**CausalBO: A Python Package for Adding Causality to Bayesian Optimization**

**Introduction**

Bayesian optimization (BO) is a powerful technique in the field of optimization which seeks to find the optimal configuration of parameters for a given objective function while minimizing the number of costly evaluations. This task is achieved through the use of a probabilistic model to balance exploration and exploitation, making it particularly suited for optimizing expensive or time-consuming black-box functions.

Causal Bayesian Optimization (CBO) is an extension of Bayesian optimization, incorporating causal reasoning into the evaluation process. CBO introduces the concept of separate observational and interventional datasets. Unlike classical optimization, where observations are treated as passive measurements, Causal BO enables active interventions in the form of do-calculus that can uncover causal relationships, inform decision-making, and suggest optimal actions even in the presence of confounding variables.

**Comparison to Original Implementation**

Initially, the CBO algorithm was developed by Virginia Aglietti et. al., written in Python using GPy and emukit wrappers. It featured hard-coded do-calculus expressions bound to specific datasets and primarily served as a proof-of-concept piece of software to demonstrate the viability of incorporating causality into Bayesian optimization. The code was non-modular, and could not be incorporated into existing BO solutions. In addition, several libraries used by the original implementation required significant manual setup including the installation of VS Build Tools, and compilation from source.

The CausalBO approach aims to increase code readability, usability for those with little to no knowledge of causality and do-calculus, and modularity. CausalBO is built on top of BoTorch, a popular Bayesian optimization library that will be receiving updates and bugfixes for the foreseeable future. CausalBO also uses DoWhy, a library created to perform causal inference based on known causal links and observational data. The advantage of this is that the end-user needs no knowledge of causality or do-calculus to take advantage of their benefits.

In addition, CausalBO also exposes an easy-to-use premade CBO loop, specifically the algorithm described in V. Aglietti’s paper on the original implementation. Once again, the code has been simplified and made more readable and usable, allowing the entire algorithm to be run on any dataset the user wishes, using a single function call.

**Enhanced Features**

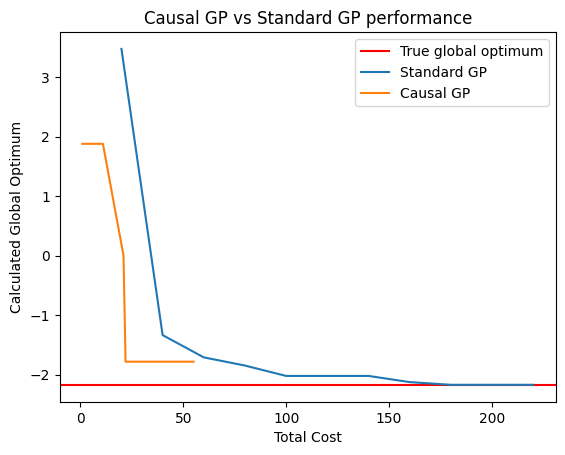
The original implementation by Aglietti et. al. required the use of specific, provided datasets due to the need for hardcoded do-calculus functions associated with each dataset’s causal model. The use of DoWhy removes this necessity, as it enables causal relationships to be learning from a causal model and any arbitrary dataset that follows the logic of that causal model. This means that CausalBO can be used on any dataset the user wishes, a massive usability improvement over the original.

The essential modules enabling causality, the CausalRBF kernel and CausalMean mean function, take advantage of the DoWhy package to allow for modularity. Due to the restrictions on do-calculus in the original implementation, the mean and kernel modules cannot function outside of the algorithm they were written for. The CausalBO kernel and mean function, however, function identically to any other mean and kernel available in BoTorch, with the addition of an arbitrary causal model included during declaration. This means that causality can be easily added to an existing solution or algorithm with little effort by simply replacing the existing mean and kernel functions, and describing a causal model as part of the declaration.

**Results**

Currently, there are still some issues with CausalBO that need to be addressed in future updates. Most notably, there are certain downsides to using DoWhy rather than hardcoded do-calculus. The interventions provided by DoWhy are computationally expensive and result in slower runtime than the original implementation. Due to the need to learn causal relationships from data using a regression approach, DoWhy suffers when there is a lack of observational data, both in a specific region of the interventional space and across the entire space. This can lead to suboptimal solutions for some functions.

A graph with blue and orange lines

Description automatically generated

Here, we can see that the mean function on the left (orange) provided by DoWhy does not always capture a good representation of the ground truth function (blue). In this case, the causal GP will never explore the area around the true global minimum due to the incorrectness of the mean function in that space, leading to a less optimal result than standard BO, as shown in the graph on the right. However, when this is not the case, CausalBO vastly outperforms standard Bayesian Optimization.

A graph of a graph

Description automatically generated with medium confidence A graph of a graph with numbers and a line

Description automatically generated

In the graphs above, where the mean function on the left (orange) provides a better representation of the ground truth (blue), the Causal GP takes advantage of this to converge to the global optimum at a much lower cost than the standard GP.

**Future Work**

For future versions of CausalBO, my main goal is to address the errors in the mean function provided by DoWhy. This entails delving deeper into the mechanics of the package to explore opportunities for accelerating intervention execution and enhancing mean function accuracy. If no improvements are feasible, I intend to account for potential errors within the mean and kernel modules themselves, minimizing their impact on the acquisition function, allowing it to choose points from areas that may be erroneously marked as sub-optimal.

Additionally, legacy code which does not handle tensors, instead performing computationally expensive for loops, will be purged and replaced with code that properly handles tensors to take advantage of the speed improvements.

Lastly, while the code supports exploration sets containing multiple variables, it currently performs worse on these sets than those containing single variables. Future work will include an investigation into the cause behind this issue and the implementation of solutions to remove this issue.