NEURAL ZEROTH-ORDER OPTIMIZATION: EMPIRICAL STUDY AND INSIGHTS

Akhil Nadigatla
Advised by Biswajit Paria, Arun Suggala, Pradeep Ravikumar

Background

Consider the problem of minimizing a function f over a set $\mathcal{X} \subseteq \mathbb{R}^d$

$$f^* = \min_{x \in \mathcal{X}} f(x).$$

Suppose we only have access to the function via a black-box (zeroth-order) oracle, which outputs a noisy estimate of the function when queried at any $x \in \mathcal{X}$

$$y = f(x) + \xi$$

where ξ is a mean-zero random variable. The goal in zeroth-order optimization (ZOO) is to find an *approximate* minimizer of f while making as few queries to the oracle as possible. Hence, any ZOO algorithm makes a sequence of queries $x_1, \ldots, x_T$ and outputs some point $\hat{x}_T$ as a minimizer of f.

The performance of any ZOO technique is typically measured using of the following optimality criteria

$$f(\hat{x}_T) - f^* \qquad \text{(Simple Regret)}$$

$$\frac{1}{T}\sum_{t=1}^{T}(f(x_t) - f^*) \qquad \text{(Cumulative Regret).}$$

Applications

ZOO problems arise in a number of fields. For example, in machine learning, ZOO techniques are often used for **hyper parameter tuning**, where we need to tune the several 'knobs' of a statistical model's architecture and identify the configuration which provides the best model [1]. Similarly, the problem of **neural architecture search** can also be formulated as a ZOO problem [2].

ZOO problems also appear in **robust machine learning**, where an adversary tries to make imperceptible changes (perturbations) to the inputs of a neural network with the goal of making the network misclassify its inputs [3, 4]. Defending against such attacks often requires ZOO techniques that can efficiently identify the worst possible perturbation for any given input, which in turn is used to train robust neural networks.

Another application of ZOO is in **engineering design** where these techniques help expedite the search for promising designs [5]. More recently, ZOO algorithms have even been used to design better culinary recipes [6]!

Existing Techniques

Existing techniques for solving zeroth-order optimization problems can be broadly be categorized into:

1. **Model-based:** construct a surrogate model to approximate and optimize f.
2. **Model-free:** perform random walks through the search space.

Our focus in this work was model-based approaches, which can further be broken down as ones:

1. **Assuming structure:** assume that f is linear, (strongly) convex, etc.
2. **Assuming no structure:** make no major structural assumptions and/or minimal ones like Lipschitz smoothness.

Both these categories have their own downsides: structural assumptions made by the former often do not hold in practice, while techniques in the latter usually have high sample complexity.

Neural networks can get the best of both worlds. Current neural ZOO techniques extend the classic Upper Confidence Bound (UCB) and Thompson Sampling (TS) approaches used in bandit optimization. The problem with these is that they either require posterior sampling or the construction of confidence bands for neural network predictions - which are non-trivial or computationally expensive.

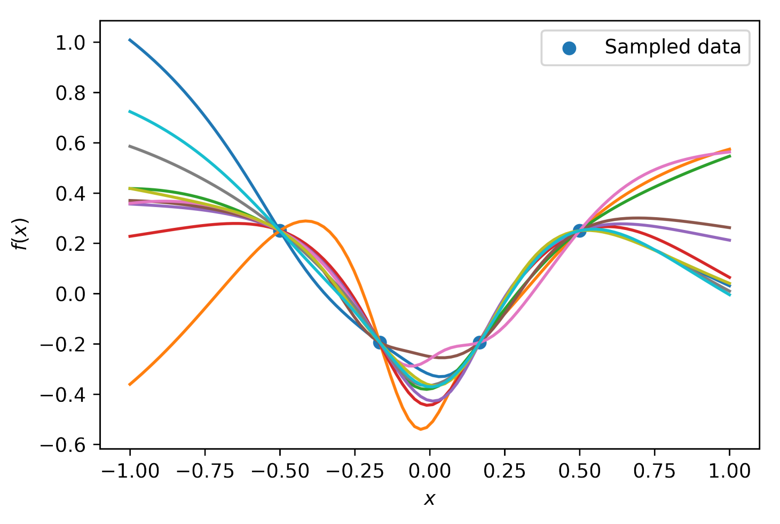
Algorithm & Motivation

We propose a simple, greedy algorithm that, at each round $t \in \{1, \ldots, T\}$ fits a neural network that yields zero square loss *i.e.* solving the following ℓ_2-regularized objective

$$\min_{\theta_t} L(\theta_t) = \sum_{t'=0}^{t}\left(-\left(f_{\theta_{t\|t'}}(x_{t'}) - f(x_{t'})\right)^2 + \|\theta_{t\|t'}\|_2^2\right).$$

```
Algorithm: Greedy
Input: Hyperparameter $\sigma^2$, budget $T$.
Initialize $\mathcal{D}_0 = \emptyset$.
Determine $t_e$ using $T$.
for $t = 1, 2, \ldots t_e$ do
    Sample $x_t$ randomly from the domain.
    Query $f$ at $x_t$ and obtain $y_t$.
    Update $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$.
end
for $t = t_e, t_{e+1}, \ldots, T$ do
    Initialize $\theta_{t,0}$.
    Train neural network $\theta_t$ on $\mathcal{D}_{t-1}$.
    Find $x_t = \mathrm{argmin}_x\, f(x; \theta_t)$.
    Query $f$ at $x_t$ to obtain $y_t$.
    Update $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$.
end
```

Our goal with this algorithm is *simulate* posterior sampling as is the case in Bayesian optimization techniques like Gaussian processes. There are multiple possible neural networks that can fit the data as per the aforementioned objective (analogous to the multiple samples that can possibly be drawn from a posterior distribution). The value of the hyperparameter σ^2 (initialization variance for the neural network weight parameters) determines the nature of the final fit achieved, and acts as an implicit regularizer.



The benefits of using neural networks for this procedure are (1) we avoid defining a prior (doing which often requires extensive domain knowledge for good performance), and (2) Gaussian processes are known to perform poorly in high dimensional settings, contrary to neural networks [7].

Future Work

Some interesting directions to pursue following our findings are:

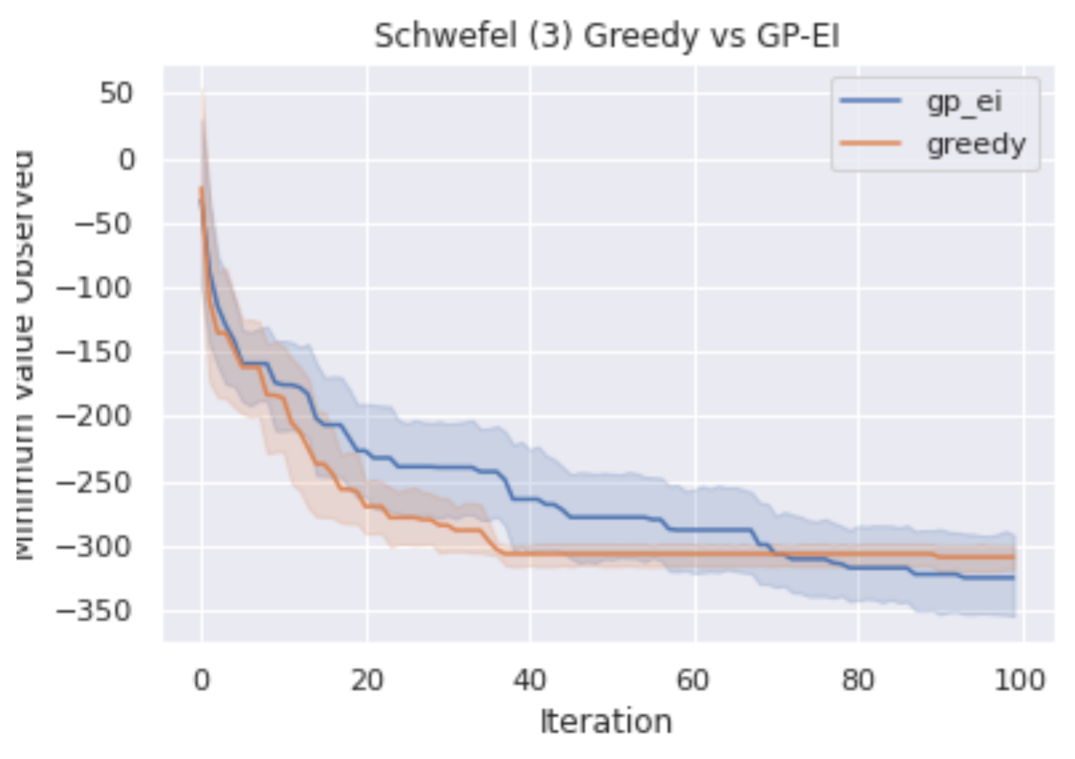
1. **Tuning** σ^2 - the value of this hyperparameter determines the smoothness of the neural network (smaller values lead to smoother fits). This has a great impact on the quality of our surrogate approximation. Our implementation uses brute-force grid search. Ideally, we desire an automatic, more principled way to identifying the best value. For example, we could play a meta-bandit problem on top of our algorithm, solved using an Exp3-style approach [10].
2. **Regret bounds** - we need to provide explicit regret bounds that would allow for easier comparison with more baselines. Existing complexity measures like Eluder dimension [11] may be insufficient given the discrepancy witnessed between theory and our empirical performance.
3. **Benchmarking** - our algorithm should be compared with a wider variety of baseline techniques in order to fully understand its merits (and flaws).
4. **Real-world experiments** - we need to test our algorithm on real-world datasets, moving beyond synthetic experiments.

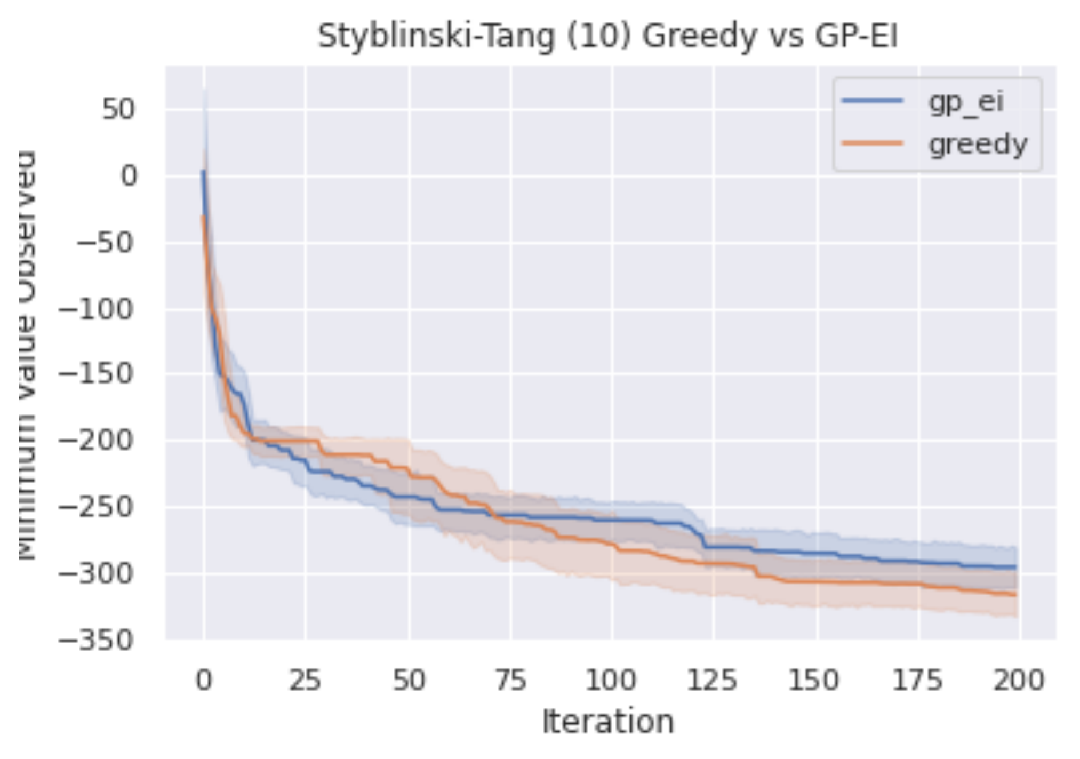
References

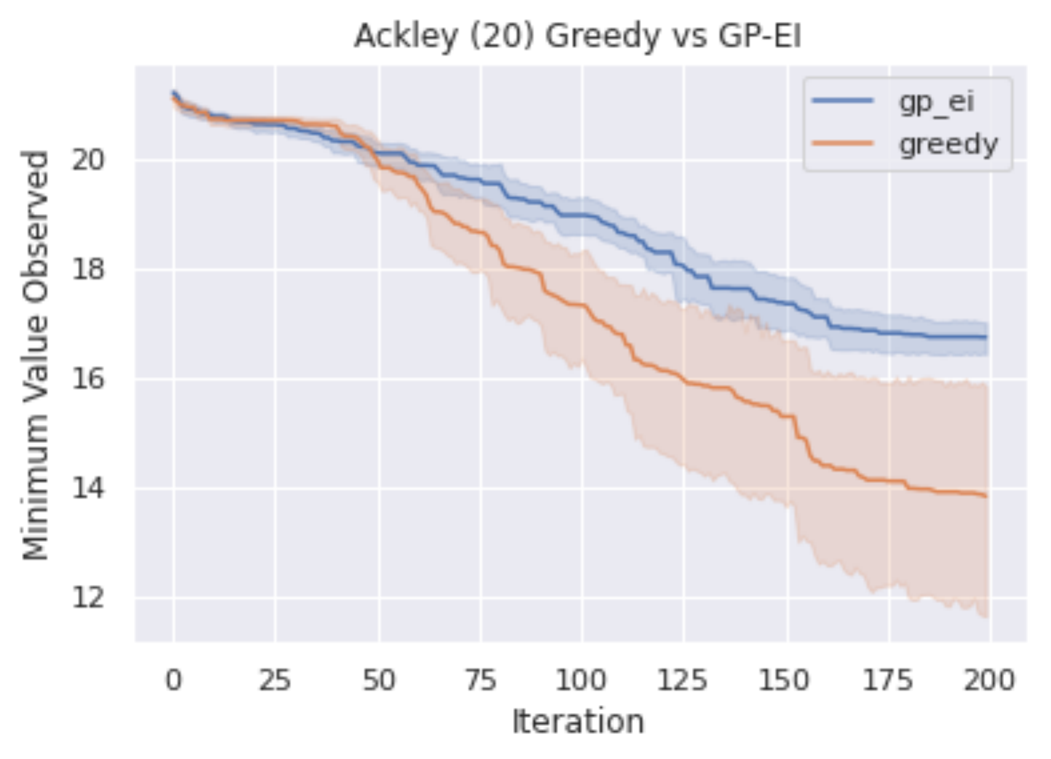
[1] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems, 25.*
[2] Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., & Xing, E. P. (2018). Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems, 31.*
[3] Bhagoji, A. N., He, W., Li, B., & Song, D. (2018). Practical black-box attacks on deep neural networks using efficient query mechanisms. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 154-169).
[4] Liu, S., Chen, P. Y., Kailkhura, B., Zhang, G., Hero III, A. O., & Varshney, P. K. (2020). A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, *37*(5), 43-54.
[5] Sobester, A., Forrester, A., & Keane, A. (2008). Engineering design via surrogate modelling: a practical guide. John Wiley & Sons.
[6] Solnik, B., Golovin, D., Kochanski, G., Karro, J. E., Moitra, S., & Sculley, D. (2017). Bayesian optimization for a better dessert.
[7] Verleysen, M. (2003). Learning high-dimensional data. Nato Science Series Sub Series III Computer And Systems Sciences, 186, 141-162.
[8] Surjanovic, S. & Bingham, D. (2013). Virtual Library of Simulation Experiments: Test Functions and Datasets. Retrieved April 28, 2022, from http://www.sfu.ca/~ssurjano.
[9] Bull, A. D. (2011). Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, *12*(10).
[10] Lu, Z., Xia, W., Arora, S., & Hazan, E. (2022). Adaptive Gradient Methods with Local Guarantees. *arXiv preprint arXiv:2203.01400.*
[11] Russo, D., & Van Roy, B. (2013). Eluder dimension and the sample complexity of optimistic exploration. *Advances in Neural Information Processing Systems, 26.*

Results

We present results from a subset of the experiments run on synthetic functions commonly used for black-box optimization benchmarking [8]. The comparison is between our algorithm above with Gaussian processes with the expected improvement criterion for optimizing the acquisition function (GP-EI) [9].







As the dimensionality and complexity of the synthetic function increases, we generally see that the greedy technique outperforms GP-EI in terms of simple regret. There are instances where the greedy algorithm plateaus early - falling into a local minima and failing to exit it. A possible solution to this could be to add noise during the neural network optimization so as to encourage exploration. In terms of cumulative regret, the experiments we conducted were unable to assign a clear winner. GP-EI, unfortunately, does outperform our approach when it comes to wallclock time (running on a CPU). The bottleneck here is the fresh retraining of the neural network in each step: an alternative to explore here could be to perform a warm-start and/or only train the neural network fully at regular intervals.