LPBF Bayesian Optimization Tool

1. Overview

A flexible Bayesian Optimization (BO) toolkit built with Botorch, designed to simplify the optimization of single or multi-objective black-box functions. This framework encapsulates basic BO workflows, warm-starting BO workflows and embedding-BO workflows.

2. Environmental Installation

Option 1: Using pip (no conda)

```
pip install -r requirements.txt
```

Option 2: Using conda (recommended)

```
conda create -n lpbf_bo python=3.10
conda activate lpbf_bo
pip install -r requirements.txt
```

3. Quick start

These two files will output the next step of BO based on the example data set in folder 'data'. You can change the configurence in the code to element different task

Example:

```
python warm_start.py
```

```
python embedding.py
```

4. Core API Reference

4.1 BaseBo Class

This class is defined in `./optimization/base_bo_class.py`

It is a primary class to implements BO process, include two main different functions:

1. Random sampling when you start optimizing a new task without any samples
2. Get candidtates solve by BO process when you already have some samples

Initialization

```
def __init__(self, input_dim: int, objective_dim: int, seed: int = None, device:
torch.device = None):
```

Args:

- `input_dim`: Number of input parameters (dimensions of the search space).
- `objective_dim`: Number of target objectives to optimize.
- `seed`: Random seed for reproducibility (None = no fixed seed, only used when you want to reproduce the results).
- `device`: Torch device for computations (auto-selects CUDA/CPU if None).

Key methods

add_data(X_new, Y_new)

Append new observations to the existing training data.

Converts input data to appropriate tensor format and merge with existing data.

Args:

- `X_new`: New input parameters to add. Can be array-like or tensor. Should have shape [`n_samples`, `input_dim`]

- `Y_new`: Corresponding objective values. Can be array-like or tensor. Should have shape [`n_samples`, `objective_dim`]

Raises:

```
AssertionError: If the number of samples in X_new and Y_new do not match
```

clear_data()

Reset training data (X and Y) to empty tensors while preserving dtype and device.

set_batch_size(batch_size, mini_batch_size

Set the batch sizes for BO iterations.

Args:

- `batch_size`: Number of points to query in each BO iteration

- `mini_batch_size`: Mini-batch size for internal optimization steps

set_batch_size(batch_size, mini_batch_size)

Set the batch sizes for BO iterations.

Args:

- `batch_size`: Number of points to query in each BO iteration
- `mini_batch_size`: Mini-batch size for internal optimization steps

set_bounds(lower, upper):

Set the parameter space boundaries.

Args:

- `lower`: Lower bounds for each input dimension. Array-like of length input_dim
- `upper`: Upper bounds for each input dimension. Array-like of length input_dim

Raises:

```
AssertionError: If bounds dimensions don't match input_dim or if upper <= lower
for any dimension
```

set_ref_point(slack)

Automatically determine a reference point for hyper-volume calculation.

The reference point is calculated as the minimum observed value for each objective minus a slack value, used in multi-objective optimization.

Args:

- `slack`: Slack value to subtract from the minimum observed values. Can be a scalar (same for all objectives) or array-like of length `objective_dim`

Returns:

```
torch.Tensor: The calculated reference point with shape `objective_dim`
```

Raises:

```
AssertionError: If Y is empty (no training data available)
```

run_start_sampling():

Generate initial random samples within the parameter bounds.

Used when no initial training data is available to start the BO process.

Returns:

torch.Tensor: Initial sample points with shape [`batch_size`, `input_dim`]

run_bo()

Run a single iteration of Bayesian Optimization.

Builds the model (if not already built) and queries new points using the BO strategy.

Returns:

```
torch.Tensor: Next set of points to evaluate with shape [`batch_size`,
`input_dim`]
```

4.2 WarmStartBO Class (Inherits from BaseBO)

This class is defined in `./optimization/warm_start_bo_class.py`,this class inherits directly from BaseBO with no changes to the initialization logic. It focuses on extending warm-start capabilities to reuse historical data for faster convergence in Bayesian Optimization workflows.

It implements warm start of BO, that is, you can add existing similar tasks (source task) as training samples and use them as references in the BO process of new tasks (target task). The effect obtained varies depending on the gap between tasks. Generally, the more similar the tasks are, the better the warm start effect is.

Initialization

same as BaseBo

Key methods

add_source_data(self, X_src, Y_src) -> None:

Append new observations from the source task to the existing source data.

Converts input data to appropriate tensor format and concatenates with existing source data.

Args:

- `X_src`: New source task input parameters. Can be array-like or tensor of shape (`n_samples`, `input_dim`)
- `Y_src`: Corresponding source task objective values. Can be array-like or tensor of shape (`n_samples`, `objective_dim`)

Raises:

```
AssertionError: If the number of samples in X_src and Y_src do not match
```

4.3 EmbeddingBO Class (Inherits from BaseBO)

A Bayesian Optimization (BO) class with embedding-based representations.

This class extends the base BO functionality by incorporating feature vector embeddings, allowing for augmented input spaces that combine original parameters with embedding features.

Initialization

```
def __init__(self, input_dim: int, objective_dim: int, vector_dim: int, seed: int
= None, device: torch.device = None)
```

Args:

- `vector_dim`: Dimension of the embedding feature vectors

Key methods

attach_feature_vector(x, v):

Augment input data with feature vectors by appending them as additional dimensions.

Combines original input features with embedding vectors to create an augmented input space for the Gaussian Process model.

Args:

- `x`: Original input data. Tensor of shape [`n_samples`, `n_features`], where n_samples is the number of data points and n_features is the input dimension
- `v`: Feature/embedding vector to attach. Can be a list or tensor of shape [1, `n_embedding_features`], where n_embedding_features is the dimension of the embedding vector

Returns:

```
    torch.Tensor: Augmented input data with shape [n_samples, n_features +
n_embedding_features]
```

add_augment_data(X_new, Y_new, v)

Append new observations with embedded feature vectors to training data.

Augments the input data with the provided feature vector before adding it to the existing training dataset.

Args:

- `X_new`: New input parameters to add. Can be array-like or tensor of shape [n_samples, `input_dim`]
- `Y_new`: Corresponding objective values. Can be array-like or tensor of shape [n_samples, `objective_dim`]
- `v`: Feature vector to embed with the input data. Shape [1, `vector_dim`]

Raises:

```
AssertionError: If the number of samples in X_new and Y_new do not match
```