Lecture 11: Black-Box Optimization

Dabeen Lee

Industrial and Systems Engineering, KAIST

2025 Winter Lecture Series on Combinatorial Optimization

January 17, 2025

Outline

- Introduction to black-box optimization
- Discretization-based search
- Optimistic optimization
- Optimizing over a trained neural network

Non-Convex Optimization

- Many problems in practice involve non-convex loss functions.
- Loss functions that arise in real-world applications can be as complex as the example in Figure 1.

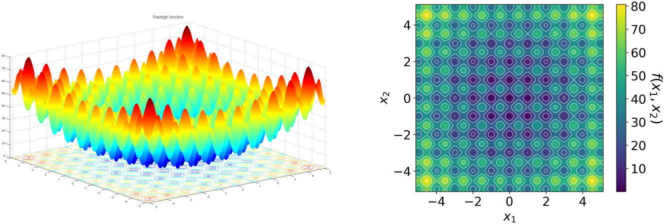

Figure: Rastrigin Function in 2D

Non-Convex Optimization

- Various algorithms exist for non-convex optimization.
- For example, gradient descent with Hessian steps, the cubic regularization method, and perturbed gradient descent.
- These algorithms are designed to find second-order stationary points or local minima under appropriate assumptions on the loss function.
- There indeed exist many applications where it is difficult to analyze the gradient and Hessian of the underlying loss function.

Black-Box Optimization

Application Scenarios

- **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.

Black-Box Optimization

Application Scenarios

- **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.

Black-Box Optimization

- 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**.

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 1 is indeed continuous, even though its structure is highly complex.

Discretization-Based Search

- 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.

Discretization-Based Search

- 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.

Discretization-Based Search

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

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

Discretization-Based Search

- Then a box has the form

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

Discretization-Based Search

- 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.

Discretization-Based Search

- 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.

Issues with the Discretization-Based Search

- The algorithm 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.

Optimistic Optimization

- 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

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.

Optimistic Optimization

- 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.

Optimistic Optimization

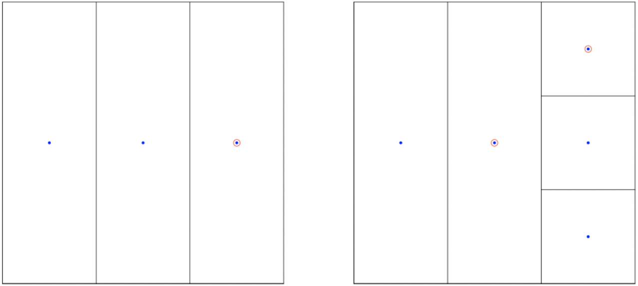

Figure: Partitioning of the domain

Optimistic Optimization

- We may continue partitioning pieces.
- From the second partition of Figure 2, we can choose one of the two large subsets or one of the three smaller subsets.
- Figure 3 shows a sequence of more refined partitions of the domain C.

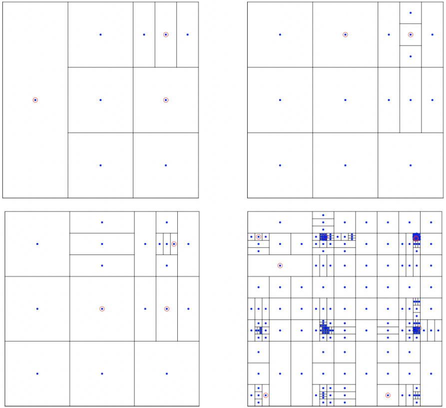

Figure: Refined partitions

Optimistic Optimization

- The hierarchical partitioning structure naturally gives rise to a tree representation as in Figure 4.

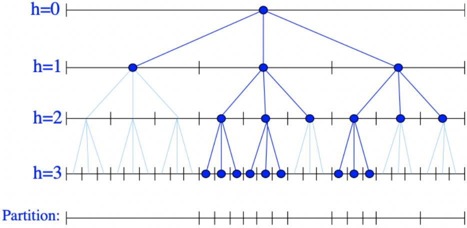

Figure: Tree representation of a partition

Optimistic Optimization

- 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.

Optimistic Optimization

- 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**.

Optimistic Optimization

- 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 level in the tree representation.

Optimistic Optimization

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 $\min\{\mathrm{depth}(\mathcal{T}_t), h_{\max}(t)\}$ **do**
    Among all leaves $(h, j) \in \mathcal{L}_t$ of depth h, select

$$(h, i) \in \mathrm{argmin}_{(h,j)\in\mathcal{L}_t} f(x_{h,j})$$

    if $f(x_{h,i}) \leq v_{\min}$ **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 $\mathrm{argmax}_{(h,i)\in\mathcal{T}_T} f(x_{h,i})$
      end if
    end if
  end for
end while

Black-Box Optimization via Supervised Learning

- Black-box optimization framework applies to settings where the objective function is not known to the decision-maker.
- Basically, we consider

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

  where the decision-maker has access to none of the gradient $\nabla f(x)$ and the Hessian $\nabla^2 f(x)$.
- We find a solution based on bandit feedback which exhibits the value $f(x)$ of a chosen solution x.
- We learned optimistic optimization methods, which explore the solution space based on continuity of the objective function.
- The main idea behind the optimistic optimization methods is that we provide a hierarchical partitioning of the search space.
- Based on the partitioning, we optimistically explore subsets of the search space.

Black-Box Optimization via Supervised Learning

- The optimistic optimization algorithms are widely used in practice because they rely on minimal structural assumptions on the objective function.
- On the other hand, they often fall into inferior performance than instance-specific methods that are implemented with some knowledge of the problem environment.
- That is because they do not exploit any underlying structures of the objective function.
- This motivates the question of **how to explore and exploit the underlying structure of the function**.

Black-Box Optimization via Supervised Learning

- In this lecture, we discuss some supervised learning methods to learn and approximate the unknown objective function.
- More importantly, based on the learned model and function, we are interested in finding a good solution that guarantees a small loss value.
- Basically, we are given n data points

$$(x_1, f(x_1)), \ldots, (x_n, f(x_n)),$$

from which we infer the underlying function f.

Optimizing over a Trained Neural Network

- One of the most practical supervised learning is to use a neural network to learn the underlying model.
- Based on a data set of n points $(x_1, y_1), \ldots, (x_n, y_n)$ with $y_i = f(x_i)$ for $i \in [n]$, one may train a neural network by considering

$$\min_{\theta} \quad \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i).$$

- Here, the trained neural network f_θ provides an approximation of the objective function f.
- Then, we may find a solution that achieves a small f value by considering

$$\min_{x \in C} \quad f_\theta(x).$$

Optimizing over a Trained Neural Network

- Feed-forward neural networks with ReLU activations functions are commonly used for approximating the unknown objective function in practice [PTA+22].
- We discuss how to find an input solution that optimizes the output value of a trained feed-forward neural network with ReLU activation.
- In particular, we explain the basic formulation due to Fischetti and Jo [FJ18] and Serra et al. [STR18].

Optimizing over a Trained Neural Network

- Let us discuss the case of a neural network with a single hidden layer.
- Let $x \in \mathbb{R}^d$ be the input, prepared by d input neurons. There are m neurons in the single hidden layer.
- Let the input of the ith neuron in the hidden layer be given by ${w_i}^\top x + b_i$. Then the output of the neuron is

$$\mathrm{ReLU}({w_i}^\top x + b_i).$$

- Let a_i denote the weight between the ith neuron in the hidden layer and the output node.
- Then the output of the neural network is given by

$$f_\theta(x) = \sum_{i=1}^{n} a_i \cdot \mathrm{ReLU}({w_i}^\top x + b_i).$$

Optimizing over a Trained Neural Network

- Then the problem boils down to solving

$$\begin{aligned} \min_{x \in C} \quad & \sum_{i=1}^{n} a_i t_i \\ \text{s.t.} \quad & t_i = \mathrm{ReLU}(w_i^\top x + b_i), \quad i \in [n]. \end{aligned} \tag{1}$$

Optimizing over a Trained Neural Network

- Recall that
$$\mathrm{ReLU}(x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases}$$
- Let ℓ_i and u_i denote the lower and upper bounds of ${w_i}^\top x + b_i$ over C given by
$$\ell_i = \inf_{x \in C} \left\{ {w_i}^\top x + b_i \right\}, \quad u_i = \sup_{x \in C} \left\{ {w_i}^\top x + b_i \right\}.$$

- Then, we can argue that $t_i = \mathrm{ReLU}(w_i^\top x + b_i)$ holds if and only if t_i satisfies

$$\begin{aligned}
t_i &\geq 0, \\
t_i &\geq w_i^\top x + b_i, \\
t_i &\leq u_i^\top z_i, \\
t_i &\leq w_i^\top x + b_i - \ell_i(1 - z_i),
\end{aligned}$$

for some $z_i \in \{0, 1\}$.

Optimizing over a Trained Neural Network

- Therefore, (1) can be formulated as

$$
\begin{aligned}
\min_{x \in C} \quad & \sum_{i=1}^{n} a_i t_i \\
\text{s.t.} \quad & t_i \geq 0, \quad i \in [n] \\
& t_i \geq w_i^\top x + b_i, \quad i \in [n] \\
& t_i \leq u_i^\top z_i, \quad i \in [n] \\
& t_i \leq w_i^\top x + b_i - \ell_i (1 - z_i), \quad i \in [n] \\
& z_i \in \{0, 1\}, \quad i \in [n].
\end{aligned} \tag{2}
$$

- The formulation simply extends to the case of multiple hidden layers.

- More recently, Anderson et al. [AHM+20] and Tsay et al. [TKTM21] developed computationally improved formulations for optimizing a trained feed-forward neural network with ReLU activation.

[AHM+20] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming, 183:3–39, 2020.

[FJ18] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. Constraints, 23:296–309, 2018.

[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. Weinberger, editors, Advances in Neural Information Processing Systems, volume 24. Curran Associates, Inc., 2011.

[PTA+22] Theodore P Papalexopoulos, Christian Tjandraatmadja, Ross Anderson, Juan Pablo Vielma, and David Belanger. Constrained discrete black-box optimization using mixed-integer programming. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, Proceedings of the 39th International Conference on Machine Learning, volume 162 of Proceedings of Machine Learning Research, pages 17295–17322. PMLR, 17–23 Jul 2022.

[STR18] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In Jennifer Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 4558–4566. PMLR, 10–15 Jul 2018.

[TKTM21] Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained relu neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, volume 34, pages 3068–3080. Curran Associates, Inc., 2021.