Kernel Sums Optimization: Project for 91762 - Combinatorial Decision Making and Optimization

Valerio Tonelli
valerio.tonelli2@studio.unibo.it

Academic Year 2020-2021

1 Introduction

I would like to study the following problem, of which, to the best of my knowledge, there is not much discussion for in the literature:

Given a set of n kernel functions $f_i : R \to [0, 1]$ and a target function $\hat{f}$ defined in the same interval, we wish to figure out how to sum up the kernels, each of which may be hidden and each of which may be translated by any real number in $[0, 1]$, in such a way as to get as close to the target function as possible.

More formally, we wish to solve the following constrained optimization problem:

$$argmin_{(c_i,t_i)} \frac{1}{n} \cdot \int_0^1 \left(\sum_{i=1}^{n} c_i \cdot f_i(x + t_i) - \hat{f}(x) \right)^2 dx$$

$$\text{s. t.}$$

$$\forall i, c_i \in \{0, 1\}$$

$$\forall i, t_i \in [-1, 1]$$

where c_i is a boolean determining whether the i^{th} kernel function contributes to the sum and t_i is the x-axis offset for the i^{th} function.

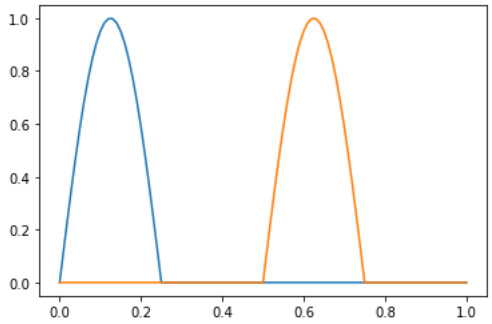
2 Problem Analysis and Design

This Section discusses the main issues found during analysis and the initial development of a Python application solving the optimization problem.

2.1 Initial Problem Design

Firstly, we may notice that we can reduce the optimization problem to a discrete sum over s sampled terms. This being said, we must also notice that Gradient

Descent (GD) is a poor optimization strategy for this problem, due to the presence of flat regions and local minima for (at least!) every flat region or local minima of each kernel as well as the target. To visualize this, it is sufficient to consider a very simple example:



If we attempt to nudge the kernel (the blue hill) to the right (towards the orange hill which is the target), the loss function would not change. Only when the two hills are partially overlapping can we expect an hill climbing algorithm to perform correctly. Thus, a better strategy could be a Genetic Algorithm (GA) or an Evolutionary algorithm such as PSO. Both have been attempted, but the latter has proven to produce better results, also due to an issue which is discussed in Subsection 2.2.

2.2 Hybrid Optimization to Real-only Optimization

The problem that we have shown in Section 1 is hybrid, in the sense that it involves both integer (the c_i) and real quantities (the t_i). The former is particularly problematic, since integer optimization is known to be a hard problem which we would like to avoid. Thankfully, by extending the range of the t_i to $[-1-\epsilon, 1+\epsilon]$ for some small $\epsilon \in \mathrm{R}$, we can shift a function either to the left or to the right in such a way that its contribution to the optimization is null, thus simulating the c_i values within the t_i rather than trying to solve multiple optimization problems on the t_i for the possible combinations of c_i.

This does, however, introduce a bias: cutting functions is a local minima for the optimization problem since the error term (ignoring the square operation) is proportional to the sums of the kernels. This means that, in practice, GA is practically unusable, because among the possible t_i values, $-1-\epsilon$ and $1+\epsilon$ can become very common due to the initial advantage they provide w.r.t. the rest of the population, and as such are 'traits' which easily spread to the rest of the population and to the other t_i values. Even PSO suffers from this issue, although much less severely.

To correct this behaviour, we introduce a Cut Penalty term as a regulariza-

tion to the loss function:

$$CT = \sum_{i=1}^{n} \left(\int_0^1 |f_i(x + t_i)|dx - \int_0^1 |f_i(x)|dx \right)$$

which is larger the more of the kernels we cut (even partially). The overall *training loss* is then a weighted combination of the original loss and of the regularization term, while the *test loss* should stay the same since the final loss should be solely based on the produced results, not on how much we cut the kernels.

2.3 Pseudocorrelation

In Subsection 2.1 we have tackled the problem of local optimization via GD (or, equivalently, hill climbing), arguing that the method is not sufficient for usage in the general case but that there could be situations in which it would be beneficial. We could thus perform some local optimization at the end of each (or some) PSO iterations.

Unfortunately, attempting to perform GD at the end of a PSO iteration would considerably slow down the learning process, as it would need to be performed for multiple particles within a loop which already executes for multiple iterations. A faster, although more approximate solution, is to compute the discrete *correlation* between the sum of the kernels and the signal, in order to figure out a *global* shift on the x-axis (that is, a scalar increase δ to be added to each t_i).

We define our discrete correlation as follows:

$$pseudocorr\,(f(x), g(x)) = \sum_{i=-s}^{s} f'(x+i) \cdot g(x)$$

$$\text{where } f'(x+i) = \begin{cases} f(x+i) & \text{if } f(x+i) \leq g(x) \\ 2 \cdot g(x) - f(x+i) & \text{if } f(x+i) > g(x) \end{cases}$$

and s are the discrete samples.

This operator has a similar definition to the classic correlation operator. In fact, it is exactly the same whenever $f(x+i) \leq g(x)$. Whenever the alternative case holds true, instead, we replace $f(x+i)$ with a term such that, if f is greater than g by an amount ψ, the result is the same as if we were considering the case in which g was exactly the same and greater than f by an amount ψ. This is done so that this new correlation operator has its maximum whenever f and g take the same value, rather than being simply proportional to each of them.

It is important to note that, although this operator should theoretically improve performances and results, for simplicity the actual implementation uses a maximum number of iterations as its threshold for termination, rather than a measure of the loss, thus the pseudo-correlation in practice slows down the computation, although it should still improve accuracy.

Furthermore, it should be noted that this local optimization reintroduces the bias discussed in Subsection 2.2, since its results are not regularized by a penalty term; as such, in the implementation it is deployed only after a fixed amount of iterations, when we suppose that the learning process has progressed enough that the current minima is below the local minima of this bias.

3 Training

Training on the hyper-parameters of PSO as well as the multiplicative factor of the Cut Penalty has been done using a simple Grid Search, considering a set target function and set kernels. For each combination of parameters, due to the random nature of the algorithm, the optimizer has been run 3 times, averaging the final losses.

The best combination of parameters was found to be the following:

$$\lambda_{CT} = 1,\ w = 0.8,\ \phi_p = 0.8,\ \phi_g = 0.8$$

However, further testing with different targets and kernels, as well as fine-tuning of these values is still necessary. Intuitively, these values are only a 'generally decent' option, as different scenarios may require very different values for these hyper-parameters. Furthermore, some hyper-parameters have not been tested due to time constraints.