Generative HPO with Diffusers

Prof. Dr. Josif Grabocka
University of Technology Nuremberg
Email: josif.grabocka@utn.de

July 5, 2024

1 Introduction

This challenge focuses on tackling the Hyperparameter Optimization (HPO) problem using Generative Models. This exercise is not just about achieving high performance; it's an opportunity to demonstrate your ability to think critically, solve problems, and quickly iterate on ideas to create a Proof of Concept.

2 Method

You will develop and empirically test a novel HPO solver based on Diffusion Models [3] and in-context transformers [6, 4].

2.1 Brief background

A hyperparameter optimization task demands finding hyperparameter configurations $x \in \mathcal{X}$ that maximize the performance of a Machine Learning pipeline. Assume that the performance of an ML pipeline x is measurable as $y = \text{eval}(x)$, which typically involves training an ML model with the hyperparameter configuration x and measuring the validation accuracy y.

As a result, we typically discover the optimal pipelines as a search problem $x^* = \text{argmax}_{x \in \mathcal{X}} \text{evaluate}(x)$. In contrast to the optimization of model parameters which relies on gradients, algorithms for Hyperparameter Optimizations use only black-box evaluations (i.e. evaluate(x)) because the gradient of the validation accuracy concerning the hyperparameter configuration x is not easily computable. Bayesian optimization techniques are a common black-box strategy for optimizing hyperparameters of Machine Learning models and use a performance estimator combined with an acquisition function. For further reading on Bayesian optimization in the context of HPO, please see [2].

Such HPO methods start with an initial set of evaluations, called the history $H = \{(x_i, y_i)\}_{i=1}^{K}$ and then recommend the next configuration x^{next} to be evaluated. The idea is that the recommended configuration has the highest likelihood of improving the best-observed performance (e.g. validation accuracy)

compared to the evaluations in H. In other words, the HPO method recommends a better configuration than all the observed ones concerning predictive performance.

2.2 Method to be developed

In this challenge, you will implement a new type of black-box optimization technique for HPO based on Difussion Models as a generative model [3].

2.2.1 Sampling a supervised dataset from the history

To train a Diffusion model for HPO, we will need to define a supervised dataset from the history of observations $H = \{(x_i, y_i)\}_{i=1}^{K}$. Our supervised dataset consists of triples (x, C, I) which are derived by sampling observations from H.

More concretely $(x, I = 1, C)$ means that the configuration x achieves a validation accuracy y which is higher than all the evaluated configurations in the set $C \subseteq H$. Similarly, $(x, I = 0, C)$ means the accuracy y of the configuration x is **not** higher than all the observations in C. We can therefore, sample x, I, C instances from H as follows:

- Sample a single evaluation from the history $(x, y) \sim H$
- Sample a subset C of evaluations from H, called the context set $C \subseteq H, C \sim H$
- Define a binary variable $I \in \{0, 1\}$ indicating whether the accuracy y of x is higher than all the accuracies of the configurations in C, i.e.:

$$I = \begin{cases} 0 & \exists (x', y') \in C \text{ s.t. } y' > y \\ 1 & \nexists (x', y') \in C \text{ s.t. } y' > y \end{cases} \tag{1}$$

The above procedure for sampling one instance x, I, C from H will be denoted as $(x, I, C) \sim p_H$ for brevity.

Let us illustrate the aforementioned sampling procedure. Assume we are tuning the initial learning rate x for optimizing neural networks from a range $x \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. Assume we have a history of evaluated learning rates as $H = \{(10^{-2}, 0.95), (10^{-4}, 0.97), (10^{-5}, 0.9)\}$, where, for instance, $(10^{-2}, 0.95)$ means that we trained a neural network with a learning rate of 10^{-2} and the achieved validation accuracy was 95%. We can sample triples (x, I, C) from the history as illustrated in Table 1.

2.2.2 Fitting a Denoising Diffusion generative model

At the heart of this challenge lies the training of a generative model that generates configurations x based on a context set of evaluations C, and a known effect (either improve $I = 1$ or not improve $I = 0$) on the validation accuracy.

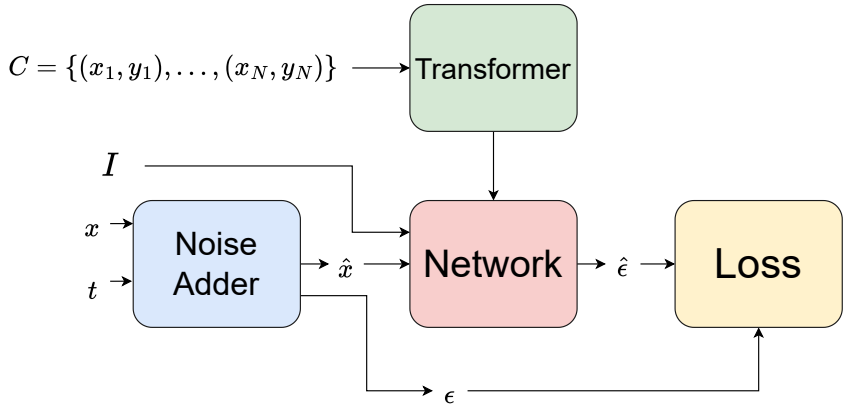

Figure 1: Illustration of Diffusion Model training (Iterative).

x	$I \in \{0, 1\}$	$C \subseteq H$
10^{-2}	$I = 1$	$\{(10^{-5}, 0.9)\}$
10^{-5}	$I = 0$	$\{(10^{-2}, 0.95), (10^{-4}, 0.97), (10^{-5}, 0.9)\}$
10^{-4}	$I = 1$	$\{(10^{-2}, 0.95), (10^{-5}, 0.9)\}$
10^{-5}	$I = 0$	$\{(10^{-2}, 0.95), (10^{-4}, 0.97)\}$
$\vdots$	$\vdots$	$\vdots$

Table 1: Sampled instances from $H = \{(10^{-2}, 0.95), (10^{-4}, 0.97), (10^{-5}, 0.9)\}$

You are going to train a Denoising Diffusion Probabilistic Model (DDPM) to minimize the error between the expected noise ϵ and the predicted noise $\hat{\epsilon}$ at timestep t given a total of T timesteps:

$$\theta^* = \text{argmin}_\theta \, \mathbb{E}_{(x,I,C)\sim p_H, t\sim U(0,T)} (\epsilon_t(x) - \hat{\epsilon}_t(\hat{x}, I, C; \theta))^2 \tag{2}$$

An overall suggestion for the DDPM architecture is illustrated in Figure 1. You are allowed to improve the architecture if necessary. The full details of the architecture and the training pipeline for fitting DDPM are not provided, because that is exactly the essence of the challenge. However the algorithm for training should have the following functions. Please stick to this specified name convention when coding.

- **Sampler**: Given a dataset of performances H, obtain randomly: a subset C, a query timestep $t \in [0, T]$, a query point x, an improvement flag I such that $I = 1$ if $y(x) > y(x')$ for $(x', y(x')) \in C$, otherwise $I = 0$.
- **Noise Adder**: receives the configuration x and adds noise given the

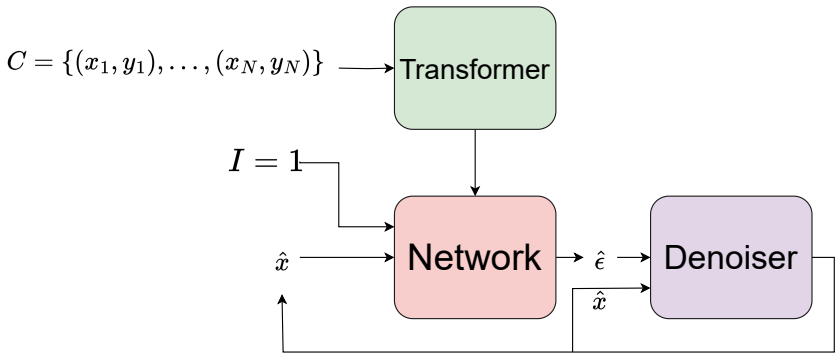

Figure 2: Illustration of Diffusion Model Inference (Iterative).

parameter t, according to the DDPM formulation [3]. The output is the noisy configuration $\hat{x}$.

- **Network**: receives the noisy configuration and its flag I, plus an embedding vector of the context C. The embedding vector is generated by a **Transformer**. The output is the noise in the configuration $\hat{\epsilon}$.
- **Loss**: it compute the loss function in Equation 2, given the actual (ϵ) and predicted noise ($\hat{\epsilon}$).

After computing this loss, you can perform gradient descent to optimize the parameters θ from the network. You might need to execute several steps of gradient descent until the loss gets low values.

2.2.3 Generative HPO Search: Inference with the Difussion Models

The trained generative DDPM will be used as an HPO policy by sequentially recommending the next configuration through a generation/sampling process. In contrast to standard Bayesian optimization, we do not have both a surrogate and an acquisition phase [2]. We need to run the denoising step (Figure 2) for several iterations (e.g. 1000 iterations).

The inference step comprises three components (Transformer, Network, Denoiser) that should be programmed as functions:

- **Network**: it is the network trained in the previous section. The difference now is in the inputs. Initially, it starts with random noise as query input $\hat{x}_0 \sim \mathcal{N}(0, I)$ sampled from a multivariate gaussian distribution. Subsequently, the input will be the denoised value $\hat{x}$. The context C is the set of observed configurations, while $I = 1$ is constant embedded by a **Transformer**. The output of the network is the predicted noise $\hat{\epsilon}$.

- **Denoiser**: it subtracts the predicted noise from the configuration $\hat{x}$ that we are iteratively denoising. The output is an "intermediate" denoised configuration.

Notice that the **Denoiser** and **Noise Adder** modules implement the specific equations from the original paper [3]. After some denoising steps (e.g. 1000), we should obtain a configuration $\hat{x}$ with a high value $y(\hat{x})$ compared to the elements in the set C.

In this project, you are responsible for implementing the HPO search technique with the DDPM generative model and evaluating it on HPO tasks (detailed below).

3 Evaluation Protocol

Your task encompasses both the implementation and testing of the proposed method, utilizing the HPO-B benchmark [5], a widely recognized standard in HPO problem evaluation. HPO-B offers an extensive range of search spaces, datasets, and baseline performances as a testing foundation. To guide your Proof of Concept, we supply you with a specific search space, and dataset, thereby concentrating your efforts on a singular task to enable a thorough and focused exploration.

- Use the interface at `https://github.com/releaunifreiburg/HPO-B` and implement a new HPO Algorithm into the interface by defining a new HPO Algorithm and overriding the "observe_and_suggest" method.
- Use the continuous search space version of the benchmark with surrogates.
- Report results using the following experimental settings:
  - Meta-dataset version: "v3-test"
  - Search space: "5971"
  - Dataset: all available ("10093", "3954", "43", "34536", "9970", "6566")
  - Seeds: all available ("test_0-4")
- You have to report the mean normalized regret along with the average rank for 50 trials, following the experimental protocol in the benchmark paper [5].

have an uncountably infinite number of states.

look at Figure 3

4 Deliverable

In presenting your final empirical results, please use the HPO-B plot format, presented in Figure 3. Your results should include an evaluation of your method in a continuous setting and a comparison with three baselines, specifically Random Search, Gaussian Process [1], and Deep Gaussian Process [7].

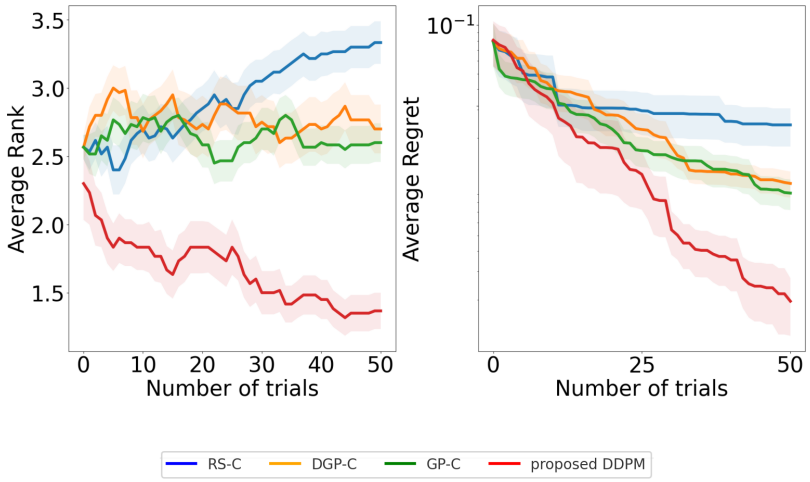

Figure 3: Empirical evaluation of the proposed method vs baselines. The red line is only imaginary and you might get a different HPO performance curve in your experiments.

In addition to your empirical results, we require a concise summary (max 4 pages) detailing your solution, your findings throughout the implementation process, and the final results. This summary should not only present the outcomes but also offer an interpretation of these results. It should reflect on the challenges encountered, the decisions made, and how these influenced the outcome. Your ability to critically analyze and articulate your journey through this task is as important as the technical results themselves, providing valuable insights into your analytical skills.

5 Questions

Finally, please add an answer to the following questions:

1. What is the mathematical formulation for the Noise Adder given the inputs in Figure 1?
2. What is the mathematical formulation for the Denoiser given the inputs in Figure 2?
3. Do we need a transformer architecture with positional encoding? Why?
4. Does the loss function decreases during training? Include a plot.
5. What modification would you do to make the system more efficient?

6 Scoring

The scoring will be based on:

1. Implementation of the training code: [5 Points]
   (a) Network and Transformer [1 Point]
   (b) Noise Adder [2 Points]
   (c) Loss and training loop [2 Points]
2. Implementation of inference code: [3 points]:
   (a) Denoiser [2 Points]
   (b) Inference loop [1 Point]
3. Answers to Questions in Section 5. [2 Points]

7 Submission LAST DATE FOR QUESTIONS: BEFORE DEADLINE

Please submit the source code of your implementation, the concise summary, answers to questions and the result plots as a zip file.

References

[1] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[2] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges, 2021.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[4] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019.

[5] 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. *Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[7] Martin Wistuba and Josif Grabocka. Few-shot Bayesian Optimization with Deep Kernel Surrogates. *Ninth International Conference on Learning Representations*, 2021.