1 Outline

In this lecture, we cover

- black-box optimization by supervised learning,
- kernel ridge regression,
- optimizing over a trained neural network.

2 Black-Box Optimization by Supervised Learning

In the last lecture, we introduced the black-box optimization framework which applies to settings where the objective function is not known to the decision-maker. 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 which we can optimistically explore subsets of the search space.

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

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.

2.1 Kernel Ridge Regression

Although we consider general non-convex functions for black-box optimization, let us start with the simple linear case to explain the basic idea.

Recall that linear regression seeks to find a linear model between the vector $x \in \mathbb{R}^d$ of features and the response variable $y \in \mathbb{R}$ given by

$$y = \theta_{\text{true}}^\top x$$

where $\theta_{\text{true}} \in \mathbb{R}^d$ is the coefficient vector. Then we want to infer the true coefficient vector θ_{true}, based on a given set of n data points $(x_1, y_1), \ldots, (x_n, y_n)$. For our setting, we have $y_i = f(x_i)$

for $i \in [n]$. For the linear case, learning the coefficient vector θ_{true} is basically learning function $f(x) = \theta_{\text{true}}^\top x$.

We learned the **ridge regression** framework that takes ℓ_2 regularization. The formulation is given by

$$\min_{\theta} \quad \sum_{i=1}^{n} \left(y_i - \theta^\top x_i\right)^2 + \lambda \|\theta\|_2^2. \tag{23.1}$$

Note that

$$\sum_{i=1}^{n} \left(y_i - \theta^\top x_i\right)^2 + \lambda \|\theta\|_2^2 = \theta^\top \left(\sum_{i=1}^{n} x_i x_i^\top + \lambda I\right) \theta - 2 \left(\sum_{i=1}^{n} y_i x_i\right)^\top \theta + \sum_{i=1}^{n} y_i^2.$$

Then the optimal solution to (23.1) is given by

$$\widehat{\theta} = \left(\sum_{i=1}^{n} x_i x_i^\top + \lambda I_d\right)^{-1} \sum_{i=1}^{n} y_i x_i$$

where I_d is the $d \times d$ identity matrix. As a result, we may find a solution by solving the following approximate problem:

$$\min_{x \in C} \quad \widehat{f}(x) \quad = \quad \min_{x \in C} \quad \widehat{\theta}^\top x.$$

To summarize, we went through the following steps.

1. Deduce a function $\widehat{f}(x)$ by ridge regression based on data $(x_1, y_1), \ldots, (x_n, y_n)$.
2. Compute a solution that optimizes $\widehat{f}(x)$.

Of course, this framework based on linear regression is limited as the objective function can be non-convex in general. Then our next question is about how to extend the procedure to the case of non-convex objective functions. To answer this, we explain **kernel ridge regression**.

To elaborate, we define Φ and y as

$$\Phi = (x_1, \ldots, x_n) \in \mathbb{R}^{d \times n}, \quad y = (y_1, \ldots, y_n)^\top \in \mathbb{R}^n.$$

Then we have

$$\sum_{i=1}^{n} x_i x_i^\top = \Phi \Phi^\top, \quad \sum_{i=1}^{n} y_i x_i = \Phi y.$$

As a result,

$$\widehat{\theta} = \left(\Phi \Phi^\top + \lambda I_d\right)^{-1} \Phi y.$$

Note that both $\Phi \Phi^\top + \lambda I_d$ and $\Phi^\top \Phi + \lambda I_n$ are invertible. Then

$$\left(\Phi \Phi^\top + \lambda I_d\right)^{-1} \Phi = \Phi \left(\Phi^\top \Phi + \lambda I_n\right)^{-1}$$

holds if and only if

$$\left(\Phi \Phi^\top + \lambda I_d\right) \left(\Phi \Phi^\top + \lambda I_d\right)^{-1} \Phi \left(\Phi^\top \Phi + \lambda I_n\right) = \left(\Phi \Phi^\top + \lambda I_d\right) \Phi \left(\Phi^\top \Phi + \lambda I_n\right)^{-1} \left(\Phi^\top \Phi + \lambda I_n\right)$$

which is equivalent to $\Phi\Phi^\top\Phi + \lambda\Phi = \Phi\Phi^\top\Phi + \lambda\Phi$. Therefore, we have $(\Phi\Phi^\top + \lambda I_d)^{-1}\Phi = \Phi(\Phi^\top\Phi + \lambda I_n)^{-1}$, and thus,

$$\widehat{\theta} = \Phi\left(\Phi^\top\Phi + \lambda I_n\right)^{-1} y, \quad \widehat{\theta}^\top x = x^\top\Phi\left(\Phi^\top\Phi + \lambda I_n\right)^{-1} y.$$

Here, let us expand the terms $x^\top\Phi$ and $\Phi^\top\Phi$ to express $\widehat{\theta}^\top x$. Note that

$$x^\top\Phi = \left(x^\top x_1, \ldots, x^\top x_n\right) \in \mathbb{R}^{1\times n}, \quad \Phi^\top\Phi = \begin{pmatrix} x_1^\top x_1 & \cdots & x_1^\top x_n \\ \vdots & \ddots & \vdots \\ x_n^\top x_1 & \cdots & x_n^\top x_n \end{pmatrix} \in \mathbb{R}^{n\times n}.$$

Note that the components of $x^\top\Phi$ are of the form $x^\top x_i$ while the entries of $\Phi^\top\Phi$ are given by $x_i^\top x_j$. Here, given two vectors x and x', one may introduce and consider the following **kernel function**:

$$k(x, x') = x^\top x'.$$

This is often called the linear kernel. By considering other types of kernel functions, one may extend linear models to non-linear models, and this is the basic idea of kernel ridge regression. For example,

- the polynomial kernel:
$$k(x, x') = (x^\top x')^\ell,$$
- the squared exponential kernel:
$$k(x, x') = \exp\left(-\frac{\|x - x'\|_2^2}{2\sigma^2}\right),$$
- the Matérn kernel:
$$k(x, x') = \frac{\sigma^2}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\|x - x'\|_2}{\lambda}\right)^\nu \mathcal{B}_\nu\left(\frac{\|x - x'\|_2}{\lambda}\right)$$
where γ denotes the Gamma function, $\mathcal{B}_\nu$ denotes the modified Bessel function of the second kind, ν is a parameter controlling the smoothness of the function.

Given a function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, let us define $k(x)$ and K as follows.

$$k(x) = \begin{pmatrix} k(x_1, x) \\ \vdots \\ k(x_n, x) \end{pmatrix} \in \mathbb{R}^n, \quad K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_n, x_1) \\ \vdots & \ddots & \vdots \\ k(x_1, x_n) & \cdots & k(x_n, x_n) \end{pmatrix} \in \mathbb{R}^{n\times n}.$$

For the linear kernel, we have that $k(x) = \Phi^\top x$ and $K = \Phi^\top\Phi$. For a function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ to be kernel function, the matrix K needs to be symmetric and positive semidefinite. Then, with a kernel function k equipped with function $k(x)$ and matrix K, we take

$$\widehat{f}(x) = k(x)^\top(K + \lambda I_n)^{-1} y.$$

Here, this function $\widehat{f}(x)$ indeed depends on x. This means that by considering

$$\min_{x\in C} \quad k(x)^\top(K + \lambda I_n)^{-1} y,$$

we can find a solution x that achieves a loss value of $f(x)$.

Let us discuss some computational aspects of kernel ridge regression. First of all, for a given set of n data points, the matrix $K + \lambda I_n$ is an $n \times n$ matrix. Using a standard matrix inversion algorithm, it often takes $O(n^3)$ time to compute $(K+\lambda I_n)^{-1}$. Moreover, the approximate objective $k(x)^\top (K+\lambda I_n)^{-1} y$ is non-linear and can be non-convex depending our choice of the kernel function k. As a consequence, finding a solution that minimizes the approximate objective can be difficult. In practice, we often discretize the solution space C and choose the best solution among the points in C.

2.2 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

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

Feed-forward neural networks with ReLU activations functions are commonly used for approximating the unknown objective function in practice [PTA+22]. Throughout this section, 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].

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

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

Then the problem boils down to solving

$$\begin{aligned} \min_{x \in C} \quad & \sum_{i=1}^{n} a_i t_i \\ \text{s.t.} \quad & t_i = \mathrm{ReLU}(w_i^\top x + b_i), \quad i \in [n]. \end{aligned} \tag{23.2}$$

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 $z_i \in \{0, 1\}$. Therefore, (23.2) can be formulated as

$$\begin{aligned} \min_{x \in C} \quad & \sum_{i=1}^{n} a_i t_i \\ \text{s.t.} \quad & t_i \geq 0, \quad i \in [n] \\ & t_i \geq w_i^\top x + b_i, \quad i \in [n] \\ & t_i \leq u_i^\top z_i, \quad i \in [n] \\ & t_i \leq w_i^\top x + b_i - \ell_i(1 - z_i), \quad i \in [n] \\ & z_i \in \{0, 1\}, \quad i \in [n]. \end{aligned} \tag{23.3}$$

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.

References

[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. 2.2

[FJ18] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23:296–309, 2018. 2.2

[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. 2.2

[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. 2.2

[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. 2.2