Lecture 18: Bayesian Optimization II

CS4787 — Principles of Large-Scale Machine Learning Systems

Questions from last lecture: what are the main computational costs of hyperparameter optimization? Which aspects of the system still need to be determined before we can actually run this algorithm in practice?

Last lecture, we saw how Bayesian optimization gives us a principled way of optimizing hyperparameters using a gaussian process, a kernel, and an acquisition function. Recall:

- $x \in \mathcal{X}$ was an assignment of the hyperparameters
- $f : \mathcal{X} \to \mathbb{R}$ is the objective function. Typically $f(x)$ represents the expected empirical risk on the validation set that would result from running the entire learning pipeline with hyperparameters x, but this could be whatever you want to optimize.
- a was the acquisition function and determines what we search. Typically, a is a function of the mean, variance, and the best observed value, i.e. $a(\mu, \sigma, y_{\text{best}})$.

But despite its nice properties, we don't always use Bayesian optimization all the time. **Why?**

- Sensitivity to choice of kernel—need to choose good hyperparameters for kernel
- Need to solve the inner optimization problem efficiently somehow—not clear how to do this
- Difficulty with scaling—scary cubic terms in the computational complexity
- Default setup doesn't handle varying cost to compute objective—but we can see substantial variation here in practice, especially when we are exploring systems parameters

Kernel selection. Choosing a good prior is very important for the performance of Bayesian optimization. We want to choose a kernel that is effective for machine learning. We could just use the RBF kernel,

$$K_{\text{RBF}}(x, y) = \exp\left(\gamma \cdot \|x - y\|^2\right).$$

But imagine we were trying to optimize, say, the momentum and the number of hidden units in a layer. **What problem might we have?**

One way to address hyperparameter scaling is to use the *automatic relevance determination (ARD) squared-exponential* kernel. This is defined, for $x, y \in \mathbb{R}^d$, as

$$K_{\text{ARD}}(x, y) = \theta_0 \cdot \exp\left(-\frac{1}{2} r^2(x - y)\right) \qquad \text{where} \qquad r^2(x - y) = \sum_{i=1}^{d} \frac{(x_i - y_i)^2}{\theta_i^2}.$$

Here, $\theta_0, \theta_1, \ldots, \theta_d$ are hyperparameters of the kernel (we can think of these as hyper-hyperparameters). **How does this help with hyperparameter scaling?**

For many tasks, the ARD kernel makes predictions that are too smooth to accurately match the true loss function f. To address this, it's common to use the **Matèrn kernel**, which allows for less smooth predictions.

It's usually written as

$$K_{M(\nu)}(x,y) = \sigma^2 \cdot \frac{2^{1-\nu}}{\Gamma(\nu)} \cdot \left(\frac{2\nu r^2(x-y)}{\rho^2}\right)^{\frac{\nu}{2}} K_\nu \left(\sqrt{\frac{2\nu r^2(x-y)}{\rho^2}}\right),$$

where K_ν denotes the modified Bessel function of the second kind, and the parameters for this kernel are ν, σ^2, ρ, and the θ_i from the definition of r above. But this is super messy, so we normally fix a value of ν (this determines how rough our predictions will be) which allows us to simplify the definition substantially. It's common to choose $\nu = 5/2$, in which case the **ARD Matèrn 5/2 kernel** is given by

$$K_{M(5/2)}(x,y) = \theta_0 \cdot \left(1 + \sqrt{5r^2(x-y)} + \frac{5}{3}r^2(x-y)\right) \cdot \exp\left(-\sqrt{5r^2(x-y)}\right).$$

Here, the hyperparameters are $\theta_0, \theta_1, \ldots, \theta_d$ as before for the ARD kernel. Using this lets us both solve the hyperparameter scaling problem and avoid making predictions that are too smooth.

Choosing the hyper-hyperparameters. One problem with what we just did: now we have a good kernel that we can use for general problems, but we just added in some hyper-hyperparameters that we need to set (the θ_i). Worse, there's *more* hyper-hyperparameters than we had hyperparameters, so the problem seems to have gotten harder! There are a couple of standard ways to try to automatically set these hyper-hyperparameters.

One way to do it is to use **maximum likelihood estimation**. We collect a sample of points from f (usually this is our random sampling we used to warm up the Gaussian process) and find the hyper-hyperparameters θ that maximize the probability of observing those points in the Gaussian process model. That is, if we observe f at D different points $x_1, \ldots, x_D$ and observe it having values $y_1, \ldots, y_D$, then we select the parameters as

$$\theta = \arg\max_\theta \mathbf{P}\left(f(x_1) = y_1, f(x_2) = y_2, \ldots, f(x_D) = y_D\right)$$

where this probability is taken within the Gaussian process model with parameters θ.

Another way to do it (that is more expensive, but can sometimes produce better results) is the fully Bayesian approach. Here, we marginalize over the hyper-hyperparameters to compute the **integrated acquisition function**

$$\begin{aligned}
\hat{a}(x_*) &= \int a(x_*;\theta) \cdot \mathbf{P}\left(\theta | f(x_1) = y_1, f(x_2) = y_2, \ldots, f(x_D) = y_D\right)\ d\theta \\
&= \int a(x_*;\theta) \cdot \frac{\mathbf{P}\left(\theta\right) \cdot \mathbf{P}\left(f(x_1) = y_1, f(x_2) = y_2, \ldots, f(x_D) = y_D | \theta\right)}{\mathbf{P}\left(f(x_1) = y_1, f(x_2) = y_2, \ldots, f(x_D) = y_D\right)}\ d\theta \\
&\propto \int a(x_*;\theta) \cdot \mathbf{P}\left(\theta\right) \cdot \mathbf{P}\left(f(x_1) = y_1, f(x_2) = y_2, \ldots, f(x_D) = y_D | \theta\right)\ d\theta
\end{aligned}$$

where $a(x_*;\theta)$ denotes the value the acquisition function would have if we were using hyper-hyperparameters θ, and this probability denotes the probability that the hyper-hyperparameters are θ given all the observations we've made of f. We then use this integrated acquisition function instead of the Of course, we need to define some prior over θ to be able to do this...which is effectively a hyper-hyper-hyperparameter we need to set. But this method often works quite well.

This is only a bit of what we can say about choosing kernels for Bayesian optimization. It's a good place to get started. But sometimes you'll want to do something more sophisticated, and the nice thing about Bayesian optimization is that it gives you an easy way to do that.

Solving the inner optimization problem. To run Bayesian optimization, recall that we needed to solve the optimization problem

$$\begin{aligned} \text{minimize: } & a(\mu(x_*), \sigma(x_*), y_{\text{best}}) \\ \text{over: } & x_* \in \mathcal{X}. \end{aligned}$$

How do we solve this? Some ways to do it:

- We usually can differentiate a, so gradient descent is an option. But a is usually non-convex so this can be tricky.
- Common approach: choose a random starting point and run gradient descent until it converges. Then choose another random starting point and repeat many times.

Modeling costs. Hypothetical scenario: suppose that the amount of time it takes to compute f varies by an order of magnitude for different hyperparameters x. We want to minimize $f(x)$ while not spending too much time on computing f.

A new acquisition function for this setting: **expected improvement per-second**. Idea: model not only the value of f but also the time it will take to compute f as a Gaussian process (i.e. we have *two* Gaussian processes here). If $c(x_*)$ denotes the predicted compute cost for computing $f(x_*)$ from our Gaussian process, then the expected improvement per second is

$$\mathbf{E}\left[\frac{\min(f(x_*) - y_{\text{best}}, 0)}{c(x_*)}\right]$$

which leads naturally to an acquisition function like

$$a_{\text{EI/s}}(y_{\text{best}}, \mu, \sigma, x_*) = -\left(\phi\left(\frac{y_{\text{best}} - \mu}{\sigma}\right) + \frac{y_{\text{best}} - \mu}{\sigma} \cdot \Phi\left(\frac{y_{\text{best}} - \mu}{\sigma}\right)\right) \cdot \frac{\sigma}{\mathbf{E}\left[c(x_*)\right]}$$

What else can we do to incorporate cost into our hyperparameter optimization?