Supplementary Materials: Generative Evolutionary Strategy for Black-box Optimization

Anonymous Author(s)
Affiliation
Address
email

This appendix elaborates on the algorithm design details and additional experimental results of the Generative Evolutionary Strategy for Black-box Optimization, not covered in the main text.

A Model implementation details

A.1 Neural Network

In our research, the selection and design of the neural network type plays a fundamental role as our model is grounded on a surrogate neural network. For this particular study, we used a modification of the Transformer [49] (the multi-head self-attention network) model. This structure is suitable because it can adjust to experiments of different sizes without needing any changes. Another advantage is that it can avoid a spatial correlation problem among variables.

While it is true that Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are beneficial when it comes to dimension expansion, they do come with their share of challenges, notably the spatial correlation problems. In contrast, a problem of attention-based neural network is computational complexity, which escalates to $\mathcal{O}(d^2)$ when the variable size hits d. This often results in Graphics Processing Unit (GPU) memory shortages. Therefore, we had to look for other options. We could either choose a network like CNN, which has complexity $\mathcal{O}(d)$, or find a completely new solution.

To address this, we introduced a strategy we have termed the trunk-branch trick. While this method continues to employ the attention mechanism, it subtly modifies the structure to reduce complexity. First, we create a trunk network. Then, from the trunk, we attach M branch networks. Each of these branches predicts a segment of the total dimension d of the target variable, specifically d/M. The structures of the generator and critic are mirror images of each other, meaning they are symmetrical. For the generator, it starts processing in the trunk and then spreads out to the branches to create the target variable x. The critic works the other way around: its branches take parts of x and bring them back together in the trunk.

Implementing this method effectively brings down the complexity to $\mathcal{O}(d^2/M)$, offering a solution to GPU memory shortages. However, the trunk-branch affects the optimization performance of GEO. The related experiment is described in the Additional experiments chapter.

A.2 Non-dominated sorting

When dealing with scenarios that involve multiple target functions, there is often a competitive interaction between each function's optimal points. Imagine a variable x that increases the value of one function, $f_1(x)$, while it simultaneously reduces the value of another function, $f_2(x)$, and vice versa. In such situations, we use a method known as non-dominated sorting to identify the optimal point.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

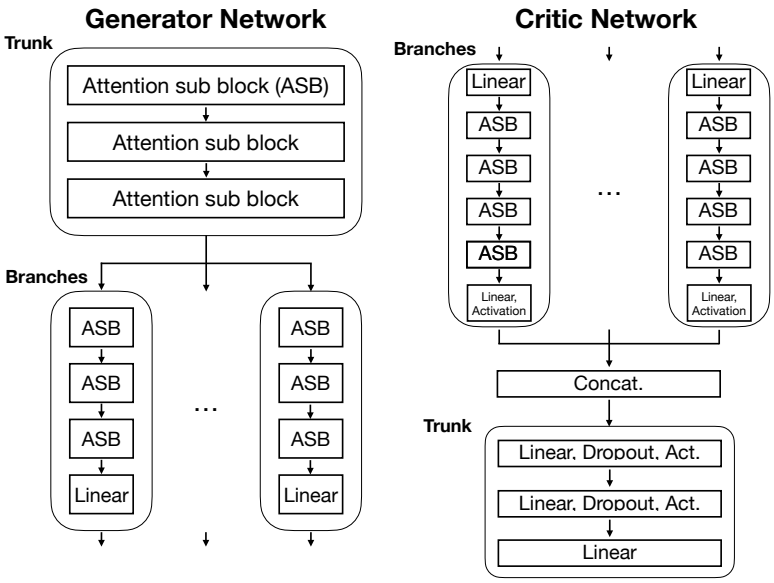

Figure 1: Generator and critic network structure.

Let's look at minimization problems to understand this better. We label a specific point, x_0, as non-dominated when we cannot find another point x that fulfills the condition $f_i(x) < f_i(x_0)$ for all the target objective functions f_i. These non-dominated points form a group known as the first Pareto-front. Consequently, we can arrange these Pareto-fronts in an order, creating a sequence like the 1st Pareto-front, 2nd Pareto-front, and so on. This ordered arrangement is what we refer to as non-dominated sorting.

There are many different methods in non-dominated sorting, each with its own computational complexity. However, we did not place too much emphasis on this aspect, as our pool size was small enough that the time required for non-dominated sorting was considerably less than that required for neural network computations.

A.3 Training details

In the Methods section of the main text, we noted the potential tendency of GEO towards the exploitation with regard to the exploit-explore strategy. This trait primarily emerges from its design, a combination of ES and GSN. Furthermore, this tendency could limit the algorithm's potential for exploration. Therefore, we added several strategies to boost its exploration abilities.

One approach we used was to try various learning rates. We prepared a wide range of learning rates, from small to large values, and applied them either randomly or all at once for each mutation event. This approach can provide an escape path if the algorithm gets stuck in a local optimum. We also considered implementing random mutations, changing the parameters of the sampled generator network or specific layers to create mutants. At the very least, these additional techniques do not compromise performance, although we could not definitively establish that they enhanced it.

We also tested various hyperparameters which directly impact experimental results. For example, the pool size is important because it plays a key role in shaping the Pareto-front.

For certain functions, such as the ZDT functions, it is necessary to set boundaries. The boundary construction method can affect performance. We can find the results of these experiments in the Additional experiments section.

For the training of the generator (mutation), we chose to train in the n separate directions using n independent critic networks. There can be another approach where we sum up n fitness scores and then increase the average score. However, this method could possibly lead to a bias towards points in the middle of the Pareto front. Moreover, this approach introduces new hyperparameters related

to the normalization of each fitness score, which significantly influences the algorithm's behavior. Therefore, to circumvent the introduction of new hyperparameters, we chose to conduct the training n times independently.

B Experiment details

B.1 Non-convex test functions

In our experiment, we analyzed multiple dimensions, each requiring comparative investigation. To make the analysis process consistent, we normalized all test functions in line with their dimension sizes. Most of test functions have the ground state as 0; however, the Styblinski-Tang function does not. Therefore, we recalibrated its value to set 0 as the ground state. Although all these functions fall under the non-convex category, the Rosenbrock function has distinct properties, as its optimal solution resides within a flat valley region

B.2 Computational details

We aimed to run the algorithm with minimal hardware acceleration. Therefore, we designed the algorithm to be compatible with a single GPU. When we faced memory shortages, we employed the trunk-branch trick, a method we previously discussed in the neural networks chapter. By using the NVIDIA Tesla v100 32GB GPU, it took about three days to execute 100,000 function calls and trainings. The majority of this computational time was dedicated to training the neural networks.

We could enhance the efficiency of neural networks by incorporating a CNN or optimizing the attention model. However, given the scope of our task, which involved 100,000 function calls across 8192 dimensions, the model we currently have is sufficient. Thus, we did not devote substantial time to network optimization. Moreover, we have to avoid excessively refining the algorithm for a specific test function. Such over-tuning could lead to overfitting and consequently limit the model's performance on different problems especially for real-world problems.

In our study, we made 100,000 function calls for each task. Although this might seem like a considerable number, it is not particularly large when viewed within the context of machine learning and neural networks. Therefore, we found a smaller model that could run on a single GPU to be the most suitable choice; note that if we increase the model size, we can potentially see a decrease in performance. However, if the number of function calls were to increase significantly, we might require a larger model. This could occur in scenarios where the target black-box is readily parallelizable.

B.3 Further investigations

As we previously noted in the main text, Bayesian optimization forms a vital branch of optimization research. This is particularly the case for the Gaussian process-based Bayesian optimization, a practical method employed in machine learning and hyperparameters optimization. Despite its utility, it is important to note that it comes with an $\mathcal{O}(N^3)$ complexity, which stands as a significant hurdle. A proposed potential solution to this issue is a neural-process. [61] This approach is receiving increasing interest because it offers Gaussian process-like uncertainty estimation while simultaneously reducing time complexity. However, it still demands the accumulation of search point data for uncertainty prediction. This requirement results in a continuous escalation in computation time, preventing it from meeting the $\mathcal{O}(N)$ complexity when employed in optimization tasks.

In the ES domain, numerous modifications have been developed based on well-known and commonly used algorithms such as GA and CMA-ES [54–56]. In reality, though, optimization experiments typically deal with around 100 dimensions. Sometimes, these experiments may handle larger dimensions; in that case however, they usually focus on simpler convex shapes such as the spherical function and Rosenbrock function; note that Rosenbrock has long flat region around the global minimum.

When it comes to experiments in the 100-1000 dimension range, GSN-based optimization studies have given promising results. [29]. However, the test functions used in these studies are often special types of test function rather than common test functions that have been widely used in ES studies, making it difficult to compare performance. In some cases, Generative Neural Network (GNN) is used, but not the surrogate model. A GNN-based study [28] has shown optimization results around 10

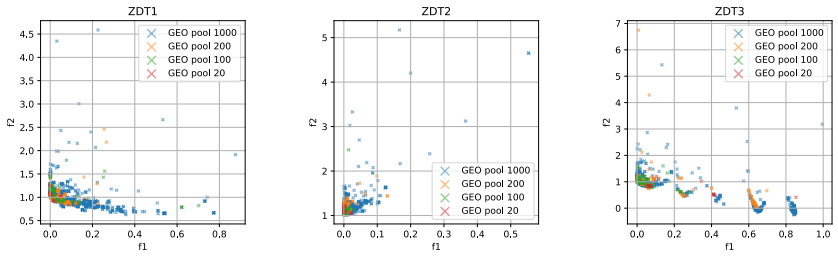

Figure 2: Pool size experiments. Optimization results in 8192 dimension after 100,000 function calls.

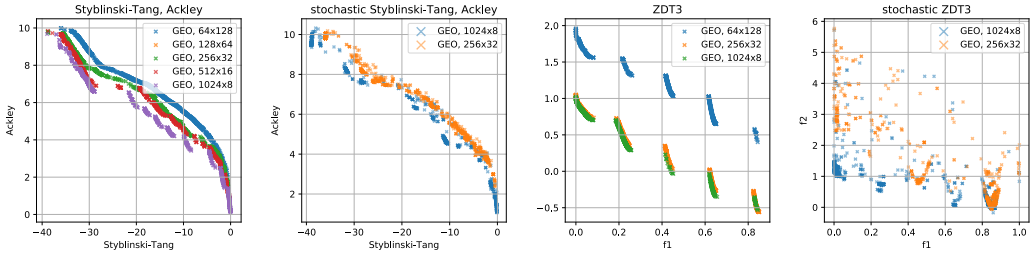

Figure 3: Trunk-branch trick experiments. Optimization results in 8192 dimension after 100,000 function calls. For M branches, $8192 = (8192/M) \times M$.

dimensions; however, this is considerably distanced from our target level of dimensions. Therefore, we did not take this algorithm into account.

Research based on surrogate models is also ongoing. For instance, Pysamoo [57] offers packaged optimization algorithms based on surrogate models. Nonetheless, we found that these offered models cannot be effectively applied in high dimensions due to time complexity problems. However, as the package is regularly updated, it could overcome this problem in the future.

Finally, Particle Swarm Optimization (PSO) [5] is a different type of algorithm from ES, but it has some similar features. Here, a certain number of elements swarm towards the global optimum while preserving their group. Therefore, we can consider how to combine PSO and GSN. However, within the scope of this study, we were unable to devise a way to integrate PSO. While ES provides a simple means to link GSN's backpropagation and non-dominated sorting, establishing a similar connection in PSO poses a challenge.

C Additional experiments

C.1 Pool size

The pool size has a direct impact on performance, especially if it is too small. In the experiment, a small pool GEO encounters problems when identifying the overall shape of the Pareto-front. This problem may arise when the algorithm loses some lines of the Pareto-front in non-dominated sorting, making them difficult to recover.

C.2 Trunk-branch structure

In the neural network section, we mentioned that the trunk-branch trick is a temporary solution to address the problem of GPU memory shortage. Hence, we also explored the performance related to the trunk-branch strategy.

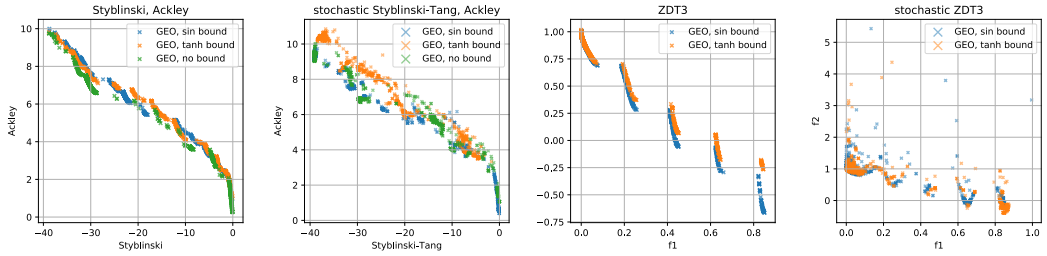

Figure 4: Boundary function experiments. Optimization results in 8192 dimension after 100,000 function calls.

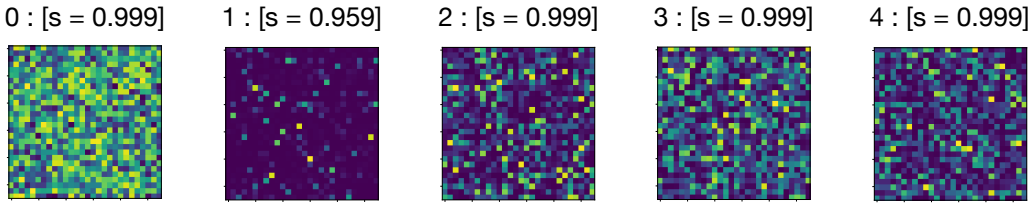

Figure 5: An experiment to black-box optimization of LeNet-5 trained with MNIST. Maximum score $s = 1.0$. Although optimization is successful, it do not produce the intended smooth handwriting image.

The experimental result clearly shows that as the number of branches increases, the performance decreases. The trunk-branch structure enhances the time and space efficiency of the attention network but at the expense of optimization performance. Therefore, these results suggest that there are limits to increasing the number of branches in extremely high dimensions. It might imply the necessity for fundamental changes, such as adopting CNNs.

C.3 Boundary conditions

ZDT test functions require boundary conditions in the search space X. To implement these boundary conditions, we attached an additional function at the end of the generator to enforce boundaries.

$$x = (bound_{max} - bound_{min})\frac{B\left(G(z)\right) + 1}{2} + bound_{min}$$

Boundaries could be implemented with functions such as $B = tanh$ and $B = sin$, with the periodic boundary condition of the sin function showing slightly better results. This might occur due to the use of the $tanh$ function, which could create a bias at the edges, leading to a concentration of points that exceed the boundary at the edge. This, in turn, generates redundant data during the training of the critic network. Hence, if a boundary is necessary, it is recommended to use a periodic boundary condition.

C.4 Manifold issue

In the L-GSO research, it was suggested that the surrogate could effectively discern manifold structures, and that optimization performance would likely improve within manifold structures than without a manifold structure. As GEO employs a similar surrogate neural network-based algorithm, the same circumstances may arise.

To visually verify this, we conducted an image generation experiment. We considered a simple LeNet-5 [44], trained on the MNIST dataset, as a black-box and optimized it. If GEO prefers manifold structures, images created through black-box optimization should exhibit a smooth shape (likely resembling actual human hand-drawn images).

However, the results are contrary. Even though the optimization is successful, a smooth shape does not emerge; instead, it produces a noise image resembling an adversarial attack. This casts doubt on previous research findings suggesting that GSN-based optimization is better suited for learning manifold structures.

Table 1: Optimization results of Ackley function in low dimensions. 20,000 function calls. 10 repeats.

	Ackley			
Dimension	2	4	8	16
GEO	0.0000 ± 0.0000	0.0071 ± 0.0076	0.1009 ± 0.0432	0.3014 ± 0.2451
GEO 1-layer	0.0000 ± 0.0000	0.0030 ± 0.0023	0.8575 ± 0.7513	1.9020 ± 0.2054
GA	0.0001 ± 0.0002	0.0016 ± 0.0008	0.0073 ± 0.0033	0.0411 ± 0.0066
CMAES	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
LSM $\epsilon 1.0$	0.0053 ± 0.0081	0.0261 ± 0.0118	0.1056 ± 0.0292	0.2036 ± 0.1139
LSM $\epsilon 0.2$	0.0006 ± 0.0003	0.6754 ± 1.3312	0.0354 ± 0.0076	0.8967 ± 0.9222
	32	64	128	
GEO	0.0694 ± 0.0576	0.0361 ± 0.0185	0.0296 ± 0.0136	
GEO 1-layer	2.7931 ± 0.1655	3.4449 ± 0.1133	3.8488 ± 0.1002	
GA	0.1132 ± 0.0163	0.2795 ± 0.0304	0.5510 ± 0.0369	
CMAES	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0001 ± 0.0000	
LSM $\epsilon 1.0$	0.2430 ± 0.1160	0.3432 ± 0.1225	0.8251 ± 0.2573	
LSM $\epsilon 0.2$	2.5080 ± 1.2227	3.4657 ± 0.3694	3.3817 ± 0.3004	

Table 2: Optimization results of Rosenbrock function in low dimensions. 20,000 function calls. 10 repeats.

	Rosenbrock			
Dimension	2	4	8	16
GEO	0.0000 ± 0.0000	0.0543 ± 0.1507	0.5090 ± 0.4545	0.5378 ± 0.5966
GEO 1-layer	0.0000 ± 0.0000	0.3062 ± 0.2330	0.8592 ± 0.2276	3.2036 ± 1.7442
GA	0.0001 ± 0.0001	0.0888 ± 0.0488	0.5197 ± 0.1102	0.8267 ± 0.0755
CMAES	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
LSM $\epsilon 1.0$	0.3805 ± 0.3780	0.8514 ± 0.3439	1.3913 ± 0.1683	1.7246 ± 0.1924
LSM $\epsilon 0.2$	0.5814 ± 0.3466	0.5992 ± 0.3083	0.6444 ± 0.3757	0.7882 ± 0.2399
	32	64	128	
GEO	0.1705 ± 0.2817	0.0564 ± 0.0975	0.0164 ± 0.0170	
GEO 1-layer	11.3029 ± 1.3108	32.3401 ± 3.7944	61.4009 ± 4.9140	
GA	1.7519 ± 0.5589	4.0404 ± 0.4313	5.9195 ± 0.2543	
CMAES	0.6532 ± 0.0278	0.9068 ± 0.0150	0.9734 ± 0.0102	
LSM $\epsilon 1.0$	1.8820 ± 0.5490	2.1025 ± 0.7888	2.0884 ± 0.7405	
LSM $\epsilon 0.2$	1.5623 ± 0.7663	17.4755 ± 25.1455	38.9798 ± 23.4691	

Table 3: Optimization results of Rastrigin function in low dimensions. 20,000 function calls. 10 repeats.

	Rastrigin			
Dimension	2	4	8	16
GEO	0.0000 ± 0.0000	0.1027 ± 0.1635	0.1434 ± 0.2483	0.8459 ± 0.6891
GEO 1-layer	0.0000 ± 0.0000	0.3483 ± 0.1219	0.4758 ± 0.2170	1.2868 ± 0.3246
GA	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0010 ± 0.0009	0.0127 ± 0.0048
CMAES	0.2985 ± 0.3300	0.4477 ± 0.2168	0.5721 ± 0.2021	0.4166 ± 0.1446
LSM $\epsilon 1.0$	0.5375 ± 0.4406	3.6183 ± 1.9796	5.6259 ± 1.2943	5.5754 ± 1.3568
LSM $\epsilon 0.2$	0.0000 ± 0.0000	0.5076 ± 0.4908	0.3248 ± 0.3222	0.4995 ± 0.2248
	32	64	128	
GEO	1.8010 ± 1.2062	1.9690 ± 1.3727	0.8947 ± 1.1321	
GEO 1-layer	2.8961 ± 0.3151	4.9785 ± 0.2506	6.4839 ± 0.2165	
GA	0.1580 ± 0.0452	0.5013 ± 0.0478	0.9218 ± 0.0550	
CMAES	0.5006 ± 0.1526	0.5208 ± 0.0961	0.6630 ± 0.0978	
LSM $\epsilon 1.0$	5.1430 ± 1.5115	7.8261 ± 0.8673	8.8664 ± 0.6537	
LSM $\epsilon 0.2$	0.6955 ± 0.3504	4.9844 ± 2.2791	7.5184 ± 1.6464	

Table 4: Optimization results of Styblinski function in low dimensions. 20,000 function calls. 10 repeats.

	Styblinski-Tang			
Dimension	2	4	8	16
GEO	0.0000 ± 0.0000	0.0009 ± 0.0014	0.0023 ± 0.0017	0.0161 ± 0.0176
GEO 1-layer	2.1695 ± 3.2087	8.5127 ± 2.2338	15.9916 ± 1.1828	22.4661 ± 0.5406
GA	0.7068 ± 2.1205	0.7069 ± 1.4137	3.5346 ± 2.3707	5.2253 ± 0.9198
CMAES	7.0684 ± 5.4751	12.7231 ± 3.2391	10.9560 ± 2.5971	9.7190 ± 2.3708
LSM $\epsilon 1.0$	26.2230 ± 1.9996	7.6978 ± 1.2559	6.4853 ± 3.9141	9.5768 ± 4.0347
LSM $\epsilon 0.2$	27.6515 ± 2.8802	16.1667 ± 1.6806	12.1698 ± 2.0510	19.3622 ± 3.1020
	32	64	128	
GEO	0.0064 ± 0.0092	1.4138 ± 4.2410	0.0000 ± 0.0000	
GEO 1-layer	25.3427 ± 0.5938	27.5193 ± 0.6898	29.3574 ± 0.3288	
GA	7.9346 ± 1.0712	11.2910 ± 0.5126	18.3960 ± 0.3724	
CMAES	8.4820 ± 2.2159	9.9841 ± 0.6612	9.2883 ± 0.5663	
LSM $\epsilon 1.0$	9.7605 ± 4.3111	8.7600 ± 2.9911	17.5349 ± 4.3426	
LSM $\epsilon 0.2$	18.9152 ± 2.6356	32.2802 ± 2.4185	33.5116 ± 2.6515	

Table 5: Optimization results of test functions in high dimensions. 50,000 function calls. 10 repeats

	Ackley		
Dimension	256	512	1024
GEO	0.0091 ± 0.0036	0.0117 ± 0.0037	0.0084 ± 0.0029
GA	0.3294 ± 0.0219	0.7342 ± 0.0273	1.4256 ± 0.0477
CMAES	0.0000 ± 0.0000	0.0003 ± 0.0000	0.0291 ± 0.0035
	Rosenbrock		
Dimension	256	512	1024
GEO	0.0006 ± 0.0006	0.0004 ± 0.0003	0.0005 ± 0.0004
GA	5.0726 ± 0.2486	5.9915 ± 0.2358	6.9435 ± 0.1485
CMAES	0.9742 ± 0.0062	1.0011 ± 0.0337	1.0292 ± 0.0301
	Rastrigin		
Dimension	256	512	1024
GEO	0.2034 ± 0.4046	0.0018 ± 0.0020	0.0034 ± 0.0057
GA	0.5636 ± 0.0317	0.9810 ± 0.3457	1.7443 ± 0.0638
CMAES	0.9573 ± 0.1040	1.3981 ± 0.1383	3.7305 ± 0.6978
	Styblinski-Tang		
Dimension	256	512	1024
GEO	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
GA	14.4536 ± 0.3616	22.3592 ± 0.2035	28.8298 ± 0.1541
CMAES	9.6913 ± 0.4508	9.3711 ± 0.1878	9.2048 ± 0.2450

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] Frazier, P. I. (2018). A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*.

[3] Lan, G., Tomczak, J. M., Roijers, D. M., & Eiben, A. E. (2022). Time efficiency in optimization with a bayesian-evolutionary algorithm. *Swarm and Evolutionary Computation, 69,* 100970.

[4] Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The computer journal, 7*(4), 308-313.

[5] Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (Vol. 4, pp. 1942-1948). IEEE.

[6] Hooke, R., & Jeeves, T. A. (1961). "Direct Search"Solution of Numerical and Statistical Problems. *Journal of the ACM (JACM), 8*(2), 212-229.

[7] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation, 6*(2), 182-197.

[8] Deb, K., & Sundar, J. (2006, July). Reference point based multi-objective optimization using evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (pp. 635-642).

[9] Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation, 9*(2), 159-195.

[10] Hansen, N. (2006). The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation*, 75-102.

[11] Price, K., Storn, R. M., & Lampinen, J. A. (2006). *Differential evolution: a practical approach to global optimization.* Springer Science & Business Media.

[12] Runarsson, T. P., & Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on evolutionary computation, 4*(3), 284-294.

[13] Runarsson, T. P., & Yao, X. (2005). Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 35*(2), 233-243.

[14] Deb, K., & Jain, H. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation, 18*(4), 577-601.

[15] Blank, J., Deb, K., & Roy, P. C. (2019, March). Investigating the normalization procedure of NSGA-III. In *International Conference on Evolutionary Multi-Criterion Optimization* (pp. 229-240). Springer, Cham.

[16] Seada, H., & Deb, K. (2015). A unified evolutionary optimization procedure for single, multiple, and many objectives. *IEEE Transactions on Evolutionary Computation, 20*(3), 358-369.

[17] Vesikar, Y., Deb, K., & Blank, J. (2018, November). Reference point based NSGA-III for preferred solutions. In *2018 IEEE symposium series on computational intelligence (SSCI)* (pp. 1587-1594). IEEE.

[18] Carvalho, R. D., Saldanha, R. R., Gomes, B. N., Lisboa, A. C., & Martins, A. X. (2012). A multi-objective evolutionary algorithm based on decomposition for optimal design of Yagi-Uda antennas. *IEEE Transactions on Magnetics, 48*(2), 803-806.

[19] Li, K., Chen, R., Fu, G., & Yao, X. (2018). Two-archive evolutionary algorithm for constrained multiobjective optimization. *IEEE Transactions on Evolutionary Computation, 23*(2), 303-315.

[20] Panichella, A. (2019, July). An adaptive evolutionary algorithm based on non-Euclidean geometry for many-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 595-603).

[21] Shirobokov, S., Belavin, V., Kagan, M., Ustyuzhanin, A., & Baydin, A. G. (2020). Black-box optimization with local generative surrogates. *Advances in Neural Information Processing Systems, 33*, 14650-14662.

[22] Wang, C., Xu, C., Yao, X., & Tao, D. (2019). Evolutionary generative adversarial networks. *IEEE Transactions on Evolutionary Computation, 23*(6), 921-934.

[23] Jiang, J., & Fan, J. A. (2019). Global optimization of dielectric metasurfaces using a physics-driven neural network. *Nano letters, 19*(8), 5366-5372.

[24] Yang, J., Sell, D., & Fan, J. A. (2018). Freeform metagratings based on complex light scattering dynamics for extreme, high efficiency beam steering. *Annalen der Physik, 530*(1), 1700302.

[25] Hughes, T. W., Minkov, M., Williamson, I. A., & Fan, S. (2018). Adjoint method and inverse design for nonlinear nanophotonic devices. *ACS Photonics, 5*(12), 4781-4787.

[26] Jensen, J. S., & Sigmund, O. (2011). Topology optimization for nano-photonics. *Laser & Photonics Reviews, 5*(2), 308-321.

[27] Molesky, S., Lin, Z., Piggott, A. Y., Jin, W., Vucković, J., & Rodriguez, A. W. (2018). Inverse design in nanophotonics. *Nature Photonics, 12*(11), 659-670.

[28] Faury, L., Calauzenes, C., Fercoq, O., & Krichen, S. (2019). Improving evolutionary strategies with generative neural networks. *arXiv preprint arXiv:1901.11271*.

[29] Trabucco, B., Kumar, A., Geng, X., & Levine, S. (2021, July). Conservative objective models for effective offline model-based optimization. In *International Conference on Machine Learning* (pp. 10358-10368). PMLR.

[30] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems, 27*.

[31] Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

[32] Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 4401-4410).

[33] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[34] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928-1937). PMLR.

[35] Rastrigin, L. A. (1974) Systems of extremal control. Mir, Moscow.

[36] Hoffmeister, F., & Bäck, T. (1990, October). Genetic algorithms and evolution strategies: Similarities and differences. In *International Conference on Parallel Problem Solving from Nature* (pp. 455-469). Springer, Berlin, Heidelberg.

[37] Mühlenbein, H., Schomisch, M., & Born, J. (1991). The parallel genetic algorithm as function optimizer. Parallel computing, 17(6-7), 619-632.

[38] Ackley, D. (2012). *A connectionist machine for genetic hillclimbing* (Vol. 28). Springer Science & Business Media.

[39] Rosenbrock, H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal, 3*(3), 175-184.

[40] Dixon, L. C. W., & Mills, D. J. (1994). Effect of rounding errors on the variable metric method. *Journal of Optimization Theory and Applications, 80*(1), 175-179.

[41] Styblinski, M. A., & Tang, T. S. (1990). Experiments in nonconvex optimization: stochastic approximation with function smoothing and simulated annealing. *Neural Networks, 3*(4), 467-483.

[42] Deb, K., Thiele, L., Laumanns, M., & Zitzler, E. (2002, May). Scalable multi-objective optimization test problems. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)* (Vol. 1, pp. 825-830). IEEE.

[43] Kumar, S. (2020). Balancing a CartPole System with Reinforcement Learning–A Tutorial. *arXiv preprint arXiv:2006.04938*.

[44] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.

[45] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation, 1*(4), 541-551.

[46] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature, 323*(6088), 533-536.

[47] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9*(8), 1735-1780.

[48] Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

[49] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems, 30*.

[50] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research, 15*(1), 1929-1958.

[51] Blank, J., & Deb, K. (2020). Pymoo: Multi-objective optimization in python. *IEEE Access, 8*, 89497-89509.

[52] Iman, R. L., Davenport, J. M., & Zeigler, D. K. (1980). *Latin hypercube sampling (program user's guide).[LHC, in FORTRAN]* (No. SAND-79-1473). Sandia Labs., Albuquerque, NM (USA).

[53] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[54] Ros, R., & Hansen, N. (2008, September). A simple modification in CMA-ES achieving linear time and space complexity. *In International conference on parallel problem solving from nature* (pp. 296-305). Springer, Berlin, Heidelberg.

[55] Akimoto, Y., Auger, A., & Hansen, N. (2014, July). Comparison-based natural gradient optimization in high dimension. *In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (pp. 373-380).

[56] Loshchilov, I. (2017). LM-CMA: An alternative to L-BFGS for large-scale black box optimization. *Evolutionary computation*, 25(1), 143-171.

[57] Blank, J., & Deb, K. (2022). pysamoo: Surrogate-Assisted Multi-Objective Optimization in Python. *arXiv preprint arXiv:2204.05855.*

[58] Tian, Y., Wang, H., Zhang, X., & Jin, Y. (2017). Effectiveness and efficiency of non-dominated sorting for evolutionary multi-and many-objective optimization. *Complex & Intelligent Systems*, 3(4), 247-263.

[59] Long, Q., Wu, X., & Wu, C. (2021). Non-dominated sorting methods for multi-objective optimization: review and numerical comparison. *Journal of Industrial & Management Optimization, 17*(2), 1001.

[60] Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864.*

[61] Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M., & Teh, Y. W. (2018). Neural processes. *arXiv preprint arXiv:1807.01622.*