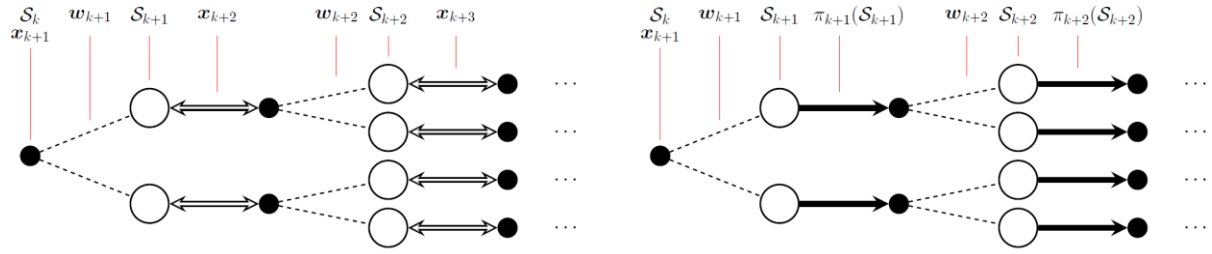
Lookahead Bayesian Optimization with Inequality Constraints

Remi R. Lam, Karen E. Willcox

1. Problem to solve: Optimization problems with expensive constraints.
2. Current methods: Modeling constraints g_i also as GPs and finding the next design using greedy (one-step) optimizations.
   Popular Choice: $EI_c = EI * P(g_i \leq 0, \forall i)$
3. Improvement: Introducing an approximate dynamic programming algorithm.
4. Dynamic Programming and approximation:



Left: Illustration of original dynamic programming algorithm. Evaluating expected long-term rewards requires the optimal policy to choose design x_{k+1}based on dataset S_k and the integration over possible result w_{k+1}…
Right: Illustration of approximate dynamic programming algorithm. Use heuristic policy instead of optimal policy, and limit the steps. Use Gauss–Hermite quadrature for the integration.

5. Algorithm:

Algorithm 2 Rollout Utility Function

Function: `utility`$(\boldsymbol{x}, h, \mathcal{S})$
Construct GPs using $\mathcal{S}$
if $h = 0$ **then**
  $U \leftarrow EI_c(\boldsymbol{x}; \mathcal{S})$
else
  $U \leftarrow EI_c(\boldsymbol{x}; \mathcal{S})$
  Generate N_q Gauss-Hermite quadrature weights $\alpha^{(q)}$ and points $\boldsymbol{w}^{(q)}$ associated with $\boldsymbol{x}$
  for $q = 1$ **to** N_q **do**
    $\mathcal{S}' \leftarrow \mathcal{S} \cup \{(\boldsymbol{x}, \boldsymbol{w}^{(q)})\}$
    if $h > 1$ **then**
      $\boldsymbol{x}' \leftarrow \pi(\mathcal{S}')$ using Eq. 13 $\quad \boldsymbol{x}_{k+1} = \underset{\boldsymbol{x} \in \mathcal{X}}{\operatorname{argmax}}\, EI_c(\boldsymbol{x}; \mathcal{S}_k).$
    else
      $\boldsymbol{x}' \leftarrow \pi(\mathcal{S}')$ using Eq. 14 $\quad \boldsymbol{x}_{\tilde{N}+1} = \underset{\boldsymbol{x} \in \mathcal{X}}{\operatorname{argmin}}\, \overline{\mu}_{\tilde{N}}(\boldsymbol{x}; f) \quad \text{s.t.} \quad PF(\boldsymbol{x}; \mathcal{S}_{\tilde{N}}) \geq 0.99,$
    end if
    $U \leftarrow U + \gamma \alpha^{(q)}$`utility`$(\boldsymbol{x}', h-1, \mathcal{S}')$
  end for
end if
Output: U

I think the last x' is chosen differently because it is to be evaluated at, thus it must be feasible.

6. Utility gap metric:

$$e_n = \begin{cases} |f(\boldsymbol{x}_n^*) - f^*| & \text{if } \boldsymbol{x}_n^* \text{ is feasible,} \\ |\Psi - f^*| & \text{else,} \end{cases}$$
$$\boldsymbol{x}_n^* = \underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmin}}\, \overline{\mu}_n(\boldsymbol{x}; f) \quad \text{s.t.} \quad PF(\boldsymbol{x}; \mathcal{S}_n) \geq 0.975.$$

   Note that x_n^* is not the best design so far. I think this metric evaluates the GP model itself.

7. Result:

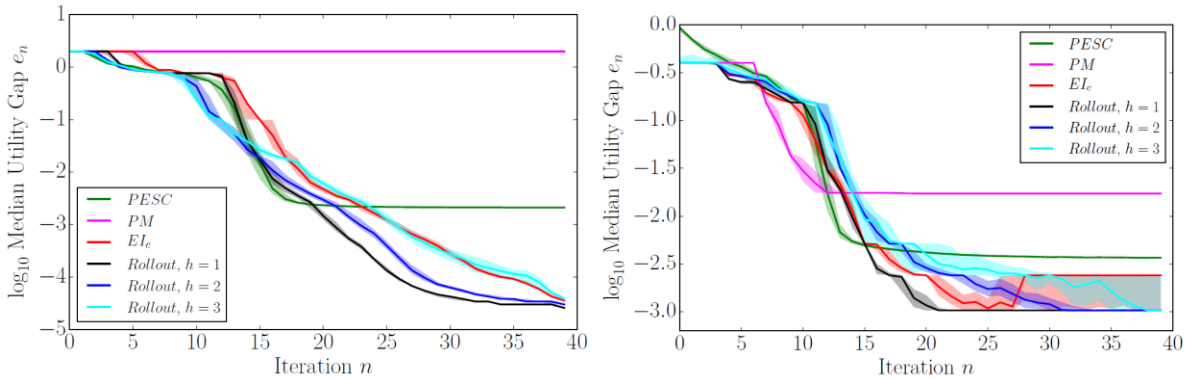


Figure 2: Left: Multi-modal objective and single constraint (P1). Right: Linear objective and multiple non-linear constraints (P2). Shaded region indicates 95% confidence interval of the median statistic.

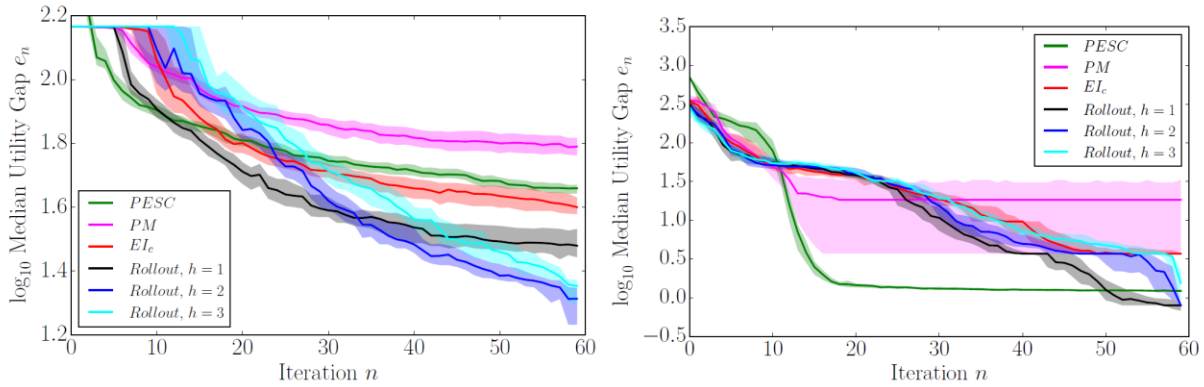


Figure 3: Left: Multi-modal 4-d objective and constraint (P3). Right: Reacting flow problem (P4). The awareness of the remaining budget explains the sharp decrease in the last iterations for the rollout.

8. Discussion: Generally, the method works well. It seems the lookahead steps h is not the larger the better. The author thinks it is because the model is imperfect, and the method relies more on the GP model than greedy methods.

The Parallel Knowledge Gradient Method for Batch Bayesian Optimization

Jian Wu, Peter I. Frazier

1. Problem to solve: Paralleled BO problems.
2. Current methods: There are parallel versions of BO algorithms for different acquisition functions such as EI and UCB. A popular way to parallelize those methods is to construct the batch of points in an iterative greedy fashion, assuming the unevaluated points have already returned some value (choices can be made here) during the optimization for the next point.
3. Improvement: Introducing a new acquisition function, the parallel knowledge gradient (q-KG), and developing a method based on infinitesimal perturbation analysis (IPA) to evaluate q-KG's gradient efficiently.
4. q-KG acquisition function:

$$q\text{-}KG(\boldsymbol{z}^{(1:q)}, \mathbb{A}) = \min_{x\in\mathbb{A}} \boldsymbol{\mu}^{(n)}(x) - \mathbb{E}_n\left[\min_{x\in\mathbb{A}} \boldsymbol{\mu}^{(n+q)}(x) | \boldsymbol{y}(\boldsymbol{z}^{(1:q)})\right]$$

   It's the expected improvement of the minimum of posterior mean function after adding q points.
5. Algorithm:

Algorithm 1 The *q-KG* algorithm

Require: the number of initial stage samples I, and the number of main stage sampling iterations N.

1: Initial Stage: draw I initial samples from a latin hypercube design in $\mathbb{A}$, $\boldsymbol{x}^{(i)}$ for $i = 1, \ldots, I$.
2: Main Stange:
3: **for** $s = 1$ to N **do**
4: Solve (4.2), i.e. get $(z_1^*, z_2^*, \cdots, z_q^*) = \text{argmax}_{\boldsymbol{z}^{(1:q)}\subset\mathbb{A}} q\text{-}KG(\boldsymbol{z}^{(1:q)}, \mathbb{A})$
5: Sample these points $(z_1^*, z_2^*, \cdots, z_q^*)$, re-train the hyperparameters of the GP by MLE, and update the posterior distribution of f.
6: **end for**
7: **return** $x^* = \text{argmin}_{x\in\mathbb{A}} \mu^{(I+Nq)}(x)$.

   Note that the point returned as the optimum may not be a previously sampled point
6. q-KG can be evaluated by taking average of results by Monte Carlo sampling after discretizing the domain A.
7. The gradient of q-KG can also be calculated using infinitesimal perturbation analysis, thus gradient based optimizers can be used to optimize q-KG.
8. Result: In most cases, the q-KG method performs significantly better than other benchmark algorithms, especially in noisy problems.

Bayesian Optimization with Gradients

Jian Wu, Matthias Poloczek, Andrew Gordon Wilson, Peter I. Frazier

1. Extended from previous paper: The Parallel Knowledge Gradient Method for Batch Bayesian Optimization
2. Problem to solve: Paralleled BO problems with gradient available.
3. Current methods: Several groups have studied how to incorporate the gradient information with the GP model. Most of them worked on Expected Improvement.
4. Improvement: Introducing the derivative-enabled knowledge-gradient (d-KG) acquisition function. Improving the algorithm evaluating KG such that it can be applied to continuous search spaces.
5. Incorporating the gradient to GP model:

$$\tilde{\mu}(x) = (\mu(x), \nabla\mu(x))^T, \quad \tilde{K}(x,x') = \begin{pmatrix} K(x,x') & J(x,x') \\ J(x',x)^T & H(x,x') \end{pmatrix}$$

where $J(x,x') = \left(\frac{\partial K(x,x')}{\partial x'_1}, \cdots, \frac{\partial K(x,x')}{\partial x'_d}\right)$ and $H(x,x')$ is the $d \times d$ Hessian of $K(x,x')$.

$$\begin{aligned} \tilde{\mu}^{(n)}(x) &= \tilde{\mu}(x) + \tilde{K}(x,X) \\ &\quad \left(\tilde{K}(X,X) + \text{diag}\{\sigma^2(x^{(1)}), \cdots, \sigma^2(x^{(n)})\}\right)^{-1} \left((y, \nabla y)^{(1:n)} - \tilde{\mu}(X)\right) \\ \tilde{K}^{(n)}(x,x') &= \tilde{K}(x,x') - \tilde{K}(x,X)\left(\tilde{K}(X,X) + \text{diag}\{\sigma^2(x^{(1)}), \cdots, \sigma^2(x^{(n)})\}\right)^{-1} \tilde{K}(X,x'). \end{aligned}$$

6. d-KG is defined similarly to q-KG:

$$d\text{-}KG(z^{(1:q)}) = \min_{x\in\mathbb{A}} \tilde{\mu}_1^{(n)}(x) - \mathbb{E}_n\left[\min_{x\in\mathbb{A}} \tilde{\mu}_1^{(n+q)}(x) \;\middle|\; x^{((n+1):(n+q))} = z^{(1:q)}\right]$$

It can also be defined at a certain direction θ (only directional derivative observed):

$$d\text{-}KG(z^{(1:q)}, \theta) = \min_{x\in\mathbb{A}} \tilde{\mu}_1^{(n)}(x) - \mathbb{E}_n\left[\min_{x\in\mathbb{A}} \tilde{\mu}_1^{(n+q)}(x) \;\middle|\; x^{((n+1):(n+q))} = z^{(1:q)}; \theta\right]$$

7. Algorithm:

Algorithm 1 *d-KG* with Relevant Directional Derivative Detection

1: **for** $t = 1$ to N **do**
2: $(z^{(1:q)^*}, \theta^*) = \text{argmax}_{z^{(1:q)},\theta} d\text{-}KG(z^{(1:q)}, \theta)$
3: Augment data with $y(z^{(1:q)^*})$ and $\theta^{*T}\nabla y(z^{(1:q)^*})$. Update our posterior on $(f(x), \nabla f(x))$.
4: **end for**
Return $x^* = \text{argmin}_{x\in\mathbb{A}} \tilde{\mu}_1^{Nq}(x)$

8. The gradient of d-KG can be evaluated by constructing an unbiased estimator, then d-KG can be optimized using gradient based optimizers such as stochastic gradient ascent.
9. Result: For most problems, the d-KG method gives the best performance.

10. Code available at: https://github.com/wujian16/Cornell-MOE

Practical Bayesian Optimization for Model Fitting with Bayesian Adaptive Direct Search

Luigi Acerbi, Wei Ji Ma

1. Problem to solve: Relatively low-cost optimization problems.
2. Contribution: Developing a method (MADS) that combines BO with adaptive direct search, which can be used widely as a general optimizer like fmincon or CMA-ES.
3. Idea: Use adaptive direct search to explore the search space, and use local BO to find local optima.
4. Use a subset of training data close to search center to build the GP model and perform BO. If improvement is sufficient, keep doing this, otherwise try to explore further spaces and set new search centers. Exploring is done by sampling points that have a certain distance to the center, and evaluate whether these points can provide sufficient improvement. The sequence of evaluation can be determined by evaluating the acquisition function. Target improvement and traveling distances are adaptive during this process.
5. BO kernel: rational quadratic kernel:

$$k_{\mathrm{RQ}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left[1 + \frac{1}{2\alpha} r^2(\boldsymbol{x}, \boldsymbol{x}')\right]^{-\alpha}, \qquad \text{with} \quad r^2(\boldsymbol{x}, \boldsymbol{x}') = \sum_{d=1}^{D} \frac{1}{\ell_d^2} (x_d - x_d')^2,$$

6. Acquisition function: lower confidence bound.
7. **Algorithm 1** Bayesian Adaptive Direct Search

Input: objective function f, starting point $\boldsymbol{x}_0$, hard bounds LB, UB, (*optional*: plausible bounds PLB, PUB, barrier function c, additional `options`)

Initialization: $\Delta_0^{\mathrm{mesh}} \leftarrow 2^{-10}$, $\Delta_0^{\mathrm{poll}} \leftarrow 1$, $k \leftarrow 0$, evaluate f on initial design ▷ Section 3.1
repeat
  (update GP approximation at any step; refit hyperparameters if necessary) ▷ Section 3.2
  for $1 \ldots n_{\mathrm{search}}$ **do** ▷ SEARCH stage, Section 3.3
    $\boldsymbol{x}_{\mathrm{search}} \leftarrow$ SEARCHORACLE ▷ local Bayesian optimization step
    Evaluate f on $\boldsymbol{x}_{\mathrm{search}}$, **if** improvement is *sufficient* **then break**
  if SEARCH is NOT *successful* **then** ▷ optional POLL stage, Section 3.3
    compute poll set P_k
    evaluate opportunistically f on P_k sorted by acquisition function
  if iteration k is *successful* **then**
    update incumbent $\boldsymbol{x}_{k+1}$
    if POLL was *successful* **then** $\Delta_k^{\mathrm{mesh}} \leftarrow 2\Delta_k^{\mathrm{mesh}}$, $\Delta_k^{\mathrm{poll}} \leftarrow 2\Delta_k^{\mathrm{poll}}$
  else
    $\Delta_k^{\mathrm{mesh}} \leftarrow \frac{1}{2}\Delta_k^{\mathrm{mesh}}$, $\Delta_k^{\mathrm{poll}} \leftarrow \frac{1}{2}\Delta_k^{\mathrm{poll}}$
  $k \leftarrow k + 1$
until `fevals` $>$ `MaxFunEvals` **or** $\Delta_k^{\mathrm{poll}} < 10^{-6}$ **or** stalling ▷ stopping criteria
return $\boldsymbol{x}_{\mathrm{end}} = \arg\min_k f(\boldsymbol{x}_k)$ (or $\boldsymbol{x}_{\mathrm{end}} = \arg\min_k q_\beta(\boldsymbol{x}_k)$ for noisy objectives, Section 3.4)

8. Result: MADS performs the best or among the best ones in every test.

9. Interesting observation: vanilla BO implemented on Matlab (bayesopt) performs poorly in every test. The author says that method becomes slow when having too many training points, thus changes it to a strange setup. I think that's why it cannot perform well.

Process-constrained batch Bayesian Optimisation

Pratibha Vellanki, Santu Rana, Sunil Gupta, David Rubin, Alessandra Sutti, Thomas Dorin, Murray Height, Paul Sandars, Svetha Venkatesh

1. Problem to solve: Parallel optimization problem with shared parameters in each batch.
2. Background: Some real-world parallel problems require shared parameters among experiments that are done at the same time such as temperature control or mass-produced materials.
3. Based on parallelized upper confidence bound BO (GP-UCB-PE).
4. Contribution: Proposed two methods that can generate batch of new designs with shared parameters.
5.

Algorithm 1 pc-BO(basic): Basic process-constrained pure exploration batch Bayesian optimisation algorithm.

while $(t < MaxIter)$
  $x_{t,0} = \left[x_{t,0}^{uc} x_{t,0}^{c}\right] = \text{argmax}_{x \in \mathcal{X}} \alpha^{GP-UCB}\left(x_{t,0} \mid D\right)$
  for $k = 1, .., K-1$
    $x_{t,k}^{uc} = \text{argmax}_{\ x^{uc} \in \mathcal{X}^{uc}} \sigma\left(x_{t,k}^{uc} \mid D, x_{t,0}^{c}, \left\{x_{t,k'}^{uc}\right\}^{k'<k}\right)$
  end
  $D = D \cup \left\{\left[x_{t,k}^{uc} x_{t,1}^{c}\right], f\left(\left[x_{t,k}^{uc} x_{t,1}^{c}\right]\right)\right\}_{k=0}^{K-1}$
end

Algorithm 2 pc-BO(nested): Nested process-constrained batch Bayesian optimisation algorithm.

while $(t < MaxIter)$
  $x_t^c = \text{argmax}_{x^c \in \mathcal{X}^c} \alpha_c^{GP-UCB}\left(x_t^c \mid D_O\right)$
  $x_{t,0}^{uc} = \text{argmax}_{x^{uc} \in \mathcal{X}^{uc}} \alpha_{uc}^{GP-UCB}\left(x_t^{uc} \mid D_I, x_t^c\right)$
  for $k = 1, ..., K$-1
    $x_{t,k}^{uc} = \text{argmax}_{x^{uc} \in \mathcal{X}^{uc}} \sigma_{uc}\left(x_t^{uc} \mid D_I, x_t^c, \left\{x_{t,k'}^{uc}\right\}^{k'<k}\right)$
  end
  $D_O = D_O \cup \left\{x_t^c, f\left(\left[(x_t^{uc})^+ x_t^c\right]\right)\right\}$
  $D_I = D_I \cup \left\{\left[x_{t,k}^{uc} x_t^c\right], f\left(\left[x_{t,k}^{uc} x_t^c\right]\right)\right\}_{k=0}^{K-1}$
end

The first algorithm optimizes the first design in a batch regularly, then fixes the shared parameters and optimizes the rest unshared parameters for the following designs. The second algorithm splits the design space into shared part and unshared part, optimizes the shared parameters, then optimizes the unshared parts for each design.

6. Provided analysis of convergence for the second algorithm.
7. The two new algorithms are compared only with sequential BO (GP-UCB) because there are no other method addressing the shared parameters in batches. Algorithm 2 has better performance.
8. This strategy can be easily extended to other BO methods.

Bayesian optimization for automated model selection

Gustavo Malkomes, Chip Schaff, Roman Garnett

1. Problem to solve: Model selection problem.
2. Background: The performance of a variety of kernel methods relies on the choice of kernel, thus how to choose the kernel from the infinite kernel space is a problem.
3. Related works: Generative grammars have been proposed to enumerate a countably infinite space of arbitrarily complex kernels via exploiting the closure of kernels under additive and multiplicative composition. A greedy search is introduced to optimize the kernel.
4. Improvement: Using BO to optimize the kernel.
5. Objective function: Given dataset, try to build a GP with zero mean and a kernel to explain the dataset. Then the marginal likelihood can be evaluated for that kernel, and that's the objective function to be optimized.

$$\log p(\mathbf{y} \mid \mathbf{X}, \mathcal{M}) \approx \log p(\mathbf{y} \mid \mathbf{X}, \hat{\theta}, \mathcal{M}) + \log p(\hat{\theta} \mid \mathcal{M}) - \tfrac{1}{2} \log \det \Sigma^{-1} + \tfrac{d}{2} \log 2\pi$$

6. After the dataset is normalized, the hyperparameters for different kernels are assumed to have some distribution (take log, then they are normal distributions) (defined by the authors).
7. The kernels are generated by randomly applying the generative grammas (adding or multiplying a base kernel).
8. Hellinger distance is introduced to measure the distance of kernels.
9. Kernel for the BO is a squared exponential kernel, and the acquisition function is EI.
10. Due to the behavior of kernel space, for each round, candidates are generated by neighbors of best kernels so far and random walks from current region. The candidates with high EI are evaluated and added to training set.
11. Result: Model-evidence regression experiments are done, and the proposed method gives the best mean square error compared with regular methods. It also outperformed the greedy method in most tests.

Bayesian Optimization for Probabilistic Programs

Tom Rainforth, Tuan Anh Le, Jan-Willem van de Meentz, Michael A. Osborne, Frank Wood

1. Problem to solve: Optimization problems on Probabilistic programming systems.
2. Contribution: Introducing BO to probabilistic programs (e.g. Anglican programs) and provided features like automatic domain scaling and constraint satisfaction.

3. This paper provides very little implementation details, and the introduction to probabilistic programs makes me confused.
4. The significant improvement compared to prominent BO packages including spearmint is not clearly reasoned, and part of the comparison used old and poor data.
5. I will try to find more details about this.
6. Code available at: https://github.com/probprog/deodorant, but it's written in Clojure.

Bayesian Optimization with Robust Bayesian Neural Networks

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, Frank Hutter

1. Problem to solve: Optimization problems that the function may not be modeled by GP.
2. Contribution: Developing a method that uses neural networks as models (instead of GP) while keeping the Bayesian treatment.
3. Formulation:

$$p(f_t(\mathbf{x}) \mid \mathbf{x}, \theta) = \mathcal{N}(\hat{f}(\mathbf{x}, t; \theta_\mu), \theta_{\sigma^2})$$

Where f_t is the target function from the multi-task setup, and $\hat{f}(x, t; \theta_\mu)$ is the output of neural network surrogate with parameters θ_μ.

$$p(\mathcal{D}, \theta) = p(\theta_\mu)p(\theta_{\sigma^2}) \prod_{k=1}^{K} \prod_{i=1}^{|\mathcal{D}_k|} \mathcal{N}(y_i^k | \hat{f}(\mathbf{x}_i^k, k; \theta_\mu), \theta_{\sigma^2})$$

$$p(f_t(\mathbf{x})|\mathbf{x}, D) = \int_\theta p(f_t(\mathbf{x}) \mid \mathbf{x}, \theta)p(\theta \mid D)d\theta \approx \frac{1}{M} \sum_{i=1}^{M} p(f_t(\mathbf{x}) \mid \mathbf{x}, \theta^i)$$

Given $\theta^i \sim p(\theta|D)$ generated by stochastic gradient Hamiltonian Monte Carlo (SGHMC) method. Details can be checked at https://arxiv.org/pdf/1402.4102.pdf.

$$\mu(f_t(\mathbf{x})|\mathcal{D}) = \frac{1}{M} \sum_{i=1}^{M} \hat{f}(\mathbf{x}, t; \theta_\mu^i), \quad \sigma^2(f(\mathbf{x})|\mathcal{D}) = \frac{1}{M} \sum_{i=1}^{M} \left(\hat{f}(\mathbf{x}, t; \theta_\mu^i) - \mu(f_t(\mathbf{x})|\mathcal{D})\right)^2 + \theta_{\sigma^2}^i$$

These two values can be used to evaluate EI. The partial derivatives of EI can also be calculated by backpropagation through all functions $\hat{f}(x, t; \theta_\mu^i)$, thus gradient based optimizers can be used.

4. The generation of $\theta^i \sim p(\theta|D)$ is most the important part of regression. This paper also proposed a scale adaptation to improve the robustness of SGHMC.
5. Result: It performs better than the recently proposed DNGO method, which uses features extracted from a maximum likelihood fit of a neural network as the basis for a Bayesian linear regression fit. It performs worse than regular BO with GP on Brainn function (smooth) (not surprising), but it performs better on multi-task Bayesian optimization for the WINE (for a random forest) and HEPATITIS dataset (using an svm).
6. It is an interesting idea to use NN to model the behaviors of NN.
7. This paper uses a three layer (fully connected?) neural network with 50 tanh units as the surrogate. But it is never discussed how the choice of different NN affects the result.
8. Code available at: https://github.com/automl/RoBO