1 Derivative Free Optimization

In previous lectures, we have considered optimization problems where one gets access to function evaluations $f(x)$ as well as gradients $\nabla f(x)$. This is usually a reasonable setting in practice. An additional setting (which we do not consider here, but is an exciting research direction) is studying *higher order* methods for optimization, where in addition receiving function evaluations and gradients, one also gets to see higher order derivatives. For example, if one can readily calculate the Hessian $\nabla^2 f(x)$, there are many algorithms that can use such information. These are called *second-order* methods.

Derivative Free Convex Optimization. In this lecture, however, we go the other direction and study what is called *derivative-free optimization* (also called *black-box* or *zero-th order* optimization). Here the goal is same as before: find a solution to the problem

$$\min_{x \in \Omega} f(x).$$

where $\Omega \subseteq \mathbb{R}^d$ is a constraint set. However, now we restrict ourselves to only receiving function values $f(x)$ and not getting access to gradients $\nabla f(x)$!

Before we study methods for solving this problem, we motivate why to study it. Often, the feedback we get comes from a system for which we cannot efficiently compute derivatives for. Here are several examples:

- Hyperparameter selection. Even for the first order methods we have studied so far in this course, we have had to make certain parameter choices. These are called "hyperparameters". Especially in the deep learning era, one can think of many tunable hyperparameters: width of the model, depth of the model, learning rate, how much acceleration to use, etc. If we want to select the best model, we can also optimize over these hyperparameters.
- Control Problems. Imagine trying to design a system to control a drone to hover at a certain height with the minimum amount of power (In fact later today we will program a strategy to do so). A common strategy is PID control, which has 3 parameters. We can try to select the optimal choice of parameters using derivative free optimization.
- Routing Problems. Imagine every day we want to go from home to school, and we pick a different path $x \in \Omega$. When we get to school, we can see how long it took (some measure of cost which we denote $f(x)$), but we do not get to see gradients.
- Other examples in protein-folding, circuit design, etc.

1.1 Random Search Algorithm

Let's introduce a simple algorithm to solve this problem. It is stated in Algorithm 1.

Algorithm 1 is mostly meant to illustrate the central ideas. Basically, whatever was true for gradient descent that we showed before is true for Algorithm 1, up to a multiplicative cost of d. This is what we would expect, because for gradient descent, in every step we get $O(d)$ information, but here we only get $O(1)$.

Algorithm 1 Random Search

1: Pick an initial point $x_0 \in \Omega$.
2: **for** $t = 1, 2, \cdots, T$:
3: Let $u_t \in \mathbb{S}^{d-1}$ be a random direction.
4: Set step size $\eta_t = \operatorname{argmin}_{\eta > 0} f(x_t - \eta u_t)$.
5: Update $x_{t+1} = x_t - \eta_t u_t$.
6: **Return**

Analysis for Smooth Functions. Let's assume that f is a β-smooth convex differentiable function. Recall that this means that for any η, we have

$$\begin{aligned} f(x_t - \eta u_t) &\leq f(x_t) - \eta \langle u_t, \nabla f(x_t) \rangle + \frac{\eta^2 \beta}{2} \|u_t\|^2 \\ &= f(x_t) - \eta \langle u_t, \nabla f(x_t) \rangle + \frac{\eta^2 \beta}{2}, \quad \text{since } u_t \in \mathbb{S}^{d-1}. \end{aligned}$$

We can optimize over η now. For some choice of $\eta^\star$ we have

$$f(x_t - \eta^\star u_t) = f(x_t) - \frac{1}{\beta} \langle u_t, \nabla f(x_t) \rangle^2.$$

Moreover because of our choice for the step size η_t we have

$$f(x_{t+1}) = f(x_t - \eta_t u_t) \leq f(x_t - \eta^\star u_t) = f(x_t) - \frac{1}{\beta} \langle u_t, \nabla f(x_t) \rangle^2.$$

Taking expectations we have

$$\begin{aligned} \mathbb{E}[f(x_{t+1})] &\leq f(x_t) - \frac{1}{\beta} \mathbb{E}\big[\langle u_t, \nabla f(x_t) \rangle^2\big] \\ &\leq f(x_t) - \frac{1}{\beta d} \mathbb{E}\big[\|\nabla f(x_t)\|^2\big]. \end{aligned}$$

In expectation, this is similar to the type of results for gradient descent that we saw before, up to an additional factor of d. One can show similar convergence guarantees as before, but we will not do so now.

Line Search. Algorithm 1 is not that practical, since step 4 involves doing an exact line search. In practice however, we can approximate this with some kinds of heuristics. For 1D minimization, we can use something in Golden-Section Search (GSS), see, e.g., https://en.wikipedia.org/wiki/Golden-section_search. This will be implemented for you in code. It is also implemented in scipy: https://docs.scipy.org/doc/scipy/reference/optimize.minimize_scalar-golden.html.

Other Methods. There are other methods for solving derivative-free optimization. The wikipedia page has more details: https://en.wikipedia.org/wiki/Derivative-free_optimization. For example, there is a more sophisticated version of this due to Nelder and Mead https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method. Nelder-Mead works well in practice but does not have strong theoretical guarantees. However, if you are looking for a numerical method for zeroth-order optimization, it is implemented in packages like `scipy`.

2 RL and Optimal Control

We introduce the problem of optimal control. The optimal control problem has a few features:

- The states are represented by $x \in \mathbb{R}^{d_x}$.
- The actions are represented by $a \in \mathbb{R}^{d_a}$.
- There is noise or random disturbance which can be represented by some e_h.

The optimal control problem has a dynamical system model which is given by:

$$x_{h+1} = f(x_h, a_h, e_h).$$

The transition function f tells us for any input state x and action a, as well as noise e how the system transitions. For example, one popular model is the so-called linear dynamical system:

$$x_{h+1} = Ax_h + Ba_h + e_h.$$

However, we can imagine more complicated transition functions f. Associated with the optimal control problem is a cost function:

$$c_h(x_h, a_h).$$

This measures the amount of cost we incur. For example, if we want to control a robotic car, the cost can capture the distance from the desired trajectory as well as the amount of energy consumed.

The goal of the optimal control problem is to compute a good policy $\widehat{\pi} : \mathcal{X} \to \mathcal{A}$ that minimizes the total cost:

$$\min_{\widehat{\pi}} \sum_{h=1}^{H} \mathbb{E}_{e_h}[c_h(x_h, a_h)].$$

The policy tells us in every given state what action to take.

- So this is exactly a stochastic optimization problem! Often we will actually parameter the policy function with some parameter θ, so the problem can be equivalently written as

$$\min_{\theta} \sum_{h=1}^{H} \mathbb{E}_{e_h}[c_h(x_h, \pi_\theta(x_h))].$$

- We are working here with what is called Markovian policies, which only take as input the current state. One can also consider policies which are "history-dependent": they take into account the entire previous sequence of states and actions, i.e., the policy at time t is a function of $\tau_t = (x_1, a_1, x_2, a_2, \cdots, x_h)$.
- Here, we are using notation which is standard in RL. It is different than what we have been using before. The parameter to optimize here is θ.
- How can we solve this problem? In general, the dynamics function f may not be known to us. For example if f models the heat flow in a massive data center, then knowing f requires a deep understanding of physics and heat transfer. Even if we know f, computing the gradients of this cost function might be very difficult. Another approach is to learn an approximate model

of f: for example we can approximate it as a linear system. This is approach is called different names in different communities. In reinforcement learning, its often called model-based RL; another name from control theory is system identification. Once we have a simpler model $\widehat{f}$, then it might be possible to find a good policy. For linear dynamical systems, there is a rich literature on how to compute a good policy.

- We will go with a direct, model-free approach. In this case, our goal is to directly optimize the function without worrying about modeling the transition model. We can actually do so using the derivative-free optimization methods that we introduced before.

2.1 Solving Optimal Control with Random Search

We will illustrate this with the Flappy Bird game. Imagine we have a bird flying at a constant velocity, and it is being pulled down by gravity. The goal is to control the bird so that it doesn't hit the ground while minimizing the amount of acceleration.

- The bird's state can be written as two variables: $x = (x_h, x_v)$ where x_h represents the current height and x_v represents the velocity.
- The action that the bird can take is (upwards) accleration, represented by a scalar $a > 0$.
- The cost function is given by $c_h(x_h, a_h) = -10 \times \mathbb{1}\{x_h < 0\} + |a_h|$. That is, we get a large cost if our height is negative, and we get a smaller cost every time we accelerate. The goal is for the bird to not hit the ground while minimizing the amount of acceleration that it uses.
- The last thing to do is to instantiate the set of policies we are optimizing over. In this case, we will use linear policies parameterized by θ, so that the next action can be written as:

$$\pi(\tau) = \theta_1 \cdot x_{h,h} + \theta_2 \cdot x_{v,h} + \theta_3 \cdot x_{h,h-1} + \theta_4 \cdot x_{h,h-1}.$$

  Of course, this is just a design choice. We can also use other sort of policies. For example, we can use a Two-Layer NN (you will be asked to implement this).

We will use the random search algorithm to solve this problem. It seems to work ok (?).

2.2 Policy Gradient

It turns out that random search is not the predominant way to solve this direct policy optimization problem, although it is often a strong baseline. We will now introduce an alternative way to solve the RL problem, which can also be thought of as a zeroth order optimization problem for RL.

This is called the policy gradient method, and it underlies a lot of more state-of-the-art methods in RL that have received a lot of attention recently (TRPO, PPO). [gl: See also https://arxiv.org/pdf/1806.09460.pdf and https://lilianweng.github.io/posts/2018-04-08-policy-gradient/]. Even though there is the word "gradient" in the name, it is still a derivative free optimization method. The gradients are taken with respect to the policy parameters, and not the entire system.

Preliminaries. Before we do so, we need to introduce a bit more notation and background which is standard in the field of the reinforcement learning (RL). As before, we will consider a fixed horizon problem for $H \geq 0$ time steps. The initial state is given by x_1. We will alternatively write the transition function to be of the form $P(\cdot|x_t, a_t)$. And instead of working with costs, we will work with rewards $r(x_t, a_t)$. However, basically everything else is the same in RL as in optimal

control! Also, we will work with random Markovian policies. That is, we will write $\pi_\theta(x) \in \Delta(\mathcal{A})$ or sometimes denote the entries as $\pi_\theta(\cdot|x)$.

Lastly, we will introduce a notion of "reward to go":

$$V_h^\pi(s) = \mathbb{E}^{M,\pi}\left[\sum_{h'=h}^{H} r(s_h, a_h) \mid s_h = s\right]$$
$$Q_h^\pi(s) = \mathbb{E}^{M,\pi}\left[\sum_{h'=h}^{H} r(s_h, a_h) \mid s_h = s, a_h = a\right].$$

The goal of RL is to find a policy $\widehat{\pi}$ which maximizes the value at the initial state. That is, we want to solve the problem

$$\max_{\theta\in\Theta} J(\theta) \coloneqq \mathbb{E}_{s_1\sim\mu} V^{\pi_\theta}(s_1).$$

CartPole. Now we will introduce the CartPole problem, which is a classical control theory benchmark task. There are many ways to solve the CartPole problem. Today we will present one way which goes through this idea of "direct policy optimization". It is not that good. For programming, we will use the OpenAI gym environment.

For the CartPole environment, we will explain how the previous concepts can be instantiated. A good reference is the API https://www.gymlibrary.dev/environments/classic_control/cart_pole/.

- The state space is a vector of dimension 4, it includes the position and velocity of the cart as well as the pole.
- The action space is binary $\{0, 1\}$, which represents moving the cart left or right.
- The goal is to keep the cart upright for as long as possible. This is encoded by the following reward function, which is 1 if we are within the boundaries, and 0 otherwise. Once the pole falls, we get a reward of 0 for the rest of the episode.

We will use a policy class parameterization which is given by a two-layer NN. See the code for details.

How do we optimize? We would like to do gradient ascent (note that we are maximizing so we want to move in the positive direction of the gradient). So we want to do something like $\theta_{t+1} = \theta_t + \eta\nabla J(\theta_t)$. But how do we compute the gradient? We will use this trick.

$$\begin{aligned}
\nabla J(\theta) &= \nabla \int_\tau V(\tau) p_\theta(\tau)\, d\tau \\
&= \int_\tau V(\tau) \nabla p_\theta(\tau)\, d\tau \\
&= \int_\tau V(\tau) \frac{\nabla p_\theta(\tau)}{p_\theta(\tau)} \cdot p_\theta(\tau)\, d\tau \\
&= \int_\tau V(\tau) \nabla \log p_\theta(\tau) \cdot p_\theta(\tau)\, d\tau \\
&= \mathbb{E}_{\tau\sim\theta}[V(\tau)\nabla \log p_\theta(\tau)].
\end{aligned}$$

This gives rise to the REINFORCE algorithm. It is actually more similar to a derivative free method, since we only access the thing we are optimizing $V(\tau)$ through function evaluations.

Further, we can observe that:

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \log \prod_{h=1}^{H} p(x_{h+1}|x_h, a_h) \cdot \pi_\theta(a_h|x_h) = \sum_{h=1}^{H} \nabla_\theta \log \pi_\theta(a_h|x_h).$$

Now we will implement the REINFORCE algorithm for CartPole.

2.3 Bibliographical Note

Some of the material in this section comes from the book [CSV09] as well as the lecture notes https://ee227c.github.io/code/lecture20.html.

References

[CSV09] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. *Introduction to derivative-free optimization*. SIAM, 2009 (cited on page 6).