddle point.

Example 1. Take $l(x, y) = xy$ with $x, y \in \mathbb{R}$. Then l has a saddle point at the origin, but GDA gives

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & -s \\ s & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix}.$$

We can see that the norm of the new iterate is $\sqrt{1 + s^2}$ times the old one, and so GDA diverges.

For the rest of this report we write $z = (x, y)$ the concatenation of x and y and $\nabla l(z) = (-\nabla_x l(x, y), \nabla_y l(x, y))$ so that GDA can be written simply as $z_{n+1} = z_n + s\nabla l(z_n)$ Given that, many different algorithms have been proposed to find saddle points. We give here some of the most popular ones:

- (EG) Extra gradient: $z_{n+1} = z_n + s\nabla l(z_n + s\nabla l(z_n))$
- (OGDA) Optimistic gradient descent-ascent: $z_{n+1} = z_n - 2s\nabla l(z_n) + s\nabla l(z_{n-1})$

These algorithms have been shown to converge and so we would like to understand if there is a fundamental difference that causes the difference in behaviour.

Explain in what conditions, cite?

1.1 Deriving continuous time dynamics from discrete time algorithms

Following the paper of [SBC16], there has been a lot of attention towards deriving differential equations $\dot{Z}(t) = f(Z(t), t)$ whose trajectory follows a given discrete time algorithms. In this subsection we show that by simply letting the timestep $s \to 0$ all of the algorithms presented so far yield the same ODE, gradient flow (GF). The main idea is to assume that the discrete algorithm is obtained from a continuous curve at fixed length times

$$z_n = Z(t_n)$$

where $t_n = t_0 + ns$. Using this, algorithms of the form

$$z_{n+1} = z_n + s\varphi(z_n)$$

can be rewritten as

$$\frac{Z(t_n + s) - Z(t_n)}{s} = \varphi(Z(t_n)).$$

Finally, taking limits $s \to 0$ gives

$$\dot{Z}(t) = \varphi(Z(t)),$$

where we dropped the index from t_n.

Example 2. Given that in GDA $\varphi(z_n)$ takes the form $\nabla l(z_n)$ we obtain a continuous time approximation

$$\dot{Z}(t) = \nabla l(Z(t)),$$

which we call gradient flow.

Example 3. For EG, the update step gives

$$\frac{Z(t+s) - Z(t)}{s} = \nabla l\big(Z(t) + s\nabla l(Z(t))\big),$$

and taking the limits gives gradient flow again.

todo: OGDA

Given that GDA diverges even in simple cases and that the other two algorithms are known to converge in a large class of situations, we are quite dissatisfied in doscovering that they present the same continuous time dynamics. In Chapter 2 we will present a new method that makes use of modified differential equations to obtain continuous time approximations that more closely follow their discrete time counterparts.

todo: natural descent from mirror descent

2 Methods to derive ODEs

While scouring the literature, there are two main methods we have found on how to derive ODEs that involve high-order terms. We present both methods in this subsection.

2.1 High-resolution differential equations

The idea of this approach is to imagine that a sequence of iterates from a discrete time algorithm $z_1, \ldots, z_n$ are obtained at regular intervals from a sufficiently smooth curve $z : \mathbb{R} \to \mathbb{R}^n$, so that $z(n\delta) = z_n$ for some $\delta \in \mathbb{R}$. In doing so, we can now take a Taylor expansion of

$$z_{n+1} = z(n\delta + \delta) = z(n\delta) + \dot{z}(n\delta)\delta + \frac{\delta^2}{2}\ddot{z}(n\delta) + O(\delta^3)$$

Now we can replace the left-hand side with whatever algorithm $z_{n+1} = \varphi(z_n, s)$ we were planning to use to obtain the high-resolution equation.

Example 4. GDA has for update rule $z_{n+1} = z_n + sF(z_n)$. Using this expression in the left-hand side of Equation (2.1) we find that

$$F(z(t)) = \frac{1}{s}\left(\dot{z}(t)\delta + \frac{1}{2}\ddot{z}(t)\delta^2 + O(\delta^3)\right).$$

By setting $\delta = s$ and letting $s^2 \to 0$ we find that

$$\ddot{z}(t) = \frac{2}{s}F(z(t)) - \frac{2}{s}\dot{z}(t).$$

Example 5. OGDA has for update rule $z_{n+1} = z_n - 2sF(z_n) + sF(z_{n-1})$. So now, we also need to derive a Taylor expansion of the term $F(z(ns - s))$. Which is given by $F(z(n\delta - \delta)) = F(z(n\delta)) - \delta\nabla F(z(n\delta))\dot{z}(n\delta) + O(\delta^2)$ So now we can replace the terms in (2.1) and we find

$$\dot{z}(t)\delta + \frac{\delta^2}{2}\ddot{z}(t) + o(\delta^3) = -sF(z(t)) - s\delta\nabla F(z(t))\dot{z}(t) + so(\delta^2).$$

By setting $\delta = s$ and letting $s^2 \to 0$ this becomes

$$\ddot{z}(t) = -\frac{2}{s}F(z(t)) - 2\dot{z}(t)\left(\frac{1}{s} + \nabla F(z(t))\right)$$

2.2 $O(s^r)$-resolution framework

In light of [Lu21] we present here a method to obtain differential equations from discrete algorithms. We assume that the velocity depends on the current position in the following way

$$\dot{Z}(t) = f_0(Z(t)) + sf_1(Z(t)) + s^2 f_2(Z(t)) + \dots$$

We want to find what are the f_i functions. To alleviate the notation we will write Z instead of $Z(t)$. We obtain a Taylor expansion

$$\begin{aligned} Z(t+s) &= Z + s(f_0(Z) + sf_1(Z) + \dots) \\ &\quad + \frac{s^2}{2}(f_0'(Z) + sf_1'(Z) + \dots)(f_0(Z) + sf_1(Z) + \dots) + \dots + o(s^r) \end{aligned}$$

On the other hand, given a numerical algorithm $z_{n+1} = \varphi(z_n, s)$ we can also express it as a Taylor expansion on s,

$$\varphi(z,s) = \varphi(z,0) + s\partial_s\varphi(z,s) + \frac{s^2}{2}\partial_{ss}\varphi(z,s) + \dots + o(s^r).$$

In order to obtain $Z(t_n) = z_n$ like the intuition in the introduction, we need to have $\varphi(z,s) = Z(t+s)$. By equating the coefficients in s for both polynomials we find that

$$\begin{aligned} f_0 &= \partial_s\varphi \\ 2f_1 &= \partial_{ss}\varphi - f_0' f_0, \end{aligned}$$

and the other coefficients are found in the same manner.

Example 6. With GDA we find that $\partial_s\varphi(z,s) = \nabla l(z)$ and that $\partial_{ss}\varphi(z,s) = 0$. So, the first two coefficients are

$$\begin{aligned} f_0 &= \nabla l(z) \\ f_1 &= -\frac{1}{2}\nabla^2 l(z)\nabla l(z) \end{aligned}$$

Example 7. With EG the update rule is $\varphi(z,s) = z + s\nabla l(z + s\nabla l(z))$. By Taylor expansion of the second term, we find that this expression equals

$$z + s\nabla l(z) + s^2\nabla^2 l(z)\nabla l(z) + \frac{s^3}{2}\nabla^3 l(z)\big(\nabla l(z), \nabla l(z)\big).$$

By equating the two polynomials as usual we find that

$$\begin{aligned} f_0 &= \nabla l(z) \\ f_1 &= \frac{1}{2}\nabla^2 l(z)\nabla l(z) \end{aligned}$$

2.3 Example: Mirror descent $O(s^r)$-resolution ODE

Let $\psi : \mathbb{R}^n \to \mathbb{R}$ be what we call a mirror map. Then, mirror descent uses as update rule

$$g(z,s) = \nabla\psi^{-1}(\nabla\psi(z) + s\nabla l(z)).$$

We denote $\alpha_n(z) = \nabla^n\psi^{-1}(\nabla\psi(x))$ and $L^k(z) = \prod_{i=1}^k \nabla l(z)$ which is an expression that makes sense only when multiplied on the left by a tensor of sufficient order. So we see that $\nabla\alpha_n(z) = \alpha_{n+1}(z)\nabla^2\psi(z)$. Given that, the first derivative with respect to s is

$$\begin{aligned} &\nabla^2\psi^{-1}(\nabla\psi(z) + s\nabla l(z))\nabla l(z) \\ &= \alpha_2(z)L^1(z) + s\alpha_3(z)\nabla^2\psi(z)L^2(z) + \frac{s^2}{2}\big(\alpha_4(z)(\nabla^2\psi(z))^2 + \alpha_3(z)\nabla^3\psi(z)\big)L^3(z) \end{aligned}$$

where we performed a Taylor expansion of the first term.

3 Convergence properties of ODEs

3.1 Under the $O(s^r)$-framework

3.2 Under the high-resolution ODE framework

4 Constrained optimization setting

In this chapter we will decribe a variety of min-max optimization algorithms in the setting where the iterates have to remain in a subset K of $\mathbb{R}^n$. Ideally, we should obtain results similar to the ones mentioned in [WLZL21] but with much simpler and elegant proofs. As a reminder, the setting is the following: we want to solve

$$\begin{aligned} \min_x \max_y \quad & l(x,y) \\ \text{s.t.} \quad & z=(x,y)\in K. \end{aligned}$$

4.1 Projected gradient descent

We want the iterates to be in a subset $K \subseteq \mathbb{R}^n$, so we define

$$\begin{cases} \tilde{z}_{n+1} & = z_n + sF(z_n) \\ z_{n+1} & = \tilde{z}_{n+1} + v_K(\tilde{z}_{n+1}) \end{cases} \tag{1}$$

where $v_K(z)$ is the differnece between z and the projection onto K of z, let us call it the "projection vector". In what follows we require a certain number of properties from K, in particular we will need that the projection vector has a Jacobian.

4.1.1 Using $O(s^r)$-resolution framework

We write $\varphi(z,s) = z + sF(z) + v_K(z + sF(z))$ using the same methods as before we find that

$$\begin{aligned} f_0(z) = \partial_s\varphi(z,s) &= F(z) + \nabla v_K(z+sF(z))F(z) \\ f_0'(z) &= (Id + \nabla v_K(z+sF(z)))\nabla F(z) + \nabla^2 v_K(z+sF(z))[Id + s\nabla F(z)]F(z) \\ \partial_{ss}\varphi(z,s) &= \nabla^2 v_K(z+sF(z))(F(z),F(z)) \end{aligned}$$

So that $f_1(z) = \partial_{ss}\varphi(z,s) - 1/2 f_0' f_0(z)$ as usual. The main downside of this approach is that it severely restricts the kind of sets K we are allowed to use since the associated projection vector v_K needs to have high-order derivatives.

4.1.2 Using high-order ODEs

Using the ideas from Subsection 2.1 we can write Equation (1) as

$$\delta\dot{z}(t) + \frac{\delta^2}{2}\ddot{z}(t) + o(\delta^3) = sF(z(t)) + v_K(z(t) + sF(z(t)))$$

and in fact, we could keep expanding the Taylor polynomial to get high-order terms without having to differentiate v_K even once. We choose to set $\delta = s$ and we let $\delta^2 \to 0$ to obtain a high-resolution ODE for (1).

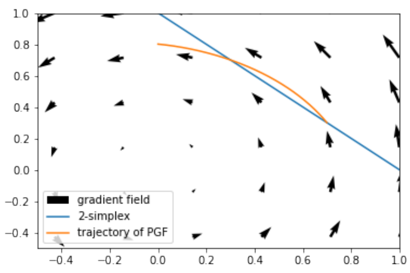
4.2 Trying out projected gradient flows

In the setting where K is a manifold

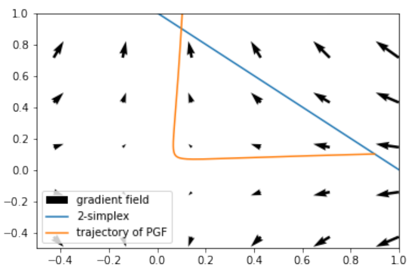
$$M = \{x \in \mathbb{R}^n | h_j(x) = 0, j \in J, g_i(x) \leq 0, i \in I\},$$

where the functions $h_j, g_i : \mathbb{R}^n \to \mathbb{R}$ are smooth and $|J| < n, |I| < \infty$. We will use the projected gradient flow derived by [SS09]. We will study the case in which M is given by

- $|J| = 1$ and $h(x) = \sum_{i=1}^n x_i$



(a) Using loss $l(x, y) = xy$



(b) Using loss $l(x, y) = x^2 + y^2$

Figure 1: Euler discretization of the projected gradient flow in the case of a 2-simplex, with two different loss funcions

- $|I| = n$ and $g_i(x) = -x_i$

so that in the end, M is the n-dimensional simplex. In this case, the projected gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$grad_{\tilde{R},\tilde{M},x} f = \nabla f - \mathfrak{F}\nabla f.$$

Where $\mathfrak{F}$ is given by the following expression

$$\mathfrak{F} = (Id + \mathrm{diag}(2x_i))^{-1} - \frac{1}{2\sum_{i=1}^n x_i}\left[\left(1 - \frac{1}{1+2x_i}\right)\left(1 - \frac{1}{1+2x_j}\right)\right]_{i,j}.$$

On Figure 1 we display the sequence of iterates given by the following discrete time algorithm

$$x_{n+1} = x_n - grad_{\tilde{R},\tilde{M},z} f(x_n)$$
$$y_{n+1} = y_n + grad_{\tilde{R},\tilde{M},z} f(y_n)$$

Where z is the concatenation of x and y. We see that both of the flows fail to remain on the manifold. We have tried many different step sizes and the result is the same. This is quite disappointing, but the authors do mention that the discretization is only meant to work on manifolds that have no constraints on equalities, namely $J = \emptyset$.

References

[Lu21] Haihao Lu. An \$O(sr̂)\$-Resolution ODE Framework for Understanding Discrete-Time Algorithms and Applications to the Linear Convergence of Minimax Problems. *arXiv:2001.08826 [cs, math]*, July 2021.

[SBC16] Weijie Su, Stephen Boyd, and Emmanuel J Candes. A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights. *Journal of Machine Learning Research*, 2016.

[SS09] V. Shikhman and O. Stein. Constrained Optimization: Projected Gradient Flows. *Journal of Optimization Theory and Applications*, 140(1):117–130, January 2009.

[WLZL21] Chen-Yu Wei, Chung-Wei Lee, Mengxiao Zhang, and Haipeng Luo. Linear Last-iterate Convergence in Constrained Saddle-point Optimization. *arXiv:2006.09517 [cs, stat]*, March 2021.