ProBO: Versatile Bayesian Optimization Using Any Probabilistic Programming Language

W. Neiswanger et al. 2019

Paper review by Wanru Zhao

Bayesian Optimization (BO)

Consider a 'well behaved' function $f : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X} \subseteq \mathbb{R}^D$ is a bounded domain.

$$x_M = \arg\min_{x \in \mathcal{X}} f(x).$$

- Expensive to evaluate
- Black box
- Derivative-free
- (Maybe) noisy

Bayesian Optimization (BO)

Flow

- Methodology to perform global optimisation of multimodal black-box functions.
- 1. Choose some prior measure over the space of possible objectives *f*.
- 2. Combine prior and the likelihood to get a posterior measure over the objective given some observations.
- 3. Use the posterior to decide where to take the next evaluation according to some acquisition function.
- 4. Augment the data.
- Iterate between 2 and 4 until the evaluation budget is over.

Bayesian Optimization (BO)

Surrogate Model

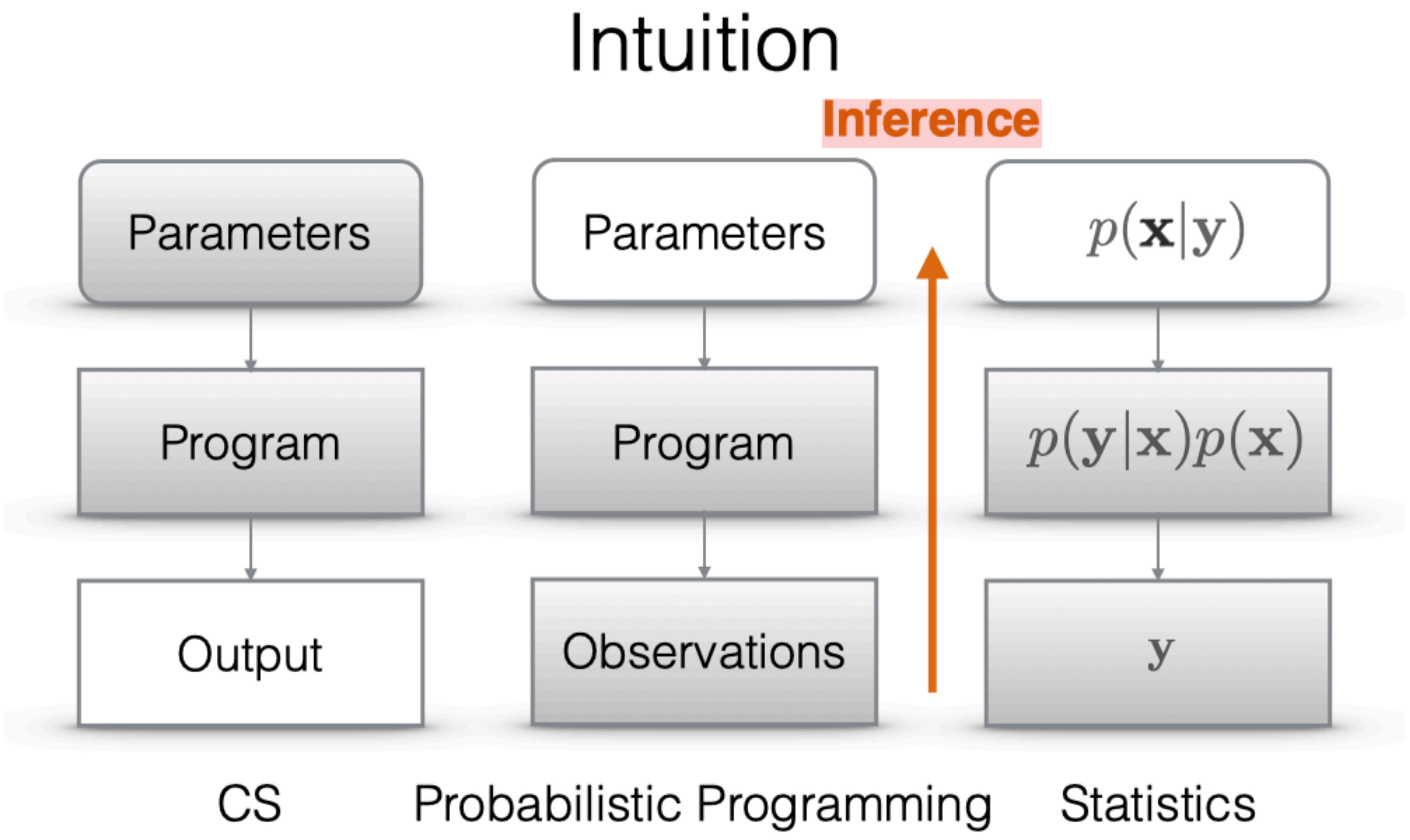
- Gaussian process
- Random Forrest
- t-Student processes
- Neural Networks

Bayesian Optimization (BO)

Acquisition Function

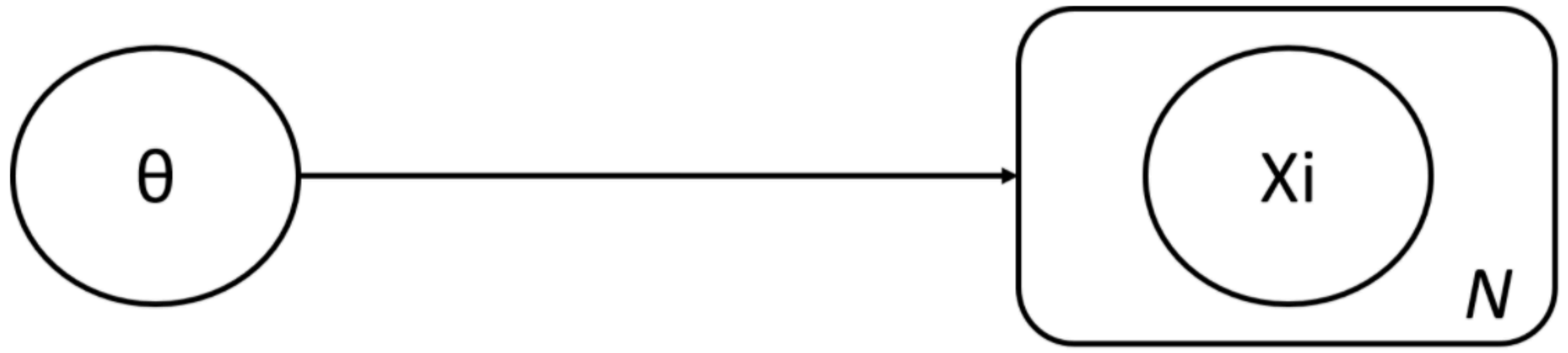
- expected improvement (EI)
- probability of improvement (PI)
- GP upper confidence bound (UCB)
- Thompson sampling (TS)

Probabilistic Programming Languages (PPLs)



Probabilistic Programming Languages (PPLs)

Example: a biased coin toss



- Calculate the bias of a coin:
  - Bernoulli distribution with latent variable θ
    - $P(x_i = 1 \mid \theta) = \theta$ and $P(x_i = 0 \mid \theta) = 1 - \theta$
    - Infer θ based on previous results of coin toss - $P(\theta \mid x_1, x_2, \ldots, x_N)$

```
# Model
theta = Uniform(0.0, 1.0)
x = Bernoulli(probs=theta, sample_shape=10)
Data 5 data = np.array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1])
Inference
qtheta = Empirical( 8 tf.Variable(tf.ones(1000) * 0.5))
inference = ed.HMC({theta: qtheta},
data={x: data})
inference.run()
Results 13 mean, stddev = ed.get_session().run( [qtheta.mean(),qtheta.stddev()])
print("Posterior mean:", mean)
print("Posterior stddev:", stddev)
```

Probabilistic Programming Languages (PPLs)

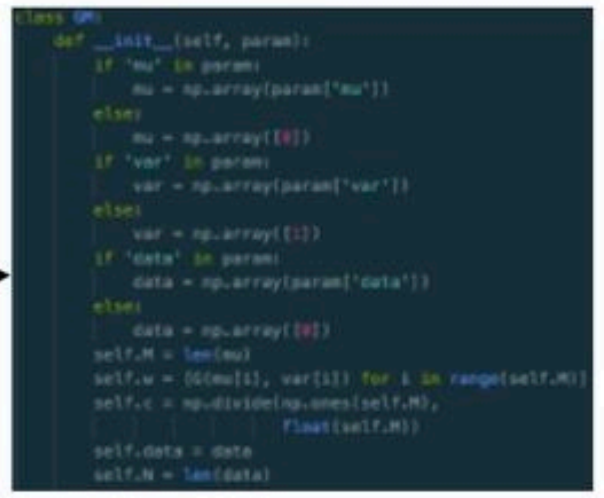
Probabilistic Programming (PP) in Julia: New Inference Algorithms

Day of a biologist who wants to use Gaussian Mixtures in his research

Without PP

1. write and code model

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{m=1}^{M} c_m \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$$



okay…
20 lines of code is not too bad

2. derive and code inference algorithm

$$\boldsymbol{\mu}_m^{(k+1)} = \frac{\sum_{i=1}^{n} P(\omega_m|\boldsymbol{x}_i, \boldsymbol{\theta}^{(k)})\boldsymbol{x}_i}{\sum_{i=1}^{n} P(\omega_m|\boldsymbol{x}_i, \boldsymbol{\theta}^{(k)})}$$

$$\sigma_m^{(k+1)2} = \frac{1}{d}\frac{\sum_{i=1}^{n} P(\omega_m|\boldsymbol{x}_i, \boldsymbol{\theta}^{(k)})\|\boldsymbol{x}_i - \boldsymbol{\mu}_m^{(k+1)}\|^2}{\sum_{i=1}^{n} P(\omega_m|\boldsymbol{x}_i, \boldsymbol{\theta}^{(k)})}$$

$$c_m^{(k+1)} = \frac{1}{n}\sum_{i=1}^{n} P(\omega_m|\boldsymbol{x}_i, \boldsymbol{\theta}^{(k)})$$

math…
coding…
debugging…
optimisation…

3. amend model and iterate 1&2

wait… I need to do them again???

With PP

1. write and code model (in a PP syntax)

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{m=1}^{M} c_m \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$$

```julia
@model MoG begin
  μs = TArray[zeros(Float64, D) for i = 1:J]
  for i = 1:J
    @assume μs[i] ~ MvNormal(Float64[0 for j = 1:D], Float64[1 for j = 1:D])
  end
  hs = tzeros(Float64, length(xs))
  for i = 1:length(xs)
    @assume hs[i] ~ Categorical(Float64[1/J for j = 1:J])
    @observe xs[i] ~ MvNormal(Float64[μs[hs[i]][j] for j = 1:D], Float64[0.25 for j = 1:D])
  end
  @predict μs hs
end
```

only 12 lines of code!

2. run the inference engine

```julia
samples = sample(MoG, HMC(500))
```

1 line of code!

3. amend model and iterate 1&2

easy!

Probabilistic Programming Languages (PPLs)

Recent popular PPL examples

- Often built upon existing languages
- PyMC3/PyMC4 (Python)
- Stan (C++, Python, R)
- Turing.jl (Julia)
- WebPPL (JavaScript)
- Edward (Tensorflow)
- Pyro (PyTorch)

ProBO

a BO system for PPL models

- Computes and optimizes acquisition functions via operations that can be implemented in a broad variety of PPLs
- Goal: allow a custom model written in an arbitrary PPL to be plugged in and immediately used in BO.

Related Work

- BOPP
  - describes a BO method for marginal maximum a posteriori (MMAP) estimates of latent variables
  - uses BO (with GP models) to help estimate latent variables in a given PPL
- BOAT
  - provides a custom PPL involving composed GP models with parametric mean functions for use in BO
  - uses exact inference & expected improvement

ProBO

Abstraction for Probabilistic Programs

- Three core PPL operations:
  - `inf(D)` - returns `post` (PPL dependent)
  - `post(s)` - returns a sample from the posterior distribution
  - `gen(x, z, s)` - returns sample from generative distribution

Algorithm 1 ProBO($\mathcal{D}_0$, $\texttt{inf}$, $\texttt{gen}$)

for $n = 1, \ldots, N$ **do**
  $\texttt{post} \leftarrow \texttt{inf}(\mathcal{D}_{n-1})$ ▷ Run inference algorithm to compute **post**
  $x_n \leftarrow \text{argmin}_{x \in \mathcal{X}} a(x, \texttt{post}, \texttt{gen})$ ▷ Optimize acquisition using **post** and **gen**
  $y_n \sim s(x_n)$ ▷ Observe system at x_n
  $\mathcal{D}_n \leftarrow \mathcal{D}_{n-1} \cup (x_n, y_n)$ ▷ Add new observations to dataset
Return $\mathcal{D}_N$.

ProBO

PPL Acquisition Functions

- Expected Improvement (EI)
- Probability of Improvement (PI)
- Upper Confident Bound (UCB)
- Thompson Sampling (TS)

Algorithm 2 $a_{\text{EI}}(x, \texttt{post}, \texttt{gen})$ ▷ EI

for $m = 1, \ldots, M$ **do**
  $z_m \leftarrow \texttt{post}(s_m)$
  $y_m \leftarrow \texttt{gen}(x, z_m, s_m)$
$f_{\min} \leftarrow \min_{y \in \mathcal{D}} f(y)$
Return $\sum_{m=1}^{M} \mathbb{1}\left[f(y_m) \leq f_{\min}\right]\left(f_{\min} - f(y_m)\right)$

Algorithm 3 $a_{\text{PI}}(x, \texttt{post}, \texttt{gen})$ ▷ PI

for $m = 1, \ldots, M$ **do**
  $z_m \leftarrow \texttt{post}(s_m)$
  $y_m \leftarrow \texttt{gen}(x, z_m, s_m)$
$f_{\min} \leftarrow \min_{y \in \mathcal{D}} f(y)$
Return $\sum_{m=1}^{M} \mathbb{1}\left[f(y_m) \leq f_{\min}\right]$

Algorithm 4 $a_{\text{UCB}}(x, \texttt{post}, \texttt{gen})$ ▷ UCB

for $m = 1, \ldots, M$ **do**
  $z_m \leftarrow \texttt{post}(s_m)$
  $y_m \leftarrow \texttt{gen}(x, z_m, s_m)$
Return $\widehat{\text{LCB}}\left(f(y_m)_{m=1}^{M}\right)$ ▷ See text for details

Algorithm 5 $a_{\text{TS}}(x, \texttt{post}, \texttt{gen})$ ▷ TS

$z \leftarrow \texttt{post}(s_1)$
for $m = 1, \ldots, M$ **do**
  $y_m \leftarrow \texttt{gen}(x, z, s_m)$
Return $\sum_{m=1}^{M} f(y_m)$

ProBO

Computational Considerations

- `inf()` cost dependent on PPLs inference algorithm
    - e.g. MCMC algorithms - $O(n)$ per iteration
- `inf()` only executed **once per query**
- Acquisition optimisation executed 100s times per query
    - `post()` & `gen()` cheaply implemented - $O(1)$

ProBO

Acquisition function optimisation

- `post()` & `gen()` not analytically differentiable
- Authors explored zeroth-order optimisation of a_{MF}
  - `post()` & `gen()` called M_f times
  - Any zeroth-order optimisation algorithm can be used

Algorithm 6 $a_{\text{MF}}(x, \texttt{post}, \texttt{gen})$

$a_{\min} \leftarrow$ Min value of a seen so far
$\ell = -\infty,\ f = 1$
while $\ell \leq a_{\min}$ **do**
  $\ell \leftarrow$ LCB-bootstrap $(\texttt{post}, \texttt{gen}, M_f)$
  $f \leftarrow f + 1$
Return $a(x, \texttt{post}, \texttt{gen})$ using $M = M_f$

Algorithm 7 LCB-bootstrap$(\texttt{post}, \texttt{gen}, M_f)$

$y_{1:M_f} \leftarrow$ Call **post** and **gen** M_f times
for $j = 1, \ldots, B$ **do**
  $\tilde{y}_{1:M_f} \leftarrow$ Resample$(y_{1:M_f})$
  $a_j \leftarrow \lambda(\tilde{y}_{1:M_f})$ ▷ See text for details
Return LCB $(a_{1:B})$

Examples and Experiments

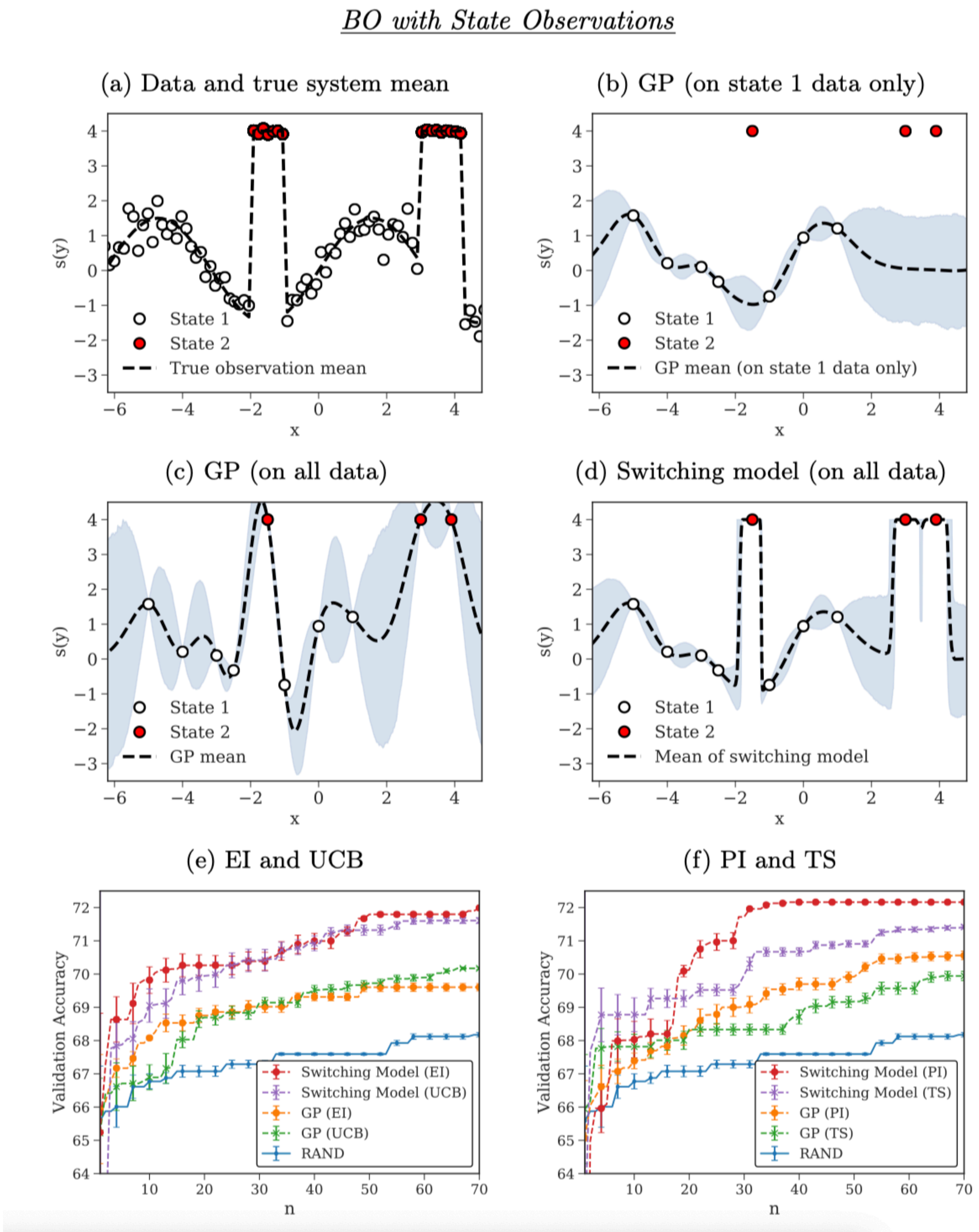
Experiment Setting

- PPL Implementations:
    - Stan + No U-Turn Sampler (a form of Hamiltonian Monte Carlo)
    - Edward + black box variational inference
- GP comparisons:
    - George
    - GPy

Examples and Experiments

BO with State Observations

- Switching Model: ProBO using a dynamic value of M_f
- Task: neural network architecture and hyperparameter search
  - multi-layer perceptron (MLP) neural networks
  - x = (number of layers, layer width, learning rate, batch size)
  - Pima Indians Diabetes Dataset
- ProBO+switching model **vs** BO+GP

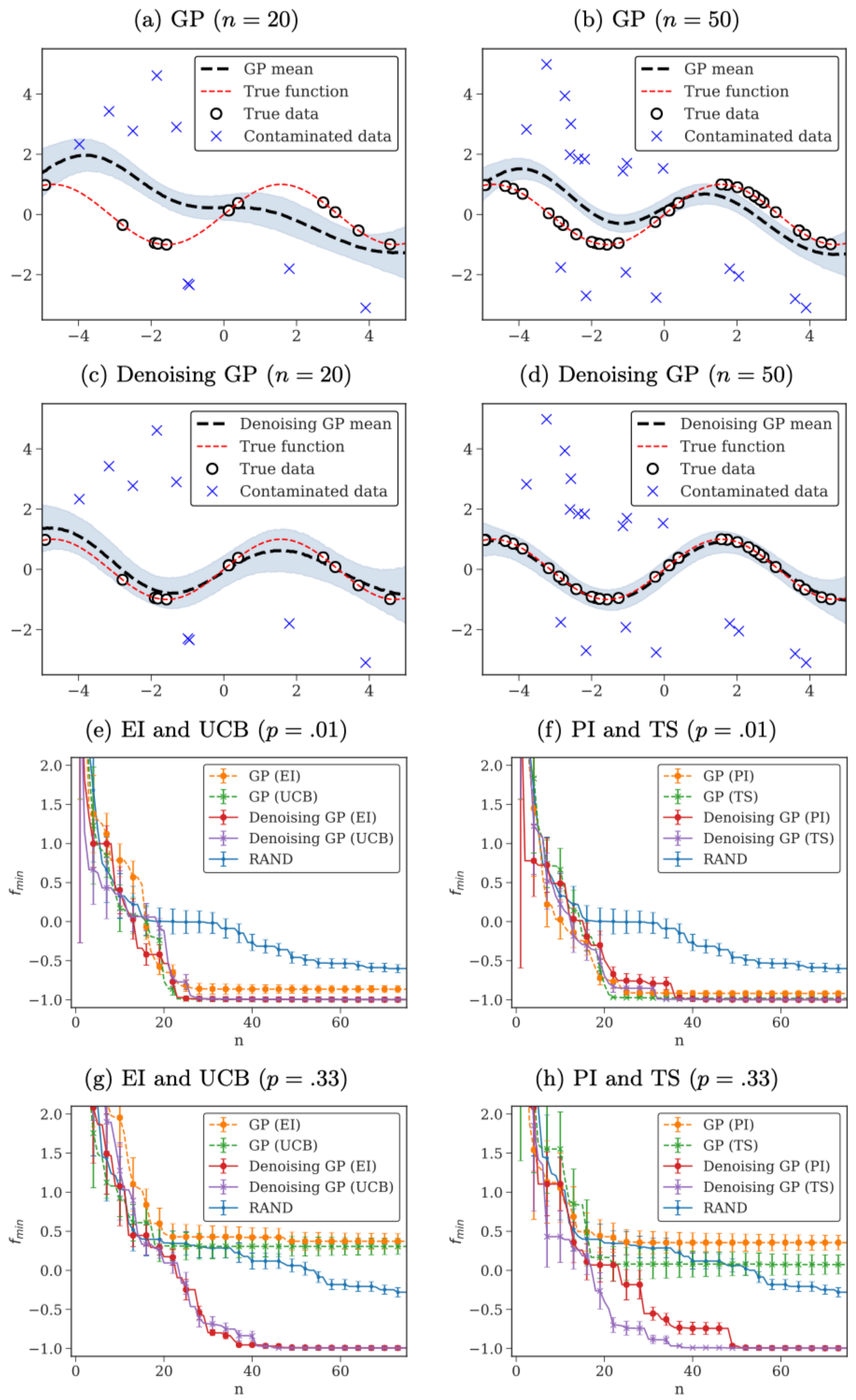


Examples and Experiments

Robust Models for Contaminated BO

- not have access to state observations
- Task: synthetic optimization task
    - system model M_s
    - contamination model M_c
    - denoising model $y \sim w_s M_s(\cdot \mid z_s; x) + w_c M_c(\cdot \mid z_c; x)$
- ProBO+denoising model **vs** BO+GP

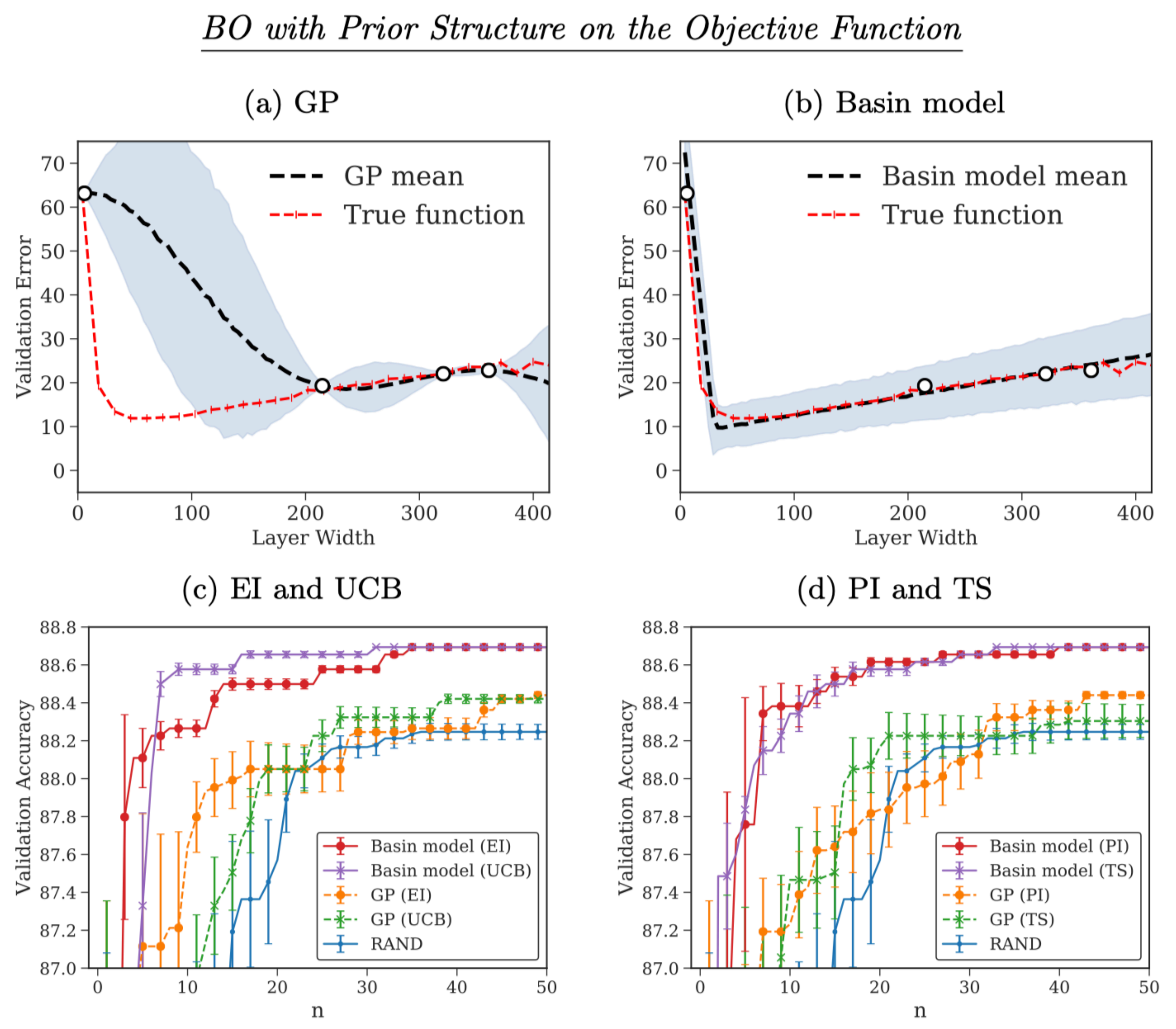
Contaminated BO



Examples and Experiments

BO with Prior Structure on the Objective Function

- basin model: have prior knowledge about properties of the objective function
- Task: tuning model complexity
    - number of units of hidden layers in 4-layer MLP
    - Wisconsin Breast Cancer Diagnosis dataset
- ProBO+basin model **vs** BO+GP

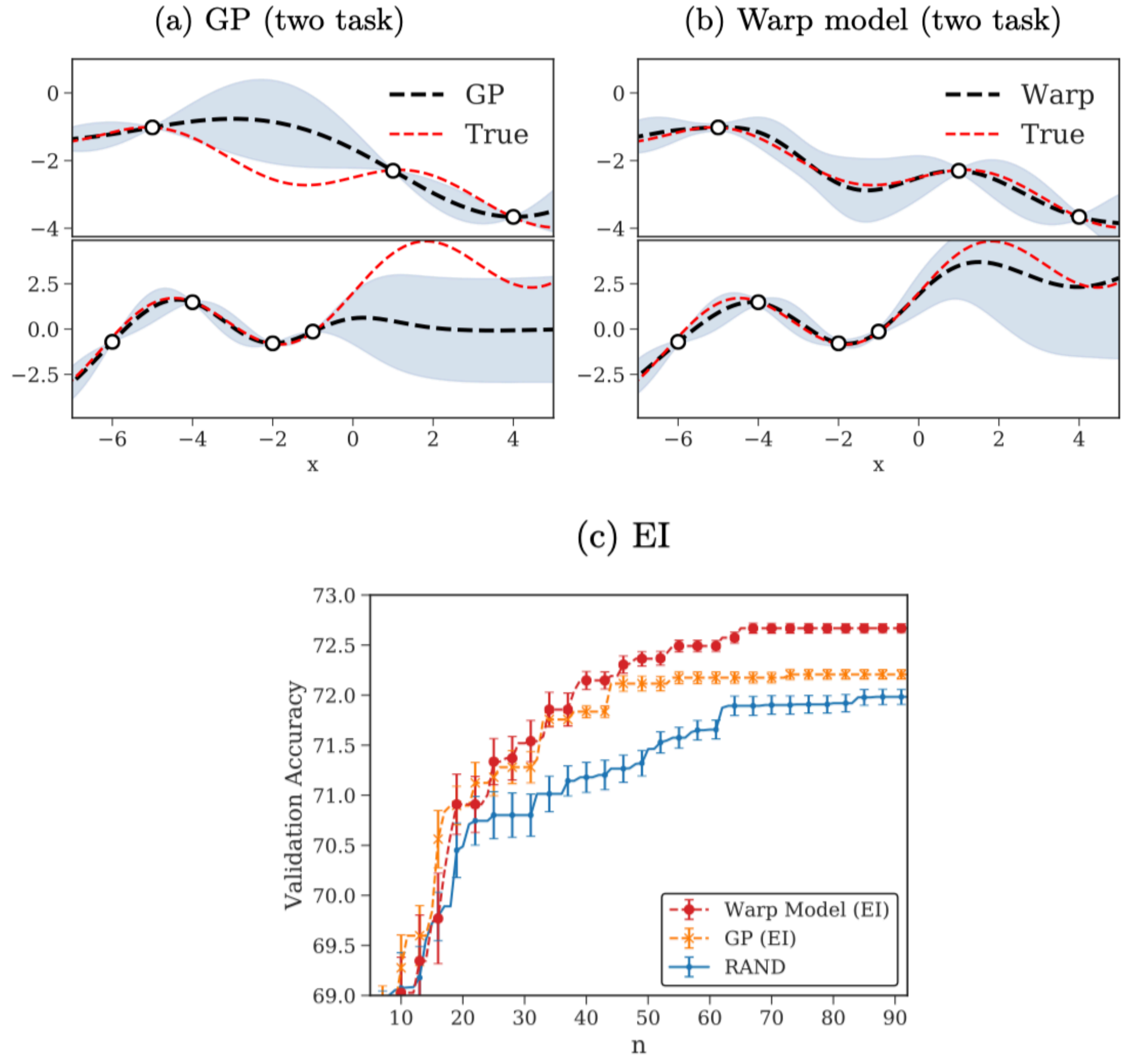


Examples and Experiments

Structured Models for Multi-task and Contextual BO, and Model Ensembles

- optimize multiple systems jointly, where there is some known relation between the systems
- a finite set of systems (multi-task BO)
- systems are each indexed by a context vector c ∈ R d (contextual BO)
- warp model: incorporate prior structure about the relationship among these systems, warps a latent model based on context/ task-specific parameters
- parametric model: model with a specific trend, shape, or specialty for a subset of the data
- posterior predictive densities of multiple PPL models, using only our three PPL operations

Structured Multi-task BO

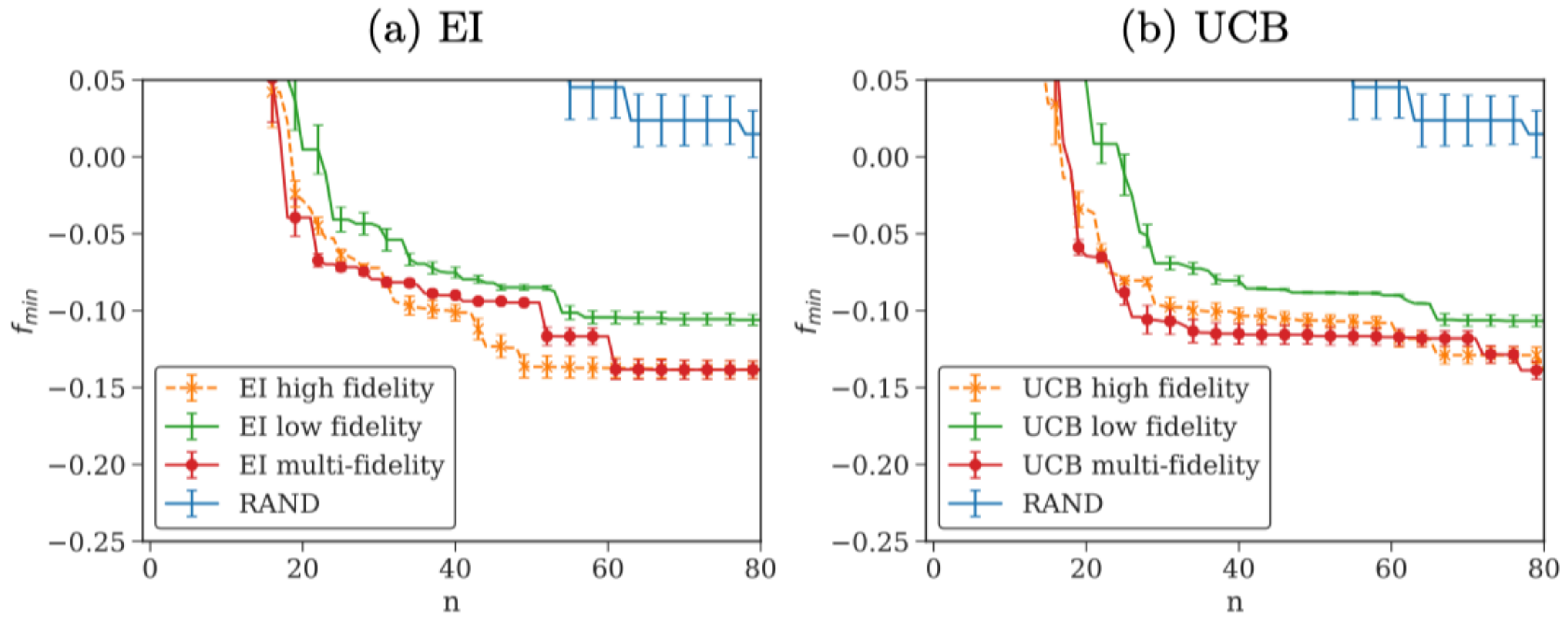


Examples and Experiments

Multi-fidelity Acquisition Optimization

- two-fidelity setting:
  - high-fidelity a (M = 1000)
  - low-fidelity a (M = 10)
  - multi-fidelity a_{MF}
- 3x better performance than high-fidelity in terms of calls to gen()

Multi-fidelity Acquisition Functions



(c) Calls to **gen**

PPL acquisition method $a(x)$	Avg. number **gen**/$a(x)$
EI high-fidelity	1000
EI multi-fidelity	347.89
EI low-fidelity	10
UCB high-fidelity	1000
UCB multi-fidelity	324.65
UCB low-fidelity	10

Opinion of the paper

Key Takeaway

- Present ProBO, a system for versatile Bayesian optimization using models from any PPL
- Use PPLs to implement new models for optimization settings that are difficult for standard BO methods and models

Following Work

- BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. NIPS 2020.
    - a library for Bayesian Optimization built on PyTorch.
    - it benefit from gradient-based optimization provided by differentiable programming, as well as algebraic methods designed to exploit GPU acceleration.
- BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search
    - BO + neural predictor framework as a high-performance framework for NAS.
    - use the ProBO implementation.

Opinion of the paper

Criticism

- The writing
  - the structure is clear and easy to follow
  - related work is not sufficient enough
- The release of ProBO is not open-sourced at all
  - Make it difficult to understand the implementation details
  - Limit the spread of the author's idea
  - No further maintenance or extension

References

- A Gentle Introduction to Probabilistic Programming Languages
- Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference
- An Introduction to Probabilistic Programming

Thanks for listening!

Q&A