Pre-training helps Bayesian optimization too

a.k.a. Prior learning for Bayesian optimization

Zi Wang @ Google Brain

zi-wang.com

Google Research

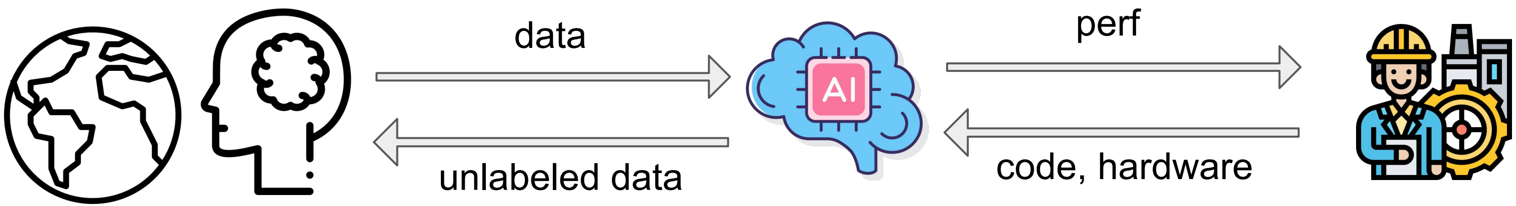
What is Bayesian optimization (BayesOpt)?

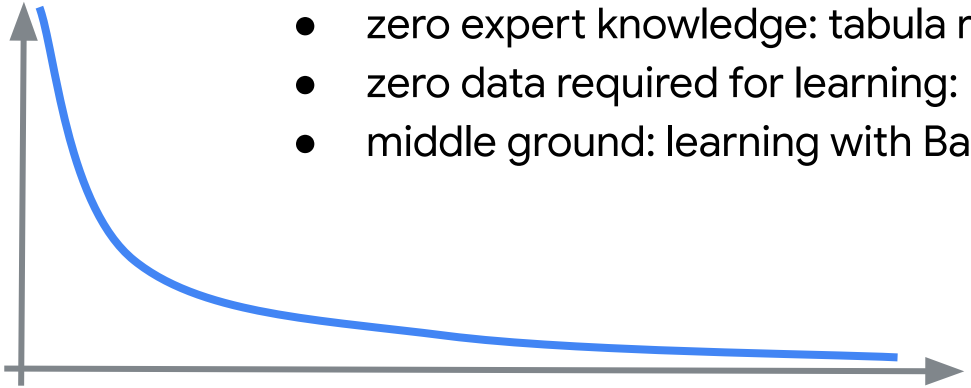
BayesOpt as an abstraction of intelligent decision making systems that collect data to gain knowledge

- BayesOpt as global optimization: how to use ML to help optimization.
- BayesOpt in AutoML: how to use ML to automate ML.
- BayesOpt for experimental design: how use ML to design experiments.

Data collection in AI / ML systems

image: Flaticon.com





- zero expert knowledge: tabula rasa models
- zero data required for learning: hard coding
- middle ground: learning with Bayesian priors

[Wang, PhD Thesis 2020]
[Wang*&Garrett*&Kaelbling&Lozano-Perez, IJRR 2020]

Robot learning as “BayesOpt” with strong priors

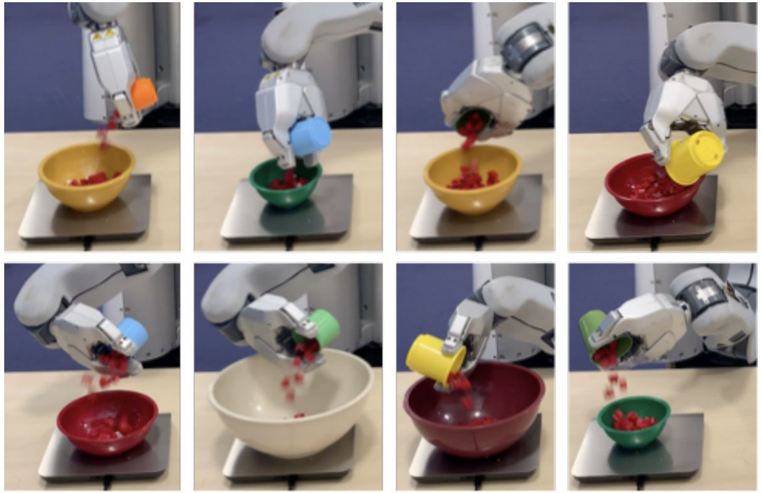


Figure 2. Examples of a real-world robot executing a trained pouring primitive in *KitchenPR2* for several contexts parameter values (cup dimensions) and control parameters values (relative cup poses).

Strong priors / built-in knowledge: modularity, robust planning algorithms...
Learning: choose which data points to collect and incorporate into the posterior.

[Wang, PhD Thesis 2020]
[Wang*&Garrett*&Kaelbling&Lozano-Perez, IJRR 2020]

Robot learning as “BayesOpt” with strong priors

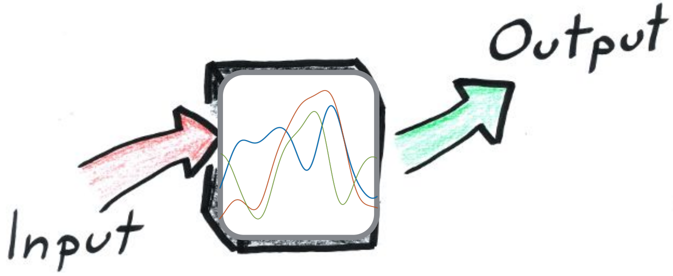
BayesOpt is not "black-box function optimization"*

Start with a model

LOOP

choose new query point(s) to evaluate

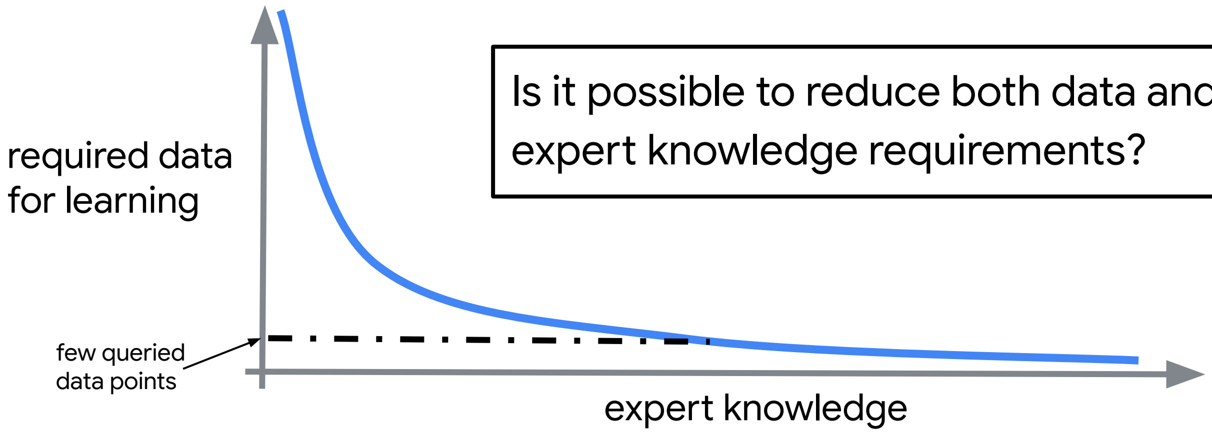
update model



What is the initial model (a.k.a. prior)?

BayesOpt and its initial model (a.k.a. prior)

- BayesOpt aims to optimize an expensive function with as few queries as possible.
- Priors are encoded by experts who have intuitions and past experience about the expensive function, e.g. wiggliness, smoothness, differentiability, etc.
- When such intuitions are lacking (e.g. hyperparameter optimization of deep learning models), BayesOpt typically needs more data.

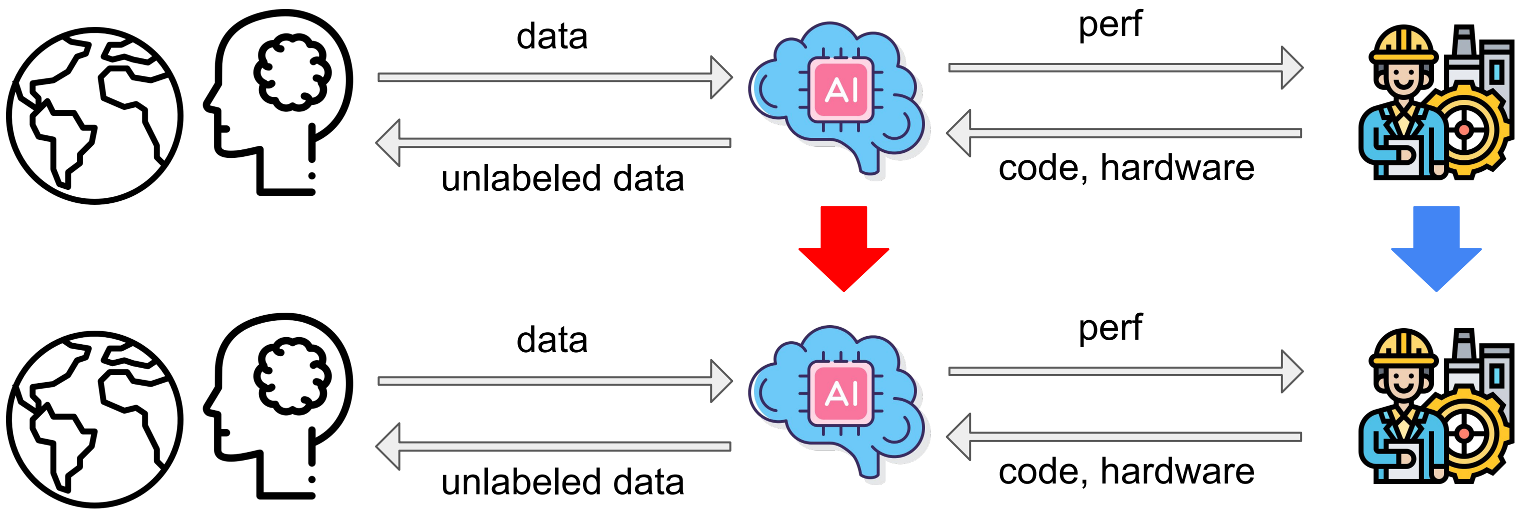


Is it possible to reduce both data and expert knowledge requirements?

How to reduce data & expertise requirements?

TL;DR: pre-training, a.k.a. meta learning, learning to learn, prior learning

image: Flaticon.com



- Improving the prior model from increased expert knowledge on this type of functions.
- Pre-training the prior on data from past experience with this type of functions.

Concepts: pre-training, prior learning and more

- Prior learning: “learning the prior” with “point sets”, a set of iid sets of potentially non-iid points. [Baxter, 1996; Minka&Picard, 1997]
- Pre-training: a more procedural and less Bayesian perspective of prior learning; i.e. emphasizing that prior learning happens before training on a new task.
- Meta learning: roughly, a frequentist way of calling prior learning.
- Learning to learn: an interpretation of meta learning (or vice versa) [Schmidhuber, 1995].

Pre-training a Gaussian process (GP)

What is pre-training in BayesOpt?

[Wang et al., 2018; Wang et al., 2022]

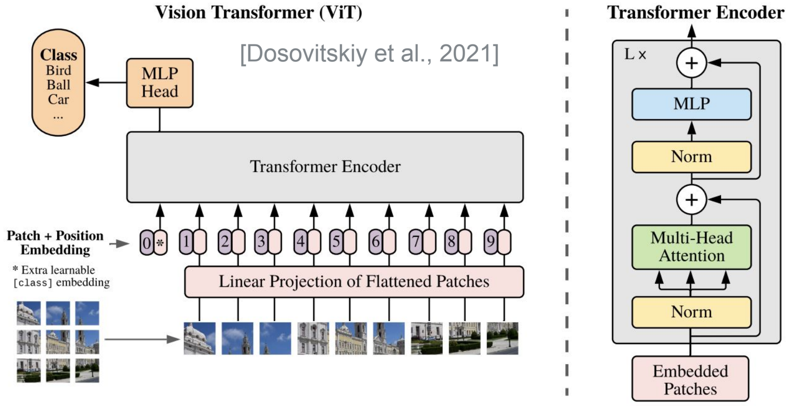
Pre-train and fine-tune for deep learning models

Pre-train

- Train the model on a very large dataset, e.g. ImageNet-21K, with a cross entropy loss.
- Save the (pre-)trained model.

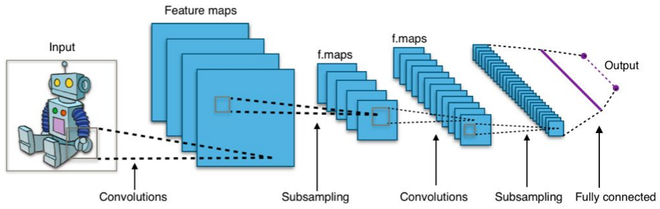
Fine-tune

- Restore the pre-trained model.
- Continue training the entire model or part of the model (e.g. last-layer weights) on a relatively small dataset, e.g. CIFAR-100.
- Now you have the fine-tuned model specific to the new task.



[Dosovitskiy et al., 2021]

Typical CNN architecture

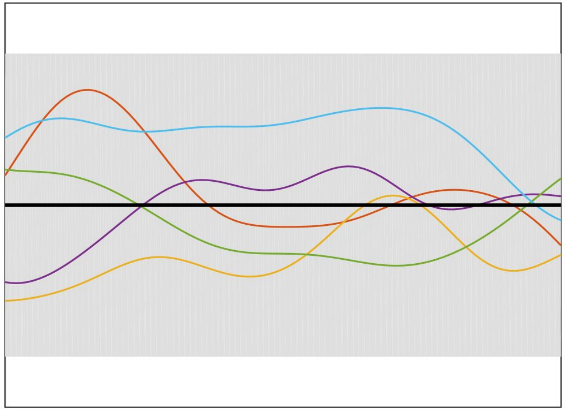


How pre-training lifted deep learning

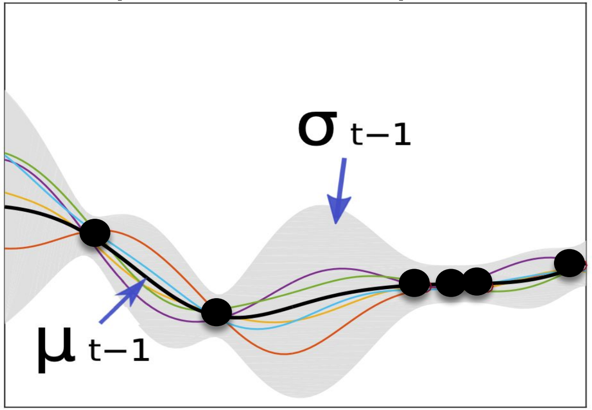
- (Supervised) pre-training on ImageNet and fine-tuning on ImageNet competition datasets led to one of the initial breakthroughs of deep learning. [Krizhevsky et al., 2012; Sermanet et al., 2014]
- “supervised pre-training on a large auxiliary dataset (ILSVRC), followed by domain-specific fine-tuning on a small dataset (PASCAL), is an effective paradigm for learning high-capacity CNNs when data is scarce. ” [Girshick et al., 2014]

Gaussian processes (GPs) in BayesOpt

Samples from the prior



Samples from the posterior



Given observations $D_t = \{(x_\tau, y_\tau)\}_{\tau=1}^{t-1}$, predict posterior mean and variance in closed form via conditional Gaussian

$$\mu_{t-1}(x) = k_{t-1}(x)^{\mathrm{T}}(K_{t-1} + \sigma^2 I)^{-1} y_{t-1}$$
$$\sigma_{t-1}(x)^2 = k(x,x) - k_{t-1}(x)^{\mathrm{T}}(K_{t-1} + \sigma^2 I)^{-1} k_{t-1}(x)$$

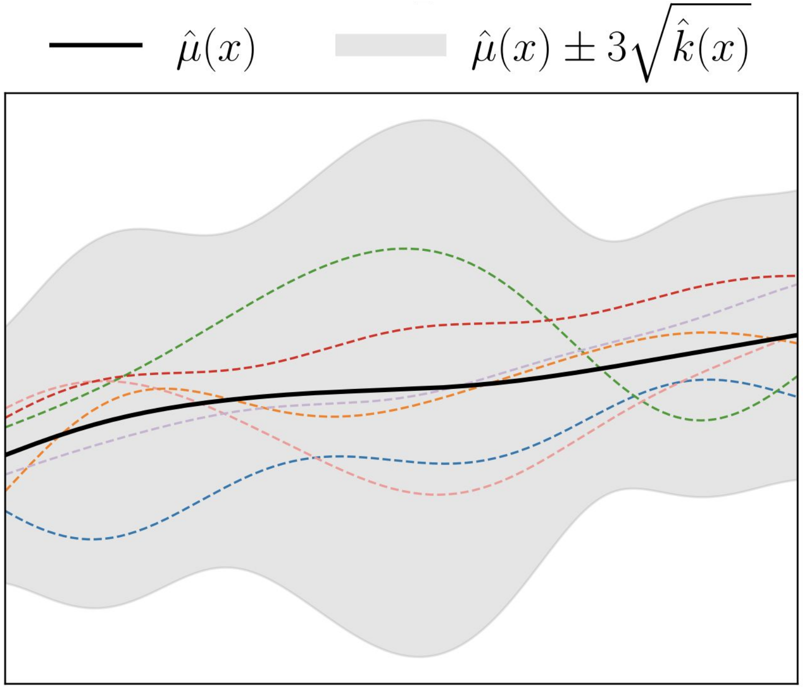
Pre-train a GP on data from a range of tasks

Task 1	(x_11, y_11)	(x_12, y_12)	…...	(x_1M, y_1M)
Task 2	(x_21, y_21)	(x_22, y_22)	…...	(x_2M, y_2M)
…...	…...	…...	…...	…...
Task N	(x_N1, y_N1)	(x_N2, y_N2)	…...	(x_NM, y_NM)
New Task	?	?	…...	?

- Each task corresponds to a function.
- Different observations may occur on different functions.
- Set the pre-trained GP as the prior for the new task.

Pre-training in BayesOpt is pre-training a GP

- Given observations on many functions (colored lines), train the GP before BayesOpt on a new function.
- Goal: train the GP model by optimizing how good observed functions fit the model.



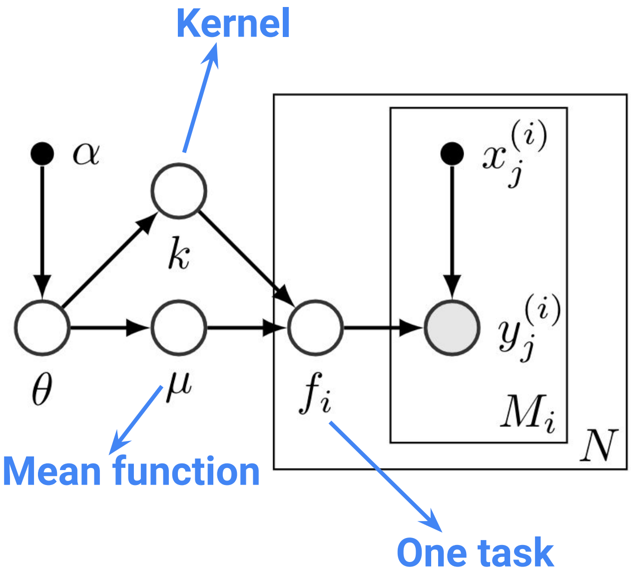
Pre-training a Gaussian process (GP)

How?

Pre-train a "multi-task" GP via hierarchical Bayes

All functions are IID samples from a GP

- Draw parameter θ from $p(\theta; \alpha)$.
- Draw mean function μ and kernel function k from $p(\mu, k \mid \theta)$.
- For each outer iteration i from 1 to N,
  - Draw a function f_i from $\mathcal{GP}(\mu, k)$.
  - For each inner loop iteration from 1 to M_i,
    * Given input $x_j^{(i)}$, we draw the observation $y_j^{(i)} \sim \mathcal{N}(f_i(x_j^{(i)}), \sigma^2)$.



Instead of learning correlations among tasks, we learn the GP that generated all tasks.

[Baxter, 1996; Minka&Picard, 1997]

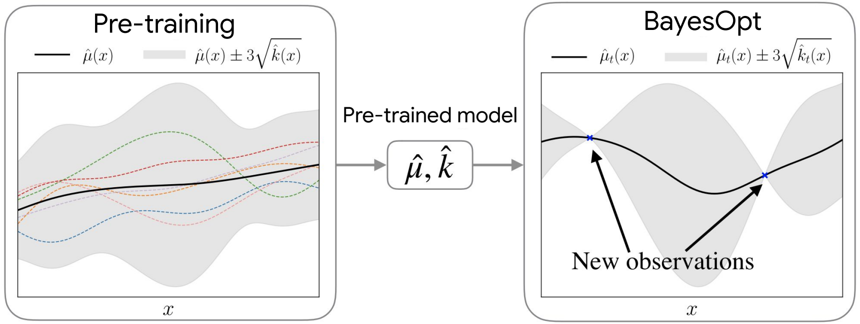
Pre-train on "a set of iid sets of potentially non-iid points"

Task f_1	$(x_1^{(1)}, y_1^{(1)})$	$\cdots$	$(x_j^{(1)}, y_j^{(1)})$	$\cdots$	$(x_{M_1}^{(1)}, y_{M_1}^{(1)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_i	$(x_1^{(i)}, y_1^{(i)})$	$\cdots$	$(x_j^{(i)}, y_j^{(i)})$	$\cdots$	$(x_{M_i}^{(i)}, y_{M_i}^{(i)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_N	$(x_1^{(N)}, y_1^{(N)})$	$\cdots$	$(x_j^{(N)}, y_j^{(N)})$	$\cdots$	$(x_{M_N}^{(N)}, y_{M_N}^{(N)})$
New task f	?	?	?	?	?

$$f_i \sim \mathcal{GP}(\mu, k)$$

$$f \sim \mathcal{GP}(\mu, k)$$

- Pre-train: train a mean function $\hat{\mu}$ and kernel $\hat{k}$ to best fit data on i.i.d. functions $f_i \sim \mathcal{GP}(\mu, k)$.
- "Fine-tune": solve $\max_{x \in \mathfrak{X}} f(x)$ via BayesOpt with an initial model $\mathcal{GP}(\hat{\mu}, \hat{k})$.



[Baxter, 1996; Minka&Picard, 1997]

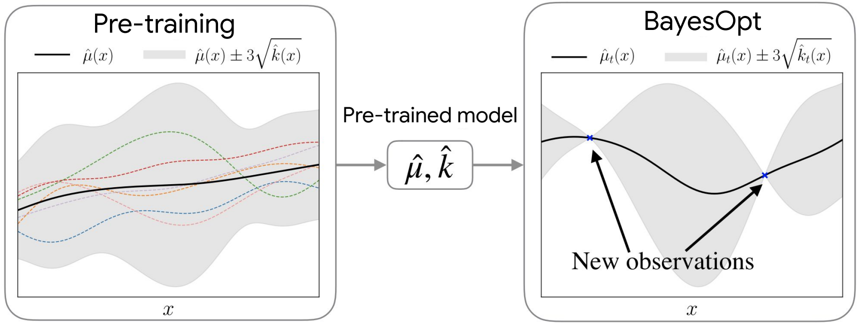
Pre-train on “a set of iid sets of potentially non-iid points”

Task f_1	$(x_1^{(1)}, y_1^{(1)})$	$\cdots$	$(x_j^{(1)}, y_j^{(1)})$	$\cdots$	$(x_{M_1}^{(1)}, y_{M_1}^{(1)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_i	$(x_1^{(i)}, y_1^{(i)})$	$\cdots$	$(x_j^{(i)}, y_j^{(i)})$	$\cdots$	$(x_{M_i}^{(i)}, y_{M_i}^{(i)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_N	$(x_1^{(N)}, y_1^{(N)})$	$\cdots$	$(x_j^{(N)}, y_j^{(N)})$	$\cdots$	$(x_{M_N}^{(N)}, y_{M_N}^{(N)})$
New task f	?	?	?	?	?

D_{f_i}

$D_N = \{D_{f_i}\}_{i=1}^N$

- Pre-train: train a mean function $\hat{\mu}$ and kernel $\hat{k}$ to best fit data on i.i.d. functions $f_i \sim \mathcal{GP}(\mu, k)$.
- “Fine-tune”: solve $\max_{x\in\mathfrak{X}} f(x)$ via BayesOpt with an initial model $\mathcal{GP}(\hat{\mu}, \hat{k})$.



[Baxter, 1996; Minka&Picard, 1997]

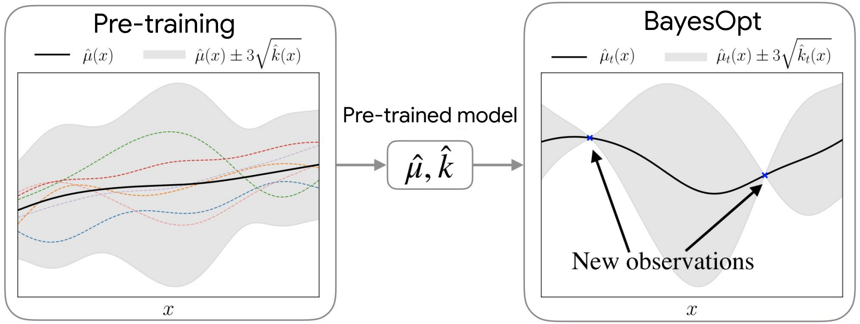
Pre-train on "a set of iid sets of potentially non-iid points"

Task f_1	$(x_1^{(1)}, y_1^{(1)})$	$\cdots$	$(x_j^{(1)}, y_j^{(1)})$	$\cdots$	$(x_{M_1}^{(1)}, y_{M_1}^{(1)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_i	$(x_1^{(i)}, y_1^{(i)})$	$\cdots$	$(x_j^{(i)}, y_j^{(i)})$	$\cdots$	$(x_{M_i}^{(i)}, y_{M_i}^{(i)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_N	$(x_1^{(N)}, y_1^{(N)})$	$\cdots$	$(x_j^{(N)}, y_j^{(N)})$	$\cdots$	$(x_{M_N}^{(N)}, y_{M_N}^{(N)})$
New task f	?	?	?	?	?

D_{f_i}

$D_N = \{D_{f_i}\}_{i=1}^N$

- Pre-train: train a mean function $\hat{\mu}$ and kernel $\hat{k}$ to best fit dataset $D_N = \{D_{f_i}\}_{i=1}^N$.
- "Fine-tune": solve $\max_{x \in \mathfrak{X}} f(x)$ via BayesOpt with an initial model $\mathcal{GP}(\hat{\mu}, \hat{k})$.



HyperBO: BayesOpt with pre-trained GP hyperparameters

Algorithm 1 HyperBO with acquisition function $\alpha(\cdot)$.

function HYPERBO(f, D_N)
  $\mathcal{GP}(\hat{\mu}, \hat{k}) \leftarrow$ PRE-TRAIN(D_N)
  $D_f \leftarrow \emptyset$
  for $t = 1, \cdots, T$ **do**
    $x_t \leftarrow \arg\max_{x \in \mathfrak{X}} \alpha\left(x; \mathcal{GP}(\hat{\mu}, \hat{k} \mid D_f)\right)$
    $y_t \leftarrow$ OBSERVE$(f(x_t))$
    $D_f \leftarrow D_f \cup \{(x_t, y_t)\}$
  end for
  return D_f
end function

Pre-train with empirical KL divergence

Pre-train with negative log likelihood

Pre-train with empirical KL divergence

Task f_1	$(x_1, y_1^{(1)})$	$\cdots$	$(x_j, y_j^{(1)})$	$\cdots$	$(x_M, y_M^{(1)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_i	$(x_1, y_1^{(i)})$	$\cdots$	$(x_j, y_j^{(i)})$	$\cdots$	$(x_M, y_M^{(i)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_N	$(x_1, y_1^{(N)})$	$\cdots$	$(x_j, y_j^{(N)})$	$\cdots$	$(x_M, y_M^{(N)})$

$$\forall i = 1, \cdots, N, \quad \begin{bmatrix} y_1^{(i)} \\ \vdots \\ y_M^{(i)} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_M) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_M) \\ \vdots & \ddots & \vdots \\ k(x_M, x_1) & \cdots & k(x_M, x_M) \end{bmatrix} + \boldsymbol{I}\sigma^2 \right)$$

i.i.d. samples from the same multivariate Gaussian

Pre-train with empirical KL divergence

Task f_1	$(x_1, y_1^{(1)})$	$\cdots$	$(x_j, y_j^{(1)})$	$\cdots$	$(x_M, y_M^{(1)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_i	$(x_1, y_1^{(i)})$	$\cdots$	$(x_j, y_j^{(i)})$	$\cdots$	$(x_M, y_M^{(i)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_N	$(x_1, y_1^{(N)})$	$\cdots$	$(x_j, y_j^{(N)})$	$\cdots$	$(x_M, y_M^{(N)})$

$$\boldsymbol{y} = \begin{bmatrix} y_1^{(1)} & \cdots & y_1^{(N)} \\ \vdots & \ddots & \vdots \\ y_M^{(1)} & \cdots & y_M^{(N)} \end{bmatrix} \in \mathbb{R}^{M \times N}$$

$$\tilde{\boldsymbol{\mu}} = \frac{1}{N}\boldsymbol{y}1_N \in \mathbb{R}^M$$

$$\tilde{K} = \frac{1}{N}(\boldsymbol{y} - \tilde{\boldsymbol{\mu}}1_N^\top)(\boldsymbol{y} - \tilde{\boldsymbol{\mu}}1_N^\top)^\top \in \mathbb{R}^{M \times M}$$

(Empirical Bayes)

Pre-train with empirical KL divergence

Task f_1	$(x_1, y_1^{(1)})$	$\cdots$	$(x_j, y_j^{(1)})$	$\cdots$	$(x_M, y_M^{(1)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_i	$(x_1, y_1^{(i)})$	$\cdots$	$(x_j, y_j^{(i)})$	$\cdots$	$(x_M, y_M^{(i)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_N	$(x_1, y_1^{(N)})$	$\cdots$	$(x_j, y_j^{(N)})$	$\cdots$	$(x_M, y_M^{(N)})$

(Our model)

$$\boldsymbol{\mu} = \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_M) \end{bmatrix} \qquad K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_M) \\ \vdots & \ddots & \vdots \\ k(x_M, x_1) & \cdots & k(x_M, x_M) \end{bmatrix} + \boldsymbol{I}\sigma^2$$

$$\mathcal{D}_{\text{KL}}\left(\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{K}), \mathcal{N}(\boldsymbol{\mu}, K)\right) = \frac{1}{2}\left(\text{tr}(K^{-1}\tilde{K}) + (\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}})^\top K^{-1}(\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}) + \ln\frac{|K|}{|\tilde{K}|} - M\right)$$

Pre-train with negative log likelihood (NLL)

Task f_1	$(x_1^{(1)}, y_1^{(1)})$	$\cdots$	$(x_j^{(1)}, y_j^{(1)})$	$\cdots$	$(x_{M_1}^{(1)}, y_{M_1}^{(1)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_i	$(x_1^{(i)}, y_1^{(i)})$	$\cdots$	$(x_j^{(i)}, y_j^{(i)})$	$\cdots$	$(x_{M_i}^{(i)}, y_{M_i}^{(i)})$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
Task f_N	$(x_1^{(N)}, y_1^{(N)})$	$\cdots$	$(x_j^{(N)}, y_j^{(N)})$	$\cdots$	$(x_{M_N}^{(N)}, y_{M_N}^{(N)})$

$$f_i \sim \mathcal{GP}(\mu, k)$$

$$\begin{aligned} L(\mu, k, \sigma^2) &= -\log p(D_N \mid \mu, k, \sigma^2) \\ &= -\sum\nolimits_{i=1}^{N} \log p(D_{f_i} \mid \mu, k, \sigma^2) \\ &= \sum\nolimits_{i=1}^{N} \left(\frac{1}{2} \left(\boldsymbol{y}^{(i)} - \mu(\boldsymbol{x}^{(i)}) \right)^\top K^{-1} \left(\boldsymbol{y}^{(i)} - \mu(\boldsymbol{x}^{(i)}) \right) + \frac{1}{2} \log |K| + \frac{M_i}{2} \log 2\pi \right) \end{aligned}$$

Besides success in deep learning,

Pre-training helps Bayesian optimization too

Near-zero regret with an unknown GP prior

Theorem 2. *Let* $N \geq 4 \log \frac{6}{\delta} + T + 2$*. With probability at least* $1 - \delta$*, simple regret in* T *iterations of HyperBO with special cases of either GP-UCB or PI satisfies*

$$R_T < O\left(\sqrt{\frac{1}{N-T}} + (\log \frac{1}{\delta})^{\frac{1}{2}}\right) O(\rho_T / T + \sigma), \qquad (1)$$

where $\rho_T = \max_{A \subset \mathfrak{X}, |A| = T} \frac{1}{2} \log |\boldsymbol{I} + \sigma^{-2} k(A)|$.

- Linear dependency on observation noise as a result of choosing the best observation.
- Pre-training on more tasks leads to better pre-trained model which leads to smaller regret.
- The dependency on T is complicated. More BO iterations push the “posterior” GP away from the ground truth posterior. But we also gain more information by observing more.
- Note that this result only applies to the KL objective and finite search space.

Improved time and memory complexity

		Time	Memory
KL	Overhead	$\mathcal{O}(M^2N)$	$\mathcal{O}(M^2)$
	Loss function	$\mathcal{O}(M^3)$	$\mathcal{O}(M^2)$
	GD	$\mathcal{O}(M^3K)$	$\mathcal{O}(M^2)$
	SGD	$\mathcal{O}(B^2MK)$	$\mathcal{O}(B^2)$
NLL	Loss function	$\mathcal{O}(M^3N)$	$\mathcal{O}(M^2)$
	Parallel	$\mathcal{O}(M^3)$	$\mathcal{O}(M^2N)$
	GD	$\mathcal{O}(M^3NK)$	$\mathcal{O}(M^2)$
	SGD	$\mathcal{O}(B^2MNK)$	$\mathcal{O}(B^2)$

- K: number of optimization steps (or epochs in stochastic optimization).
- B: mini-batch size of SGD over data points per task.
- Typical multi-task GP or contextual GP: O(M^3 N^3)

[Swersky et al., 2013; Bardenet et al., 2013; Poloczek et al., 2016; Yogatama and Mann, 2014]

New benchmark for tuning near-SOTA DL models

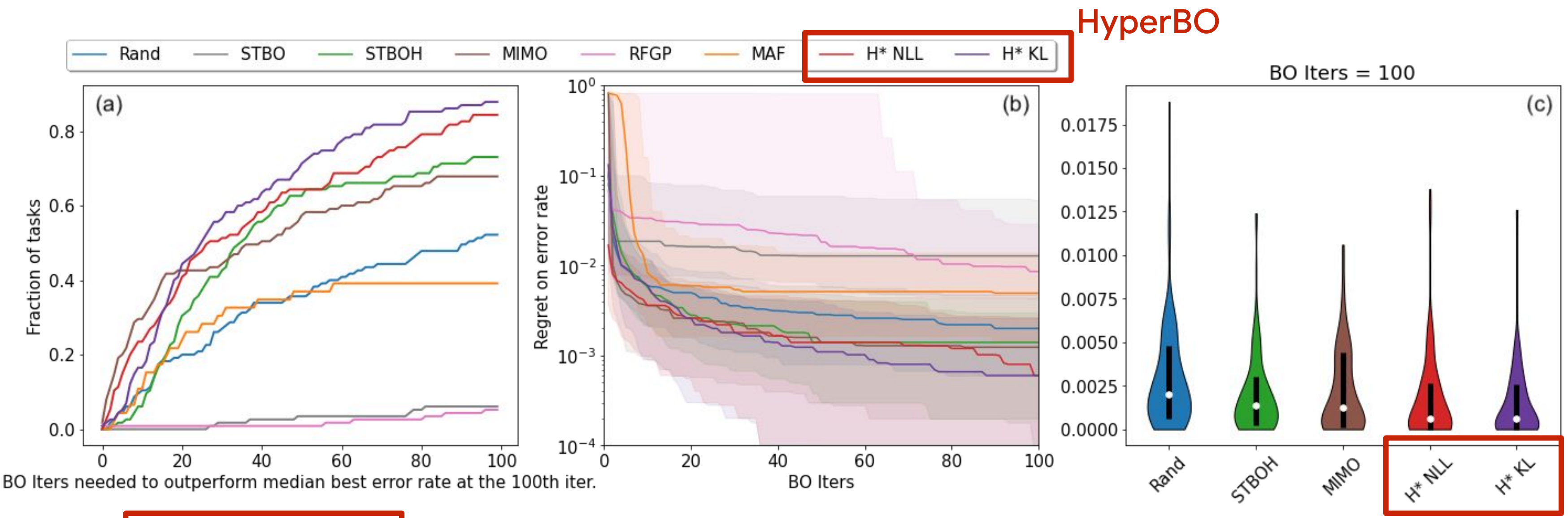
Available at https://github.com/google-research/hyperbo

The PD1 Neural Net Tuning Dataset based on open-sourced models from [Gilmer et al., 2021] https://github.com/google/init2winit

~12,000 machine-days of computation for 50,000 hyperparameter evaluations

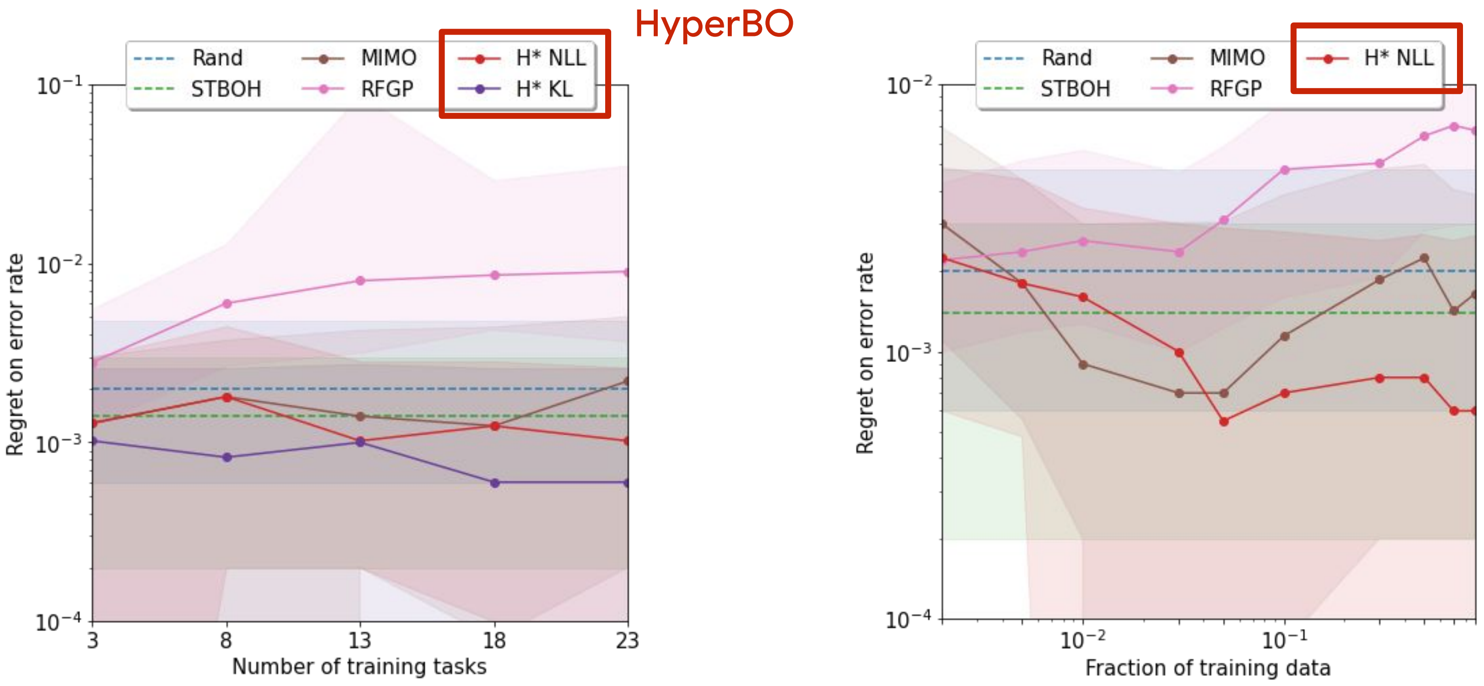
Task Dataset	Model	Batch Sizes
CIFAR10	Wide ResNet	{256, 2048}
CIFAR100	Wide ResNet	{256, 2048}
Fashion MNIST	Max pool CNN ReLU	{256, 2048}
Fashion MNIST	Max pool CNN tanh	{256, 2048}
Fashion MNIST	Simple CNN	{256, 2048}
ImageNet	ResNet50	{512, 1024, 2048}
LM1B	Transformer	{2048}
MNIST	Max pool CNN relu	{256, 2048}
MNIST	Max pool CNN tanh	{256, 2048}
MNIST	Simple CNN	{256, 2048}
SVHN (no extra)	Wide ResNet	{256, 1024}
WMT15 German-English	xformer	{64}
uniref50	Transformer	{128}

3x speed up than the best competing methods

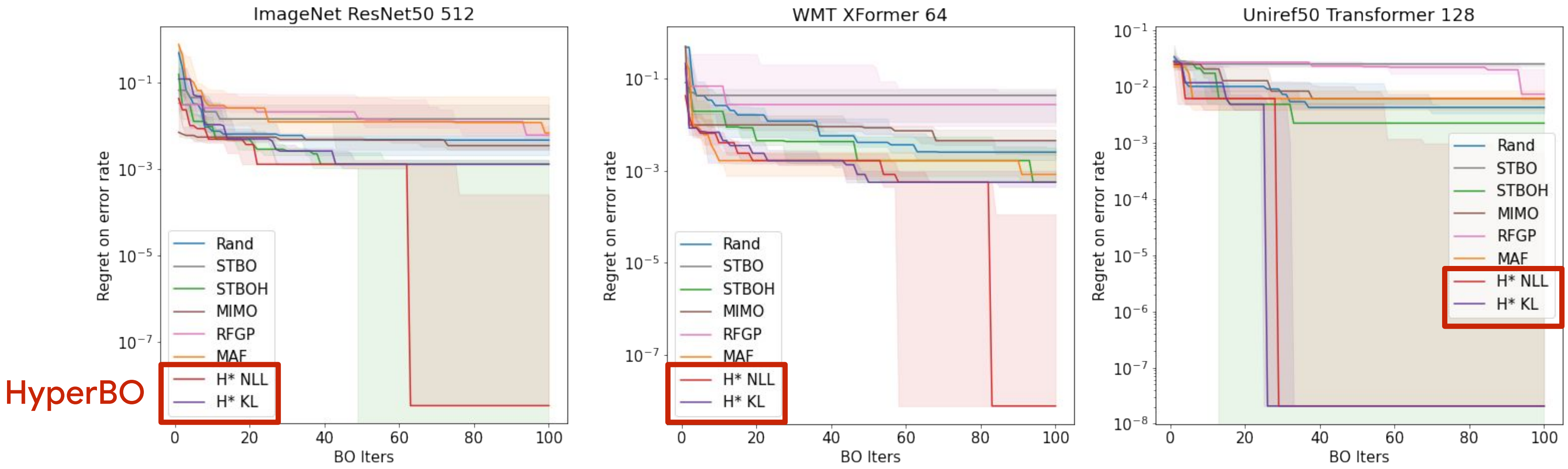


- H* NLL / KL: HyperBO with PI and 1-hidden NN mean function and Matern32 kernel on the same NN.
- STBO: Single task off-the-shelf BayesOpt with type II maximum likelihood (other settings are the same as HyperBO).
- MIMO / RFGP: Contextual BO with ensemble based Bayesian NN [Havasi et al., 2020] or random feature GP.
- MAF: Meta-learning acquisition functions for transfer learning in Bayesian optimization [Volpp et al., 2020].
- STBOH: Single task GP-UCB with hand-tuned priors on hyperparameters including UCB coefficient.

Robust performance with fewer training tasks or training data



Better performance on individual tasks

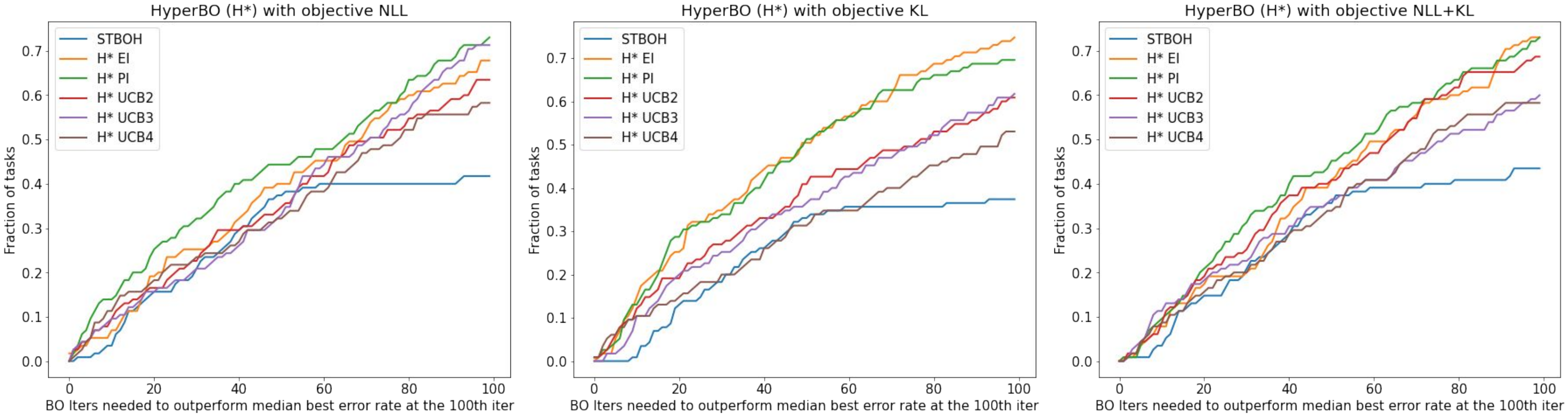


Better NLLs lead to better BayesOpt

	NLL of the test task only			NLL of all tasks		(Pseudo) KL	
Test task	No training	Single task	H* NLL	Single task	H* NLL	Single task	H* NLL
WMT XFormer 64	−301.1	159.1	**−1735.0**	1147900.5	**2264.5**	9651.9	**−40.2**
Uniref50 Transformer 128	−651.7	**−6829.4**	−1850.0	106348128.0	**867.9**	316672.2	**−25.1**
LM1B Transformer 2048	−540.6	**−2009.7**	−1692.7	18840458.0	**3565.7**	57744.1	**−23.5**
SVHN WRN 1024	9703.1	72407.5	**4267.1**	3399330.0	**9346.5**	4677.9	**−0.9**
SVHN WRN 256	10770.0	53245.5	**3794.8**	1164804.5	**9346.5**	3092.7	**−0.9**
ImageNet ResNet50 256	1196.7	7483.0	**−746.3**	7925583.5	**−74.2**	15028.1	**−30.6**
ImageNet ResNet50 512	1300.2	6930.3	**−673.1**	1778823.5	**−74.2**	9462.1	**−30.6**
MNIST CNNPoolTanh 2048	10079.7	38871.9	**794.8**	1375930.1	**97.0**	3165.5	**−32.4**
MNIST CNNPoolTanh 256	12147.7	25607.9	**550.0**	556254.6	**−606.0**	1255.1	**−41.9**
MNIST CNNPoolReLU 2048	26870.5	7149.3	**5506.6**	46538.2	**1542.2**	113.8	**−59.4**
MNIST CNNPoolReLU 256	15601.6	6734.6	**51.0**	88687.7	**−782.2**	361.2	**−41.5**
MNIST CNNReLU 2048	13939.2	40619.2	**3153.2**	743233.1	**−231.4**	877.6	**−61.7**
MNIST CNNReLU 256	10111.0	34412.4	**1365.3**	977295.0	**−779.8**	1373.3	**−46.2**
Fashion CNNPoolTanh 2048	2072.8	11433.0	**−381.0**	1139702.4	**−1051.7**	1910.5	**−37.8**
Fashion CNNPoolTanh 256	2800.7	4115.6	**−251.4**	1278018.0	**−1051.7**	4208.3	**−37.8**
Fashion CNNPoolReLU 2048	4677.4	725.2	**−405.2**	69173.3	**−1051.7**	205.1	**−37.8**
Fashion CNNPoolReLU 256	3925.7	4254.4	**−755.7**	296739.1	**−1051.7**	1027.1	**−37.8**
Fashion CNNReLU 2048	4667.3	6778.1	**251.9**	193488.4	**−1051.7**	597.0	**−37.8**
Fashion CNNReLU 256	3295.1	29348.6	**−235.1**	1526829.2	**−1051.7**	3341.4	**−37.8**
CIFAR100 WRN 2048	1271.5	15813.7	**−467.4**	3306556.5	**312.3**	25593.7	**−19.2**
CIFAR100 WRN 256	1957.6	5950.8	**−510.9**	3468309.0	**11.7**	9288.4	**−25.9**
CIFAR10 WRN 2048	5220.6	4917.6	**832.9**	334488.8	**1127.1**	1040.4	**−14.8**
CIFAR10 WRN 256	7819.1	32995.8	**463.4**	895691.2	**847.4**	1946.0	**−19.6**

- HyperBO pre-trains on 18 irrelevant tasks.
- Both HyperBO and STBO trains on 100 randomly selected data points of the test task.
- NLL on all tasks without training = 148211.2
- KL without training = 2177.2
- STBO causes severe overfitting.
- **HyperBO consistently obtains better NLL on test task, all tasks and KL on matching data.**

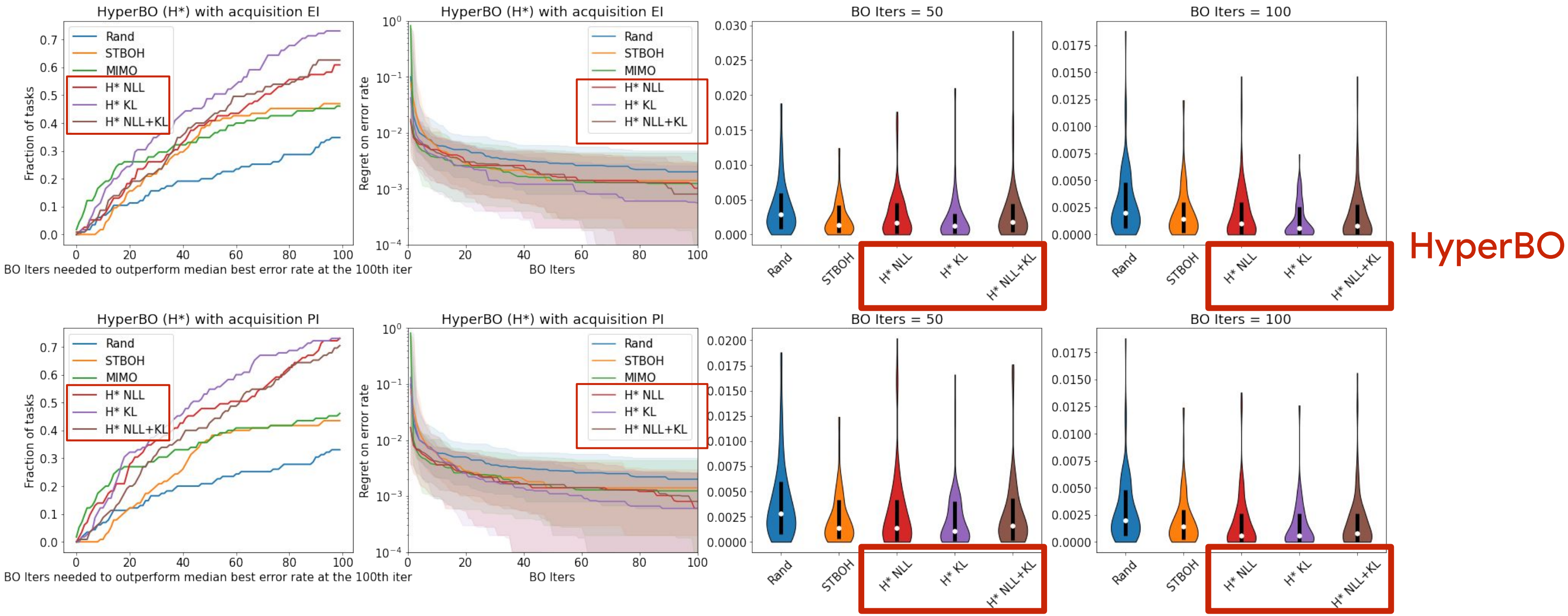
Sensitivity to acquisition functions



PI and EI achieve similar performance.

UCB's performance varies depending on its trade-off hyperparameter.

HyperBO with different objective functions



- KL always performs better than or similar to NLL.
- NLL+KL may have a slight advantage over NLL on EI but the trend gets reversed on PI.
- Overall the three objectives are much better than the best competing alternatives.

Which objective to use in HyperBO?

- Both KL and NLL works in a continuous search space.
- KL assumes same inputs across tasks.
- NLL is a more flexible objective that does not assume same inputs across tasks.
- KL can be easier to interpret: number of extra bits (or nats) to encode a multivariate Gaussian, which approaches 0 as the difference reduces.
- KL on a dataset ≠ NLL on the same dataset. NLL cannot use the matching inputs the same way as KL due to “anonymization” in mean and kernel.
- So, KL if large set of observations on same inputs across tasks. If no same input, use NLL.
- If the number of same inputs is not as large, one may use NLL with KL as regularizer but weights probably need to be tuned.

Open sourced HyperBO code and PD1 tuning dataset

- Code: github.com/google-research/hyperbo
- Data: storage.googleapis.com/gresearch/pint/pd1.tar.gz

Please let us know if you have any questions or encounter any issues by posting to github.com/google-research/hyperbo/issues.

Acknowledgement

“**Pre-training helps Bayesian optimization too**” with George E. Dahl, Kevin Swersky, Chansoo Lee, Zelda Mariet, Zachary Nado, Justin Gilmer, Jasper Snoek, Zoubin Ghahramani.

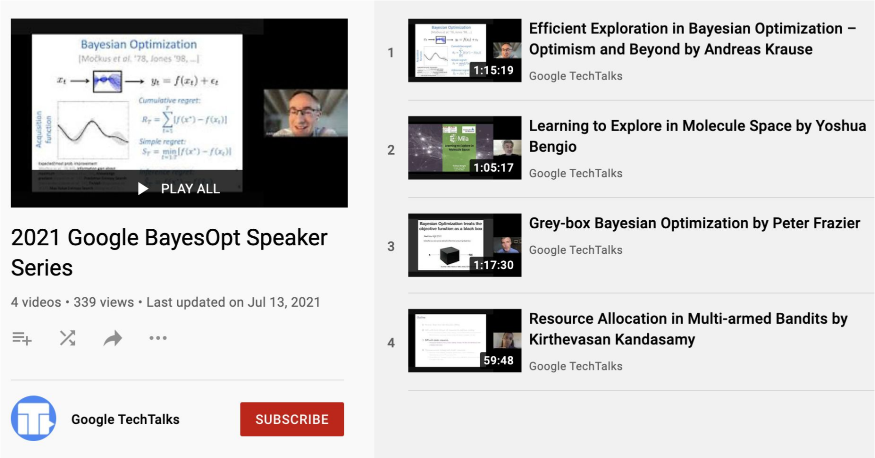
https://ziw.mit.edu/pub/hyperbo.pdf

“**Regret bounds for meta Bayesian optimization with an unknown Gaussian process prior**” with Beomjoon Kim and Leslie Pack Kaelbling.

https://ziw.mit.edu/pub/meta_bo/main.pdf

Some advertisements..

[Google BayesOpt Speaker Series](#)



[gp-seminar-series.github.io](#)

Online performance