# Tutorial5_Parallel_KG

July 20, 2022

## 0.1 Lab 5: The parallel knowledge gradient acquisition function in BoTorch

This tutorial shows two things: - how to use BayesOpt when we can do parallel objective function evaluations - how to use the knowledge-gradient (KG) acquisition function. The KG acquisition function provides better query efficiency than expected improvement (i.e., it typically requires fewer objective function evaluations to find a good solution). KG is also important in certain kinds of grey-box BO, especially multi-information source BO and multi-fidelity BO.

This tutorial is based on the BoTorch tutorial, https://botorch.org/tutorials/one_shot_kg

For problems where we are maximizing and can take batches of size $q$, the *parallel Knowledge Gradient* (KG) acquisition function [1] is defined by:

$$\text{KG}(x_{1:q}) = E_n \left[\mu^*_{n+q} - \mu^*_n \mid \text{sample at } x_{1:q}\right]$$

where - $\mu^*_n = \max_x E_n[f(x)]$ is the maximum of the posterior mean after $n$ samples, and - $\mu^*_{n+q} = \max_x E_{n+q}[f(x)]$ is the maximum of the posterior mean after ]additional samples at the $q$ points $x_{1:q}$. - $E_n$ indicates the expectation with respect to the posterior distribution after $n$ samples. This expectation is taken over the fantasized samples of $f(x_{1:q})$, which are multivariate normal under this posterior.

BoTorch implements a generalization of parallel KG that allows a composition (see [2] for applications of function compositions and computation of the EI acquisition function) with a function $g$ in which: - $\mu^*_n = \max_x E_n[g(f(x))]$ - $\mu^*_{n+q} = \max_x E_{n+q}[g(f(x))]$


[1] J. Wu and P. Frazier. The parallel knowledge gradient method for batch Bayesian optimization. NIPS 2016.

[2] R. Astudillo and P. Frazier. Bayesian optimization of composite functions. ICML 2019.

[3] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: Programmable Bayesian Optimization in PyTorch. ArXiv 2019.


### 0.1.1 Setting up a toy model

We'll fit a standard `SingleTaskGP` model on noisy observations of the synthetic function $f(x) = \sin(2\pi x_1) * \cos(2\pi x_2)$ in `d=2` dimensions on the hypercube $[0, 1]^2$.

[12]:
```
import math
import torch

from botorch.fit import fit_gpytorch_model
```

```python
from botorch.models import SingleTaskGP
from botorch.utils import standardize
from gpytorch.mlls import ExactMarginalLogLikelihood
```

[13]:
```python
# Here we create a tensor that will specify our feasible domain,
# which is the 2-dimensional unit hypercube
bounds = torch.stack([torch.zeros(2), torch.ones(2)])

train_X = bounds[0] + (bounds[1] - bounds[0]) * torch.rand(20, 2)
train_Y = torch.sin(2 * math.pi * train_X[:, [0]]) * torch.cos(2 * math.pi *␣
↪train_X[:, [1]])

train_Y = standardize(train_Y + 0.05 * torch.randn_like(train_Y))

model = SingleTaskGP(train_X, train_Y)
mll = ExactMarginalLogLikelihood(model.likelihood, model)
fit_gpytorch_model(mll);
```

### 0.1.2 Optimizing KG

[14]:
```python
from botorch.acquisition import qKnowledgeGradient
from botorch.optim import optimize_acqf
from botorch.utils.sampling import manual_seed

# increasing num_fantasies makes the method more accurate, but uses more␣
↪computation
qKG = qKnowledgeGradient(model, num_fantasies=128)

# increasing num_restarts and raw_samples makes the method more accurate, but␣
↪uses more computation
with manual_seed(1234):
    candidates, acq_value = optimize_acqf(
        acq_function=qKG,
        bounds=bounds,
        q=2,
        num_restarts=10,
        raw_samples=512,
    )
```

[15]:
```python
# This gives the set of points that maximize the KG acqusition function.
# It contains 2 points because we asked for a batch of size q=2
candidates
```

[15]:
```
tensor([[0.8250, 0.5266],
        [0.9652, 0.5132]])
```

[10]:
```python
# Since we do not pass a `current_value` argument to optimize_acqf,
# acq_value is not actually the KG value, but is instead offset by $\mu^*_n$.
acq_value
```

[10]: tensor(2.1769)

**Exercise 1** Use the cells above to write code in which we do batches of q=2 evaluations at a time for minimizing the Hartmann 6 objective function, choosing them using the KG method. Start with a copy / paste of the code from Exercise 2 in Lab 3. Create the same output as in that exercise, except each iteration will have 2 points to report.

[ ]:
```python

```