Augmenting Bayesian Optimization with Preference-based Expert Feedback

Daolang Huang [1] **Louis Filstroff** [2] **Petrus Mikkola** [1] **Runkai Zheng** [3] **Milica Todorovic** [4] **Samuel Kaski** [1 5]

Abstract

Bayesian optimization (BO) is a well-established method to optimize black-box functions whose direct evaluations are costly. In this paper, we tackle the problem of incorporating expert knowledge into BO, with the goal of further accelerating the optimization, which has received little attention so far. We design a multi-task learning architecture for this task, with the goal of jointly eliciting the expert knowledge and minimizing the objective function. In particular, this allows for the expert knowledge to be transferred into the BO task. We introduce a specific architecture based on Siamese neural networks to handle the knowledge elicitation from pairwise queries. Experiments on various benchmark functions show that the proposed method significantly speeds up BO even when the expert knowledge is biased.

1. Introduction

Bayesian optimization (BO) (Jones et al., 1998; Brochu et al., 2010) has become a well-established class of methods to optimize black-box functions, with applications in hyperparameter tuning (Snoek et al., 2012), chemistry (Hase et al., 2018), and material science (Zhang et al., 2020), to cite only a few. Formally, let $f : \mathcal{X} \to \mathbb{R}$ be a black-box function defined over some compact space $\mathcal{X} \subset \mathbb{R}^d$. We assume that evaluating f at some point $\mathbf{x}$ is possible, but expensive. The goal is to find the global optimum $\mathbf{x}^\star$, defined as

$$\mathbf{x}^\star = \arg\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}). \tag{1}$$

Each domain-specific BO problem naturally has its domain experts with their own knowledge of the problem, i.e., often tacit knowledge about the shape of f or where the global optimum might lie. Moreover, the cost of asking the expert can be significantly cheaper than the cost of obtaining the value of $f(\mathbf{x})$ in many applications. However, leveraging that expert knowledge in order to speed up BO has only started to receive attention in the literature very recently (Li et al., 2020; Ramachandran et al., 2020; Souza et al., 2021; Hvarfner et al., 2022), and none of these works properly discuss how to obtain such knowledge. The reason may be that eliciting knowledge from humans is notoriously challenging. Indeed, humans can be bad at evaluating absolute magnitudes, but can be much better at comparing instances (Millet, 1997; Shah et al., 2014). This has been utilized for preference learning through pairwise comparisons of items (Chu & Ghahramani, 2005), and has been expanded to an online learning setting (Brochu et al., 2008; González et al., 2017) to find the optimum of the preferences. Even though this approach has recently been shown to work in expert knowledge elicitation (Mikkola et al., 2020), there is still a need for methods to elicit knowledge from the expert with the goal of performing BO for f. Transferring that knowledge into the BO task also represents a challenge.

In this paper, we propose an expert knowledge-augmented BO method. We formulate the problem as a multi-task learning (MTL) problem (Caruana, 1997), and propose to solve it with a Bayesian neural network-based architecture whose goal is to learn both f and the expert knowledge. The key insight is to leverage statistical strength across the latent representations of the two tasks, which both are about the same ground truth f, but imperfect in different ways. We operate by first querying the expert, and then initialize the BO with that knowledge, which leads to a speed-up for the BO. For expert knowledge elicitation, we introduce a novel preference learning method based on Siamese neural networks. We call it a preferential Bayesian neural network (PBNN); it not only learns the instance preference relationship, but is also capable of capturing the latent function shape.

Experiments demonstrate that PBNN leads to better performance than existing GP-based approaches with limited numbers of data acquisition steps. More importantly, we show that the standard BO optimization can be significantly sped-up when the elicited expert knowledge is transferred to the BO surrogate.

[1]Department of Computer Science, Aalto University, Finland [2]ENSAI, CREST, France [3]School of Data Science, The Chinese University of Hong Kong (Shenzhen), China [4]Department of Mechanical and Materials Engineering, University of Turku, Finland [5]Department of Computer Science, The University of Manchester, UK. Correspondence to: Daolang Huang <daolang.huang@aalto.fi>.

The Many Facets of Preference Learning Workshop at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023.

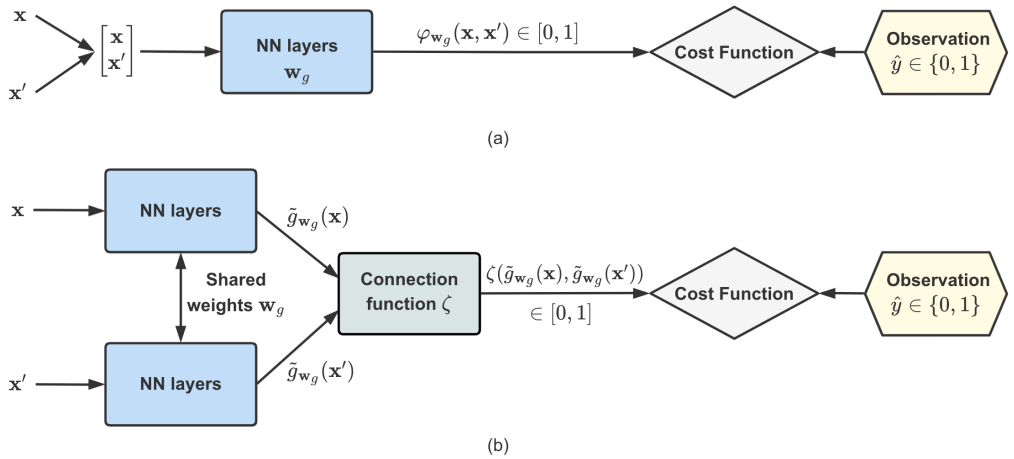

Figure 1. (a) A simple neural network architecture to handle preference learning. The neural network outputs the probability of $\hat{y}$ to be 1 given $\mathbf{x}$ and $\mathbf{x}'$, but fails at capturing the shape of the utility function of the expert, g. (b) The proposed architecture, based on a Siamese neural network. It also solves the preference learning problem, but each sub-network outputs a real-valued latent representation that we interpret as $g(\mathbf{x})$. The network is able to learn the shape of g, up to a monotonic transformation.

2. Preferential Bayesian Neural Network

A first goal is to elicit the knowledge of the domain expert. We model it as a function, denoted by g, which represents their beliefs. We can interpret g as a biased version of f. By querying pairwise comparisons from the expert, we build a probabilistic surrogate of g. Note that g corresponds to the *utility function* of the expert, which is a well-studied concept in economics (Rader, 1963).

As motivated in the introduction, it is much easier for humans to compare two items than to give the absolute magnitude of one item (for instance, it is extremely difficult for a material scientist to assess the total energy of a simulated material model, but comparing the stability between two material configurations is easier). Hence, we assume that the expert cannot directly return the value $g(\mathbf{x})$ for a certain $\mathbf{x}$. Instead, given a pair of covariates $[\mathbf{x}, \mathbf{x}'] \in \mathcal{X} \times \mathcal{X}$, we assume that the expert is able to return a preference label $\hat{y} \in \{0, 1\}$, with value $\hat{y}_i = 1$ if $g(\mathbf{x}) \geq g(\mathbf{x}')$, and $\hat{y}_i = 0$ if $g(\mathbf{x}) < g(\mathbf{x}')$. We will sequentially collect a dataset $\mathcal{D}_g = \{(\mathbf{x}_i, \mathbf{x}'_i, \hat{y}_i)\}_{i=1}^M$, which is in turn used to build a probabilistic surrogate of g. Note that based on the ordered data, we will be able to learn about the shape of g, but not about the actual magnitude and scale, meaning that any monotonic transformation of g is an equivalent solution.

We introduce a neural network architecture to handle preference learning. A natural solution is to expand the input space to $\mathcal{X} \times \mathcal{X}$ by concatenating the covariates pair. Such an architecture is displayed in Figure 1-a. However, by doing so, we would not learn anything about the function g. Instead, we propose to use an architecture based on Siamese networks (Figure 1-b), coined PBNN (preferential Bayesian neural network), which is detailed in the next subsection.

2.1. Preference Learning with Siamese Networks

Network architecture and loss function A Siamese neural network consists of two parallel, identical sub-networks that share the same set of parameters. Each sub-network takes a distinct input, and the representations produced by each network are then compared using a connection function, which we denote by ζ. They were introduced in the 90s for signature verification (Bromley et al., 1993), and have since become very popular for, e.g., one-shot/few-shot learning (Koch et al., 2015), and object tracking (Bertinetto et al., 2016). As our knowledge elicitation task amounts to a comparison between two values at a time, the Siamese network architecture naturally fits to our problem.

The proposed PBNN uses that architecture exactly. Let us denote by $\mathbf{w}_g$ the weights shared by the two sub-networks, and let us denote by $\tilde{g}_{\mathbf{w}_g}(\mathbf{x})$ and $\tilde{g}_{\mathbf{w}_g}(\mathbf{x}')$ the representations produced by forwarding $\mathbf{x}$ and $\mathbf{x}'$. PBNN models the probability of $\hat{y}$ to be 1 given two inputs $\mathbf{x}$ and $\mathbf{x}'$ by comparing $\tilde{g}_{\mathbf{w}_g}(\mathbf{x})$ and $\tilde{g}_{\mathbf{w}_g}(\mathbf{x}')$ with the connection function ζ. Contrary to the "concatenation" baseline approach previously described (Figure 1-a), the representations produced by the two sub-networks are real-valued, and we interpret them as the values of the true function g. These two values are further combined to provide a value in $[0, 1]$, i.e., our connection function is naturally chosen as

$$\zeta(\tilde{g}_{\mathbf{w}_g}(\mathbf{x}), \tilde{g}_{\mathbf{w}_g}(\mathbf{x}')) = \sigma(\tilde{g}_{\mathbf{w}_g}(\mathbf{x}) - \tilde{g}_{\mathbf{w}_g}(\mathbf{x}')), \quad (2)$$

where $\sigma(x) = \dfrac{1}{1 + e^{-x}}$ is the sigmoid function. We can then train the whole model by minimizing the negative log-likelihood, which is equivalent to using the binary cross-entropy loss. We write

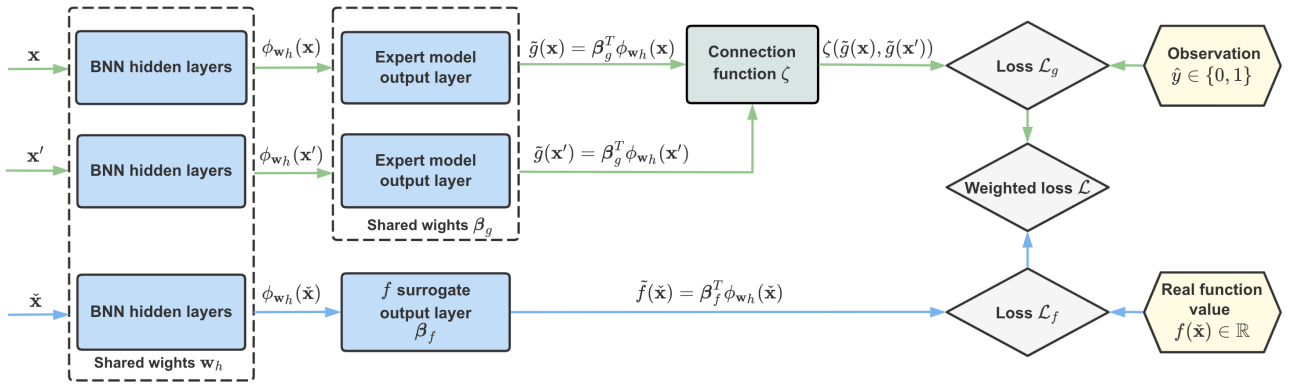

Figure 2. Multi-task learning (MTL) architecture, for the tasks of building jointly probabilistic surrogates for both the expert's beliefs g and the function to be optimized f. The green flow corresponds to surrogate of g, used in the knowledge elicitation part, and the blue flow to the surrogate of f, which is used for Bayesian optimization. The first layers (with parameters $\mathbf{w}_h$) are shared for the two surrogates, meaning that we aim at leveraging the similarity between the two functions by sharing some representations. They only differ in the output layer, parameterized by either $\boldsymbol{\beta}_g$ or $\boldsymbol{\beta}_f$. The two losses $\mathcal{L}_g$ and $\mathcal{L}_f$ are combined using a weighted scheme, which will give more and more importance to $\mathcal{L}_f$ as we get evaluations of f.

$$\log p(\mathcal{D}_g|\mathbf{w}_g) = \sum_{i=1}^{M} \log p(\hat{y}_i|[\mathbf{x}_i, \mathbf{x}_i'], \mathbf{w}_g) \tag{3}$$

$$= \sum_{i=1}^{M} \Big(\hat{y}_i \log \left(\zeta(\tilde{g}_{\mathbf{w}_g}(\mathbf{x}), \tilde{g}_{\mathbf{w}_g}(\mathbf{x}'))\right) \tag{4}$$

$$+ (1 - \hat{y}_i) \log \left(1 - \zeta(\tilde{g}_{\mathbf{w}_g}(\mathbf{x}), \tilde{g}_{\mathbf{w}_g}(\mathbf{x}'))\right) \Big).$$

To summarize, the Siamese network-based architecture solves the binary classification problem, but does so by learning an intermediate representation, which we identify as $g(\mathbf{x})$. However, the current architecture only outputs a point estimate for $\tilde{g}_{\mathbf{w}_g}(\mathbf{x})$, which is unsatisfying in our scenario where we wish to characterize uncertainties. To do so, we resort to Bayesian inference.

Bayesian inference To characterize the posterior distribution $p(\mathbf{w}_g|\mathcal{D}_g) \propto p(\mathcal{D}_g|\mathbf{w}_g)p(\mathbf{w}_g)$, the network weights $\mathbf{w}_g$ are equipped with a prior distribution $p(\mathbf{w}_g)$. The posterior is then in turn used to compute the predictive posterior distribution of $\tilde{g}_{\mathbf{w}_g}(\mathbf{x})$. We resort to variational inference to characterize the posterior distribution. Variational inference aims at finding the closest approximation in terms of Kullback-Leibler divergence to $p(\mathbf{w}_g|\mathcal{D}_g)$, among a chosen family of distributions parameterized by $\boldsymbol{\theta}_g$. Let us denote this approximation by $q(\mathbf{w}_g|\boldsymbol{\theta}_g)$ (the so-called variational posterior). It can easily be shown that this amounts to minimizing the following expression w.r.t. $\boldsymbol{\theta}_g$:

$$\mathcal{L}_g(\boldsymbol{\theta}_g) = \mathrm{KL}[q(\mathbf{w}_g|\boldsymbol{\theta}_g)||p(\mathbf{w}_g)] \tag{5}$$
$$- \mathbb{E}_{q(\mathbf{w}_g|\boldsymbol{\theta}_g)}[\log p(\mathcal{D}_g|\mathbf{w}_g)],$$

where the term $\log p(\mathcal{D}_g|\mathbf{w}_g)$ is given by (4). This expression is called the negative ELBO (evidence lower bound). The loss in (5) and its gradient are intractable, but we use Bayes by backprop (BBB) (Blundell et al., 2015) as our practical implementation. It provides Monte Carlo estimators of the loss and gradients, and ensures that back-propagation works. The minimization is then simply carried out by gradient descent. We refer the reader to the original paper for details.

2.2. Active Data Acquisition

Humans are not passive sources of information, and can only answer a certain amount of queries before growing tired or impatient. In this limited budget setting, in order to maximize the use of the expert's time, we propose to resort to active learning to learn an accurate model in a sample-efficient way. Here, we propose to use the so-called BALD (Bayesian Active Learning by Disagreement, Houlsby et al. (2011); Gal et al. (2017)), a criterion justified from an information-theoretic perspective. BALD selects the point from a pool $\mathcal{D}_{pool}$ which maximizes the mutual information between its observation and model parameters. Adapting BALD to our setting, we write

$$[\mathbf{x}, \mathbf{x}']^\star = \underset{[\mathbf{x},\mathbf{x}'] \in \mathcal{D}_{pool}}{\arg\max} \; \mathrm{I}(\hat{y}; \mathbf{w}_g|[\mathbf{x}, \mathbf{x}'], \mathcal{D}_g). \tag{6}$$

We approximate the mutual information as follows:

$$\mathrm{I}(\hat{y}, \mathbf{w}_g|[\mathbf{x}, \mathbf{x}'], \mathcal{D}_g)$$
$$= \mathcal{H}[\hat{y}|[\mathbf{x}, \mathbf{x}'], \mathcal{D}_g] - \mathbb{E}_{p(\mathbf{w}|\mathcal{D}_g)}[\mathcal{H}[\hat{y}|[\mathbf{x}, \mathbf{x}'], \mathbf{w}_g]], \tag{7}$$
$$\simeq \mathcal{H}[\hat{y}|[\mathbf{x}, \mathbf{x}'], \mathcal{D}_g] - \mathbb{E}_{q(\mathbf{w}_g|\boldsymbol{\theta}_g)}[\mathcal{H}[\hat{y}|[\mathbf{x}, \mathbf{x}'], \mathbf{w}_g]], \tag{8}$$
$$\simeq h\left(\frac{1}{T}\sum_{t=1}^{T} \hat{p}_{\mathbf{w}_g^{(t)}}(\mathbf{x}, \mathbf{x}')\right) - \frac{1}{T}\sum_{t=1}^{T} h\left(\hat{p}_{\mathbf{w}_g^{(t)}}(\mathbf{x}, \mathbf{x}')\right), \tag{9}$$

where $\mathcal{H}$ denotes the differential entropy, and the notation

$$\hat{p}_{\mathbf{w}_g}(\mathbf{x}, \mathbf{x}') = \zeta(\tilde{g}_{\mathbf{w}_g}(\mathbf{x}), \tilde{g}_{\mathbf{w}_g}(\mathbf{x}')) \tag{10}$$

denotes the predicted probability that $\hat{y} = 1$ given the pair $[\mathbf{x}, \mathbf{x}']$ and parameters $\mathbf{w}_g$ (i.e., the output of the network with parameters $\mathbf{w}_g$), and where $h(p) = -p\log(p) - (1-p)\log(1-p)$ denotes the binary entropy function. The approximation in (8) comes from swapping the true posterior distribution $p(\mathbf{w}_g|\mathcal{D}_g)$ with the variational posterior $q(\mathbf{w}_g|\boldsymbol{\theta}_g)$, and the approximation in (9) corresponds to Monte Carlo approximations given that the $\mathbf{w}_g^{(t)}$ are i.i.d. samples from $q(\mathbf{w}_g|\boldsymbol{\theta}_g)$. We will refer to this criterion as PBALD.

3. Expert Knowledge-Augmented Bayesian Optimization

We now tackle the challenge of transferring what was learned in the previous step for the BO task. To that end, we propose to plug the previously described PBNN architecture into a wider multi-task learning (MTL) one. The MTL architecture aims at building probabilistic surrogates for both f and g, by sharing the weights of the hidden layers, and having separate weights for the last layer. Indeed, we leverage the similarity between the functions f and g by sharing some of the latent representations produced by the network. The architecture is detailed in Section 3.2. As we are eliciting the knowledge of the expert in a first step, this will have the effect of providing the surrogate model for f with a good initialization, which in turn will lead to accelerating the task of optimizing f. For this second step, we sequentially update this surrogate by collecting a dataset $\mathcal{D}_f = \{(\mathbf{x}_j, f(\mathbf{x}_j))\}_{j=1}^J$. Those points are selected using the expected imporvement acquisition function, a standard BO acquisition function recalled in Appendix A.1.

3.1. Surrogate model for f

The probabilistic surrogate we use for f is a Bayesian neural network. Let us denote by $\mathbf{w}_f$ its weights, with prior distribution $p(\mathbf{w}_f)$. We further denote by $\tilde{f}_{\mathbf{w}_f}(\mathbf{x})$ the output obtained by forwarding $\mathbf{x}$. Similarly, we resort to variational inference to characterize the posterior distribution $p(\mathbf{w}_f|\mathcal{D}_f)$, i.e., we aim at minimizing the following expression w.r.t. variational parameters $\boldsymbol{\theta}_f$:

$$\begin{aligned}\mathcal{L}_f(\boldsymbol{\theta}_f) = \; & \mathrm{KL}[q(\mathbf{w}_f|\boldsymbol{\theta}_f)||p(\mathbf{w}_f)] \\ & - \mathbb{E}_{q(\mathbf{w}_f|\boldsymbol{\theta}_f)}[\log p(\mathcal{D}_f|\mathbf{w}_f)],\end{aligned} \tag{11}$$

where $q(\mathbf{w}_f|\boldsymbol{\theta}_f)$ denotes the variational posterior parameterized by $\boldsymbol{\theta}_f$. Note that the log-likelihood term $p(\mathcal{D}_f|\mathbf{w}_f)$ is here Gaussian, which corresponds to the mean squared error.

A straightforward way of transferring what was learned in the first step would be to use the posterior distribution of the weights from the trained PBNN as the prior for $\mathbf{w}_f$. However, since PBNN does not learn the actual scale of f, using it to provide the prior distribution of the weights will not help at all, and may even lead to catastrophic forgetting (French, 1999), where the shape information encoded in the posterior distribution of weights is erased during the training with $\mathcal{D}_f$.

To alleviate this problem, we consider a MTL architecture, with hard parameter sharing among the hidden layers for the surrogates of f and g. In other words, we consider a joint model, whose shared parameters are going to be initialized through the trained PBNN. This is detailed next.

3.2. Multi-task learning

We adopt hard parameter sharing among the weights of the hidden layers for the surrogates of f and g. Let us split the weights $\mathbf{w}_g$ of PBNN into $\mathbf{w}_h$ and $\boldsymbol{\beta}_g$, where $\mathbf{w}_h$ are the weights of all hidden layers and $\boldsymbol{\beta}_g$ the weights of the output layer. That is, we can write $\tilde{g}(\mathbf{x}) = \boldsymbol{\beta}_g^T \phi_{\mathbf{w}_h}(\mathbf{x})$, where $\phi_{\mathbf{w}_h}(\mathbf{x})$ represents the feature vector which is produced by forwarding $\mathbf{x}$ through all the hidden layers. The weights $\mathbf{w}_h$ are going to be shared for both surrogates, i.e., the BNN surrogate of f is parameterized by $\mathbf{w}_h$ and $\boldsymbol{\beta}_f$, such that $\tilde{f}(\mathbf{x}) = \boldsymbol{\beta}_f^T \phi_{\mathbf{w}_h}(\mathbf{x})$ is the predicted outcome.

As such, the shared representation $\phi_{\mathbf{w}_h}(\mathbf{x})$ will encode common features, such as the shape information that we wish to transfer to the surrogate of f. While expert knowledge may be biased, $\boldsymbol{\beta}_f$ can be interpreted as a calibrator to lead the surrogate of f to its actual scale and also rectify the potentially inaccurate information provided by the expert. By using the joint model, we will need fewer queries for Bayesian optimization, i.e., we save potentially expensive simulation costs. The full architecture of the MTL system is presented in Figure 2. Details regarding the combination of the losses $\mathcal{L}_g$ and $\mathcal{L}_f$ can be found in Appendix A.2. Lastly, the full algorithm corresponding to our method is detailed in Appendix A.3.

4. Experiments

In this section, we first evaluate the performance of our proposed PBNN in knowledge elicitation. We then analyze the performance of expert knowledge-augmented BO across various objective functions. Lastly, we conduct a further experiment involving actual humans on simulated data. The results of all additional experiments can be found in Appendix B.

Table 1. Accuracy of preference prediction after M acquisitions is significantly better than the earlier GP-based method on three datasets. The accuracy score corresponds to the proportion of correct binary preference prediction on a hold-out test set comprising 1000 pairs. The mean and standard deviation of this score are reported over 20 replications.

		Accuracy (%)	
DATASET	M	GP	PBNN
Machine CPU (6D)	50	67.29 ± 2.91	$\mathbf{81.07 \pm 3.74}$
	100	69.45 ± 1.82	$\mathbf{84.38 \pm 1.52}$
Boston Housing (13D)	50	67.41 ± 2.35	$\mathbf{83.40 \pm 2.14}$
	100	69.33 ± 2.50	$\mathbf{85.89 \pm 2.52}$
Pyrimidine (27D)	50	70.99 ± 2.35	$\mathbf{79.63 \pm 2.14}$
	100	79.29 ± 2.50	$\mathbf{86.59 \pm 2.52}$

4.1. Knowledge elicitation performance of PBNN

Toy example We first present a toy example using 1-dimensional benchmark functions to illustrate how the proposed PBNN architecture can learn the shape of the function g through pairwise comparisons. The model is trained by sequentially selecting 10, 20 and 50 pairwise comparisons using the PBALD criterion and getting the associated preference labels. We assume noiseless feedback in this experiment for illustration purposes. Figure 3 displays the comparison between the real function values g and elicited expert model predictions $\tilde{g}$, with the Forrester and Styblinski-Tang functions. From the figure, it can been seen that with 50 pairwise comparisons, the expert model $\tilde{g}$ can capture the ordinal information, i.e., g up to a monotonic constant, as previously explained.

Performance comparison We compare the performance of PBNN with the classical GP-based preference learning model by (Chu & Ghahramani, 2005) on three different datasets. We artificially transform three regression datasets into preference datasets by creating preference labels between all possible pairs of covariates using the target values. For the GP-based model, we use the squared exponential kernel with the hyperparameters optimized by maximum marginal likelihood. The details of the PBNN architecture are shown in Appendix.

A total of M pairs are sequentially queried by maximizing the BALD criterion (Houlsby et al., 2011) for the GP-based model, and the PBALD criterion for PBNN, respectively. After the active learning phase, the accuracy of the model is assessed by computing a binary accuracy score on a hold-out test set consisting of 1000 pairs. The results on the three datasets, averaged over 20 replications, are presented in Table 1 for $M = 50$ and $M = 100$. In all scenarios, PBNN achieved better accuracy results w.r.t. the GP-based model. We further compare the runtimes of the two methods in the Appendix, on all three datasets, PBNN is roughly 20 times faster than the GP-based baseline.

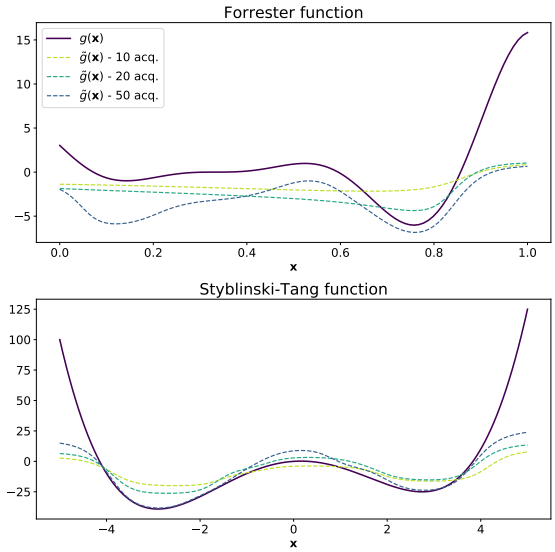

Figure 3. Toy example illustrating how the proposed PBNN architecture can learn the shape of a function using pairwise comparisons. Experiments carried out on the Forrester (top) and Styblinski-Tang (bottom) functions. The thickest, dark blue line represents the true function g, while the dotted lines represent the predicted functions $\tilde{g}$ learned by PBNN using 10, 20 and 50 pairwise comparisons.

4.2. Performance of Bayesian optimization

We first study the performance of the proposed knowledge-augmented Bayesian optimization scheme in a simulated setting where we can control the bias of an expert. More precisely, we compare how well the scheme performs w.r.t. to standard Bayesian optimization on several benchmark functions from the literature.

We assume an expert with potentially biased beliefs of the true function f, with the bias expressed as a perturbation function δ:

$$g(x) = f(x) + \delta(x), \tag{12}$$

where δ is a zero-mean Gaussian process draw with kernel $\sigma_\delta^2 k(x, x')$, which encodes the form of the expert's bias. Note that this does not reduce generality, assuming a general enough perturbation family. In the experiments reported below, we study the effect of expert's bias on the performance by choosing the SE kernel with lengthscale $\ell = 0.1$ and varying σ_δ^2 so that we obtain five levels of expert knowledge accuracy from 50% up to 90%. The visual illustration of the simulated experts is shown in Appendix A.4.

For the MTL structure, the shared hidden layers have width [100, 30, 15]. Standard BO is run using the BNN surrogate described in Section 3.1, in other words, it is the exact same architecture as the "BO branch" of the MTL, for fair comparison. Experiments were run with $M = 100$, $J = 50$ and $\alpha = 0.95$. The results, detailed in the next paragraph,

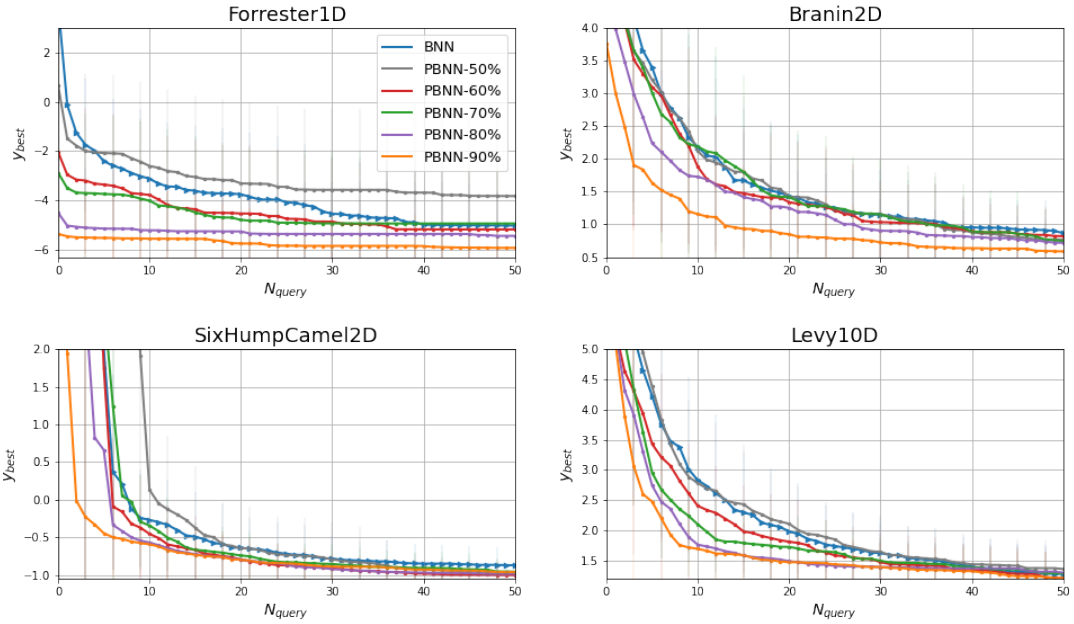

Figure 4. Comparison of the optimization performance of the expert knowledge-augmented BO using 4 benchmark functions w.r.t. standard BO. We simulate 5 experts with different levels of knowledge (denoted `PBNN-xx%`), where the percentage stands for the accuracy of the expert's preferential feedback, i.e., 50% means that the expert is simulated with fully biased knowledge. The knowledge of the simulated expert is elicited using $M = 100$ pairwise comparisons. The standard BO scheme, i.e., without expert knowledge, is denoted `BNN`. The results are averaged over 50 simulations.

are averaged over 50 replications of the experiment. The full description of experimental settings is in Appendix D.

Figure 4 shows the results on four benchmark functions[1]: "Forrester1D", "Six-hump-camel2D", "Branin2D" and "Levy10D". The results are evaluated by y_{best}, which is the current minimal value of the true objective function predicted by the surrogate of f. We can see that the more accurate the simulated expert is, the more pronounced the acceleration effect. If the expert is reliable enough, we can speed up BO significantly. When the expert does not have any knowledge, i.e. 50% preference accuracy, this actually leads to performance deterioration w.r.t. standard BO, which meets our expectation. For the all expert accuracy levels $\geq 60\%$, the final round BO performance ($J = 50$) is at least as competitive as the standard BO. However, the gain is much more striking in the early stages of the BO ($J \ll 50$). This phenomenon may be due to the challenge of making use of inaccurate expert knowledge as more accurate ground-truth data becomes increasingly available.

[1]https://www.sfu.ca/ ssurjano/optimization.html

5. Conclusion

In this paper, we tackled the incorporation of human expert knowledge into BO with the goal of speeding up the optimization. Our procedure breaks down into two steps. The first is to elicit the expert beliefs by querying them with pairwise comparisons. By doing so, we obtain the approximate shape of the objective function. The second step is to share the expert knowledge with the BO, to provide auxiliary information about the potential location of the optimum.

More precisely, we proposed PBNN, a novel preference learning architecture based on Siamese networks to efficiently elicit the expert knowledge. By sequentially querying the preferences between two objects with active learning, the proposed PBNN is more powerful in capturing the latent preference relationships compared with the former GP-based model on different datasets. To conduct the knowledge transfer, we design a well-aligned multi-task learning structure with a knowledge sharing scheme to combine our expert model with BO surrogate. Experiments on different benchmark functions and real data show that when the expert is trustworthy, we can gain significant benefit from the elicited knowledge and markedly speed up the optimization.

References

Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision (ECCV)*, pp. 850–865, 2016.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural network. In *International Conference on Machine Learning (ICML)*, pp. 1613–1622, 2015.

Brochu, E., Freitas, N. D., and Ghosh, A. Active preference learning with discrete choice data. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 409–416, 2008.

Brochu, E., Cora, V. M., and De Freitas, N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems (NIPS)*, 1993.

Caruana, R. Multitask learning. *Machine Learning*, 28(1): 41–75, 1997.

Chu, W. and Ghahramani, Z. Preference learning with gaussian processes. In *International Conference on Machine Learning (ICML)*, pp. 137–144, 2005.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Gal, Y., Islam, R., and Ghahramani, Z. Deep Bayesian active learning with image data. In *International Conference on Machine Learning (ICML)*, pp. 1183–1192, 2017.

González, J., Dai, Z., Damianou, A., and Lawrence, N. D. Preferential Bayesian Optimization. In *International Conference on Machine Learning (ICML)*, pp. 1282–1291, 2017.

Hase, F., Roch, L. M., Kreisbeck, C., and Aspuru-Guzik, A. Phoenics: a Bayesian optimizer for chemistry. *ACS central science*, 4(9):1134–1145, 2018.

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Hvarfner, C., Stoll, D., Souza, A., Nardi, L., Lindauer, M., and Hutter, F. πBO: Augmenting Acquisition Functions with User Beliefs for Bayesian Optimization. In *International Conference on Learning Representations (ICLR)*, 2022.

Jones, D. R., Schonlau, M., and Welch, W. J. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13:455–492, 1998.

Kim, S., Lu, P. Y., Loh, C., Smith, J., Snoek, J., and Soljačić, M. Scalable and flexible deep bayesian optimization with auxiliary information for scientific problems. *arXiv preprint arXiv:2104.11667*, 2021.

Koch, G., Zemel, R., Salakhutdinov, R., et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, 2015.

Li, C., Gupta, S., Rana, S., Nguyen, V., Robles-Kelly, A., and Venkatesh, S. Incorporating expert prior knowledge into experimental design via posterior sampling. *arXiv preprint arXiv:2002.11256*, 2020.

Mikkola, P., Todorović, M., Järvi, J., Rinke, P., and Kaski, S. Projective preferential bayesian optimization. In *International Conference on Machine Learning*, pp. 6884–6892, 2020.

Millet, I. The effectiveness of alternative preference elicitation methods in the analytic hierarchy process. *Journal of Multi-Criteria Decision Analysis*, 6(1):41–51, 1997.

Rader, T. The existence of a utility function to represent preferences. *The Review of Economic Studies*, 30(3): 229–232, 1963.

Ramachandran, A., Gupta, S., Rana, S., Li, C., and Venkatesh, S. Incorporating expert prior in bayesian optimisation via space warping. *Knowledge-Based Systems*, 195:105663, 2020.

Shah, N. B., Balakrishnan, S., Bradley, J., Parekh, A., Ramchandran, K., and Wainwright, M. When is it better to compare than to score? *arXiv preprint arXiv:1406.6618*, 2014.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing systems (NIPS)*, 2012.

Souza, A., Nardi, L., Oliveira, L. B., Olukotun, K., Lindauer, M., and Hutter, F. Bayesian optimization with a prior for the optimum. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 265–296, 2021.

Zhang, Y., Apley, D. W., and Chen, W. Bayesian optimization for materials design with mixed quantitative and qualitative variables. *Scientific reports*, 10(1):1–13, 2020.

Supplementary Materials

A. Additional details

A.1. Acquisition function for BO

We adopt the expected improvement (EI) as the acquisition function (Jones et al., 1998). Given $\mu_{\mathbf{x}}$ the predictive mean of BO surrogate model and $s^2_{\mathbf{x}}$ the predictive variance, the EI at point $\mathbf{x}$ is defined as:

$$\alpha_{\text{EI}}(\mathbf{x}) = s_{\mathbf{x}}[\gamma(\mathbf{x})\Phi(\gamma(\mathbf{x})) + \psi(\gamma(\mathbf{x}))], \tag{13}$$

where $\gamma(\mathbf{x}) = (y_{best} - \mu_{\mathbf{x}})/s_{\mathbf{x}}$, y_{best} is the current lowest value of the objective function, and $\Phi(\cdot)$ and $\psi(\cdot)$ are the cumulative distribution function and probability density function of a standard normal random variable. Since (13) is intractable for a BNN as there is no analytical form of the output distribution, we use Monte Carlo sampling to obtain the approximate EI (Kim et al., 2021):

$$\alpha_{\text{EI}}(\mathbf{x}) \approx \frac{1}{T}\sum_{t=1}^{T} \max\left(y_{best} - \tilde{f}^{(t)}(\mathbf{x}), 0\right), \tag{14}$$

where the $\tilde{f}^{(t)}(\mathbf{x})$ are i.i.d. predictive samples at $\mathbf{x}$.

A.2. Combining the losses

One remaining question is how to combine the loss functions $\mathcal{L}_g$ (5) and $\mathcal{L}_f$ (11). In order to put more emphasis on the actual acquisitions of f over time, we propose a weighted scheme with exponential decay for the $\mathcal{L}_g$. After the j-th BO acquisition, the loss $\mathcal{L}_j$ is such that

$$\mathcal{L}_j = \frac{\alpha^{j-1}}{\alpha^{j-1}+1}\mathcal{L}_g + \frac{1}{\alpha^{j-1}+1}\mathcal{L}_f, \tag{15}$$

with $\alpha < 1$ a hyperparameter to control the speed of the decay.

A.3. Algorithm

Our two-step, expert knowledge-augmented BO procedure (knowledge elicitation first, BO second) is summed up in Algorithm 1.

A.4. Simulated experts

Figure 5 illustrates the comparison between the simulated experts with the ground truth objective function with the Forrester function, for the 5 considered accuracy levels.

B. Additional results of the experiments

B.1. Computational cost analysis

We compare the runtimes of PBNN with the GP-based preference learning model by (Chu & Ghahramani, 2005). The inference of PBNN is typically run GPU-based architectures, however, for a fair comparison, we compare 20 runs of each method on the three datasets using $M = 50$ using the same CPU architecture[2]. The results are reported in Table 2. On all three datasets, the proposed PBNN is roughly 20 times faster than the GP-based baseline.

B.2. Experiment with human experts

We further study the performance of the proposed method in a real-world setting with actual human experts. To that end, we conduct a simple user experiment involving memory abilities. Similarly to the previous experiment, the goal is to optimize BO benchmark functions, this time 2D functions. To induce controlled knowledge about those functions, we let users memorize the shape of the objective function by displaying 3D-plots for a short time. This provides useful but

[2] 2x20 core Xeon Gold 6248 2.50GHz, 192GB RAM.

Algorithm 1 Expert knowledge-augmented BO

Input: Active learning budget M, BO acquisition budget J
Output: Minimum of f

// Start Knowledge Elicitation
Initialize the expert model $\tilde{g}$ using PBNN with a random pair, $\mathcal{D}_g = \{(\mathbf{x}_0, \mathbf{x}'_0, \hat{y}_0)\}$.
for $i = 1$ to M **do**
  $[\mathbf{x}_i, \mathbf{x}'_i] = \underset{[\mathbf{x},\mathbf{x}'] \in \mathcal{D}_{pool}}{\arg\max}\ \mathrm{I}(\hat{y}; \mathbf{w}_g | [\mathbf{x}, \mathbf{x}'], \mathcal{D}_g)$ ((9))
  Query the expert to obtain $\hat{y}_i$ associated to $[\mathbf{x}_i, \mathbf{x}'_i]$
  $\mathcal{D}_g \leftarrow \mathcal{D}_g \cup (\mathbf{x}_i, \mathbf{x}'_i, \hat{y}_i)$
  Update variational parameters $\boldsymbol{\theta}_g$ by minimizing (5)
end for
// Start Bayesian Optimization
$\mathcal{D}_f = \emptyset$
$y_{best} = \infty$
for $j = 1$ to J **do**
  $\check{\mathbf{x}}_j = \underset{\mathbf{x} \in \mathcal{X}}{\arg\max}\ \alpha_{\mathrm{EI}}(\mathbf{x})$
  Evaluate $f(\check{\mathbf{x}}_j)$
  $\mathcal{D}_f \leftarrow \mathcal{D}_f \cup (\check{\mathbf{x}}_j, f(\check{\mathbf{x}}_j))$
  Update variational parameters $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_f$ by minimizing the combined loss (15)
  $y_{best} \leftarrow \min(y_{best}, f(\underset{\mathbf{x} \in \mathcal{X}}{\arg\min}\ \tilde{f}(\mathbf{x})))$
end for
return y_{best}

Table 2. Average runtimes (in seconds) after $M = 50$ acquisitions on three datasets. The runtime of the proposed PBNN is roughly 20 times faster than the GP-based method. The comparison was carried out with 20 runs on the same CPU architecture. The standard deviation is also reported.

		Runtimes (sec.)	
DATASET	M	GP	PBNN
Machine CPU (6D)	50	899 ± 68	$\mathbf{40 \pm 4}$
Boston Housing (13D)	50	981 ± 72	$\mathbf{50 \pm 7}$
Pyrimidine (27D)	50	1003 ± 64	$\mathbf{57 \pm 9}$

biased preference information for optimization. Based on their memory of the function, the user must then answer a series of questions asked in preferential form, in asked in the format "*At which point do you think the value of the function is larger?*". The questions are determined by the PBALD criterion, detailed in Section 2.2. Finally, BO augmented with expert knowledge is then run with the proposed methodology. We compare this approach with standard BO. The intuition behind this experiment is that users cannot memorize the function in all its complexity, but still can grasp an understanding its overall shape, which could speed up the optimization.

We choose three commonly used 2D benchmark functions: Six-hump-camel2D, Three-hump-camel2D and Branin2D. This choice is motivated by the fact that these particular functions have several local minima, but are still smooth enough so that users can remember their general shape in a short time. The plots are displayed for 2 minutes, which is enough time to remember the general shape of the function, but not learn perfect information, thus mimicking expert knowledge on complex problems. The number of preferential questions is set to 25, a number not too small to effectively build the expert model nor too large to bore the users. The coordinates of the points selected by each question, as well as their locations in the coordinate system used for the visualization of the function, are provided to the user. Other settings remain the same as the experiments of Section 4.2. We also provide a brief instruction manual for the users (see Appendix C). The experiment is carried out following an existing code of conduct about user studies. The users are recruited from a student population that have no previous knowledge of the test functions.

Table 3 reports the accuracy of eight different users on the three selected objective functions. All the accuracy rates are greater than 50%, and the average accuracy is 70% or higher for each function. Figure 6 shows the comparison between

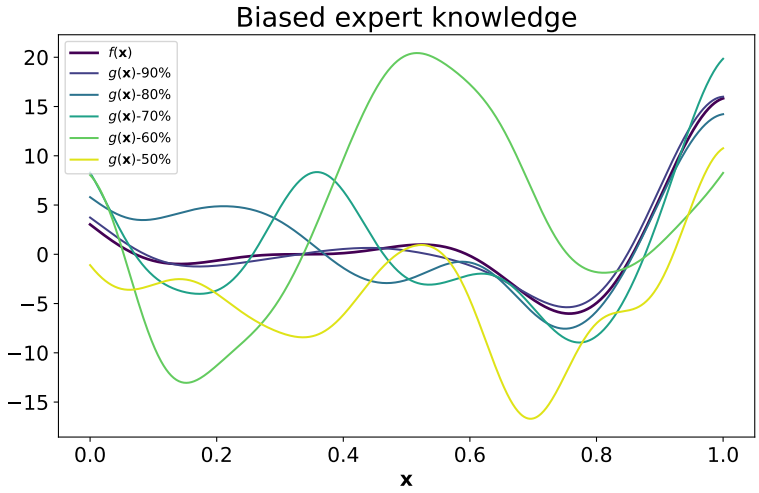

Figure 5. Illustration of the simulated expert's beliefs using the Forrester1D. The true function is the thickest, dark blue curve, and the other curves correspond to that function perturbed with a random GP draw with various variances. The variances are chosen such that the accuracy of the expert ranges from 50% to 90 %.

Table 3. User accuracy for the three BO benchmark functions, i.e., the percentage of questions each user answered correctly (out of 25).

	Accuracy								
Function (2D)	User1	User2	User3	User4	User5	User6	User7	User8	**Avg.**
3H-camel	64%	84%	72%	80%	52%	80%	72%	68%	**71.5%**
6H-camel	52%	68%	76%	88%	64%	84%	72%	56%	**70%**
Branin	80%	72%	72%	80%	68%	84%	80%	76%	**76.5%**

standard BO and our method in terms of optimization performance. Each simulation builds the expert model using PBNN with the same dataset obtained from each user, but with different network initialization. We run 10 simulations to account for the randomness. As can be seen on those plots, the help of experts leads to prominent acceleration compared with standard BO, which again proves the effectiveness of our expert knowledge-augmented BO method.

B.3. Experiments with different elicitation budgets

We further investigate the performance of our expert-augmented BO with different elicitation budgets. We use the same objective functions as in Section 4.2 and simulate four different levels of the experts. We use the same configurations as in the previous experiments, the details can be found in Section D.2.

The results are shown in Figures 7 8, 9, 10. The overall performance behaves as expected. With a larger elicitation budget, the acceleration of BO is more obvious. We notice that the performance is even worse under a very limited budget than standard BO, i.e., $M = 10$. We guess the reason behind this situation is that the insufficient preference training data makes the network prone to overfitting, hence misguiding the training of the surrogate model during the MTL stage. Moreover, in some figures, we can see the performance between $M = 50$ and $M = 100$ is very close, which implies that there is no need to over-query for the expert under some relatively easy-to-optimize functions since the expert knowledge will then dominate the actual BO regression data and slow down the optimization. In this case, we should consider lowering the value of α (Equation 15).

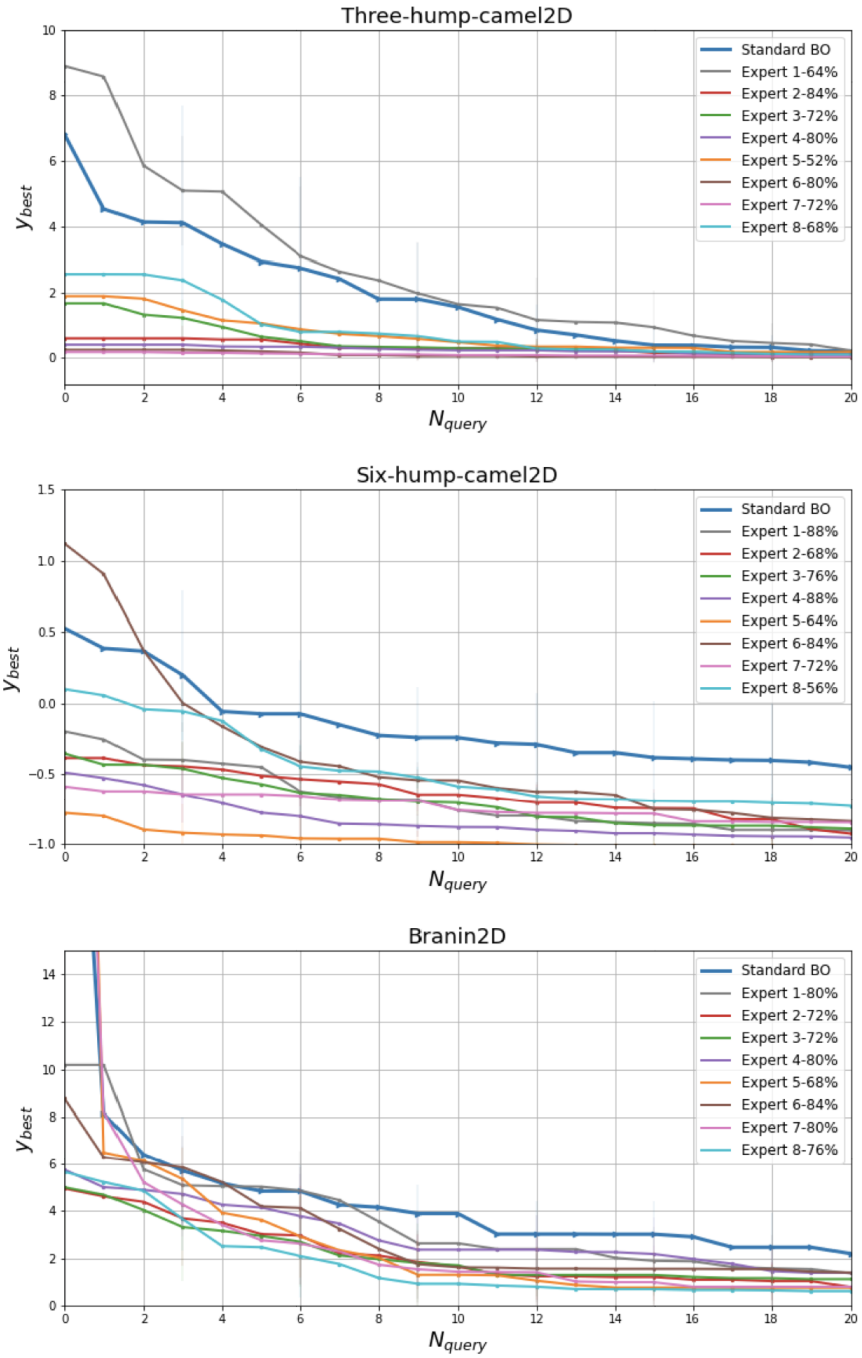

Figure 6. Comparison of the optimization performance of the expert knowledge-augmented BO with real users on 3 2D benchmark functions w.r.t. standard BO. We collect the data from 8 users, where the percentage stands for the accuracy of the expert's preferential feedback. The knowledge of the simulated expert is elicited using M = 25 pairwise comparisons. The results are averaged over 10 simulations.

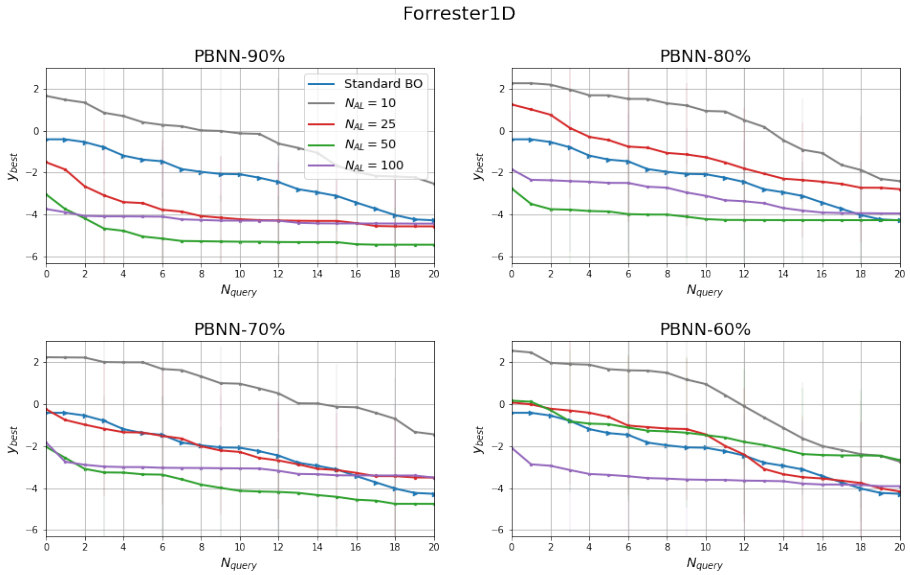

Figure 7. A BO comparison on "Forrester1D" function with different knowledge elicitation budget. We simulate 4 experts with different levels of knowledge, in each subplot we use the same level of the expert. The results are averaged over 20 simulations.

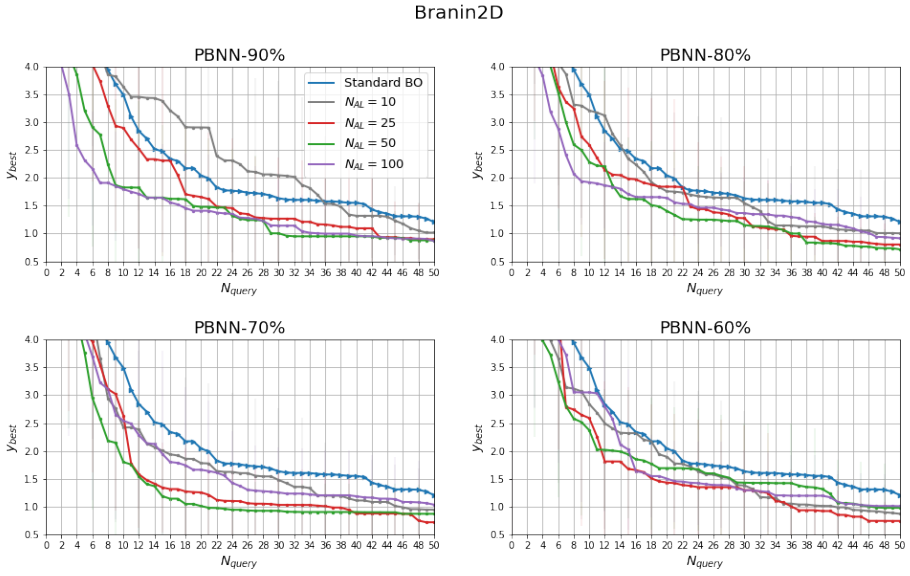

Figure 8. A BO comparison on "Branin2D" function with different knowledge elicitation budget. We simulate 4 experts with different levels of knowledge, in each subplot we use the same level of the expert. The results are averaged over 20 simulations.

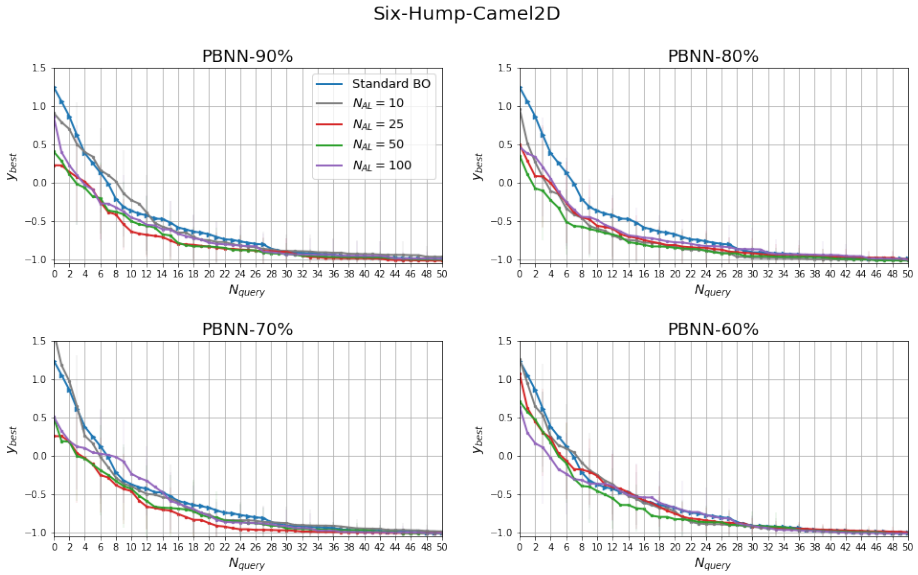

Figure 9. A BO comparison on "Six-Hump-Camel2D" function with different knowledge elicitation budget. We simulate 4 experts with different levels of knowledge, in each subplot we use the same level of the expert. The results are averaged over 20 simulations.

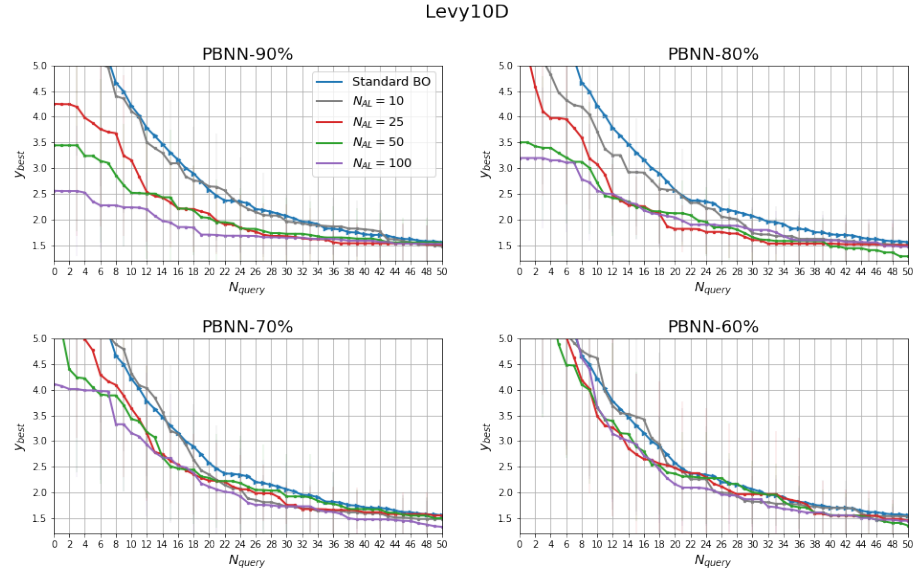

Figure 10. A BO comparison on "Levy10D" function with different knowledge elicitation budget. We simulate 4 experts with different levels of knowledge, in each subplot we use the same level of the expert. The results are averaged over 20 simulations.

C. User Manual

Introduction Welcome to the test. During this experiment, you need to try your best to remember the shapes of three different 2-D functions in limited time. After that you need to answer 25 simple questions, by telling which point do you think is larger between a pair of points.

In this test, we rely on the existing code of conduct for conducting user studies in our field. The experimental data is only used for this paper, and we will not disclose any of your private information.

Experimental details Three experiments will be conducted in random order. When each experiment starts, you will be shown a 3-D plot and a 2-D heat map of the function (demo plots are shown in Figure 11), and you can drag the 3-D plot to have a better visualization. You will have 2 minutes to remember the plots, once the time is up, you will no longer be able to view these plots.

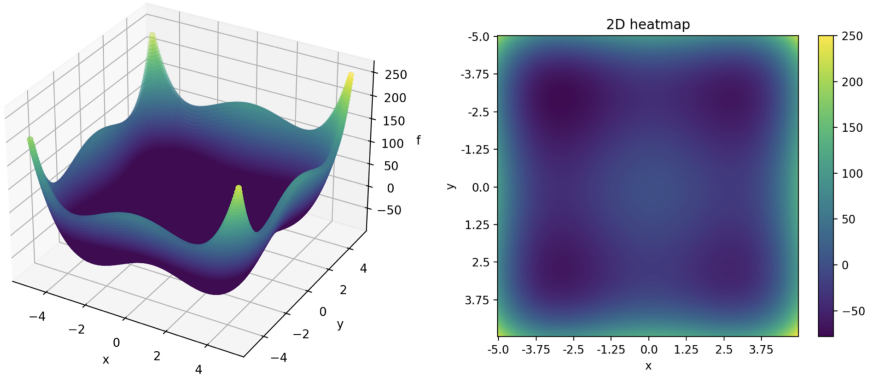

Figure 11. The left plot is the 3-D view of the objective function, and the right one is the 2-D heat map. These plots are only for demonstration, the real objective functions in the experiment will not be shown here.

After that, you will be asked 25 questions. Each question is asked in the format "At which point do you think the value of the function is larger?" And there will be no time limit for you to answer these questions. We will provide the coordinates of the two points to you and also plot their locations in the coordinate system used in the visualization of the function. The demo plot is shown in Figure 12.

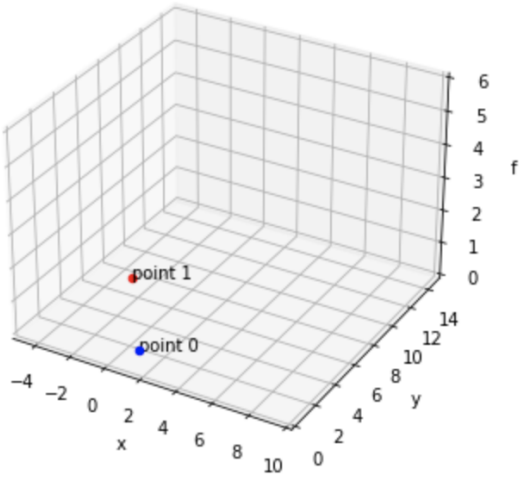

Figure 12. The plot of two points in the question stage.

After answering all the questions, you will directly jump to the next experiment. Upon finish the three experiments, the system will calculate the accuracy of your performance, and you can also view your own user model in 3D plot (see Figure 13 for reference).

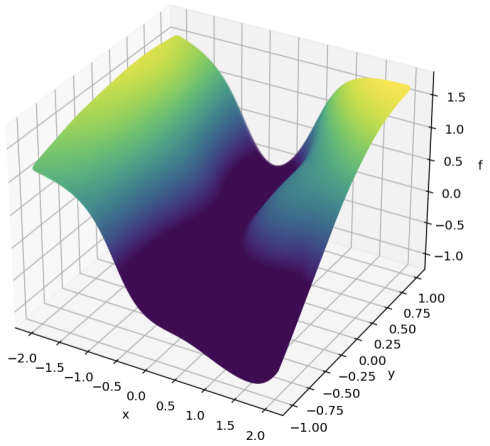

Figure 13. An example of the user model

D. Experimental settings

D.1. Elicitation experiment

DATASETS

- Machine CPU: A computer hardware dataset. The dimension is 6 and has 209 instances
- Boston housing: This dataset contains information concerning housing in the area of Boston Mass. The dimension is 13 and has 506 cases
- Pyrimidine: A pyrimidine QSAR dataset. The dimension of this dataset is 27 and has 74 instances

The initial training set contains one random pair. The query pool size for active learning is 2000 pairs, and the test set used for evaluating accuracy consists of 1000 pairs. The dataset is shuffled in each epoch.

HYPER-PARAMETERS

- Number of active acquisitions in elicitation stage: 50, 100
- Monte Carlo sampling budget in BALD: 100
- Number of simulations: 20

NEURAL NETWORK CONFIGURATIONS

- Framework: PyTorch, torchbnn
- Optimizer: ADAM with learning rate = 0.001
- Scheduler: CosineAnnealingLR with $T_{max} = 20$ and $eta_{min} = 0.0001$
- Batch size: 2
- Number of epochs: 20
- BNN hidden layers: 2 shared layers with weight prior $\mathcal{N}(0, 0.1)$, width [100, 10]
- Activation function: Tanh

D.2. BO with simulated experts

BENCHMARK FUNCTIONS

- Forrester1D: A simple one-dimensional test function, with one global minimum, one local minimum and a zero-gradient inflection point. This function is evaluated on $x \in [0, 1]$. The form of this function is:

$$f(x) = (6x - 2)^2 \sin(12x - 4). \tag{16}$$

- Branin2D: A 2D function with three global minima. We take $a = 1, b = \frac{5.1}{4\pi^2}, c = \frac{5}{\pi}, r = 6, s = 10$ and $t = \frac{1}{8\pi}$. This function is evaluated on the square $x_1 \in [-5, 10], x_2 \in [0, 15]$. The function form is:

$$f(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t)\cos(x_1) + s. \tag{17}$$

- Six-hump-camel2D: A 2D function with six local minima, two of which are global. This function is evaluated on the square $x_1 \in [-3, 3], x_2 \in [-2, 2]$. The function form is:

$$f(x) = (4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2. \tag{18}$$

- Levy10D: A 10D function evaluated on the hypercube $x_i \in [-2, 2]$, for all $i = 1, ..., d$. The function form is:

$$\begin{aligned} f(x) = \ & \sin^2(\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] \\ & + (w_d - 1)^2[1 + \sin^2(2\pi w_d)], \end{aligned} \tag{19}$$

  where $w_i = 1 + \frac{x_i - 1}{4}$, for all $i = 1, ..., d$.

The number of initial training pairs for elicitation is 1. The query pool size for active learning is 2000.

HYPER-PARAMETERS

- Number of active acquisitions in elicitation stage: 100
- Number of BO acquisition: 50
- Monte Carlo sampling budget in BALD: 100
- Monte Carlo sampling budget in EI: 30
- α in MTL shared weight: 0.95
- Number of simulations: 50

NEURAL NETWORK CONFIGURATIONS

- Framework: PyTorch, torchbnn
- Optimizer: ADAM with lr = 0.001 in elicitation stage, lr = 0.01 in BO stage
- Scheduler: CosineAnnealingLR with $T_{max} = 20$ and $eta_{min} = 0.0001$ in elicitation stage, no scheduler in BO.
- Batch size: 10 for preference data, 5 for regression data
- Number of epochs: 100 in elicitation stage, 200 in BO stage
- BNN hidden layers: 3 shared layers with weight prior $\mathcal{N}(0, 0.1)$, width [100, 30, 15]
- Activation function: Tanh

D.3. BO with actual human experts

BENCHMARK FUNCTIONS

- Three-hump-camel2D: This function has three local minima and is evaluated on the square $x_1 \in [-2, 2], x_2 \in [-2, 2]$. The form of this function is:

$$f(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2. \tag{20}$$

- Six-hump-camel2D: A 2D function with six local minima, two of which are global. This function is evaluated on the square $x_1 \in [-2, 2], x_2 \in [-1, 1]$. The function form is:

$$f(x) = (4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2. \tag{21}$$

- Branin2D: A 2D function with three global minima. We take $a = 1, b = \frac{5.1}{4\pi^2}, c = \frac{5}{\pi}, r = 6, s = 10$ and $t = \frac{1}{8\pi}$. This function is evaluated on the square $x_1 \in [-5, 10], x_2 \in [0, 15]$. The function form is:

$$f(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t)\cos(x_1) + s. \tag{22}$$

The number of initial training pairs for elicitation is 1. The query pool size for active learning is 2000.

HYPER-PARAMETERS

- Number of active acquisitions in elicitation stage: 25
- Number of BO acquisition: 20
- Monte Carlo sampling budget in BALD: 100
- Monte Carlo sampling budget in EI: 30
- α in MTL shared weight: 0.95
- Number of simulations: 10

NEURAL NETWORK CONFIGURATIONS

- Framework: PyTorch, torchbnn
- Optimizer: ADAM with lr = 0.001 in elicitation stage, lr = 0.01 in BO stage
- Scheduler: CosineAnnealingLR with $T_{max} = 20$ and $eta_{min} = 0.0001$ in elicitation stage, no scheduler in BO.
- Batch size: 10 for preference data, 5 for regression data
- Number of epochs: 100 in elicitation stage, 200 in BO stage
- BNN hidden layers: 3 shared layers with weight prior $\mathcal{N}(0, 0.1)$, width [100, 30, 15]
- Activation function: Tanh