DS 801: Advanced Optimization for Data Science **KAIST, Fall 2024**
Lecture #22: Black-Box Optimization May 29, 2024
Lecturer: Dabeen Lee

1 Outline

In this lecture, we cover

- introduction to black-box optimization,
- the discretization-based search method,
- simultaneous optimistic optimization.

2 (Non-Convex) Black-Box Optimization

In the last lecture, we learned bandit convex optimization where the decision-maker chooses a sequence of solutions based on bandit feedback. Here, bandit feedback refers to the value of a given loss function at a chosen decision. Bandit feedback is restrictive, as the decision-maker has no access to loss functions, and as a consequence, the decision-maker cannot compute the gradient. Nevertheless, one may deduce an estimator of the gradient, thereby being able to implement an online gradient-based algorithm. We covered two algorithms that guarantee sublinear regret upper bounds for bandit convex optimization.

Many problems in practice, however, involve non-convex loss functions. Loss functions that arise in real-world applications can be as complex as the example in Figure 22.1. In fact, we learned various

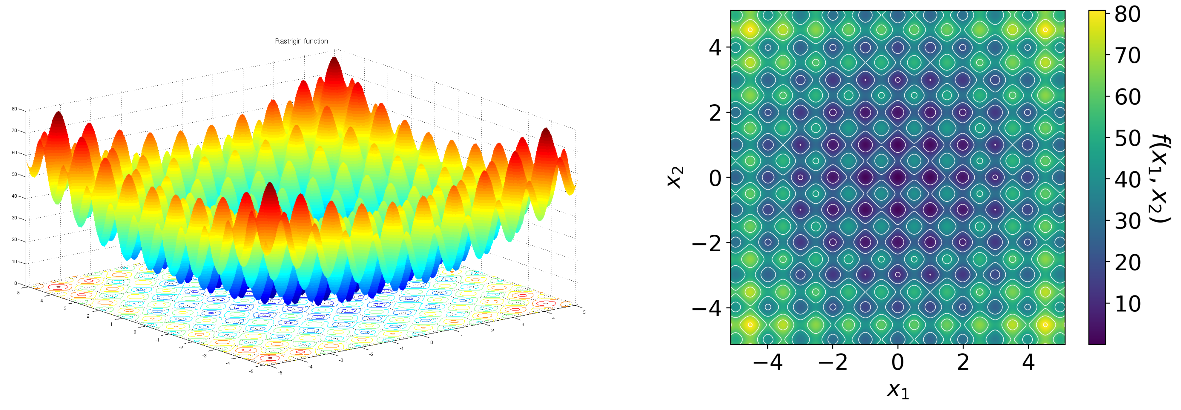

Figure 22.1: Rastrigin Function in 2D

algorithms for non-convex optimization, including gradient descent with Hessian steps, the cubic regularization method, and perturbed gradient descent. Recall that these algorithms are designed to find second-order stationary points or local minima under appropriate assumptions on the loss function.

Although the aforementioned algorithms for non-convex optimization are commonly used in practice, they require knowledge of the loss function's gradient and possibly Hessian. As discussed in the last lecture, there indeed exist many applications where it is difficult to analyze the gradient and Hessian of the underlying loss function. The following provides a list of such applications.

- Engineering Design: Optimizing the design of complex systems and structures (e.g., aerodynamics of aircraft, structural design of bridges) where simulations are used to evaluate performance.
- Machine Learning and Hyperparameter Tuning: Tuning hyperparameters of machine learning models, such as neural networks, support vector machines, and random forests, to achieve better performance on training and validation data.
- Robotics: Optimizing control parameters and policies for robotic systems where the dynamics are complex and non-linear.
- Gaming and AI: Developing and tuning artificial intelligence for games, including the optimization of strategies and behaviors in complex environments.
- Finance and Trading: Developing and optimizing trading algorithms and strategies, as well as portfolio optimization, where the financial models are often noisy and non-differentiable.
- Energy Systems: Optimizing the operation and design of energy systems, such as power grids, renewable energy installations, and energy storage systems, to improve efficiency and stability.
- Material Science: Discovering new materials with desirable properties (e.g., strength, conductivity) by optimizing the composition and processing parameters.
- Healthcare and Medicine: Personalizing treatment plans and drug formulations by optimizing the dosage and combination of therapies for individual patients.
- Chemistry and Biochemistry: Optimizing chemical reactions and biological processes for higher yield, efficiency, and reduced side products in chemical engineering and biotechnology.

In these application settings, the associated loss function is ofen complex, non-differentiable, noisy, or not explicitly known. As a result, we cannot hope for computing the gradient nor the Hessian of the underlying loss function. Thereore, we need to consider **non-convex optimization with bandit feedback**. This problem is often referred to as **black-box optimization**.

3 Discretization-Based Search

Let us consider

$$\min_{x\in C} \quad f(x)$$

where C is the domain and f is the loss function. For black-box optimization, we make minimal assumptions on the loss function f. That said, we consider the general setting where the loss function can be non-convex and non-differentiable. On the other hand, in some applications, the underlying loss function is continuous. The example in Figure 22.1 is indeed continuous, even though its structure is highly complex. Motivated by this, we consider the setting where the loss function is **Lipschitz continuous**. Throughout this section, we assume that f is L-Lipschitz continuous in a norm $\|\cdot\|$, i.e.,

$$|f(x)-f(y)|\leq L\|x-y\|.$$

The goal is to find a near-optimal solution x_ϵ for a given $\epsilon>0$ such that

$$f(x_\epsilon)\leq \min_{x\in C} f(x)+\epsilon.$$

As the loss function f is Lipschitz continuous, our approach is to find a point that is close to an optimal solution. Then, how do we find such a point? The most naïve way is to discretize the solution space and search over the discrete set of points. To be more precise, we consider the following two steps.

1. First, discretize the domain C to obtain a finite subset $C_\epsilon \subseteq C$ containing an ϵ-optimal solution.
2. Next, enumerate all points in C_ϵ.

Hence, as long as the discretization C_ϵ contains an ϵ-optimal solution x_ϵ, the search procedure will find one. The iteration complexity of this algorithm is the number of points in C_ϵ. Therefore, the part of constructing a discretization C_ϵ is crucial.

To simplify our presentation, we assume that

- the domain is given by $C = [0,1]^d$,
- we use the ℓ_∞-norm, i.e., $\|\cdot\| = \|\cdot\|_\infty$, and
- $1/L\epsilon$ is an integer.

Based on these assumptions, we partition the domain $C = [0,1]^d$ into $(1/L\epsilon)^d$ boxes by decomposing each coordinate interval $[0,1]$ into

$$[0,\epsilon/L], \quad [\epsilon/L, 2\epsilon/L], \quad \ldots, \quad [1-\epsilon/L, 1].$$

Then a box has the form

$$\begin{aligned}&\left[\frac{(i_1-1)\epsilon}{L}, \frac{i_1\epsilon}{L}\right] \times \left[\frac{(i_2-1)\epsilon}{L}, \frac{i_2\epsilon}{L}\right] \times \cdots \times \left[\frac{(i_d-1)\epsilon}{L}, \frac{i_d\epsilon}{L}\right]\\ &= \left\{x \in \mathbb{R}^d : \ \frac{(i_j-1)\epsilon}{L} \leq x_j \leq \frac{i_j\epsilon}{L} \quad \forall j \in [d]\right\}.\end{aligned}$$

For a given box, we take the center point given by

$$\left(\frac{\left(i_1-\frac{1}{2}\right)\epsilon}{L}, \quad \frac{\left(i_2-\frac{1}{2}\right)\epsilon}{L}, \quad \ldots, \frac{\left(i_d-\frac{1}{2}\right)\epsilon}{L}\right).$$

Note that there are $(1/L\epsilon)^d$ center points from the $(1/L\epsilon)^d$ boxes. Basically, the set of center points gives rise to a desired discretization C_ϵ. The algorithm is to enumerate all center points and return the one achieving the minimum loss value.

How do we establish the correctness of this approach? Note that any two points x, y in a piece satisfies

$$\|x-y\|_\infty \leq \epsilon/L,$$

which implies that

$$|f(x)-f(y)| \leq L\|x-y\|_\infty \leq \epsilon.$$

Let c^* be the center point of the box containing an optimal solution. Then it follows that

$$f(c^*) \leq \min_{x \in C} f(x) + \epsilon.$$

Let $\bar{c}$ be the center point returned by the algorithm. By the choice of $\bar{c}$, we have that

$$f(\bar{c}) \leq f(c^*) \leq \min_{x \in C} f(x) + \epsilon,$$

as required.

4 Optimistic Optimization

The algorithm from the previous section is based on a fixed discretization. As a result, the algorithm always takes $(1/L\epsilon)^d$ steps to finish search over all points in the discretization. Another issue is that we require knowledge of the Lipschitz constant L. Furthermore, the most critical issue with the method is that we need the assumption that the loss function is Lipschitz continuous over the entire domain.

In this section, we cover a framework of Munos [Mun11], referred to as **simultaneous optimistic optimization (SOO)**. The SOO framework works under the following weaker assumption than the global Lipschitz continuity assumption.

Assumption 22.1. There exists some $L > 0$ such that for any $x \in C$,

$$f(x) - f(x^*) \leq L\|x - x^*\|$$

where x^* is an optimal solution to $\min_{x\in C} f(x)$.

Hence, we assume Lipschitz continuity around an optimal solution, which is essentially a local Lipschitz continuity assumption.

Another favorable aspect of SOO is that it does not need to know the Lipschitz constant L. How is this possible? Recall that the previous approach needs to know L because it prepares a fixed discretization based on the parameter L. In contrast, instead of one fixed discretization, the SOO framework starts with a rough partition of the domain, and it gradually refines it.

To be more specific, SOO works with the idea of **hierarchical partitioning**. First, the domain C is partitioned into K subsets. Here, one may represent the K subsets as K children of paraent C. Then, we may choose one of the K subsets and partition it into K subsets, as in Figure 22.2.

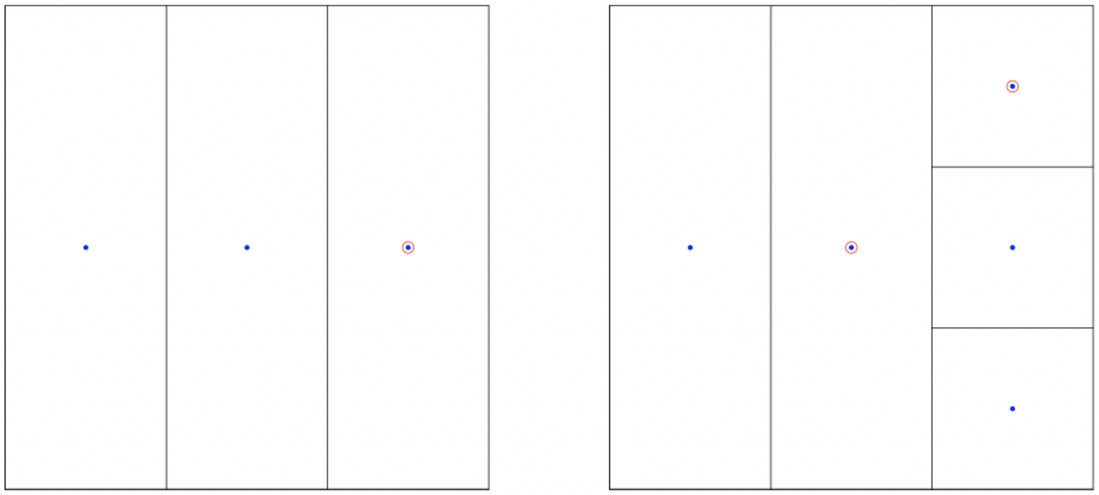

Figure 22.2: Partitioning of the domain

We may continue partitioning pieces. From the second partition of Figure 22.2, we can choose one of the two large subsets or one of the three smaller subsets. Figure 22.3 shows a sequence of more refined partitions of the domain C.

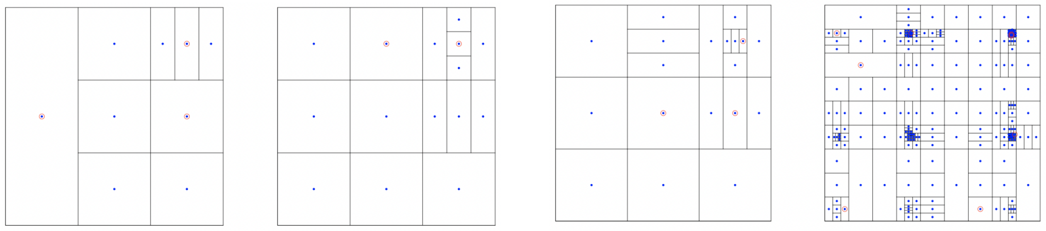
Figure 22.3: Refined partitions

The hierarchical partitioning structure naturally gives rise to a tree representation as in Figure 22.4.

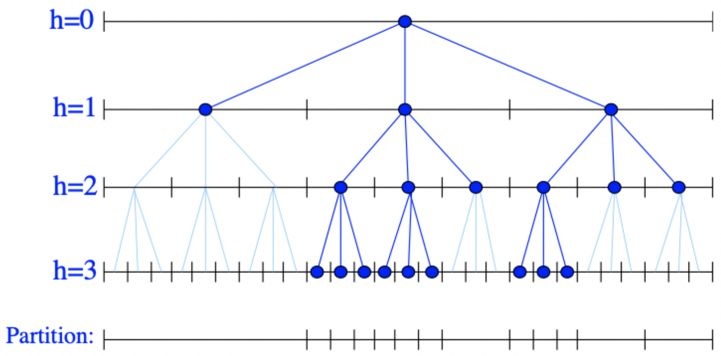

Figure 22.4: Tree representation of a partition

Note that hierarchical partitioning can be done without knowledge of the Lipschitz constant L. The main idea behind the SOO framework is to choose subsets that are expected to contain an optimal solution and refine them gradually. As the algorithm from the previous section, SOO takes a center point of each subset. Then the quality of the subset is measured by the loss value of its center point.

Another important component of SOO is the idea of **optimistic search**. At each iteration, we need to choose which subset to be partitioned. The choice is made based on two criteria. On one hand, it makes sense to focus on subsets whose center points have low loss values. On the other hand, a large subset is not explored enough yet, so its unexplored region may contain a good solution. This is similar in spirit to **the exploration-exploitation tradeoff**.

To be more specific, we use notation (h, j) to denote the jth subset at depth h. Here, $(0, 0)$ refers to the original domain C. Then we denote by $x_{h,j}$ the center point of (h, j). Then the quality of subset (h, j) is measured by $f(x_{h,j})$. Then, the next question is about how to choose a subset that is unexplored? We may select a subset at a high hierarchy in the tree representation. The SOO algorithm is given as follows.

References

[Mun11] Rémi Munos. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Wein-

Algorithm 1 Simultaneous Optimistic Optimization

```
Input: the maximum depth function $h_{\max} : \mathbb{Z} \to \mathbb{Z}$.
Initialize $\mathcal{T}_1 = \{(0,0)\}$ and $t = 1$.
while True do
    Set $v_{\min} = \infty$.
    for $h = 0$ to $\min\{\text{depth}(\mathcal{T}_t), h_{\max}(t)\}$ do
        Among all leaves $(h,j) \in \mathcal{L}_t$ of depth $h$, select
            $(h,i) \in \text{argmin}_{(h,j)\in\mathcal{L}_t} f(x_{h,j})$
        if $f(x_{h,i}) \le v_{\min}$ then
            Partition the subset $(h,i)$ into $K$ subsets $(h+1,i_1),\ldots,(h+1,i_K)$.
            Add them to $\mathcal{T}_t$.
            Evaluate $f(x_{h+1,i_1}),\ldots,f(x_{h+1,i_K})$.
            Set $v_{\min} = f(x_{h,i})$.
            if $t = T$ then
                Return
                    $\text{argmax}_{(h,i)\in\mathcal{T}_T} f(x_{h,i})$
            end if
        end if
    end for
end while
```

berger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. 4