End-to-End Meta-Bayesian Optimisation with Transformer Neural Processes

Alexandre Maraval[*1] Matthieu Zimmer[*1]
Antoine Grosnit[1,2] Haitham Bou-Ammar[1,3]

Huawei Noah's Ark Lab[1]
Technische Universität Darmstadt[2]
University College London[3]

November 2023

Table of Contents

Background I - Bayesian Optimisation

In Bayesian Optimisation (BO), the objective is to optimise a *black-box* function $f : \mathcal{X} \to \mathbb{R}$ that is expensive to evaluate (time, cost, ...)

$$x^* = \arg\max_{x \in \mathcal{X}} f(x) \tag{1}$$

We do not have access to gradients so We build a *cheap proxy* of the objective function with a probabilistic model, typically a Gaussian Process (GP) that we use to model f. We explore the space $\mathcal{X}$ with an *acquisition function* $\alpha : \mathcal{X} \to \mathbb{R}$ that evaluates the probability of regions in $\mathcal{X}$ of solving (1), using the GP model. The ability of the GP to give uncertainty estimates allows us to trade off exploration and exploitation during optimisation resulting in a very *sample-efficient* optimisation method.

Background II - Meta-Bayesian Optimisation

BO works *tabula-rasa* which means that each time we have a new function to optimise, we do it from scratch, fitting the surrogate model and using the acquisition to collect points online.
Meta-BO is the setting in which we have access to previous **source tasks** that are similar to the **test tasks** that we want to solve with BO.
Existing *meta-learning* paradigms that exploit the source data include methods that meta-learn

- a model with Supervised Loss [FSBO[1],RGPE[2]]
- a model with RL (end-to-end GP+EI) [DKAF[3]]
- an acquisition function with RL [MetaBO[4]]

None of them learn both end-to-end with RL!

[1] Wistuba and Grabocka [2021]
[2] Feurer et al. [2018]
[3] Iwata [2021]
[4] Volpp et al. [2020]

Background III - Neural Process

Neural Processes (NPs)[5] combine flexibility and model capacity from Neural Networks with the uncertainty prediction capabilities of Probabilistic models such as GPs.
NPs are *meta*-models: conditioned on some *context* data $\mathcal{C} = \{(x^{(c)}, y^{(c)})\}_{c=1}^{C}$ as well as *target inputs* $\mathrm{x}_\mathcal{T} = \{x^{(t)}\}_{t=1}^{T}$, they predict **in a single forward pass** a (Gaussian) distribution of the *target outputs* $\mathrm{y}_\mathcal{T} = \{y^{(t)}\}_{t=1}^{T}$

$$p_\theta(\mathrm{y}_\mathcal{T}|\mathcal{C}, \mathrm{x}_\mathcal{T}) = \mathcal{N}(\hat{\mu}_\mathcal{T}, \hat{\sigma}^2_\mathcal{T})$$

where the NP model is a neural network parameterised by θ that outputs a mean and variance associated to each test input.
We will make use of such models in the context of Meta-BO!

[5] Garnelo et al. [2018]

Table of Contents

Method I - Learning Acquisition Functions with RL

We directly learn to predict acquisition function values, for which there are *no true labels*. We need to use RL!
Formally we have the following MDP

$$\begin{aligned}\textbf{State:} s_t &= [\mathcal{H}_t, t, T] \\ \textbf{Action:} a_t &= \boldsymbol{x}_t \text{ (choice of next probe)} \\ \textbf{Reward:} r_t &= \max_{1 \leq \ell \leq t} y_\ell \text{ (simple regret)}\end{aligned}$$

where $\mathcal{H}_t = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{t-1}, y_{t-1})\}$ is the *history* of collected point so far up to step $t-1$.
The transition function is simply adding the new point and its corresponding function value to the history

$$\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{(\boldsymbol{x_t}, y_t)\}.$$

Method II - Learning Acquisition Functions with RL

We assume we have collected data from multiple source tasks so we extend to multi-task RL. We introduce a set of MDPs $\mathcal{M}_1, \ldots, \mathcal{M}_K$ where all have the same state and action spaces. Formally we have the following MDP

$$\begin{aligned}
\textbf{States:} s_t^{(k)} &= [\mathcal{H}_t^{(k)}, t^{(k)}, T^{(k)} \\
\textbf{Actions:} a_t^{(k)} &= \boldsymbol{x}_t^{(k)} \\
\textbf{Rewards:} r_t^{(k)} &= \max_{1 \leq \ell \leq t^{(k)}} y_\ell^{(k)}, \forall k.
\end{aligned}$$

We now seek a policy π_θ which performs well on average on all K source tasks

$$\arg\max_{\pi_\theta} J(\pi_\theta) = \arg\max_{\pi_\theta} \mathbb{E}_k \left[\mathbb{E}_{\mathcal{H}_T^{(k)}} \left[\sum_{t=1}^{T^{(k)}} \gamma^{t-1} r_t^{(k)} \right] \right]. \tag{2}$$

Method III - Auxiliary Task

In a complete black-box setting, the reward definition as simple regret is the best we can do, but it gives only *sparse information*.
For r_t to contribute to the cumulative reward, y_t must be larger than all previous values observed.
We can quantify the average number of such informative events and find that it is of order $\mathcal{O}(\log T)$.
To improve training of the policy π_θ we introduce an *auxiliary task*.
For each source task k with collected dataset $\mathcal{D}^{(k)} = \mathcal{D}^{(k)}_{\text{obs}} \sqcup \mathcal{D}^{(k)}_{\text{pred}}$, we define the *auxiliary loss* to be the **log-likelihood** of the predicted subset, conditioned on the observed subset

$$\mathcal{L}(\theta) = \mathbb{E}_{k, \mathcal{D}^{(k)}_{\text{obs}}, \mathcal{D}^{(k)}_{\text{pred}}} \left[\log p \left(y_k^{(\text{pred})} | \boldsymbol{x}_k^{(\text{pred})}, \mathcal{D}^{(k)}_{\text{obs}} \right) \right]. \tag{3}$$

Method IV - Neural Acquisition Processes

The architecture used for our policy is a transformer-based Neural Process that predicts acquisition function values directly.
Similarly to NPs it takes as inputs a *context* $\mathcal{H}$ and *query locations*. It is parameterised by θ and we denote its outputs by $\alpha_\theta(\boldsymbol{x}^{(\text{pred})}, \mathcal{H}, t, T)$

$$\pi_\theta\left(\boldsymbol{x}_t^{(\text{pred})} | \mathcal{H}_t, t, T\right) \propto \frac{e^{\alpha_\theta(\boldsymbol{x}_t^{(\text{pred})}, \mathcal{H}_t, t, T)}}{\sum_i^{n_{\text{pred}}} e^{\alpha_\theta(\boldsymbol{x}_i^{(\text{pred})}, \mathcal{H}_t, t, T)}}. \tag{4}$$

The full objective is therefore the sum of both the RL loss (2) and Supervised auxiliary one (3)

$$\mathcal{J}(\theta) = J(\theta) + \lambda \mathcal{L}(\theta)$$

where λ is a hyperparameter.

Pipeline

Algorithm Neural Acquisition Process training.

Require: Source tasks training data $\{\mathcal{D}^{(k)}\}_{k=1}^{K}$, initial parameters $\boldsymbol{\theta}$, budgets $T^{(k)} \equiv T$, discount factor γ, learning rate η

for each epoch **do**

  select task k and dataset $\mathcal{D}^{(k)}$, set $\mathcal{H}_0 = \{\emptyset\}$

  for $t = 1, \ldots, T$ **do**

    $x_t \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}_t)$, $y_t = f^{(k)}(x_t)$ ▷ predict & execute action

    $r_t = y^*_{\leq t}$ ▷ collect reward

    $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t \cup \{(x_t, y_t)\}$ ▷ update hist.

  end for

  $R = \sum_{t=1}^{T} \gamma^t r_t$ ▷ cumul. reward

  $\mathcal{D} \rightarrow \mathcal{D}_{\text{obs}} \sqcup \mathcal{D}_{\text{pred}}$ ▷ split source data

  $\mathcal{L} = p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(\text{pred})}|\boldsymbol{x}^{(\text{pred})}, \mathcal{D}_{\text{obs}})$ ▷ aux. loss

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta(\ \nabla_{\boldsymbol{\theta}} R + \nabla_{\boldsymbol{\theta}} \mathcal{L}\ )$ ▷ update $\boldsymbol{\theta}$

end for

Architecture I

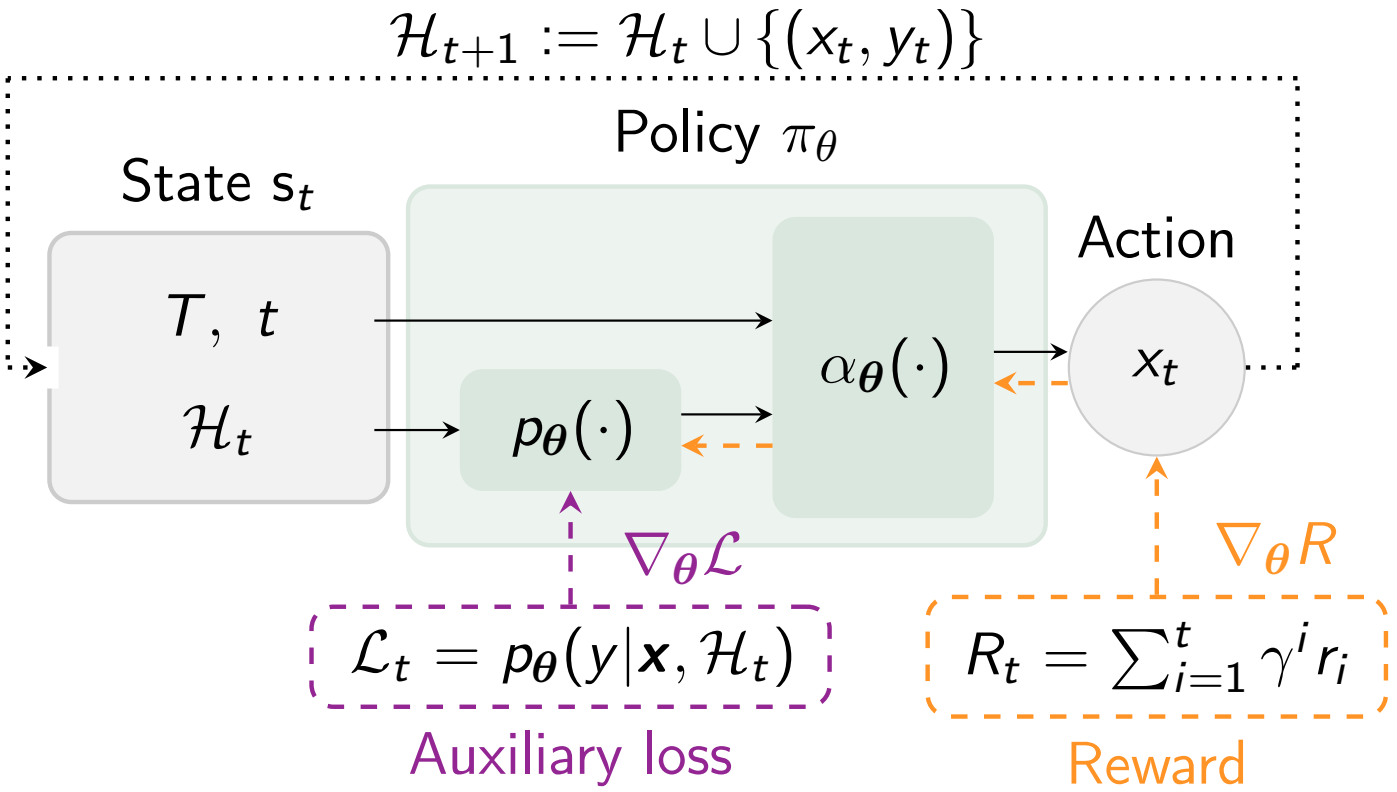

Figure: Summary of our proposed Neural Acquisition Process (NAP) architecture. The action is sampled from the policy $x_t \sim \pi_{\boldsymbol{\theta}}(\cdot|s_t)$. For a set of locations $\boldsymbol{x} \subseteq \mathcal{A}$, the gradients flow back to parameters $\boldsymbol{\theta}$ from both the cumulative regret returns R_t and the auxiliary likelihood loss $\mathcal{L}_t$.

Architecture II - Inputs and attention

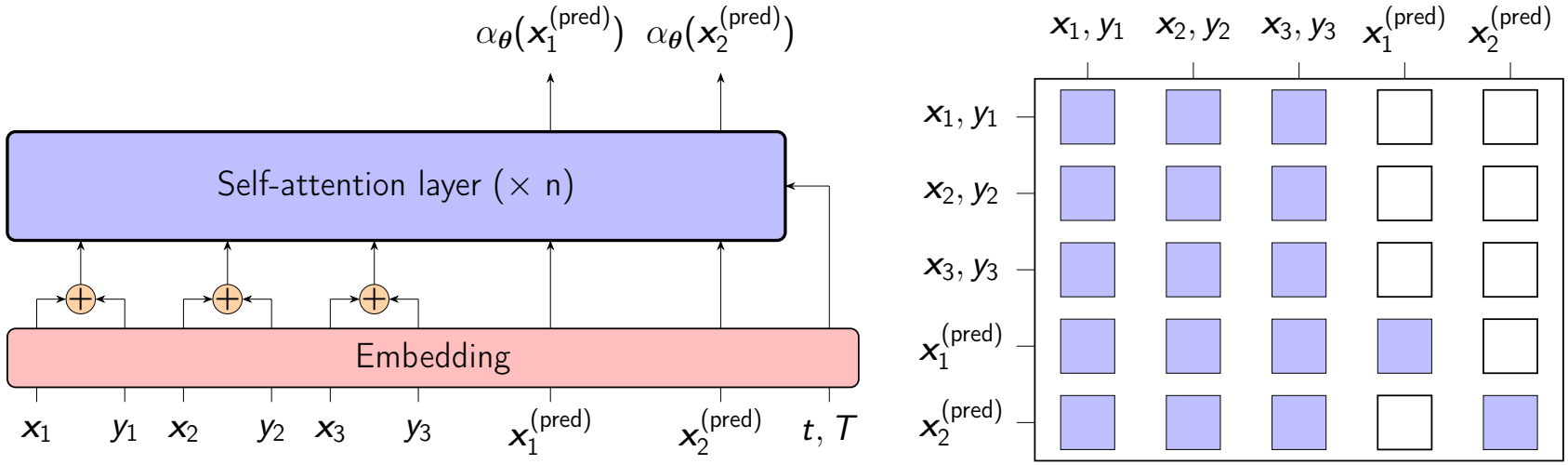

Figure: Our proposed NAP architecture (left) and an example of the masks applied during inference (right). We apply independent embedding on $\boldsymbol{x}_i$, y_i, t and T. The colored squares mean that the tokens on the left can attend the tokens on the top in the self-attention layer.

Architecture III - Comparison to other models

Table: We compare the properties of different transformer architectures. L is the number of tokens needed to encode the meta-data, and D denotes the dimensionality of $\mathcal{X}$.

	NAP (ours)	TNP[1]	OptFormer[2]	PFN[3]
History-order inv.	✔	✔	✘	✔
Query ind.	✔	✘	✘	✔
AF values	✔	✘	✘	✘
Tokens	$t + n_{\text{pred}}$	$t + n_{\text{pred}}$	$L + (D + 2) \times (t + n_{\text{pred}})$	$t + n_{\text{pred}}$

[1] Nguyen and Grover [2022]
[2] [Chen et al., 2022]
[3] [Müller et al., 2022]

Architecture IV - Properties

History-order invariance

An NP g is history-order invariant if for any choice of permutation function ψ that changes the order of the points in history, $g(\boldsymbol{x}, \psi(\mathcal{H})) = g(\boldsymbol{x}, \mathcal{H})$.

We do not use a positional encoding. NAP treats the history $\mathcal{H}$ as a set instead of an ordered sequence. in BO, the order in which we collect points is not relevant for making predictions.

Query independence

A NP g is query independent if for any choice of n queried locations $\boldsymbol{x}^{(\text{pred})} = \boldsymbol{x}_1^{(\text{pred})}, \ldots, \boldsymbol{x}_n^{(\text{pred})}$, we have
$g(\boldsymbol{x}^{(\text{pred})}, \mathcal{H}) = g(\boldsymbol{x}_1^{(\text{pred})}, \mathcal{H}), \ldots, g(\boldsymbol{x}_n^{(\text{pred})}, \mathcal{H})$.

Tokens in $\boldsymbol{x}^{(\text{pred})}$ can access all tokens in $\mathcal{H}$ but cannot access each other through the self-attention mask. Predicting AF values for BO should not depend on the other queried locations.

Table of Contents

Experiments I - Hyperparameter Optimisation

Hyperparameter Optimisation Benchmark

We use a subset of tasks from HPO-B [Pineda-Arango et al., 2021], a standard benchmark. We select a representative set of 6 search spaces and the ones with lowest amount of data to showcase the low data regime performance of NAP.

Tuning MIP Solvers

We tune the hyperparameters of SCIP, a solver for Mixed Integer Programs (MIP). The hyperparameters consist of 135 mixed-type variables. We collect data on 103 source tasks and test on 42 other tasks.

Experiments II - Sequence Optimisation

Antibody CDRH3-Sequence Optimisation

CDRH3 is a part of the amino-acid (AA) sequence representing a protein, decisive in its binding properties. It is a sequence of 11 AA out of an alphabet of cardinality 22. The goal is to optimise the CDRH3 to minimise the binding energy of a given antibody protein with a specific antigen. We collect datasets of CDRH3s and their respective binding energies (with `Absolut!`) across various antigens, 109 to train and 16 to test.

Electronic Design Automation (EDA)

Logic Synthesis (LS) is an essential step in the EDA pipeline of chip design process. A sequence of logic synthesis operators is used to optimise the And-Inverted Graph (AIG) that represents a circuit. Finding the sequence that optimises circuit area and delay is of major interest. We consider LS sequences of length 20 from an alphabet of 11 operators. We collected datasets from 30 different circuits and test on 9 new ones.

Experiments III - Results

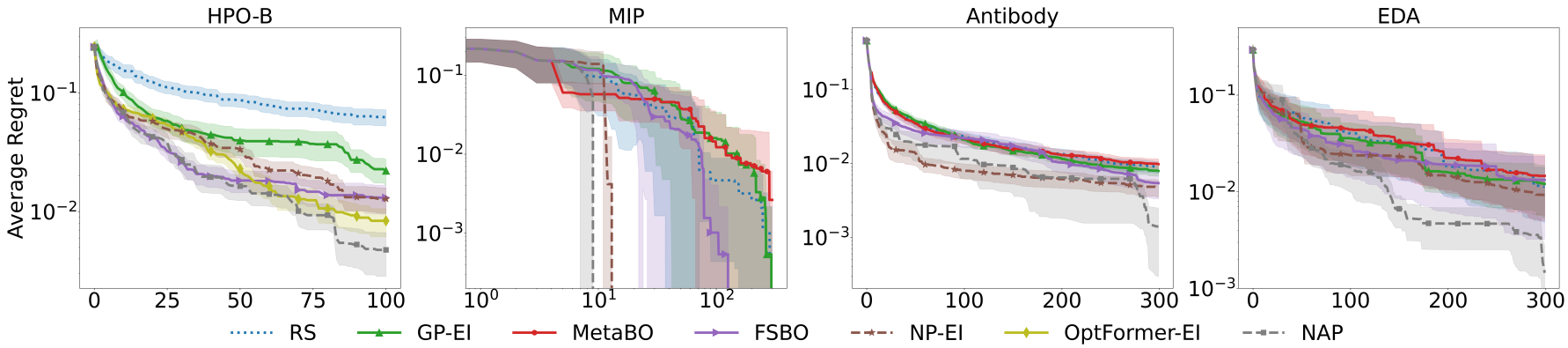

Figure: Average regret vs. BO iterations with 5 initial points. (Left) Results on 6 search spaces on the HPO-B benchmark. (Middle-left) Results tuning SCIP for solving 42 different MIPs. (Middleright) Antibody CDR3 sequence optimisation on 32 test datasets corresponding to 32 different antigens. (Right) Logic synthesis operator sequence optimisation on 9 test datasets corresponding to 9 different circuits. For each method, error bars show confidence intervals computed across 5 runs on HPO-B and 10 runs on all the others.

Ablation Study I

Table: Variations of NAP and their components.

	$p_{\boldsymbol{\theta}}$	$\alpha_{\boldsymbol{\theta}}$	RL	Supervision	End-to-end
NAP (ours)	✔	✔	✔	✔	✔
Pre-NAP	✔	✔	✔	✔	✘
NAP-RL	✔	✔	✔	✘	✔
NP-EI	✔	✘	✘	✔	✘

Ablation Study II

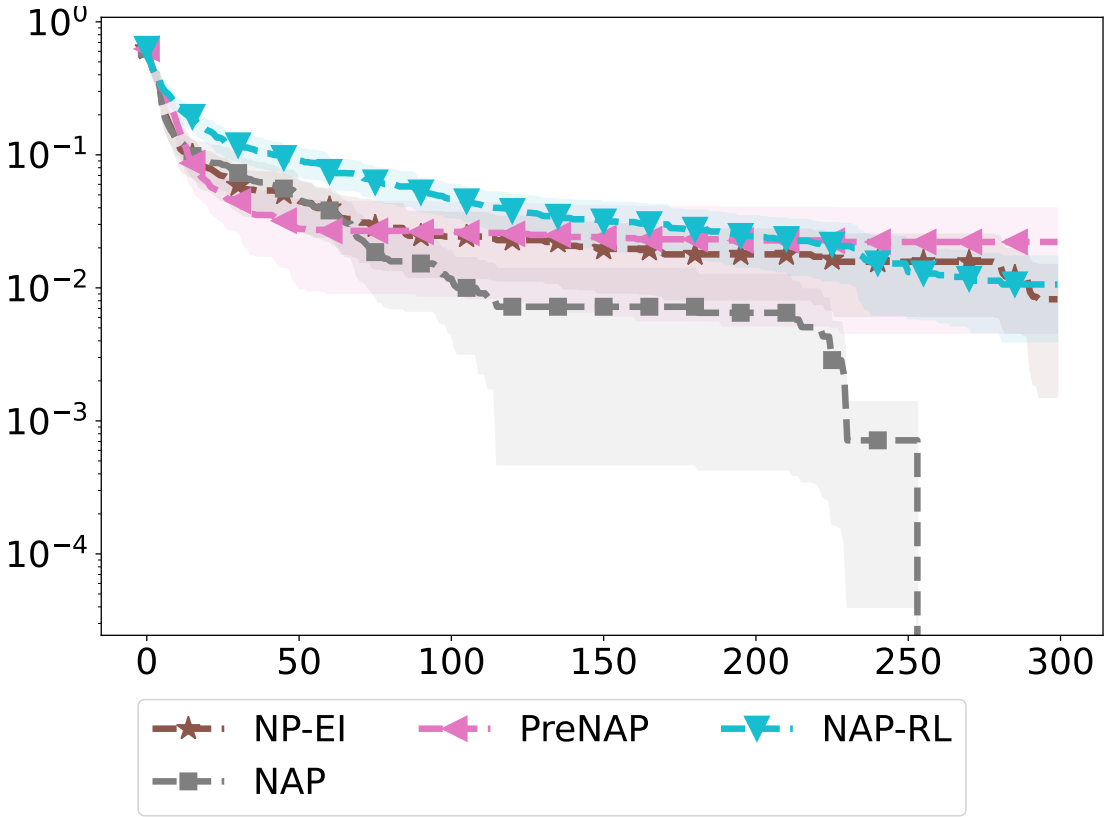

Figure: Average regret vs iterations on HPOBench dataset for XGBoost. Error bars are confidence intervals across ten runs.

Table of Contents

Conclusion

NAP is the first method that can be meta-trained *end-to-end to predict acquisition functions*.
It's applicable to a wide range of domains and gets good results on various tasks, provided little meta-training data.
It is able to outperform other metaBO baselines such as FSBO [Wistuba and Grabocka, 2021] and even bigger models such as Optformer [Chen et al., 2022].

Limitations & Future Work

NAP architecture suffers from the usual quadratic complexity of the transformer in the number of tokens. It can still handle around 5000 steps, which is enough for most BO scenarios.
Another limitation is that we need to train a new model for each search space. In future, we plan to enable our method to leverage meta-training from multiple search spaces.

Table of Contents

References I

Martin Wistuba and Josif Grabocka. Few-shot bayesian optimization with deep kernel surrogates. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=bJxgv5C3sYc.

Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for bayesian optimization using ranking-weighted gaussian process ensembles. In *AutoML Workshop at ICML*, volume 7, 2018.

Tomaharu Iwata. End-to-end learning of deep kernel acquisition functions for bayesian optimization, 2021.

Michael Volpp, Lukas P. Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=ryeYpJSKwr.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Jimenez Rezende, and S. M. Ali Eslami. Conditional neural processes. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1690–1699. PMLR, 2018. URL http://proceedings.mlr.press/v80/garnelo18a.html.

References II

Tung Nguyen and Aditya Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 16569–16594. PMLR, 2022. URL https://proceedings.mlr.press/v162/nguyen22b.html.

Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc Aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards learning universal hyperparameter optimizers with transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 32053–32068. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/cf6501108fced72ee5c47e2151c4e153-Paper-Conference.pdf.

Samuel Müller, Noah Hollmann, Sebastian Pineda-Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=KSugKcbNf9.

References III

Sebastian Pineda-Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on openml. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL `https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/ec8956637a99787bd197eacd77acce5e-Abstract-round2.html`.