Supplement to Bayesian Optimisation over Multiple Continuous and Categorical Inputs

Binxin Ru* [1] Ahsan S. Alvi* [1] Vu Nguyen [1] Michael A. Osborne [1] Stephen J Roberts [1]

A. Notation summary

Please refer to Table 1 for a description of the notations used in our paper.

B. EXP3

EXP3 is a method proposed in (Auer et al., 2002) to deal with the non-stochastic, adversarial multi-arm bandit problem which is a more general setting and can model any form of non-stationarity (Allesiardo et al., 2017). In such problem setting, the rewards g_t are chosen by an adversary at each iteration. For a categorical variable h with N categories and bounded reward ($g_t \leq 1$), the regret bound for EXP3 algorithm is

$$\begin{aligned}\mathcal{R}_T^{EXP3} &= \max_{h^* \in \{1,\dots,N\}} \sum_t^T g_t(h^*) - \mathbb{E}\left[\sum_t^T g_t(h)\right] \\ &\leq 2.63\sqrt{TN\log(N)} \qquad (1)\end{aligned}$$

where h^* is the best single action over all rounds and $g_t(\cdot)$ is the reward function. Please refer to (Auer et al., 2002) for detailed derivation.

C. Regret bound for EXP3 in CoCaBO

Assume we have c categorical variables $\mathbf{h} = [h_1, \dots, h_c]$ and each categorical variables h_j, determined by an agent, can take one of N categories. A simplified version of the cumulative regret of such multi-agent MAB setting after T iterations can be written as:

$$\begin{aligned}\sum_j^c \mathcal{R}_T^j = &\sum_j^c \sup_{\mathbf{h}_{\setminus j}} \left\{ \max_{h_j^* \in \{1,\dots,N\}} \sum_t^T g_t(h_j^*, \mathbf{h}_{\setminus j}) \right. \qquad (2) \\ &\left. - \mathbb{E}\left[\sum_t^T g_t(h_j, \mathbf{h}_{\setminus j})\right] \right\} \qquad (3)\end{aligned}$$

where $\mathcal{R}_T^j$ is the cumulative regret of agent j when we consider the decisions by the other agents $\mathbf{h}_{\setminus j}$ are controlled by the adversary. $g_t(\cdot)$ is the reward achieved by the joint decisions of all the agents and is equal to the normalised (between $[0, 1]$) objective function value in our case.

We first look at the cumulative regret for a single agent when when the decision of all the other agents (i.e. the value of all the other categorical variables) are fixed to $\mathbf{h}_{\setminus j}$:

$$\begin{aligned}\mathcal{R}_T^j = &\sup_{\mathbf{h}_{\setminus j}} \left\{ \max_{h_j^* \in \{1,\dots,N\}} \sum_t^T g_t(h_j^*, \mathbf{h}_{\setminus j}) \right. \qquad (4) \\ &\left. - \mathbb{E}\left[\sum_t^T g_t(h_j, \mathbf{h}_{\setminus j})\right] \right\} \qquad (5)\end{aligned}$$

At iteration t with given continuous inputs, the adversary's reward function $g_t(h_j, \mathbf{h}_{\setminus j})$ (with fixed $\mathbf{h}_{\setminus j}$) is equivalent to $\hat{g}_t(h_j)$ where $g_t(\cdot)$ has N^c arbitrary outputs while $\hat{g}_t(\cdot)$ has only N arbitrary outputs (Carlucci et al., 2020). By substituting $\hat{g}_t(h_j) = g_t(h_j, \mathbf{h}_{\setminus j})$ in Equation (6), we reduce the setting for a single agent to the EXP3 setting (Auer et al., 2002):

$$\begin{aligned}\mathcal{R}_T^j &= \sup_{\mathbf{h}_{\setminus j}} \left\{ \max_{h_j^* \in \{1,\dots,N\}} \sum_t^T \hat{g}_t(h_j^*) - \mathbb{E}\left[\sum_t^T \hat{g}_t(h_j)\right] \right\} \\ &= \sup_{\mathbf{h}_{\setminus j}} \left\{ \mathcal{R}_T^{EXP3,j} \right\} \leq \sup_{\mathbf{h}_{\setminus j}} \left\{ 2.63\sqrt{TN\log(N)} \right\} \qquad (6)\end{aligned}$$

Therefore, the cumulative regret of our multi-agent EXP3 setting after T iterations has the bound:

$$\sum_j^c \mathcal{R}_T^j \leq 2.63c\sqrt{TN\log(N)} \qquad (7)$$

which is sub-linear as $\lim_{T\to\infty} \frac{\sum_j^c \mathcal{R}_T^j}{T} = 0$ and increases with the number of categorical variables c and the number of categories for each variable.

*Equal contribution [1]University of Oxford. Correspondence to: Binxin Ru <robin@robots.ox.ac.uk>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

Table 1: Notation list

Notation	**Type**	**Meaning**
σ_l^2, σ^2	scalar	lengthscale for RBF kernel, noise output variance (or measurement noise)
$\mathcal{X} \in \mathbb{R}^d$	search domain	continuous search space where d is the dimension
d	scalar	dimension of the continuous variable
c	scalar	dimension of categorical variables
$\mathbf{x}_t$	vector	a continuous selection by BO at iteration t
N_c	scalar	number of choices for categorical variable c
$\mathbf{h}_t = [h_{t,1}, ..., h_{t,c}]$	vector	vector of categorical variables
$\mathbf{z}_t = [\mathbf{x}_t, \mathbf{h}_t]$	vector	hyperparameter input including continuous and categorical variables
$\mathcal{D}_t$	set	observation set $D_t = \{z_i, y_i\}_{i=1}^t$

D. Categorical kernel relation with RBF

In this section we discuss the relationship between the categorical kernel we have proposed and a RBF kernel. Our categorical kernel is reproduced here for ease of access:

$$k_h(\mathbf{h}, \mathbf{h}') = \frac{\sigma^2}{c} \sum_{i=1}^{c} \mathbb{I}(h_i - h_i'). \tag{8}$$

Apart from the intuitive argument, that this kernel allows us to model the degree of similarity between two categorical selections, this kernel can also be derived as a special case of an RBF kernel. Consider the standard RBF kernel with unit variance evaluated between two scalar locations a and a':

$$k(a, a') = \exp\left(-\frac{1}{2}\frac{(a-a')^2}{l^2}\right). \tag{9}$$

The lengthscale in Eq, 9 allows us to define the similarity between the two inputs, and, as the lengthscale becomes smaller, the distance between locations that would be considered similar (i.e. high covariance) shrinks. The limiting case $l \to 0$ states that if two inputs are not exactly the same as each other, then they provide no information for inferring the GP posterior's value at each other's locations. This causes the kernel to turn into an indicator function as in Eq. 8 above (Kulis & Jordan, 2011):

$$k(a, a') = \begin{cases} 1, & \text{if } a = a' \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

By adding one such RBF kernel with $l \to 0$ for each categorical variable in h and normalising the output we arrive at the form in Eq. 8.

E. Learning the hyperparameters in the CoCaBO kernel

We present the derivative for estimating the variable λ in our CoCaBO kernel.

$$\begin{aligned} k_z(\mathbf{z}, \mathbf{z}') =& (1-\lambda)\left(k_h(\mathbf{h}, \mathbf{h}') + k_x(\mathbf{x}, \mathbf{x}')\right) \\ &+ \lambda k_h(\mathbf{h}, \mathbf{h}') k_x(\mathbf{x}, \mathbf{x}'). \end{aligned} \tag{11}$$

The hyperparameters of the kernel are optimised by maximising the log marginal likelihood (LML) of the GP surrogate

$$\theta^* = \arg\max_{\theta} \mathcal{L}(\theta, \mathcal{D}), \tag{12}$$

where we collected the the hyperparameters of both kernels as well as the CoCaBO hyperparameter into $\theta = \{\theta_h, \theta_x, \lambda\}$. The LML and its derivative are defined as (Rasmussen & Williams, 2006)

$$\mathcal{L}(\theta) = -\frac{1}{2}\mathbf{y}^\intercal \mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}| + \text{constant} \tag{13}$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{2}\left(\mathbf{y}^\intercal \mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta}\mathbf{K}^{-1}\mathbf{y} - \text{tr}\left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta}\right)\right), \tag{14}$$

where $\mathbf{y}$ are the function values at sample locations and $\mathbf{K}$ is the kernel matrix of $k_z(\mathbf{z}, \mathbf{z}')$ evaluated on the training data.

Optimisation of the LML was performed via multi-started gradient descent. The gradient in Equation 14 relies on the gradient of the kernel k_z w.r.t. each of its parameters:

$$\frac{\partial k_z}{\partial \theta_h} = (1-\lambda)\frac{\partial k_h}{\partial \theta_h} + \lambda k_x \frac{\partial k_h}{\partial \theta_h} \tag{15}$$

$$\frac{\partial k_z}{\partial \theta_x} = (1-\lambda)\frac{\partial k_x}{\partial \theta_x} + \lambda \frac{\partial k_x}{\partial \theta_x} k_h \tag{16}$$

$$\frac{\partial k_z}{\partial \lambda} = -(k_h + k_x) + k_h k_x, \tag{17}$$

where we used the shorthand $k_z = k_z(\mathbf{z}, \mathbf{z}')$, $k_h = k_h(\mathbf{h}, \mathbf{h}')$ and $k_x = k_x(\mathbf{x}, \mathbf{x}')$.

F. Kriging Believer

The Kriging Believer (KB) is a method proposed in (Ginsbourger et al., 2010) to sequentially select batch points in the continuous space. In KB, as the name suggested, we fully trust the predictive posterior and use the posterior mean $\mu(\mathbf{x}_t^{(1)})$ at a selected batch location $\mathbf{x}_t^{(1)}$ as a proxy for the true function value $f(\mathbf{x}_t^{(1)})$. We then augment the observation data D_{t-1} with this hallucinated data $\{\mathbf{x}_t^{(1)}, \mu(\mathbf{x}_t^{(1)})\}$ to update the surrogate model as well as the the acquisition function. The next point in the batch is then selected by maximising the updated acquisition function. This process repeats until all b points in the batch are selected as shown in Algorithm 1.

Algorithm 1 Kriging Believer

```
1: Input: Observation data $\mathcal{D}_{t-1}$, batch size $b$
2: Output: The batch points $\mathcal{B}_t = \{\mathbf{x}_t^{(1)}, \dots, \mathbf{x}_t^{(b)}\}$
3: $\mathcal{D}'_{t-1} = \mathcal{D}_{t-1}$
4: for $j = 1, \dots, b$ do
5:     $\mathbf{x}_t^{(j)} = \arg\max \alpha(\mathbf{x}|\mathcal{D}'_{t-1})$
6:     Compute $\mu(\mathbf{x}_t^{(j)})$
7:     $\mathcal{D}'_{t-1} \leftarrow \mathcal{D}'_{t-1} \cup (\mathbf{x}_t, \mu(\mathbf{x}_t^{(j)}))$
8: end for
```

G. Description of the optimisation problems

G.1. Synthetic test functions

We generated several synthetic test functions: *Func-2C*, *Func-3C* and a *Ackley-cC* series, whose input spaces comprise both continuous variables and multiple categorical variables. Each of the categorical inputs in all three test functions have multiple values.

Func-2C is a test problem with 2 continuous inputs ($d = 2$) and 2 categorical inputs ($c = 2$). The categorical inputs decide the linear combinations between three 2-dimensional global optimisation benchmark functions: beale (bea), six-hump camel (cam) and rosenbrock (ros)[1].

Func-3C is similar to *Func-2C* but with 3 categorical inputs ($c = 3$) which leads to more complicated linear combinations among the three functions.

Ackley-cC comprises $c = \{2, 3, 4, 5\}$ categorical inputs and 1 continous input ($d = 1$). Here, we convert c dimensions of the $c+1$-dimensional Ackley function into 17 categories each.

The value range for both continuous and categorical inputs of these functions are summarised in Table 2.

G.2. Real-world problems

We defined three real-world tasks of tuning the hyperparameters for ML algorithms: *SVM-Boston*, *XG-MNIST*, *NAS-CIFAR10* and *NN-Yacht* .

SVM-Boston outputs the negative mean square error of support vector machine (SVM) for regression on the test set of Boston housing dataset. We use the Nu Support Vector regression algorithm in the scikit-learn package (Pedregosa et al., 2011) and use a train/test split of 7 : 3.

XG-MNIST returns classification accuracy of a XGBoost algorithm (Chen & Guestrin, 2016) on the testing set of the MNIST dataset. We use the *xgboost* package and adopt a stratified train/test split of 7 : 3.

NAS-CIFAR10 performs the architecture search on convolutional neural network topology for CIFAR10 classification. We use the public architecture dataset, NAS-Bench-101 (Ying et al., 2019) [2], which contains the precomputed training, validation, and test accuracies of $423,624$ unique neural networks on CIFAR10 after training for 108 epochs. All these networks are exhaustively generated from a graph-based search space. The search space comprises a 7-node directed acyclic graph(DAG) with the first node being the input and the last node being the output. The 5 intermediate nodes can perform one of the following 3 operations: 3x3 convolution, 1x1 convolution, and 3x3 max-pooling (i.e. 5 categorical variables, each with 3 categorical choices). There are 21 possible edges in the DAG but any valid architecture is limited to a maximum of 9 edges. We follow the encoding scheme in (Ying et al., 2019), which defines a probability value $x_i \in [0, 1]$ for each possible edge i and defines an integer parameter $x_2 2 \in [0, 9]$. An architecture is generated by activating

[1]The analytic forms of these functions are available at `https://www.sfu.ca/~ssurjano/optimization.html`

[2]Code and data are available at `https://github.com/google-research/nasbench`

Table 2: Continuous and categorical input range of the synthetic test functions

Function f	Inputs $\mathbf{z} = [\mathbf{h}, \mathbf{x}]$	Input values
Func-2C ($d = 2$, $c = 2$)	h_1	$\{\text{ros}(\mathbf{x}), \text{cam}(\mathbf{x}), \text{bea}(\mathbf{x})\}$
	h_2	$\{+\text{ros}(\mathbf{x}), +\text{cam}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x})\}$
	$\mathbf{x}$	$[-1, 1]^2$
Func-3C ($d = 2$, $c = 3$)	h_1	$\{\text{ros}(\mathbf{x}), \text{cam}(\mathbf{x}), \text{bea}(\mathbf{x})\}$
	h_2	$\{+\text{ros}(\mathbf{x}), +\text{cam}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x})\}$
	h_3	$\{+5 \times \text{cam}(\mathbf{x}), +2 \times \text{ros}(\mathbf{x}), +2 \times \text{bea}(\mathbf{x}), +3 \times \text{bea}(\mathbf{x})\}$
	$\mathbf{x}$	$[-1, 1]^2$
Ackley-cC for $c = \{2, 3, 4, 5\}$ ($d = 1$, $N_i = 17$)	h_i for $i = 1, 2, \ldots, 5$	$\{z_i = -1 + 0.125 \times (j - 1),\ \text{for } j = 1, 2, \ldots, 17\}$
	$\mathbf{x}$	$[-1, 1]$

Table 3: Continuous and categorical input ranges of the real-world problems

Problems	Inputs $\mathbf{z} = [\mathbf{h}, \mathbf{x}]$	Input values
SVM-Boston ($d = 3$, $c = 3$)	kernel type h_1	{linear, poly, RBF, sigmoid}
	kernel coefficient h_2	{scale, auto }
	shrinking h_3	{shrinking on, shrinking off}
	penalty parameter x_1	$[0, 10]$
	tolerance for stopping x_2	$10^{[10^{-6}, 1]}$
	lower bound of the fraction of support vector x_3	$[0, 1]$
XG-MNIST ($d = 5$, $c = 3$)	booster type h_1	{gbtree, dart}
	grow policies h_2	{depthwise, loss}
	training objective h_3	{softmax, softprob}
	learning rate x_1	$[0, 1]$
	maximum dept x_2	$[1, 2, \ldots, 10]$
	minimum split loss x_3	$[0, 10]$
	subsample x_4	$[0.001, 1]$
	regularisation x_5	$[0, 5]$
NAS-CIFAR10 ($d = 22$, $c = 5$)	operations for the 5 intermediate nodes in the DAG $h_1, \ldots, h_5$	{3x3 conv, 1x1 conv, and 3x3 max-pool}
	probability values for the 21 possible edges in the DAG $x_1, \ldots, x_{21}$	$[0, 1]$
	Number of edges present in the DAG x_{22}	$[0, 9]$
NN-Yacht ($d = 3$, $c = 3$)	activation type h_1	{ReLU, tanh, sigmoid}
	optimiser type h_2	{SGD, Adam, RMSprop, AdaGrad}
	suggested dropout value h_3	$\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$
	learning rate x_1	$10^{[-5, -1]}$
	number of neurons x_2	$2^{[4, 7]}$
	aleatoric variance x_3	$[0.2, 0.8]$

the $x_2 2$ edges with the highest probability. The design of the search space turns the neural architecture search into an optimisation problem involving multiple categorical variables and continuous variables. This dataset enables us to quickly evaluate the architectures proposed by BO algorithms by looking up the dataset and compare the search strategies without the need for huge computing resources.

NN-Yacht returns the negative log likelihood of a one-hidden-layer neural network regressor on the test set of Yacht hydrodynamics dataset. We follow the MC Dropout implementation and the random train/test split on the dataset proposed in (Gal & Ghahramani, 2016)[3]. The simple neural network is trained on mean squared error objective for 20 epochs with a batch size of 128. We run 100 stochastic forward passes in the testing stage to approximate the predictive mean and variance. The results of CoCaBO against other methods in the batch setting ($b = 4$) for this task is shown in Figure 2 and against, CoCaBOs outperform other methods.

The hyperparameters over which we optimise for each above-mentioned ML task are summarised in Table 3. One point to note is that we present the unnormalised range for the continuous inputs in Table 3 but normalise all continuous inputs to $[-1, 1]$ for optimisation in our experiments. All the remaining hyperparameters are set to their default values.

[3]Code and data are available at `https://github.com/yaringal/DropoutUncertaintyExps`

H. Additional experimental results

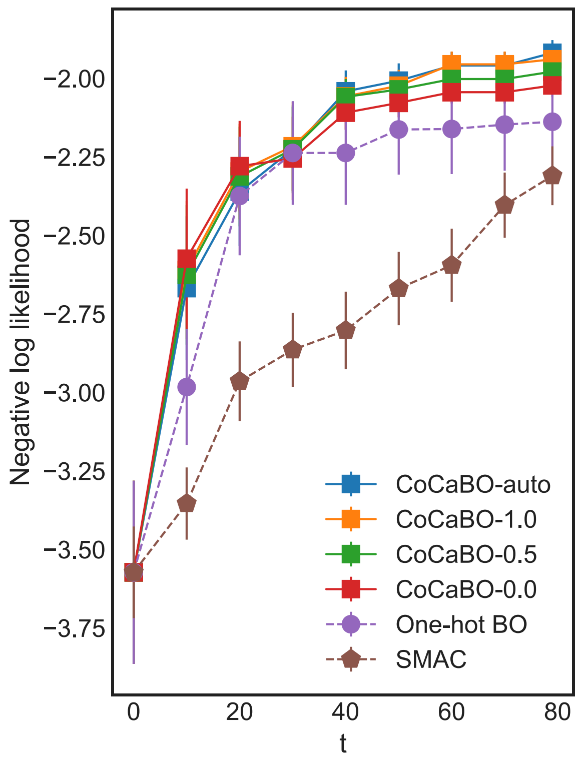

Figure 1: Performance of CoCaBOs against existing methods on *NN-Yacht* in the batch setting ($b = 4$)

I. Wall-clock time overhead of CoCaBO

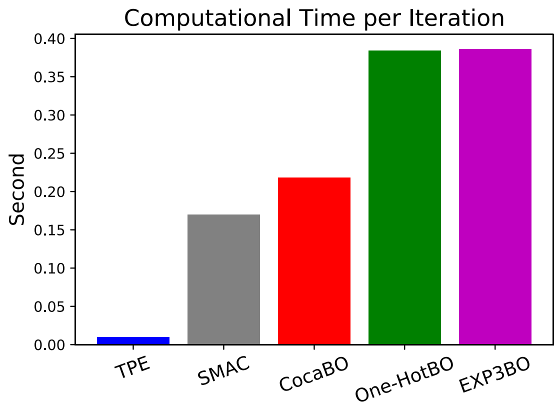

Figure 2: The mean wall-clock time overheads of different methods for each BO iterations on the same machine. Such an overhead ($0.2s$/itr) is negligible compared to the function evaluation time in real practice (for NAS it can take $2500s$ to train a query architecture even with TPUs).

References

Allesiardo, Robin, Féraud, Raphaël, and Maillard, Odalric-Ambrym. The non-stationary stochastic multi-armed bandit problem. *International Journal of Data Science and Analytics*, 3(4):267–283, 2017.

Auer, Peter, Cesa-Bianchi, Nicolo, Freund, Yoav, and Schapire, Robert E. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

Carlucci, Fabio Maria, Esperança, Pedro M, Singh, Marco, Gabillon, Victor, Yang, Antoine, Xu, Hang, Chen, Zewei, and Wang, Jun. {MANAS}: Multi-agent neural architecture search, 2020.

Chen, Tianqi and Guestrin, Carlos. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794. ACM, 2016.

Gal, Yarin and Ghahramani, Zoubin. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*, 2016.

Ginsbourger, David, Le Riche, Rodolphe, and Carraro, Laurent. Kriging is well-suited to parallelize optimization. In *Computational Intelligence in Expensive Optimization Problems*, pp. 131–162. Springer, 2010.

Kulis, Brian and Jordan, Michael I. Revisiting k-means: New algorithms via Bayesian nonparametrics. *arXiv preprint arXiv:1111.0352*, 2011.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Rasmussen, C E and Williams, C K I. *Gaussian processes for machine learning*. 2006.

Ying, Chris, Klein, Aaron, Real, Esteban, Christiansen, Eric, Murphy, Kevin, and Hutter, Frank. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.