Report on Black Box Optimization on a Function with Known Properties

Shuqing Chai
chaisq@connect.hku.hk

Abstract

In the report, I propose a methodology for solving the problem of black box optimization on a function with known properties. The problem is divided into two parts: 1) constructing a neural network to approximate a function with known properties; 2) minimizing the objective with only zero-order evaluations. The neural network architecture is inspired by Input Convex Neural Networks (ICNNs), which are known for their ability to approximate convex functions. For the optimization part, a promising initial approach would involve Gaussian smoothing (GS), a derivative-free optimization method. The methodology is evaluated on a range of test functions to assess its performance and robustness. This includes dealing with polluted data, noisy inputs, and other potential challenges. The source code of this report is available at https://github.com/ShuqingChai/test_for_TSRL.

Contents

1 Literature Review 1
1.1 Monotonic/Convex Neural Networks 1
1.2 Derivative-Free Optimization 2

2 Methodologies 3
2.1 Overview of ICNNs 3
2.2 Neural Network Architecture 4
2.3 Finding the Maximum 5

3 Experiments 5
3.1 Neural Network Training 5
3.2 Optimization 5
3.3 Robustness Testing Against Noisy Inputs 7

1 Literature Review

Upon thorough examination of the question, I divide the problem into two parts: 1) constructing a neural network to approximate a function with known properties; 2) minimizing the objective with only zero-order evaluations. This section will delve into a review of the existing methodologies for addressing these two components.

1.1 Monotonic/Convex Neural Networks

We first discuss the current neural network approaches for approximating monotonic functions. One approach involves designing neural networks that inherently ensure monotonicity, while an alternative method requires some additional regularization [22, 10]. The ways in which neural networks can be designed to guarantee monotonicity are outlined as follows:

Constrained weights: This is the simplest method, which constrains the signs of the weights of the fully connected neural network to be either non-negative or non-positive, depending on the monotonicity direction of the input variables [2]. This method works with monotonic activation functions, but has limitations when the activation function is convex, as it can only approximate convex functions [14]. To address this limitation, a modification of the constrained network layer incorporates a non-saturated monotone convex activation function, along with two additional monotonic activation functions. This modification enhances the network's capability to approximate arbitrary monotonic functions [20]. Unconstrained Monotonic Neural Networks take a different approach by employing a free-form neural network, thereby relaxing constraints on both weights and activation functions [24].

Min-Max networks: Representing piece-wise linear architectures, Min-Max networks employ maximum and minimum operations on groups of hyperplanes to acquire the capacity to learn monotonic functions [21]. Notably, these networks are extended to approximate partially monotonic functions and are universal approximators [7]. Additionally, partial monotonicity can be achieved by constraining the signs of weights within a multi-layer perceptron [12].

Deep lattice networks: This category of networks integrates linear calibrators and lattices to effectively learn monotonic functions [26, 16]. While these networks find extensive application in practical scenarios, they do come with certain drawbacks, notably the demand for a substantial number of parameters [14].

Convex neural networks: Constructing a convex neural network can be achieved through different methods. One approach is to use a convex activation function as mentioned in [14]. A more elaborate architecture designed specifically for convex functions includes multiple layers of affine transformations, followed by non-decreasing activation functions such as ReLU, softplus, or sigmoid. Importantly, the weights of the affine transformations are constrained to be non-negative, ensuring convexity in the network structure [1].

1.2 Derivative-Free Optimization

In this subsection, we review the existing methods for derivative-free optimization within an unconstrained domain.

Deterministic methods: These methods use only function values to find local or global minima of smooth or non-smooth functions. They can be classified into direct-search methods, which use comparisons of function values to determine search directions [17, 23, 19, 11], and model-based methods, which use surrogate models of the objective function to guide the search [6, 25]. Some methods also use finite-difference estimates of derivatives to improve efficiency and robustness.

Randomized methods: These methods use random sampling or perturbation to generate search directions or model points, and often have better theoretical properties than deterministic methods [13]. Random search is a simple method that generates random points in the domain and compares their function values, and can converge in probability to the global minimum under mild assumptions [27]. Nesterov random search, which is motivated by Gaussian smoothing, uses a random direction and a line search to find a local minimum, and can achieve faster convergence rates than pure random search under smoothness and convexity assumptions [18]. Randomized direct-search methods use random sampling of positive spanning sets to ensure sufficient decrease or directional derivative conditions, and can reduce the dependence on the problem dimension in the worst-case complexity bounds [9]. Randomized trust-region methods methods use random sampling of interpolation points to construct probabilistically fully linear models of the objective function [3].

2 Methodologies

The network architecture I intend to employ is inspired by Input Convex Neural Networks (ICNNs) [1]. Despite ICNNs falling within a specific parametric category of convex functions, it is imperative to assess the richness of this class concerning its representational capacity. This concern is affirmatively addressed in [5], where the authors demonstrate that ICNNs can effectively approximate any convex function within a compact domain to the desired accuracy, particularly in the supremum norm. This validation lends strong support to the choice of ICNNs as a suitable class for approximating convex functions. Furthermore, the extensive representational capabilities of ICNNs have been evidenced in prior works, such as [15, 4].

For the optimization part, a promising initial approach would involve Gaussian smoothing (GS). GS stands out as a straightforward and efficient method, complemented by an alternative forward-difference estimator. This method comes with theoretical assurances of convergence to a stationary point for non-convex objectives and convergence to the optimal point for convex ones, subject to mild regularity conditions on both the objective and the perturbations [18]. Additionally, GS can be enhanced by adjusting the distribution of perturbations through techniques like normalization, orthogonalization, and others [8]. This adaptability makes GS a versatile optimization tool with the potential for improved performance through thoughtful modifications.

2.1 Overview of ICNNs

The ICNN architecture is designed to approximate convex functions. The network is constructed by stacking multiple layers of affine transformations, followed by non-decreasing activation functions. The weights of the affine transformations are constrained to be non-negative, ensuring convexity in the network structure.

Consider a partially input convex neural network (PICNN) with k layers as shown in Figure 1. This model defines a network over the input space $(x, y) \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_y}$ and the output $f(x, y; \theta)$ where f is convex in y but not necessarily in x. The network is defined by the following equations:

$$u_{i+1} = \tilde{g}_i \left(\tilde{W}_i u_i + \tilde{b}_i \right) \tag{1}$$

$$z_{i+1} = g_i \left(W_i^{(z)} \left(z_i \circ \left[W_i^{(zu)} u_i + b_i^{(z)} \right] \right) + W_i^{(y)} \left(y \circ \left(W_i^{(yu)} u_i + b_i^{(y)} \right) \right) + W_i^{(u)} u_i + b_i \right) \tag{2}$$

$$f(x, y; \theta) = z_k, \quad u_0 = x, \tag{3}$$

where $u_i \in \mathbb{R}^{n_u}$ and $z_i \in \mathbb{R}^{n_z}$ denote the hidden units for the x-path and y-path, $W_i \in \mathbb{R}^{n_z \times n_u}$ is the weight matrix, $b_i \in \mathbb{R}^{n_z}$ is the bias vector, g_i is a non-decreasing activation function, and $\circ$ denotes the element-wise product.

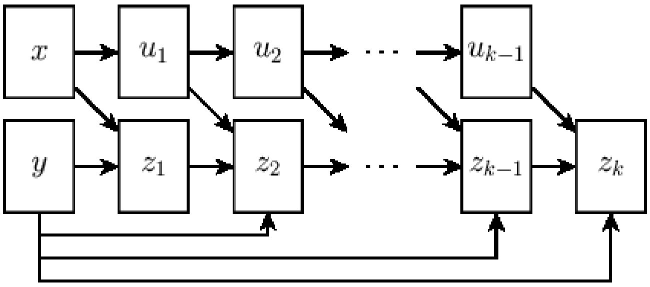

Figure 1: Input Convex Neural Network (ICNN) Architecture. Curtesy of [1].

The central propositon on convexity of ICNNs is as follows:

Proposition 1 *The function f is convex in y provided that all $W_i^{(z)}$ are non-negative, and all functions g_i are convex and non-decreasing.*

The idea is simple that non-negative sums of convex functions are convex and the composition of a convex and convex non-decreasing function is convex.

2.2 Neural Network Architecture

We adopt the PICNN architecture and make it suitable for our problem. Modifications to the PICNN are based on similar ideas to the idea behind the above proposition. We use different notations in the model from here on to avoid confusion, which are consistent with the original problem statement.

First, we consider y. The convexity of f in y is satisfied with the PICNN. To ensure the monotonicity of decreasing in y, we constrain the weights in maps from y to z_i to be non-positive.

Second, we do not specifically constrain the weights in maps from x to z_i to be non-negative to assure monotonicity. Theoretically, we can use a combination of different activation functions and constraints to ensure the monotonicity of f in x. However, in practice, we simply leave the weights unconstrained for convenience and flexibility.

Third, analyzing on v, we can see that the output f is independent of v. This means that we can set v to be a constant vector, which is equivalent to removing the v-path in the network. In practice, we use one tensor to represent v-path, i.e., set it to be a constant tensor, and train it along with x. This simplifies the network and reduces the number of parameters.

Last, considering the role of u on the convexity and monotonicity of y, we withhold the u-path until the last layer. This is because the u-path can change the convexity and monotonicity of y at the last layer without damaging the structure on y-path earlier.

A detailed architecture of a two-layer modified PICNN is shown in Figure 2.

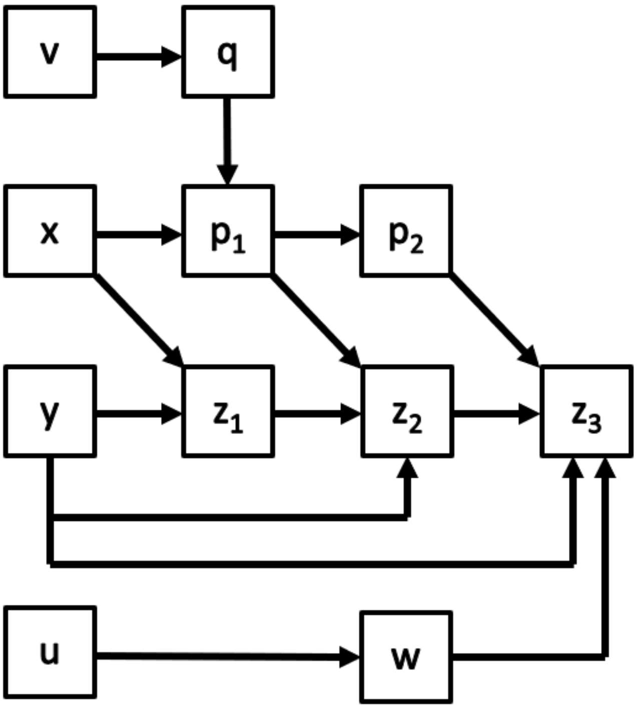

Figure 2: Modified Partially Input Convex Neural Network (PICNN) Architecture.

2.3 Finding the Maximum

We solve the problem of finding the maximum of a function with zero-order evaluations by the Gaussian smoothing (GS) method. GS estimates the gradient of a function by generating a direction from a standard normal distribution and evaluating the function at a perturbed point. The gradient is then estimated by the difference between the function values at the original and perturbed points. Specifically, the GS gradient estimator is

$$\nabla_\theta F^{GS}(\theta) = \frac{1}{c}F(\theta + c\epsilon)\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}). \tag{4}$$

Alternative estimators include the forward-difference (FD) estimator and the antithetic (AT) estimator:

$$\nabla_\theta F^{FD}(\theta) = \frac{1}{c}[F(\theta + c\epsilon) - F(\theta)]\epsilon, \tag{5}$$

$$\nabla_\theta F^{AT}(\theta) = \frac{1}{2c}[F(\theta + c\epsilon) - F(\theta - c\epsilon)]\epsilon. \tag{6}$$

We use the FD estimator for its simplicity and lower variance compared to the GS estimator.

3 Experiments

We construct two families of functions to evaluate the performance of the proposed methodology. The first family considers a linear combination of variables while the second family involves non-linear relation.

$$f_1 = \gamma_1 x + \gamma_2(-1)^u y^2 + \gamma_3 g_1(u) + \gamma_4 g_1(v), \tag{7}$$

$$f_2 = \frac{\gamma_2 \exp(\gamma_1(-1)^u x)}{(-1)^u y - 0.9} + \gamma_3 g_2(u) + \gamma_4 g_2(v), \tag{8}$$

where $g_1(\cdot) = 1$, $g_2(\cdot) = 0$, and $\gamma_i \sim U([0, 2])$.

3.1 Neural Network Training

We generate a dataset of 500 samples for each function. We use 80% of the data for training and the remaining 20% for validation. We train the modified PICNN on the training data and evaluate its performance on the test data. We use the Adam optimizer with a learning rate of 0.001. We train the network for 500 epochs and monitor the training and validation losses. Convergence of the training and validation losses is shown in Figure 3.

3.2 Optimization

We use the FD estimator to estimate the gradient of the objective function. However, estimating the maximum is a challenging task due to discontinuity brought by $(-1)^u$ and the space where u belongs to. To address this issue, we consider the functions setting u to be a constant vector. By this means, we divide the problem into smaller problems, and perform optimization on each sub-problem. In this way, we can avoid the discontinuity and reduce the dimension of the problem. The following experiments of optimization are conducted on the modified functions. We initialize the optimization at a random point and run the optimization for 100 rounds with 50 iterations each round. For each iteration, 20 perturbed points are generated to estimate the gradient. We monitor the convergence of the maximum value. The results are shown in Figure 4. Integration tests against true maximum value can be performed to verify the correctness of the optimization, which can be found in the github repository.

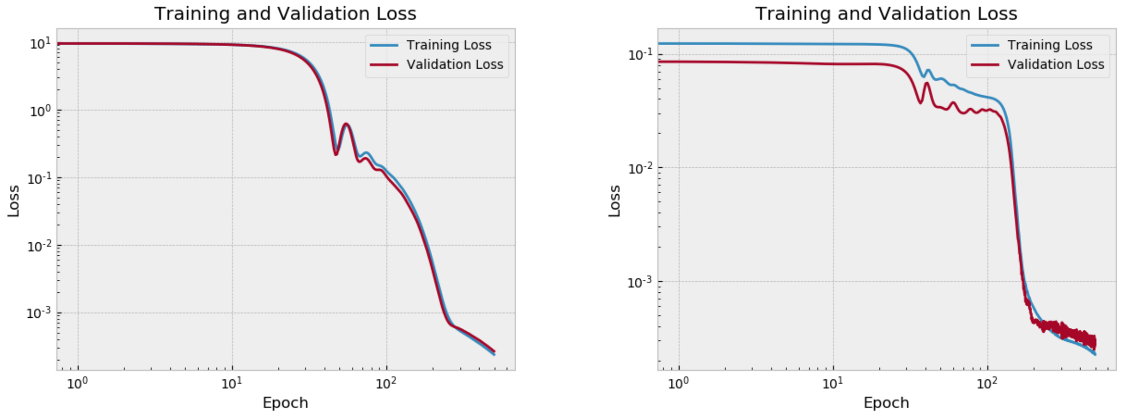

Figure 3: Convergence of the training and validation losses. Left: Approximating f_1. Right: Approximating f_2.

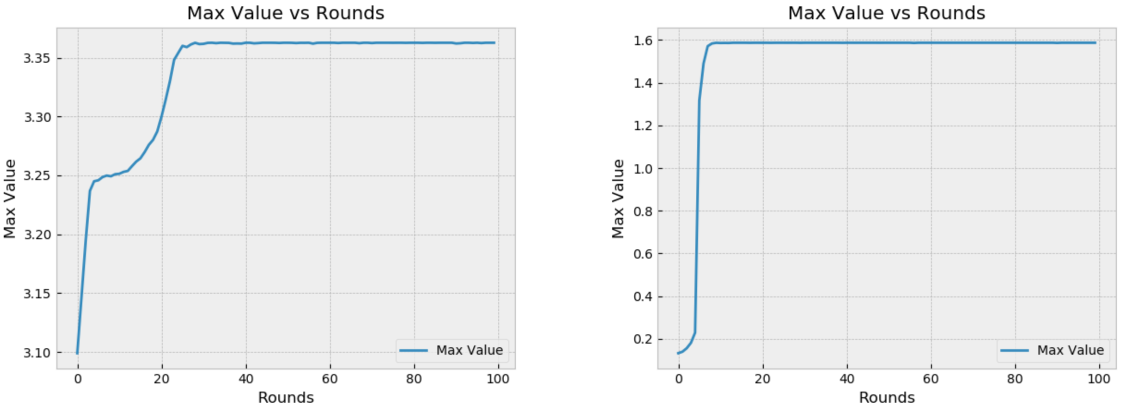

Figure 4: Convergence of the maximum value. Left: Estimated maximum value of f_1. Right: Estimated maximum value of f_2.

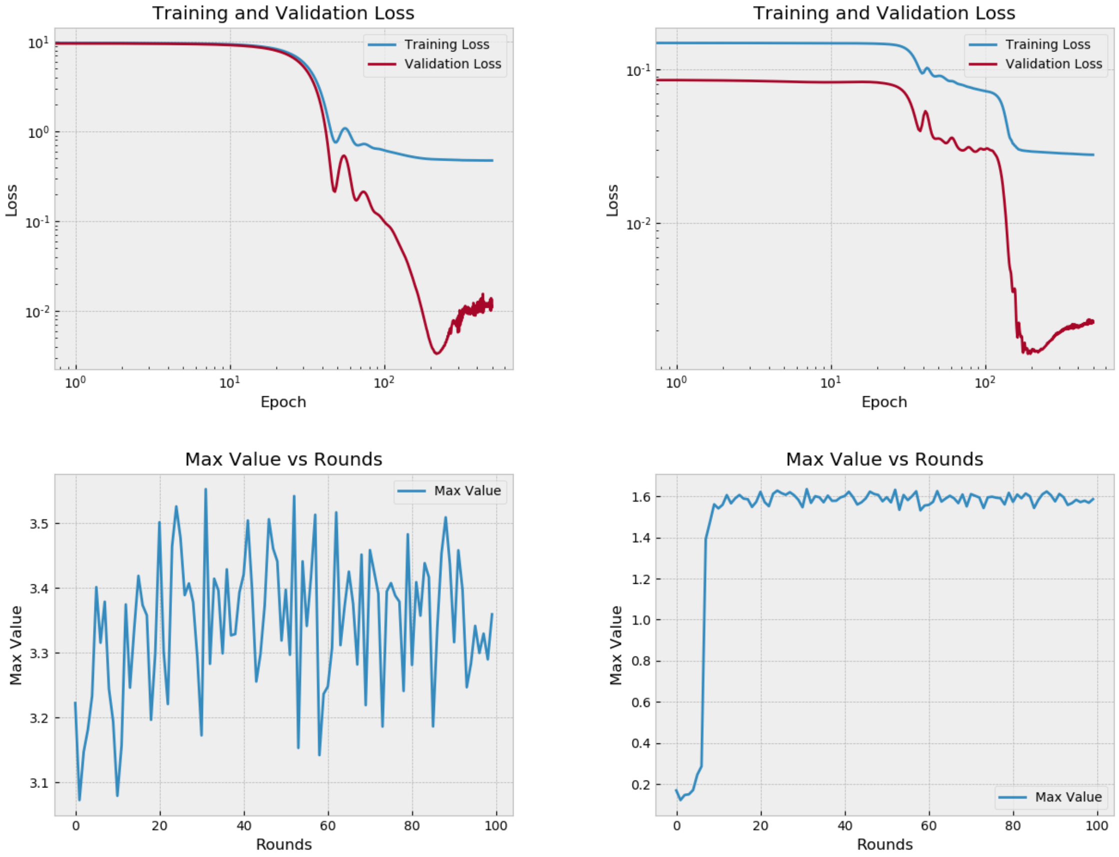

Figure 5: Robustness testing against noisy inputs. Top left: Approximating f_1. Top right: Approximating f_2. Bottom left: Estimated maximum value of f_1. Bottom right: Estimated maximum value of f_2.

3.3 Robustness Testing Against Noisy Inputs

Observation with noise is a common scenario in practice. To evaluate the robustness of the proposed methodology, we add Gaussian noise to the inputs, which is a structurally simpler than the time series noise. We add noise with a constant standard deviation to the inputs and repeat the experiments. The results are shown in Figure 5.

References

[1] B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

[2] N. P. Archer and S. Wang. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, 24(1):60–75, 1993.

[3] A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Convergence of trust-region methods based on probabilistic models. *SIAM Journal on Optimization*, 24(3):1238–1264, 2014.

[4] C. Bunne, A. Krause, and M. Cuturi. Supervised training of conditional monge maps. *Advances in Neural Information Processing Systems*, 35:6859–6872, 2022.

[5] Y. Chen, M. Telgarsky, C. Zhang, B. Bailey, D. Hsu, and J. Peng. A gradual, semi-discrete approach to generative network training via explicit wasserstein minimization. In *International Conference on Machine Learning*, pages 1071–1080. PMLR, 2019.

[6] A. R. Conn, N. I. Gould, and P. L. Toint. *Trust region methods*. SIAM, 2000.

[7] H. Daniels and M. Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.

[8] K. Gao and O. Sener. Generalizing gaussian smoothing for random search. In *International Conference on Machine Learning*, pages 7077–7101. PMLR, 2022.

[9] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM Journal on Optimization*, 25(3):1515–1541, 2015.

[10] A. Gupta, N. Shukla, L. Marla, A. Kolbeinsson, and K. Yellepeddi. How to incorporate monotonicity in deep networks while preserving flexibility? *arXiv preprint arXiv:1909.10662*, 2019.

[11] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

[12] B. Lang. Monotonic multi-layer perceptron networks as universal approximators. In *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005: 15th International Conference, Warsaw, Poland, September 11-15, 2005. Proceedings, Part II 15*, pages 31–37. Springer, 2005.

[13] J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numerica*, 28:287–404, 2019.

[14] X. Liu, X. Han, N. Zhang, and Q. Liu. Certified monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:15427–15438, 2020.

[15] A. Makkuva, A. Taghvaei, S. Oh, and J. Lee. Optimal transport mapping via input convex neural networks. In *International Conference on Machine Learning*, pages 6672–6681. PMLR, 2020.

[16] M. Milani Fard, K. Canini, A. Cotter, J. Pfeifer, and M. Gupta. Fast and flexible monotonic functions with ensembles of lattices. *Advances in neural information processing systems*, 29, 2016.

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

[18] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17:527–566, 2017.

[19] W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[20] D. Runje and S. M. Shankaranarayana. Constrained monotonic neural networks. In *International Conference on Machine Learning*, pages 29338–29353. PMLR, 2023.

[21] J. Sill. Monotonic networks. *Advances in neural information processing systems*, 10, 1997.

[22] J. Sill and Y. Abu-Mostafa. Monotonicity hints. *Advances in neural information processing systems*, 9, 1996.

[23] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4(4):441–461, 1962.

[24] A. Wehenkel and G. Louppe. Unconstrained monotonic neural networks. *Advances in neural information processing systems*, 32, 2019.

[25] S. M. Wild, R. G. Regis, and C. A. Shoemaker. Orbit: Optimization by radial basis function interpolation in trust-regions. *SIAM Journal on Scientific Computing*, 30(6):3197–3219, 2008.

[26] S. You, D. Ding, K. Canini, J. Pfeifer, and M. Gupta. Deep lattice networks and partial monotonic functions. *Advances in neural information processing systems*, 30, 2017.

[27] Z. B. Zabinsky and R. L. Smith. Pure adaptive search in global optimization. *Mathematical programming*, 53(1-3):323–338, 1992.