Global Optimization Networks

Sen Zhao [1] **Erez Louidor** [1] **Maya Gupta** [1]

Abstract

We consider the problem of estimating a good maximizer of a black-box function given noisy examples. We propose to fit a new type of function called a global optimization network (GON), defined as any composition of an invertible function and a unimodal function, whose unique global maximizer can be inferred in $\mathcal{O}(D)$ time, and used as the estimate. As an example way to construct GON functions, and interesting in its own right, we give new results for specifying multi-dimensional unimodal functions using lattice models with linear inequality constraints. We extend to *conditional* GONs that find a global maximizer conditioned on specified inputs of other dimensions. Experiments show the GON maximizers are statistically significantly better predictions than those produced by convex fits, GPR, or DNNs, and form more reasonable predictions for real-world problems.

1. Introduction

We consider the problem of predicting a maximizer $\hat{\mathbf{x}}$ for an unknown function $g(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}$, given only a fixed set of N noisy input-output training pairs $(\mathbf{x}_i, y_i)$ for $\mathbf{x}_i \in \mathbb{R}^D$, and $y_i = g(\mathbf{x}_i) + \epsilon_i \in \mathbb{R}, i = 1, \ldots, N$, where ϵ_i is zero-mean noise. The predicted maximizer $\hat{\mathbf{x}}$ will be judged by how close its predicted output $g(\hat{\mathbf{x}})$ is to the true global maximum $g(\mathbf{x}^*)$ where $\mathbf{x}^* \in \arg\max_{\mathbf{x}} g(\mathbf{x})$.

For example, a coffee chain may wish to optimize where it puts its next cafe given features like local population density and distances to other cafes. Doctors want to optimize the best dosages of a cocktail of medicines to give a patient. Businesses want to optimize the features or their new products. In each of these cases, we assume we have some past examples to learn from, and want to make one best guess.

[1]Google Research, Mountain View, CA 94043 USA. Correspondence to: Sen Zhao <senzhao@google.com>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

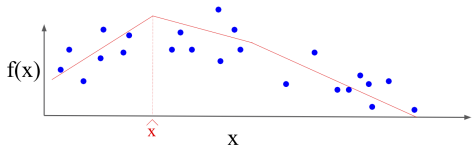

Figure 1. Toy Example: A concave piecewise-linear function $h(\mathbf{x}; \phi)$ was fit to $N = 23$ *noisy* training examples to minimize mean squared error. The fitted function's prediction of the global maximizer is marked as $\hat{x}$. Because the training data is known to be noisy, it is reasonable that the prediction's maximum is lower than the training data maximum, and in a different place.

We leave as future work extending the proposed methodology to the standard *global optimization algorithm* setting where one is allowed to make a series of guesses (Horst & Pardalos, 1995), that is, where one selects an $\hat{\mathbf{x}}_t$ and is able to acquire the additional training label $g(\hat{\mathbf{x}}_t) + \epsilon_t$, for $t = 1, \ldots, T$. Here, we make only one guess $\hat{x}$.

We take a machine-learning approach as illustrated in Fig. 1: we fit a function $h(\mathbf{x}; \phi)$ with parameters ϕ to the N training samples, and then take the global maximizer of the fitted $h(\mathbf{x}; \phi)$ as the estimate $\hat{\mathbf{x}}$.

A key question is which function class to use for $h(\mathbf{x}; \phi)$. Ideally $h(\mathbf{x}; \phi)$ will have the right amount of model expressibility, and it will be easy to find its maximizer. Box & Wilson (1951) proposed fitting a quadratic function as a surrogate function whose maximizer can be easily found, but for many real-world applications a quadratic function will be too inflexible. At the other extreme, one can fit an arbitrarily flexible function like a DNN (Gorissen et al., 2010), but that may overfit, producing a noisy estimate of the maximizer. In addition, even if we regularize DNN models well, they may still have multiple local maximizers. Hence, with any randomly chosen starting point, gradient descent is unlikely to find DNN's global maximizer. One may need to try many starting points to improve the maximizers found by gradient descent, which could become exponentially more expensive with larger input dimension D, and there is no guarantee that the global maximizer of the DNN is found.

We propose a new function class called a *global optimization network* (GON) that generalizes unimodal functions. GONs are more flexible than prior restricted surrogate functions, but have a well-defined global maximizer. We also

extend GONs to the conditional setting where some of the inputs $\mathbf{z} \in \mathbb{R}^M$ are fixed, and define *conditional global optimization networks* (CGONs) that infer the conditional maximizer $\mathbf{x}^* = \arg\max_{\mathbf{x}} g(\mathbf{x}, \mathbf{z})$.

GONs can be built with any function layers that satisfy the required invertibility and unimodality constraints. We show how to construct GONs using new constraints on *deep lattice networks* (DLNs) (You et al., 2017), which are implemented in the TensorFlow Lattice[1] library. A key benefit of the proposed DLN GON is that its D-dimensional global maximizer can be found surprisingly efficiently in $\mathcal{O}(D)$ time. Further, DLN GONs can be trained efficiently by constrained empirical risk minimization with linear inequality constraints. Experiments show the favorable and more intuitive results obtained by GONs on real problems and statistically significant simulations.

2. Related Work

GONs lie at the intersection of two classic strategies: *(i)* fit a function from a restricted function class to noisy data and use the fitted models' maximizer as the prediction of the true maximizer, and *(ii)* define a function class by *shape constraints* on the behavior of the function, such as requiring the function to be monotonically increasing in some directions. See the Supplemental for surveys of those two strategies. Next, we detail the closest related work.

2.1. Convex Neural Networks As Surrogates

Amos et al. (2017) proposed using the shape constraint of *convexity* on a deep neural network to form a surrogate function, then predicting the global minimizer to be $x^* = \arg\min_{\mathbf{x}} h(\mathbf{x})$. They constructed it as a multi-layer ReLU net with the necessary monotonicity shape constraints to produce an overall convex function. They called this ICNN and sometimes FICNN, and *partial-input convex neural network* (PICNN) when used for the *conditional* global optimization problem $\mathbf{x}^* = \arg\min_{\mathbf{x}} g(\mathbf{x}, \mathbf{z})$.

Because both FICNN and PICNN are convex in $\mathbf{x}$, the fits can be minimized with gradient-based optimization algorithms to find $\arg\min_{\mathbf{x}} h(\mathbf{x}; \phi)$ and $\arg\min_{\mathbf{x}} h(\mathbf{x}, \mathbf{z}; \phi)$. Others have found the ICNN strategy useful (Chen et al., 2019; 2020). However, ReLU-activated ICNNs are neither smooth nor strongly convex, which reduces the convergence rate in finding the minimizer.

We found that convex functions were often still too inflexible for even small problems, for examples see Fig. 3 and Fig. 4.

[1] https://www.tensorflow.org/lattice/overview

2.2. Lattice Networks

We will define GONs only by the shape constraints their layers must satisfy, but our GON constructions will be architected with calibrator layers (D separate piecewise linear functions on D inputs) and lattice layers (functions resulting from interpolating a multi-dimensional look-up table). The basics of such functions are covered most clearly in Gupta et al. (2016), but see You et al. (2017) for architecting *deep* lattice networks that mix these and other layers. The regular structure of lattice networks makes them well-suited for satisfying shape constraints (see the Supplemental) while maximizing flexibility.

2.3. Unimodal Shape Constraints

We will define GONs by using the shape constraint *unimodality*: a function is unimodal if it has a maximizer and is non-increasing along any ray that starts at that maximizer (for example, in Fig. 3 the GON, ICNN, and GPR fits are all unimodal). A few papers have studied fitting *1D* unimodal functions (Stout, 2008; Köllmann et al., 2014; Gunn & Dunson, 2005; Chatterjee & Lafferty, 2019). This paper goes beyond that prior work both in fitting 1D unimodal functions *without* prior knowledge of the maximizer using constrained empirical risk minimization, and in the ability to fit *multi-D* unimodal functions with a known maximizer (which will be sufficient to construct GONs).

Recently, Gupta et al. (2020) showed how to construct and fit a *subclass* of multi-d unimodal functions by applying linear inequality constraints on a lattice model. However, their unimodality constraints were overly restrictive in that they were separable by dimension, and hence were sufficient but not necessary for multi-d unimodality. We will give a new set of linear inequality constraints that are *both necessary and sufficient* for a multi-d lattice function to be unimodal. This paper also differs from Gupta et al. (2020) in that we use unimodality to create surrogate functions for global optimization, whereas their goal was to regularize ML functions where there was domain knowledge that the fit should be unimodal. Here we do not assume the true function is actually unimodal, we simply use a unimodal fit as a regularization tool to efficiently predict a good maximizer.

2.4. Model-based Optimization

Model-based optimization (MBO) is a recent term used by some in the ML community to describe the same problem and set-up we attack here. Much of the MBO experiments and motivation (e.g. (Kumar & Levine, 2020) and (Trabucco et al., 2021)) has focused on problems with significant train-test distribution shift.

The closest MBO work is that of Yu et al. (2021), which claims that it is better than prior MBO approaches, and

uses the classic strategy (as we do) of fitting a surrogate f(x) to the train data and then maximizing it. Their work differs in that they propose a specially smoothed DNN fit: they perturb the inputs pre-training, and then locally smooth post-training based on four smoothness hyperparameters.

Another difference to the prior MBO work is that in our experiments we use a validation set to select the hyperparameters for each of the compared methods, where our validation metric is the accuracy of the predicted maximizer on the validation set.

3. Global Optimization Networks

We propose a new multi-layer function class that we call *global optimization networks* (GONs) defined as a unimodal function composed with an invertible function, and a conditional variant we call CGONs.

3.1. Definition of Global Optimization Networks

We define a GON to be any multi-layer function $h : \mathbb{R}^D \to \mathbb{R}$ that can be expressed as $h(\mathbf{x}; \phi) = u(c(\mathbf{x}))$, where $c : \mathbb{R}^D \to \mathcal{S}_D$ is any invertible function whose image $\mathcal{S}_D$ is a convex subset of $\mathbb{R}^D$ that contains $\mathbf{0}$ (, and $u : \mathcal{S}_D \to \mathbb{R}$ is any *unimodal* function such that it is non-increasing along any ray that starts at $\mathbf{0}$.

Fig. 2 shows a 1D example of a $c(x)$ and $u(x)$, with the resulting GON $h(\mathbf{x}) = u(c(x))$ shown at the far-left of Fig. 3. The role of the $c(\mathbf{x})$ is to stretch, rotate, and shift where the outputs of c land in u's domain so that the GON maximizer $\hat{\mathbf{x}}$ satisfies $c(\hat{\mathbf{x}}) = 0$, which the unimodal function u then maps to the GON maximum.

Because the maximizer of u is constrained to be at $\mathbf{0}$, the GON maximizer is $\hat{\mathbf{x}} \equiv \arg\max_{\mathbf{x}} h(\mathbf{x}; \phi) = c^{-1}(\mathbf{0})$, where c is invertible because it was constrained to be a bijection. The maximizer $c^{-1}(\mathbf{0})$ will be efficient to find if c is efficient to invert at $\mathbf{0}$. Further, suppose $c(\mathbf{x}) = s(c'(\mathbf{x}))$ for some bijective $c' : \mathbb{R}^D \to \mathcal{S}_D$ and bijective $s : \mathcal{S}_D \to \mathcal{S}_D$ with $s(\mathbf{0}) = \mathbf{0}$. The GON maximizer $\hat{\mathbf{x}} = c^{-1}(\mathbf{0}) = c'^{-1}(s^{-1}(\mathbf{0})) = c'^{-1}(\mathbf{0})$, thus only c' must be computationally easy to invert, and s can be quite flexible to increase the expressiveness of c.

3.2. Relation of GONs to Other Function Classes

We show how GONs are related to other function classes. All proofs for this paper are in the Supplemental.

First, note that the set of GONs includes unimodal functions with an arbitrary maximizer:

Prop. 1: Let $g : \mathcal{S}_D \to \mathbb{R}$ be a unimodal function with global maximizer $\mathbf{x}^* \in \mathcal{S}_D \subseteq \mathbb{R}^D$. Then g can be expressed as a GON.

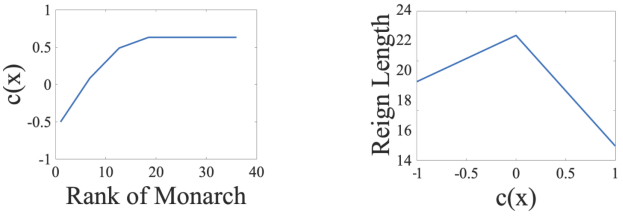

Figure 2. The component $c(x)$ and $u(x)$ fit for the best GON for the 1D Monarchs' Reigns dataset detailed in Sec. 5.2. **Left:** The first-layer $c(x)$ is a piece-wise linear function (PLF) is defined by five key-value pairs, and meets the invertibility requirement because it is strictly monotonically increasing. **Middle:** The second-layer $u(\cdot)$ is a PLF defined by three key-value pairs, and was constrained to be unimodal around 0 by making it monotonically increasing up to 0 and monotonically decreasing after 0 and forcing its middle knot to be at 0, and the other two knots were fixed at -1 and 1 so the 2nd-layer PLF can be described as a 1D lattice function. The resulting GON $h(x; \phi) = u(c(x))$ is shown in Fig. 3. As described in Sec. 4.9, the parameters of c and u were trained jointly using constrained empirical risk minimization with linear inequality constraints to ensure the needed monotonicity constraints.

Next, note concave/convex functions are a special case of unimodal functions (and thus GONs generalize ICNNs (Amos et al., 2017)):

Prop. 2: Let $g : \mathcal{S}_D \to \mathbb{R}$ be a concave function with global maximizer $\hat{\mathbf{x}} \in \mathcal{S}_D \subseteq \mathbb{R}^D$. Then g is unimodal with maximizer $\hat{\mathbf{x}}$.

Surprisingly, continuous *1D* GONs are always unimodal:

Prop. 3: Let $u : \mathcal{S}_1 \to \mathbb{R}, \mathcal{S}_1 \subseteq \mathbb{R}$ be a 1D unimodal function with maximizer at 0. Let $c : \mathbb{R} \to \mathcal{S}_1$ be continuous, bijective, and have 0 in its image. Then $h(x; \phi) = u(c(x))$ is unimodal.

3.3. Conditional Global Optimizaton Networks

Consider the conditional global optimization problem: $\mathbf{x}^* = \arg\max_{\mathbf{x}} g(\mathbf{x}, \mathbf{z})$ for $\mathbf{z} \in \mathbb{R}^M$ (Amos et al., 2017). For example, conditioned on what percentage of a job is manual labor z, predict the number of weekly work hours x^* that will maximize long-term output (Pencavel, 2015).

We extend the GON definition to a *conditional global optimization network* (CGON). Let c, u, s and c' be as defined above, $\mathbf{z} \in \mathbb{R}^M$ be an M-dimensional feature vector of conditional inputs, and $r : \mathbb{R}^M \to \mathcal{S}_D$ be any learnable function. We define a CGON to be any function that can be written as:

$$h(\mathbf{x}, \mathbf{z}; \phi) = u\left(s\left(\frac{c'(\mathbf{x}) + r(\mathbf{z})}{2}\right)\right). \quad (1)$$

Since the maximizer of u is fixed at $\mathbf{0}$, and $s(\mathbf{0}) = \mathbf{0}$, it is

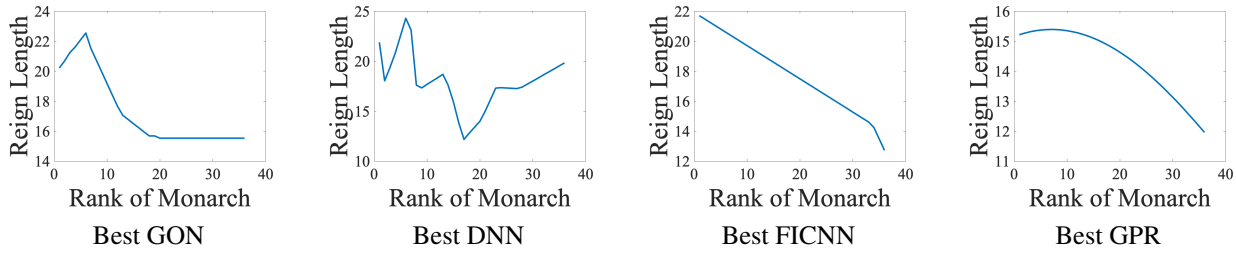

Figure 3. Illustrative Example: Best fits for four methods for the 1D Monarchs' Reigns problem detailed in Sec. 5.2: the goal is to predict the rank of the monarch in a dynasty that will rule longest. The predicted maximizer of the GON and DNN coincide at the 6th monarch, though the DNN fit has many local maxima. The convex fit (FICNN) is too rigid. The GPR fit is smooth and provides a closer prediction. The GON shown is the composition of the component functions $c(x)$ and $u(\cdot)$ that are shown in Fig. 2.

easy to show the CGON maximizer is at $\hat{\mathbf{x}} = c'^{-1}(-r(\mathbf{z}))$. Note r is unrestricted, so the CGON can have arbitrary dependence on the conditional input $\mathbf{z}$, for example r can be a DNN. Table 1 summarizes the requirements for GON and CGON.

4. Constructing GONs and CGONs

GONs are defined by their shape constraints, but how do we satisfy those constraints? Next, we describe one approach: build GONs by using a calibrator layer of piecewise linear functions (PLFs), and a second layer of their multi-d cousins, lattice functions. PLFs and lattice functions are implemented in the TensorFlow Lattice[2] library. The benefits of using these function classes are their: *(i)* **flexibility:** they are respectively universal approximators of continuous bounded 1D and multi-D functions over convex domains; *(ii)* **efficiency:** we show they enable finding the maximizers of GONs and CGONs in $\mathcal{O}(D)$ time; and *(iii)* **trainability:** we show they can be trained using constrained empirical risk minimization with linear inequality constraints.

4.1. Constructing 1D GONs with PLFs

We first build intuition by showing how to construct efficient two-layer 1D GONs using piecewise-linear functions (PLFs) for both the invertible $c()$ and unimodal $u()$, as in Fig. 2. Recall that a PLF can be defined by a set of key-value pairs, and then is evaluated at any point by linearly interpolating the values of the surrounding two keypoints. Let $c(x)$ be defined by the $K^{(c)}$ key-value pairs $(\kappa_k^{(c)} \in \mathbb{R}, \nu_k^{(c)} \in \mathbb{R})$ for $k = 1, \ldots, K^{(c)}$. Then $c(x) =$

$$\sum_{i=1}^{K^{(c)}-1} \left(\nu_i^{(c)} + \frac{x - \kappa_i^{(c)}}{\kappa_{i+1}^{(c)} - \kappa_i^{(c)}} \left(\nu_{i+1}^{(c)} - \nu_i^{(c)} \right) \right) I_{\kappa_i^{(c)} < x \leq \kappa_{i+1}^{(c)}} \tag{2}$$

In our experiments, we fix the keys of c to be the two endpoints of the feasible input domain plus the $K^{(c)} - 2$ quantiles of the inputs in the train data, and only train the PLF values $\{\nu_k^{(c)}\}$.

Recall a 1D continuous invertible function on a closed interval must be strictly monotonic. One can make a PLF monotonically increasing(decreasing) by constraining its values to be increasing(decreasing) (as done in isotonic regression (Barlow et al., 1972)). In addition, we constrain the outputs of c to lie within the input domain of the second-layer function u, which we set to be $[-(K^{(u)}-1)/2, -(K^{(u)}+1)/2]$ as explained below. Thus the parameters $\{\nu_k^{(c)}\}$ of the PLF c are constrained to satisfy:

$$-\frac{K^{(u)}-1}{2} \leq \nu_1^{(c)} < \ldots < \nu_{K^{(c)}}^{(c)} \leq \frac{K^{(u)}-1}{2}. \tag{3}$$

To construct a unimodal PLF with maximizer at 0, use an odd number of $K^{(u)}$ keypoints uniformly spaced in $\mathcal{S}_1 = [-(K^{(u)}-1)/2, (K^{(u)}-1)/2]$. Hence, $\kappa_i = -(K^{(u)}-1)/2 + i - 1, i = 1, \ldots, K^{(u)}$. $K^{(u)}$ is a hyperparameter, where a larger value of $K^{(u)}$ increases the number of parameters of u and hence the flexibility of u. Since $K^{(u)}$ must be odd, this makes 0 the middle keypoint of u. We then constrain the PLF to be increasing up to 0, and decreasing after 0. That is, a PLF with $K^{(u)}$ keypoints satisfies the unimodality constraints if their values $\nu_k^{(u)}$ satisfies the following $K^{(u)} - 1$ linear inequality constraints:

$$\nu_1^{(u)} < \ldots < \nu_{(K^{(u)}+1)/2}^{(u)} > \ldots > \nu_{K^{(u)}}^{(u)}. \tag{4}$$

Note that the domain of u is bounded by its first and last keypoints, i.e., $\mathcal{S}_1 = [-(K^{(u)}-1)/2, (K^{(u)}-1)/2]$, which is why in (3) we constrained the outputs of c to land there.

All continuous 1D functions defined on a closed interval can be approximated arbitrarily well by a PLF with enough knots. It follows that $g(x)$ can be approximated arbitrarily well by a $h(x; \phi)$ constructed with PLFs as well. Thus, this construction can approximate arbitrarily well all continuous 1D GON functions defined on closed intervals.

[2] https://www.tensorflow.org/lattice/overview

Table 1. GON and CGON summaries for $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{z} \in \mathbb{R}^M$.

	GON	CGON
FORMULATION	$h(\mathbf{x};\phi) = u(c(\mathbf{x})) = u(s(c'(\mathbf{x})))$	$h(\mathbf{x},\mathbf{z};\phi) = u\left(s\left(\frac{c'(\mathbf{x})+r(\mathbf{z})}{2}\right)\right)$
MAXIMIZER	$\hat{\mathbf{x}} = c^{-1}(\mathbf{0}) = c'^{-1}(\mathbf{0})$	$\hat{\mathbf{x}} = c'^{-1}(-r(\mathbf{z}))$
REQUIREMENT ON $c' : \mathbb{R}^D \to \mathcal{S}_D$	BIJECTIVE, EASY TO INVERT	BIJECTIVE, EASY TO INVERT
REQUIREMENT ON $s : \mathcal{S}_D \to \mathcal{S}_D$	BIJECTIVE, $s(\mathbf{0}) = \mathbf{0}$	BIJECTIVE, $s(\mathbf{0}) = \mathbf{0}$
REQUIREMENT ON $u : \mathcal{S}_D \to \mathbb{R}$	UNIMODAL, $\arg\max_{\mathbf{x}} u(\mathbf{x}) = \mathbf{0}$	UNIMODAL, $\arg\max_{\mathbf{x}} u(\mathbf{x}) = \mathbf{0}$
REQUIREMENT ON $r : \mathbb{R}^M \to \mathcal{S}_D$	-	ANY FUNCTION

Given this PLF construction, to find the maximizer of $h(x)$, we only need to invert $c(x)$ at 0. Since c is a monotonically increasing PLF, inverting it is efficient and requires a constant number of operations: first find c's smallest keypoint κ^* that satisfies $c(\kappa^*) \geq 0$, and then invert the linear segment between this keypoint and the keypoint to the left of it to get $\hat{x} = c^{-1}(0)$. Note that such a κ^* must exist since we assume that 0 is in the image of c.

4.2. Multi-D GONs Using Lattice Layers

Our multi-D GON construction is a generalization of our 1D construction. For c, we simply use D monotonic PLFs, one for each input, which is called a calibrator layer and is commonly used as a first layer for lattice networks (Gupta et al., 2016; Canini et al., 2016; You et al., 2017). We constrain their output ranges to the domain of u using linear inequality constraints like (3). One can increase the GON's flexibility by setting $s : \mathcal{S}_D \to \mathcal{S}_D$ to be cascades of no-bias hyperbolic-tangent-activated dense layers, or other invertible models (Behrmann et al., 2019), but our experiments simply use D PLFs for c. For u, we use a D-dimensional lattice function (Garcia & Gupta, 2009), and we propose new linear inequality constraints for a lattice that are both sufficient and necessary to ensure the lattice is unimodal.

4.3. Lattice Function Review

Lattice functions are multi-D look-up tables that are interpolated to form piecewise multilinear polynomial functions; see Gupta et al. (2016) and Garcia et al. (2012) for more details. Let $\mathbf{V} \in \mathbb{N}^D$ be a hyperparameter vector where $\mathbf{V}[d]$ is the number of keypoints (and hence flexibility) of the lattice function over its dth input. The lattice is defined by the set of $\prod_d \mathbf{V}[d]$ regularly-spaced keys or vertices,

$$\mathcal{M}_{\mathbf{V}} = \left\{-\left\lfloor\frac{\mathbf{V}[1]-1}{2}\right\rfloor, \ldots, \left\lceil\frac{\mathbf{V}[1]-1}{2}\right\rceil\right\} \times \ldots \times \left\{-\left\lfloor\frac{\mathbf{V}[D]-1}{2}\right\rfloor, \ldots, \left\lceil\frac{\mathbf{V}[D]-1}{2}\right\rceil\right\},$$

and corresponding $\prod_d \mathbf{V}[d]$ values, $\{\theta_{\mathbf{v}} : \mathbf{v} \in \mathcal{M}_{\mathbf{V}}\}$, where the keys $\mathcal{M}_{\mathbf{V}}$ are pre-determined and fixed, and the values $\{\theta_{\mathbf{v}}\}$ are trained. The domain of the lattice function u is the "interior" of $\mathcal{M}_{\mathbf{V}}$ given by

$$\mathcal{S}_D = \left[-\left\lfloor\frac{\mathbf{V}[1]-1}{2}\right\rfloor, \left\lceil\frac{\mathbf{V}[1]-1}{2}\right\rceil\right] \times \ldots \times \left[-\left\lfloor\frac{\mathbf{V}[D]-1}{2}\right\rfloor, \left\lceil\frac{\mathbf{V}[D]-1}{2}\right\rceil\right] \subset \mathbb{R}^D. \quad (5)$$

To evaluate the lattice function $u(\cdot)$, we find the set of 2^D vertices surrounding $\mathbf{x}$ given by $\mathcal{N}(\mathbf{x}) = \{\lfloor\mathbf{x}[1]\rfloor, \lfloor\mathbf{x}[1]\rfloor+1\} \times \ldots \times \{\lfloor\mathbf{x}[D]\rfloor, \lfloor\mathbf{x}[D]\rfloor+1\}$ and linearly interpolate their parameters using standard multilinear interpolation, i.e.,

$$u(\mathbf{x}) = \sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x})} \theta_{\mathbf{v}} \Phi_{\mathbf{v}}(\mathbf{x}), \quad (6)$$

where $\Phi_{\mathbf{v}}(\mathbf{x})$ is the linear interpolation weight on vertex $\mathbf{v}$ given by

$$\Phi_{\mathbf{v}}(\mathbf{x}) = \prod_{d=1}^{D} \left(1 + (\mathbf{x}[d] - \mathbf{v}[d])(-1)^{I_{\mathbf{v}[d]=\lfloor\mathbf{x}[d]\rfloor}}\right), \quad (7)$$

and I is the standard indicator function.

4.4. Unimodal Lattice Functions

To make a unimodal lattice, we set the number of vertices in each feature $\mathbf{V}[d]$ to be an odd number, and fix the center of the lattice's domain at $\mathbf{0}$. Then, we show one needs the following necessary and sufficient linear inequality constraints on the lattice parameters for unimodality:

Lemma 1: Let $u : \mathcal{S}_D \to \mathbb{R}$ be the function of a D-dimensional lattice of size $\mathbf{V} \in \mathbb{N}^D$. For $d = 1, \ldots, D$, denote by $\mathbf{e}_d \in \{0,1\}^D$ the one-hot vector with $\mathbf{e}_d[i] = 1$ iff $i = d$, and for $n \in \mathbb{N}$, denote by $[n]$ the set $\{1, \ldots, n\}$. Let $s \in [D]$. Every restriction of u to a function with s inputs obtained by fixing the last $D - s$ inputs to constants is unimodal with respect to the maximizer $\mathbf{0} \in \mathbb{R}^s$ iff for every $\mathbf{v} \in \mathcal{M}_{\mathbf{V}}$, $\delta_1, \ldots, \delta_s \in \{0,1\}$ such that $\mathbf{v} + \delta_d \mathbf{e}_d, \mathbf{v} - (1-\delta_d)\mathbf{e}_d \in \mathcal{M}_{\mathbf{V}}$ for all $d \in [s]$, it holds that

$$\sum_{d=1}^{s} (\theta_{\mathbf{v}+\delta_d\mathbf{e}_d} - \theta_{\mathbf{v}-(1-\delta_d)\mathbf{e}_d})\mathbf{v}[d] \leq 0. \quad (8)$$

4.5. Finding The Maximizer

Recall that the maximizer of u is at $\mathbf{0}$ by construction, so the maximizer of $h(\mathbf{x};\phi) = u(s(c'(\mathbf{x})))$ is $\hat{\mathbf{x}} = c'^{-1}(\mathbf{0})$. Because in our proposed lattice GON construction c' is D PLFs, the dth component of the maximizer is found by simply inverting the dth PLF of c', which takes $\mathcal{O}(D)$ time overall.

4.6. GONs Generalize Unimodality

Unlike 1D GONs, multi-D GONS generalize unimodal functions:

Prop. 4: Multi-dimensional GONs *generalize* unimodal functions.

4.7. Higher-D GONs with Ensemble of Lattices

A single unimodal lattice must be defined on a regular grid of at least three keypoints over each feature, thus it needs at least 3^D parameters. For better scaling in D, we use a layer made up of an ensemble of T lattices for u (Canini et al., 2016; You et al., 2017).

Let $c : \mathbb{R}^D \to \mathcal{S}_D$ be D 1D monotonic PLFs, with $\mathcal{S}_D = [-V, V]^D$, for some uniform lattice side size $2V + 1 \in \mathbb{N}$. We define the ensemble GON as

$$h(\mathbf{x};\phi) = \alpha_0 + \sum_{t=1}^{T} a_t u_t(\pi_t(c(\mathbf{x}))), \tag{9}$$

where each $\pi_t : \mathcal{S}_D \to \mathcal{S}_Q$ for $t = 1,\dots,T$ with $S_Q = [-V, V]^Q$, is a random projection given by $\pi_t(\mathbf{x}) = (\mathbf{x}[i_{t,1}], \mathbf{x}[i_{t,2}], \dots, \mathbf{x}[i_{t,Q}])$, and each $u_t(x) : \mathcal{S}_Q \to \mathbb{R}, \mathbf{0} \in \mathcal{S}_Q \subseteq \mathbb{R}^Q$ is a unimodal lattice as described above that acts on a (randomly selected) subset of Q entries of $\mathbf{x}$. The T and $Q \leq D$ are hyperparameters; larger T and Q increases the flexibility of the fit. The α_0 and $\alpha_t \geq 0, t = 1,\dots,T$ are learned ensemble parameters.

Prop. 5 shows that the ensemble function in (9) is still unimodal with maximizer $\mathbf{0}$, and thus one can again find its maximizer by simplying invert the first layer PLFs: $\hat{\mathbf{x}} = c^{-1}(\mathbf{0})$.

Prop. 5: Let $I \subseteq \mathbb{R}$ be an interval containing 0. For an integer $d > 0$ denote by $\mathcal{S}_d$ the Cartesian product I^d. Fix an integer $Q > 0$, let $u_t : \mathcal{S}_Q \to \mathbb{R}$, $t = 1,\dots,T$ be unimodal functions with maximizer $\mathbf{0} \in \mathcal{S}_Q$ and let $\pi_t : \mathcal{S}_D \to \mathcal{S}_Q$ be projections given by $\pi_t(\mathbf{x}) = (\mathbf{x}[i_{t,1}], \mathbf{x}[i_{t,2}], \dots, \mathbf{x}[i_{t,Q}])$. Finally, let $u : \mathcal{S}_D \to \mathbb{R}$, be the ensemble function given by $u(\mathbf{x}) = a_0 + \sum_{t=1}^{T} a_t u_t(\pi_t(\mathbf{x})), a_t \geq 0$. Then $u(\mathbf{x})$ is unimodal with maximizer $\mathbf{0} \in \mathcal{S}_D$.

4.8. CGON Maximizer

Similarly, using the above constructions for the CGON layers with D PLFs for c', the CGON global maximizer can also be computed in $\mathcal{O}(D)$ time unless the evaluation of $r(\mathbf{z}))$ requires more than $\mathcal{O}(D)$.

4.9. Training PLF and Lattice GONs

Given a standard loss L and a training set $\{\mathbf{x}_i, y_i\}$ for $i = 1,\dots,N$, collect the parameters of both c and u into a parameter vector $\phi \in \mathbb{R}^p$, collect all the linear inequality constraints to enforce the monotonicity of c and the unimodality of u into one matrix inequality $A^T\phi \geq 0$, then train by solving:

$$\arg\min_{\phi} \sum_{i=1}^{N} L(h(\mathbf{x}_i;\phi), y_i) \text{ such that } A^T\phi \geq 0. \tag{10}$$

Note that $A^T\phi \geq 0$ in (10) only forces any monotonic functions in c to be *non-decreasing*, so to force c to be *increasing* for invertibility, if there are any flat segments in any c, we simply treat the rightmost key's parameter to be larger.

To solve (10), we extended the TensorFlow Lattice library (Milani Fard, 2020), which already provides PLF layers, lattice layers, and monotonicity constraints, to also support our new joint unimodality constraints, which are now in the open-sourced TensorFlow Lattice library. As recommended in Milani Fard (2020), we *fixed* the keypoints of c at initialization based on the endpoints and quantiles of the input data, did not train the keypoints of c, and we project onto the linear inequality constraints in (10) after each batch using 10 steps of Dykstra's projection algorithm (Boyle & Dykstra, 1986).

5. Experiments

We compare GONs to DNN's, the convex neural networks of Amos et al. (2017), and GPR at predicting the maximizer (or minimizer) given the same set of N noisy training samples and only one guess. We start with three real-data problems to build intuition. Then we provide statistically significant comparisons for the problem of selecting the best hyperparameters for image classifiers on five datasets, and two popular global optimization benchmark simulations. Table 2 summarizes the experiments.

5.1. Experimental Details

For each experiment and for each method, we train a set of models with different hyperparameter choices, select the best model according to a validation or cross-validation metric (metric described below), then use the global maximizer

Table 2. Summary of Experiments.

EXPERIMENT	# OF FEATURES	# TRAINING SAMPLES	# TEST CANDIDATES FOR x^*
MONARCH	1	373	2 TO 28 FOR EACH OF 30 DYNASTIES
PUZZLE	2	36	27
WINE	61	84,642	24,185
HYPERPARAMETERS	7	25	PRACTICALLY INFINITE
GRIEWANK	4–16	100–10,000	INFINITE
ROSENBROCK	4–16	100–10,000	INFINITE

of a model trained on the selected hyperparameters as the method's predicted maximizer.

In practice, given a model $h(x)$, one would predict the maximizer over the entire input domain: $\hat{x} = \arg\max_{x \in \mathbb{R}^D} h(x)$, and that is what we do for our two simulations. However, for our real-data experiments we cannot judge arbitrary predictions, because we do not have the true label for every x, so we limit the prediction to the inputs seen in the test set: $\hat{x} = \arg\max_{x \in \mathcal{X}_{\text{Test}}} h(x)$, where $\mathcal{X}_{\text{Test}}$ is the test set inputs for which we have labels.

For all experiments, we score each predicted maximizer $\hat{x}$ by the test output corresponding to test input $\hat{x}$.

GPR was trained with sklearn's GPR function. GONs and CGONs are trained with the TensorFlow Lattice library. All other models were trained in TensorFlow with Keras layers, and used ADAM (Kingma & Ba, 2015) with a default learning rate of .001 (preliminary experiments with learning rates of 0.0003 as suggested in Liu et al. (2020) yielded similar results). Batch size was N for $N < 100$, 1000 for the larger wine experiment in Sec 5.4, and 100 otherwise. FICNN and PICNN used the formulations in (2) and (3) respectively, from Amos et al. (2017). For a CGON with M-dimensional conditional inputs, we use $r(z)[j] = \sum_{i=1}^{M} PLF_i^j(z[i]), j = 1, \ldots, D$, where $z[i]$ and $r(z)[i]$ denote the i-th entry of z and $r(z)$. For simplicity we use $S(x) = x$, i.e., the identity function. Hyperparameter choices are detailed in the Appendix. For training, labels were scaled to lie in $[0, 1]$ to make it easier to specify hyperparameter options. All TensorFlow models were trained to minimize MSE loss. Code for some experiments can be found at https://github.com/google-research/google-research/tree/master/gon.

5.2. Predict the Longest-Reigning Monarch

Do royal dynasties become more stable over time? Here we predict the rank of the monarch in a dynasty that rules the longest. Input $x \in [1, 36]$ is the rank-order of a monarch, and label y is how many years that monarch reigned. Fig. 3 shows the different validated functions given 373 such training samples from 30 dynasties. The 1d GON model is unimodal, with its peak at the 6th monarch. The DNN model is less smooth with more peaks and valleys, but agrees with the GON model that the global maxima should be at the 6th monarch. The convex neural network (FICNN) is over-regularized for this problem, and predicts the first monarch will rule the longest. The GPR model predicts the 7th monarch will rule the longest. See the Supplemental for more details and results.

5.3. Predict the Best Selling Jigsaw Puzzle

We partnered with the jigsaw puzzle company Artifact Puzzles to predict what kind of jigsaw puzzle sells best. This data has been made publicly available at www.kaggle.com/senzhaogoogle/puzzlesales. Each puzzle is characterized by $D = 2$ features: the number of pieces in the puzzle in the range $[79, 1121]$, and the century of the artwork rounded to the nearest century from 1500 to 2000. The non-IID train/validation/test sets had $36/32/27$ puzzles that were new in 2017/2018/2019, each puzzle's label was that puzzle's sales that year during the holiday season.

We optimized over 8 different hyperparameter choices for each model type (see the Supplemental for details), scoring candidate hyperparameters by the actual sales of the validation-set puzzle it predicted would sell best. Similarly, the test metric was the actual sales of the test puzzle predicted to have the best sales by the optimized trained model.

Figure 4 shows the best model for each function class. Table 3 shows the GON predicted best seller from the test set did have the highest actual sales. The sales of the DNN and FICNN models are close, but if one did not restrict the maximizer to the test set and instead took the global maximizer over the feature space, the DNN and FICNN models predicted sales would be maximized by a puzzle with 0 pieces.

5.4. Predict the Highest-Rated Wine

Using Kaggle data from Wine Enthusiast Magazine[3], we predict which wine will have the highest quality rating in [80, 100]. We take as given the wine's real-valued price in dollars, 21 Boolean features denoting the country of origin,

[3] www.kaggle.com/dbahri/wine-ratings

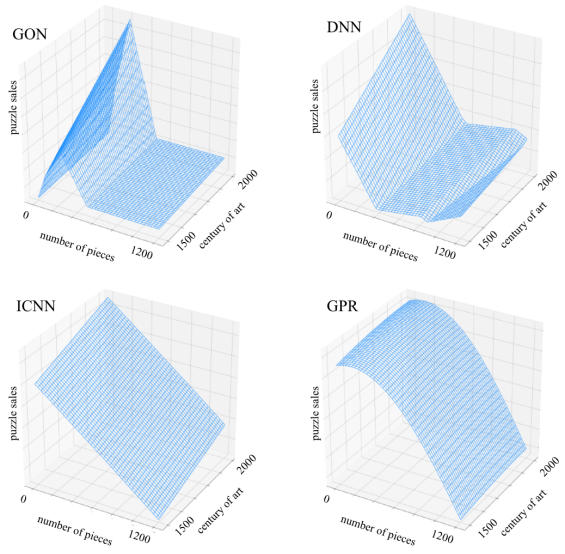

Figure 4. Validated models for predicting best-selling puzzles as a function of number of pieces and century of art. The *global* arg max of the DNN and FICNN predicts the best-selling puzzle would have 0 pieces! That bad extrapolation was fairly stable over hyperparameters (see Supplemental).

Table 3. New Puzzle Sales: Metrics for the Trained Models With Best Validation Scores. Bold is best. Train Root MSE and the actual Test Sales (of the test puzzle the surrogate function predicts will sell best) are puzzles sold (scaled). Global Arg Max is the surrogate function's exact global maximizer.

	TRAIN RMSE	TEST SALES	GLOBAL ARG MAX
DNN	78.2	173	0 PIECES, YEAR 2000
FICNN	78.8	173	0 PIECES, YEAR 2000
GPR	87.3	2	146 PIECES, YEAR 2000
GON	77.4	**182**	230 PIECES, YEAR 2000

and 39 Boolean features describing the wine by Wine Enthusiast Magazine for a total of $D = 61$ features. There are 84,642 train samples, 12,092 validation samples, and 24,185 test samples, all IID. We omit results for GPR for this problem because we could not train GPR in sklearn using our machines with 128GB of memory. We validated each model over 15 hyperparameter choices (details in Appendix); the validation score was the actual quality of the model's highest quality prediction over the validation set.

Table 4 reports the validated models and their predicted best wines. Consistent across hyperparameter choices, the DNNs and FICNNs relied heavily on the price feature, and the best DNN wrongly predicted that the most expensive test wine would be the highest-quality. The GON achieved the best test score by choosing a test wine with many refined

Table 4. Best Wine: Results for Models With Best Validation Scores. Units are *quality points* from $[80, 100]$. Bold is best.

MODEL	TRAIN RMSE	TEST PTS	PREDICTED BEST TEST WINE
DNN	2.54	88	$3300, ACID, JUICY, TANNIN, FRANCE
FICNN	2.20	94	$1100, COMPLEX, EARTH, LEES, TIGHT, AUSTRIA
GON	2.28	**97**	$375, ACID, BRIGHT, COMPLEX, ELEGANT, REFINED, STRUCTURE, TANNIN, ITALY

Boolean features known to correlate with higher-quality wines.

We also compared the ability of CGON, PICNN, and DNN models to predict the highest-quality wine conditioned on six different price points. We used the same hyperparameters for these models as for the unconditioned experiments. The CGON won or tied 5 of the 6 experiments.

5.5. Image Classifier Hyperparameter Optimization

The next experiment predicts the best hyperparameters for image classifiers as shown in Table 5. Note that the goal of this experiment is not to produce another state-of-the-art image classifier, but to demonstrate that GON is more effective in selecting hyperparameters than other methods. Also note that we focused on the problem of making one single best guess of the maximizer. Other hyperparameter selection methods generate a sequence of guesses for exploration, such as bilevel optimization using iterative differentiation, approximate implicit differentiation, and Bayesian optimization. One can surely modify GON to generate a sequence of guesses, but one would need to consider the exploration-exploitation tradeoff and evolving training set, which is beyond the scope of this paper.

We ran experiments on five benchmark datasets: CIFAR-10/100 (Krizhevsky, 2009), Fashion MNIST (Xiao et al., 2017), MNIST (LeCun et al., 2010) and cropped SVHN (Netzer et al., 2011) datasets with their default train/test splits, and use 10% of the train set as validation. We use ReLU-activated image classifiers: $Conv(f1, k) \rightarrow MaxPool(p) \rightarrow Conv(f2, k) \rightarrow MaxPool(p) \rightarrow Conv(f3, k) \rightarrow Dense(u) \rightarrow Dense(\#classes)$, where filters/units $f1, f2, f3, u \in [8, 128]$, kernel/pool size $k, p \in [2, 5]$ and training epochs $e \in [1, 20]$ are treated as hyperparameters.

To train the optimizers, we randomly sample $N = 25$ sets of hyperparameters $(f1, f2, f3, u, k, p, e)$, then train $N = 25$ image classifiers on each set of hyperparameters, and use their $N = 25$ validation errors as the train labels to fit the

Table 5. Mean Test Accuracy ± 95% error margin with predicted best hyperparameters. Bold is stat. sig. best or tied for best at 95% level.

METHOD	CIFAR-10	CIFAR-100	FASHION MNIST	MNIST	SVHN
GON	**70.9%** ± *0.4%*	**37.0%** ± *0.4%*	**91.1%** ± *0.1%*	**98.9%** ± *0.1%*	**87.8%** ± *0.3%*
FICNN	67.7% ± *1.6%*	34.5% ± *1.6%*	**91.0%** ± *0.1%*	**99.0%** ± *0.1%*	**88.3%** ± *0.4%*
DNN	67.8% ± *1.0%*	33.6% ± *1.2%*	90.7% ± *0.2%*	**99.0%** ± *0.1%*	**86.1%** ± *3.1%*
GPR	66.0% ± *3.5%*	34.9% ± *0.9%*	90.7% ± *0.2%*	**98.9%** ± *0.1%*	85.6% ± *1.8%*
CGON	**69.7%** ± *0.5%*	**35.5%** ± *0.6%*	**91.3%** ± *0.1%*	**98.9%** ± *0.1%*	**87.8%** ± *0.3%*
PICNN	65.5% ± *3.5%*	32.4% ± *1.0%*	91.1% ± *0.1%*	**99.0%** ± *0.1%*	**87.3%** ± *2.8%*
DNN	67.5% ± *1.1%*	32.5% ± *1.7%*	90.9% ± *0.2%*	**98.9%** ± *0.1%*	**87.2%** ± *1.4%*
GPR	**68.1%** ± *2.1%*	**34.7%** ± *0.8%*	91.1% ± *0.1%*	**98.9%** ± *0.1%*	**87.0%** ± *1.9%*

response surfaces over the $D = 7$ dimensional feature space of hyperparameter choices. For the conditional models, we conditioned on $e = 10$ training epochs.

For GON and CGON, we found the global maximizer of the response surface over the $D = 7$ hyperparameter space by inverting the PLFs. For FICNN and PICNN, we used ADAM to find their global maximizers, taking advantage of the fact that their response surfaces are concave, similar to the original work of Zico et al. (Amos et al., 2017). For DNN and GPR, we first randomly generated a candidate set $\mathcal{X}_{\text{candidates}}$ of 100,000 hyperparameter-sets from the $D = 7$-dim domain, and set $\hat{\mathbf{x}} = \arg\max_{\mathbf{x}\in\mathcal{X}_{\text{candidates}}} h(\mathbf{x};\phi)$, and use that predicted best hyperparameters to re-train the image classifier and report the test error rates. For each of the 5 image classification problems, we ran the entire experiment 50 times, each with a different random draw of the $N = 25$ random hyperparameters set used to train the response surface. See the Supplemental for details.

Table 5 shows that GON and CGON are statistically significantly the best or tied for the best for all 5 image datasets.

5.6. Simulations on Benchmark Functions

We ran extensive simulations with two popular benchmark functions: the banana-shaped Rosenbrock and pocked-convex Griewank functions (Horst & Pardalos, 1995). Table 6 shows the results for increasing D, N and train noise σ (Supplemental has full experimental details and results). GON was statistically significantly the best predictor of the global minimizer for all the simulation set-ups for both Rosenbrock (6) and Griewank. CGON was also consistently best for Rosenbrock. For Griewank, CGON was the best or tied for the best in 6 slices, and PICNN, DNN and GPR were the best or tied for the best in 0, 5 and 3 slices, respectively.

Table 6. Rosenbrock simulation results with 95% conf. intervals. Bold is stat. sig. best or tied for best.

D	GON	FICNN	DNN	GPR
4	**213** ±*24*	833 ±*92*	2259 ±*151*	2310 ±*186*
8	**492** ±*37*	2370 ±*188*	5019 ±*209*	4791 ±*241*
12	**734** ±*47*	3575 ±*278*	7407 ±*220*	7022 ±*257*
16	**1004** ±*21*	5750 ±*128*	9466 ±*91*	9133 ±*107*
σ	GON	FICNN	DNN	GPR
0.25	**282** ±*8*	818 ±*34*	4064 ±*110*	6582 ±*201*
0.5	**419** ±*13*	1273 ±*52*	5183 ±*116*	3830 ±*117*
1.0	**557** ±*16*	2805 ±*97*	6216 ±*113*	5737 ±*95*
2.0	**797** ±*22*	4382 ±*118*	7075 ±*105*	6445 ±*77*
4.0	**999** ±*26*	6383 ±*139*	7651 ±*105*	6478 ±*70*
N	GON	FICNN	DNN	GPR
$1e2$	**473** ±*9*	2983 ±*78*	6462 ±*83*	5820 ±*90*
$1e3$	**897** ±*19*	4281 ±*104*	6237 ±*87*	5923 ±*97*
$1e4$	**463** ±*13*	2133 ±*71*	5414 ±*97*	5700 ±*103*

6. Conclusions

We defined GONs by the shape constraints they must obey: invertible layers and unimodal layers. We showed they provide better or comparable accuracy as DNNs, convex functions, and GPR for predicting a maximizer. We focused on using PLF and lattice layers because they are arbitrarily flexible models and amenable to shape constraints, but other invertible layers could be used (e.g. Behrmann et al. (2019)), or other unimodal (or even convex) layers. Computationally, we found the time to fit a GON was similar to ICNNs and DNNs using Tensorflow for the same number of parameters, but a DLN GON maximizer can be found exactly in $\mathcal{O}(D)$ time.

We hypothesize that GONs will generally work well in practice when there are large-scale trends that GON's unimodality shape constraint can take advantage of. If the true landscape has many separate local minima without a larger-scale trend, then the GON shape may be less useful.

We focused on the set-up where only one guess is needed, but an important open question is how to use GONs as a response function as the core of a global optimization algorithm that gets new labels and can make a series of guesses.

References

Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *ICML*, pp. 146–155, 2017.

Barlow, R. E., Bartholomew, D. J., and Bremner, J. M. *Statistical inference under order restrictions; the theory and application of isotonic regression*. Wiley, 1972.

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 573–582, 2019.

Box, G. E. P. and Wilson, K. B. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society: Series B*, 13:1–45, 1951.

Boyle, J. P. and Dykstra, R. L. A method for finding projections onto the intersection of convex sets in Hilbert spaces. In *Advances in Order Restricted Statistical Inference*, pp. 28–47. 1986.

Canini, K., Cotter, A., Gupta, M., Milani Fard, M., and Pfeifer, J. Fast and flexible monotonic functions with ensembles of lattices. In *Advances in Neural Information Processing Systems 29*, pp. 2919–2927. 2016.

Cannon, A. J. Non-crossing nonlinear regression quantiles. *Stochastic Environmental Research and Risk Assessment*, 32:3207–3225, 2018.

Chatterjee, S. and Lafferty, J. Adaptive risk bounds in unimodal regression. *Bernoulli*, 2019.

Chen, Y. and Samworth, R. J. Generalized additive and index models with shape constraints. *Journal Royal Statistical Society B*, 2016.

Chen, Y., Shi, Y., and Zhang, B. Optimal control via neural networks: A convex approach. In *International Conference on Learning Representations*, 2019.

Chen, Y., Shi, Y., and Zhang, B. Data-driven optimal voltage regulation using input convex neural networks. *Electric Power Systems Research*, 189, 2020.

Chetverikov, D., Santos, A., and Shaikh, A. The econometrics of shape restrictions. *Annual Review of Economics*, 10:31–63, 2018.

Cotter, A., Gupta, M., Jiang, H., Louidor, E., Muller, J., Narayan, T., Wang, S., and Zhu, T. Shape constraints for set functions. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 1388–1396, 2019.

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., and Garcia, R. Incorporating functional knowledge in neural networks. *Journal of Machine Learning Research*, 10(42): 1239–1262, 2009.

Duindam, J. Dynasties. *Medieval Worlds*, 2:59–78, 2015.

Feldman, S., Gupta, M. R., and Frigyik, B. A. Revisiting Stein's paradox: Multi-task averaging. *Journal of Machine Learning Research*, 15(106), 2014.

Garcia, E. and Gupta, M. Lattice regression. In *Advances in Neural Information Processing Systems 22*, pp. 594–602. 2009.

Garcia, E., Arora, R., and Gupta, M. R. Optimized regression for efficient function evaluation. *IEEE Transactions on Image Processing*, 21(9):4128–4140, 2012.

Gasthaus, J., Benidis, K., Wang, Y., Rangapuram, S. S., Salinas, D., Flunkert, V., and Januschowski, T. Probabilistic forecasting with spline quantile function RNNs. In *AIStats*, volume 89, pp. 1901–1910, 2019.

Gorissen, D., Couckuyt, I., Demeester, P., Dhaene, T., and Crombecq, K. A surrogate modeling and adaptive sampling toolbox for computer based design. *Journal of Machine Learning Research*, 11(68):2051–2055, 2010.

Groeneboom, P. and Jongbloed, G. *Nonparametric estimation under shape constraints*. Cambridge University Press, 2014.

Gunn, L. H. and Dunson, D. B. A transformation approach for incorporating monotone or unimodal constraints. *Biostatistics*, 6(3):434–449, 2005.

Gupta, M., Cotter, A., Pfeifer, J., Voevodski, K., Canini, K., Mangylov, A., Moczydlowski, W., and van Esbroeck, A. Monotonic calibrated interpolated look-up tables. *Journal of Machine Learning Research*, 17(109):1–47, 2016.

Gupta, M., Bahri, D., Cotter, A., and Canini, K. Diminishing returns shape constraints for interpretability and regularization. In *Advances in Neural Information Processing Systems 31*, pp. 6834–6844. 2018.

Gupta, M. R., Louidor, E., Mangylov, O., Morioka, N., Narayan, T., and Zhao, S. Multidimensional shape constraints. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Horst, R. and Pardalos, P. M. *Handbook of Global Optimization*. Springer, 1995.

Howard, A. and Jebara, T. Learning monotonic transformations for classification. In *Advances in Neural Information Processing Systems 20*, pp. 681–688. 2008.

Jones, D. R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

Kennedy, J. and Eberhart, R. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

Kim, J., Lee, J., Vandenberghe, L., and Yang, C.-K. K. Techniques for improving the accuracy of geometric-programming based analog circuit design optimization. In *IEEE/ACM International Conference on Computer Aided Design*, pp. 863–870, 2004.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Köllmann, C., Bornkamp, B., and Ickstadt, K. Unimodal regression using Bernstein–Schoenberg splines and penalties. *Biometrics*, 70(4):783–793, 2014.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Kumar, A. and Levine, S. Model inversion networks for model-based optimization. In *NeurIPS*, 2020.

LeCun, Y., Cortes, C., and Burges, C. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020.

Magnani, A. and Boyd, S. P. Convex piecewise-linear fitting. *Optimization and Engineering*, 10(1):1–17, 2009.

Milani Fard, M. TensorFlow Lattice: Flexible, Controlled, and Interpretable ML, 2020.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. 2011.

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer, 2006.

Pencavel, J. The productivity of work hours. *The Economic Journal*, 125:2052–2076, 2015.

Pinker, S. *The Better Angels Of Our Nature: Why Violence Has Declined*. Viking Penguin, 2011.

Pya, N. and Wood, S. N. Shape constrained additive models. *Statistics and Computing*, 2015.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Shi, Y. and Eberhart, R. A modified particle swarm optimizer. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 69–73, 1998.

Stout, Q. F. Unimodal regression via prefix isotonic regression. *Computational Statistics and Data Analysis*, 53: 289–297, 2008.

Trabucco, B., Kumar, A., Geng, X., and Levine, S. Conservative objective models for effective offline model-based optimization. In *ICML*, 2021.

Wang, S. and Gupta, M. R. Deontological ethics by monotonicity shape constraints. In *AIStats*, 2020.

Wehenkel, A. and Louppe, G. Unconstrained monotonic neural networks. *Advances in Neural Information Processing Systems*, 2019.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

You, S., Ding, D., Canini, K., Pfeifer, J., and Gupta, M. Deep lattice networks and partial monotonic functions. In *Advances in Neural Information Processing Systems 30*, pp. 2981–2989. 2017.

Yu, S., Ahn, S., Song, L., and Shin, J. Robust model adaptation for offline model-based optimization. In *NeurIPS*, 2021.

A. Broader Related Work

In the next two subsections, we review the broader set of related work for GONs: strategies in fitting functions for optimization, and shape constraints.

A.1. Related Work in Fitting Functions for Optimization

The idea of fitting a function and then predicting the maximizer to be the maximizer of the fitted-function (see Fig. 1) dates back to at least Box and Wilson's 1951 paper (Box & Wilson, 1951), which considered fitting interpolating high-order polynomials through all the data (but in practice restricted their experiments to linear and quadratic functions). Such fits are often called *response surfaces* or *surrogates*. This strategy is also used as an intermediary step for convex optimization in *trust region methods* that fit a quadratic function locally to a neighborhood, and then expand or contract the region over which the quadratic is fitted (Nocedal & Wright, 2006). They also considered two issues we do not address in this paper. First, they considered the selection of training examples that would lead to good estimates, e.g., by properly covering the input space, whereas in this paper we take the training examples as given. Second, they noted that one might need to fit a series of such surrogate functions over different subregions of the input space, and we leave this question of specifying a good multi-pass global optimization algorithm open for future work.

Amos et al. (2017) proposed fitting flexible convex (or concave) functions to all the training data. They constructed convex functions through a multi-layer ReLU-activated machine-learned model with the appropriate monotonicity shape constraints to get the convexity. They proposed a *fully-input convex neural network* (referred to as FICNN or just ICNN) for solving the global optimization problem $x^* = \arg\min_{\mathbf{x}} g(\mathbf{x})$, and a *partial-input convex neural network* (PICNN) for the *conditional* global optimization problem $\mathbf{x}^* = \arg\min_{\mathbf{x}} g(\mathbf{x}, \mathbf{z})$. Because their machine-learned functions $h(\mathbf{x}; \phi)$ and $h(\mathbf{x}, \mathbf{z}; \phi)$ are convex in $\mathbf{x}$, they can be minimized numerically to find $\arg\min_{\mathbf{x}} h(\mathbf{x}; \phi)$ and $\arg\min_{\mathbf{x}} h(\mathbf{x}, \mathbf{z}; \phi)$. Others have found this strategy useful (Chen et al., 2019; 2020). However, note that ReLU-activated ICNNs are neither smooth nor strongly convex, which reduces the convergence rate in finding the minimizer of an ICNN.

For non-convex problems, (Jones, 2001) contended that fitting quadratics is "unreliable" because "the surface may not sufficiently capture the shape of the function." Arbitrary machine-learning models have been used as surrogate models (Gorissen et al., 2010). However, for those methods, we cannot use gradient-based methodologies to find their maximizers, and hence the second stage of finding the global optimizer of such models becomes computationally restrictive in high-dimensions. In addition, using an arbitrary surrogate misses the chance to semantically regularize the fitted function to have a shape with a unique global optimum.

A different flexible fitting strategy is kriging, also called Gaussian process regression (GPR) (Rasmussen & Williams, 2006). GPR *interpolates* the training set (Jones, 2001). Computing GPR has complexity $O(N^3)$ for N training examples, and finding its optimizer is problematic as the number of inputs D increases (Jones, 2001; Rasmussen & Williams, 2006).

Compared to the prior work, the proposed GON functions are more flexible than concave functions, but do have a unique global maximizer. Further, the global maximizer of GONs can be specified analytically and found in $\mathcal{O}(D)$ time, without the need for gradient-based algorithms. Further, unlike methods which use arbitrarily flexible fits like DNNs, the proposed GONs use a semantically meaningful regularization strategy, which produces more interpretable and often more accurate results, as shown in Sec. 5.

A.2. Related Work in Shape Constraints

Shape constraints define function classes by specifying their model shape properties (Groeneboom & Jongbloed, 2014; Chetverikov et al., 2018). Fig. 5 shows examples of 1D functions that satisfy the shape constraints of monotonicity, concavity, and unimodality.

The most common and popular shape constraint is *monotonicity*. For example, linear functions with positive slopes are monotonically increasing. In general, a 1D function with $x \in \mathbb{R}$, a function is monotonically increasing if $f(x)$ is non-decreasing as x increases, or monotonically decreasing if the opposite. Here we use the shorthand *monotonic* for either direction. For differentiable functions, a function is monotonic if the first derivative is non-negative everywhere.

A popular flexible 1D function class for satisfying shape constraints is piecewise linear functions (PLF) (Barlow et al., 1972; Howard & Jebara, 2008; Groeneboom & Jongbloed, 2014; Garcia et al., 2012; Gupta et al., 2016), as shown in Figure 5.

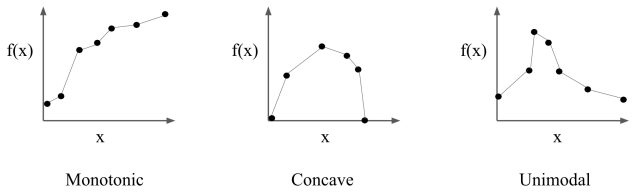

Figure 5. Examples of piece-wise linear functions (PLFs) that satisfy different shape constraints. Each PLF is parameterized by the key-value pairs marked by the black dots.

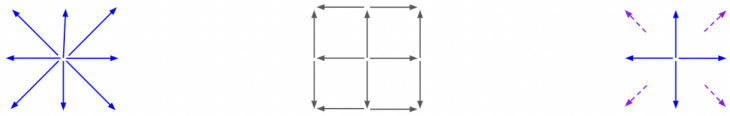

Figure 6. Arrows illustrate different unimodality constraints for a two-dimensional function, with the maximizer at the center of each plot. **Left:** A function is defined to be *unimodal* if it has a maximizer and is non-increasing along all rays starting at the maximizer. **Middle:** The arrows show the prior art: separable unimodality constraints given in (Gupta et al., 2020) on a 3×3 grid of knots that is later bilinearly-interpolated to form the lattice function. Each arrow signifies that the parameter value at the arrow tip must be smaller than or equal to the parameter value at its tail. These separable constraints are sufficient but not necessary for unimodality, as they un-necessarily enforce unimodality on every orthogonal slice of the function. **Right:** The joint unimodality constraints proposed in this paper in (15) for a 3×3 lattice. The solid blue arrows indicate that the parameter value at the arrow tip must be smaller than or equal to the parameter value at the arrow tail. The dashed purple arrows signify that the parameter value at the purple tip must be smaller than or equal to the average of the two knot values at the diagonal corners. This set of constraints in (15) is shown to be both sufficient and necessary for a lattice function to be unimodal.

Monotonicity constraints can also be applied to multi-d functions with $\mathbf{x} \in \mathbb{R}^D$, where the usual definition is that $f(\mathbf{x})$ is increasing in the dth feature, $\mathbf{x}[d]$, if $f(\mathbf{x})$ is non-decreasing as $\mathbf{x}[d]$ increases, with all other features held fixed. A function can be monotonic with respect to a subset of its features. Flexible multi-dimensional monotonic functions have been created by constraining neural networks, (**?**)e.g.,][]Archer:93,Sill:98,ZhangZhang:1999,Daniels:2010,Minin:2010,QuHu:11, Zhu:2017,Cannon:2018,Louppe:2019, support vector machines (Howard & Jebara, 2008), decision trees (**?**)e.g.,][]Qian:2015,PeiHu:2018, and lattices (**?**)e.g.,][]GuptaEtAl:2016,canini:2016,You:2017, or by post-processing (**?**)e.g.,][]Chernozhukov:2010,Lafferty:2018.

Monotonicity shape constraints are usually imposed on machine-learned functions to incorporate domain knowledge that the true function should be monotonic (Gupta et al., 2016), for example, if all else is equal, a house price should be monotonically increasing if it has more square footage. Monotonicity shape constraints have also been used to impose policies such as fairness on functions and provide guarantees the functions work in ways people want (Wang & Gupta, 2020).

Other shape constraints that have been used in the statistics and machine-learning communities are diminishing returns (Pya & Wood, 2015; Chen & Samworth, 2016; Gupta et al., 2018), complementary inputs (Gupta et al., 2020), and dominance between inputs (Gupta et al., 2020).

Another overly-restrictive special case of unimodality is jointly concave functions. These have been produced by summing jointly concave basis functions (Kim et al., 2004; Magnani & Boyd, 2009), or by DNN's with ReLU activations that are constrained to be jointly convex over a subset of features (Dugas et al., 2009; Amos et al., 2017). We show experimentally that concave functions are generally too restrictive for finding and understanding global maximizers.

Shape constraints are often applied to lattice functions, as we do in this paper. Lattices are linearly-interpolated multidimensional look-up tables (Garcia et al., 2012): in one-dimension a lattice is just a piecewise linear function with regular knots. Lattices are arbitrarily flexible, just add more knots (parameters). Because the lattice is parameterized by a regular grid of function values, many shape constraints turn into sparse linear inequality constraints, making training them easy (**?**Gupta et al., 2018; 2020). Higher-dimensional lattice functions are achieved through ensembles (Canini et al., 2016) and multi-layer models (You et al., 2017; Cotter et al., 2019). In the next section, we will show how to construct efficient GONs using multi-layer lattice models with the appropriate shape constraints. Tensorflow Lattice provides an open source library

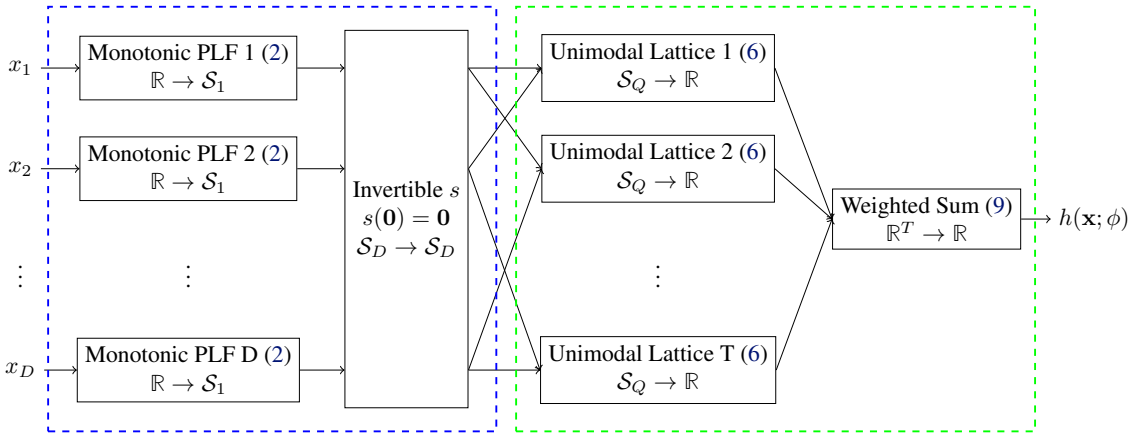

Figure 7. Block diagram for the proposed multi-dim GON using a PLF layer for $c(x)$ and an ensemble of weighted unimodal lattices for $u(\cdot)$. Each unimodal lattice $u(\cdot)$ takes a subset of features as the input. The blue box denotes the invertible function and the green box denotes the unimodal function.

for lattice functions (Milani Fard, 2020), we provide extensions to the Tensorflow Lattice library for GONs.

B. Block Diagrams for Ensemble GON and CGON Models

Fig. 7 gives a block diagram for a DLN GON using an ensemble of lattices as a layer (for more on lattice ensembles see Canini et al. (2016), and for more on ensembles of lattices as a layer in a multi-layer model see You et al. (2017).

Fig. 8 gives a block diagram for a DLN CGON.

C. Puzzles Experiment More Details

To further build intuition, in Figure 9 we show the trained functions with the most flexible hyperparameter choices we validated over. The most flexible GON model used 9 keypoints for the PLF for each of the two inputs for c, and then a 3×3 lattice for u. It has a steep peak at 213 pieces and year 2000. The most flexible DNN, with 4 layers and 8 hidden nodes, is a reasonable model with a peak at 353 pieces and art from year 2000. The GPR model with $\alpha = 1e-6$ overfit good sales data for one of the largest puzzles. The most flexible ICNN model, with 4 layers and 8 hidden nodes, still advises the company to make puzzles with zero pieces.

D. Proofs

Below are the proofs for all of the results in the paper. See Fig. 10 for a Venn diagram summarizing Propositions 1,2,3 and 4.

D.1. Unimodal Functions Are GONs

Prop. 1: Let $g : \mathcal{S}_D \to \mathbb{R}$ be a unimodal function with global maximizer $\mathbf{x}^* \in \mathcal{S}_D \subseteq \mathbb{R}^D$. Then g can be expressed as a GON.

Proof. All unimodal functions are GONs. This is because for any unimodal function g with maximizer $\hat{\mathbf{x}}$, we can reparametrize it as $g(\mathbf{x}) = u(\mathbf{x} + \hat{\mathbf{x}})$ for some u that is unimodal with $\arg\max_{\mathbf{x}} u(\mathbf{x}) = \mathbf{0}$. This can then be written $u(c(x))$ where $c(\mathbf{x}) = \mathbf{x} + \hat{\mathbf{x}}$ is invertible, thus forming a GON. □

D.2. Concave Functions Are Unimodal

Prop. 2: Let $g : \mathcal{S}_D \to \mathbb{R}$ be a concave function with global maximizer $\hat{\mathbf{x}} \in \mathcal{S}_D \subseteq \mathbb{R}^D$. Then g is unimodal with maximizer $\hat{\mathbf{x}}$.

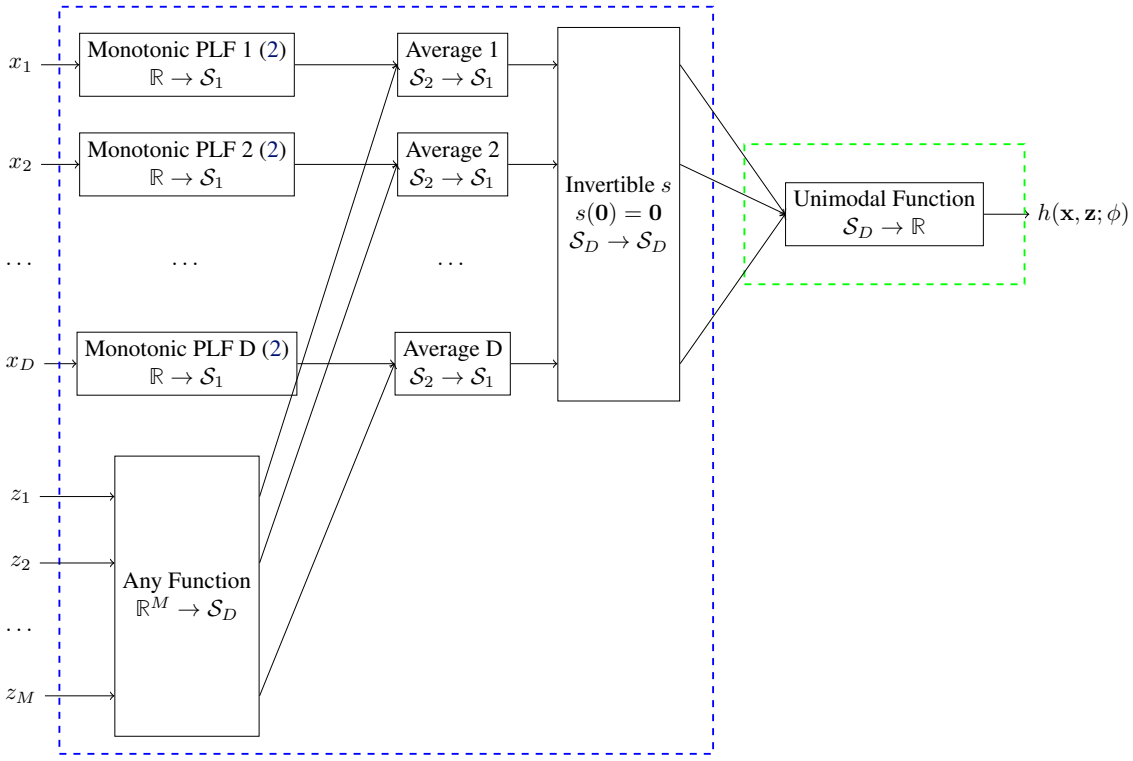

Figure 8. Block diagram of a CGON constructed with DLN layers. The blue box marks the invertible function, and the green box marks the unimodal function and is identical to the green box in Figure 7. There are D inputs $\mathbf{x}$ to optimize over given values for the M conditional inputs $\mathbf{z}$. The model uses D one-dimensional monotonic PLFs to calibrate each of the D inputs $\mathbf{x}$, and uses any function (e.g. a DNN) to map the M conditional inputs $\mathbf{z}$ to D outputs. We add each of the D calibrated $\mathbf{x}$ to one of the D outputs of $\mathbf{z}$, resulting in D inputs s to the unimodal function. Note the whole model will be jointly trained, so the D outputs of $r(\mathbf{z})$ will be optimized to be linearly combined with the $c'(\mathbf{x})$. Then the D outputs from s are separated into an ensemble of unimodal lattices as in (9), whose outputs are linearly combined to get the final prediction.

Proof. Let $r(t) = \hat{\mathbf{x}} + t\mathbf{v}$, $t \geq 0$, be a ray in $\mathbb{R}^D$ originating at $\hat{\mathbf{x}}$. To prove concave g is unimodal, we need to show that $g(r(t))$ is decreasing. Let $r(t_1), r(t_2)$ be two points on the ray with $0 \leq t_1 \leq t_2$. Then it's easily verified that $r(t_1) = ((t_2 - t_1)/t_2)\hat{\mathbf{x}} + (t_1/t_2)r(t_2)$. Now by the concavity of g, we have

$$\begin{aligned} g\left(r\left(t_1\right)\right) &= g\left(\frac{t_2 - t_1}{t_2}\hat{\mathbf{x}} + \frac{t_1}{t_2}r(t_2)\right) \\ &\geq \frac{t_2 - t_1}{t_2}g\left(\hat{\mathbf{x}}\right) + \frac{t_1}{t_2}u\left(r\left(t_2\right)\right) \\ &\geq \frac{t_2 - t_1}{t_2}g\left(r\left(t_2\right)\right) + \frac{t_1}{t_2}g\left(r\left(t_2\right)\right) \\ &= g(r(t_2)), \end{aligned}$$

where the last inequality follows since $g(\hat{\mathbf{x}})$ is the maximum of g. □

D.3. One-Dimensional GONs With Monotonic c Are Unimodal

We show that for one-dimensional GONs because a continuous one-to-one function c defined on a convex set must be monotonic, the GON is unimodal.

Prop. 3: Let $u : \mathcal{S}_1 \to \mathbb{R}, \mathcal{S}_1 \subseteq \mathbb{R}$ be a 1D unimodal function with maximizer at 0. Let $c : \mathbb{R} \to \mathcal{S}_1$ be continuous, bijective, and have 0 in its image. Then $h(x; \phi) = u(c(x))$ is unimodal.

Proof. Let $x^* \in \mathbb{R}$ be the pre-image of 0 under c. Consider any $x_1, x_2 \in \mathbb{R}$ such that $x_1 < x_2 \leq x^*$. Note that to be

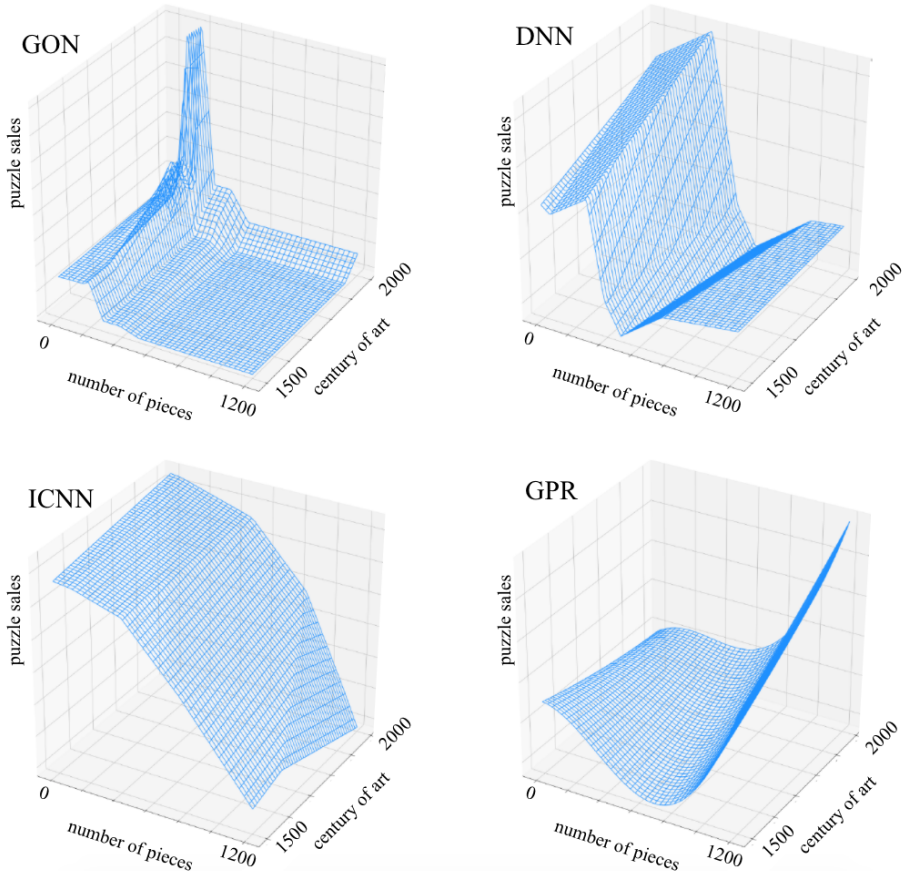

Figure 9. The most flexible models considered when validating hyperparameters for the puzzle sales experiment.

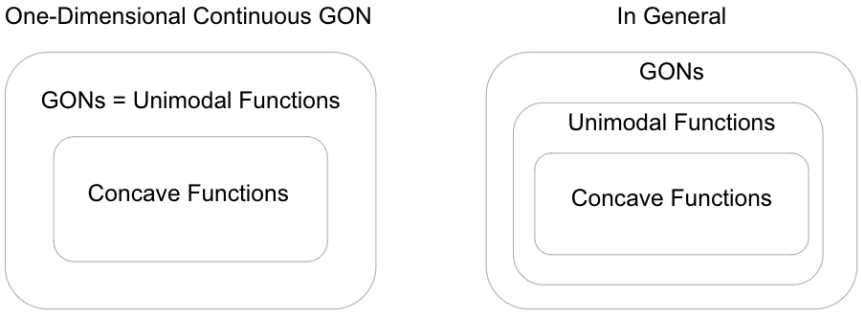

Figure 10. Relationship of GONs to other function classes, summarizing Propositions 1, 2, 3, and 4. **Left:** In the special case of an one-dimensional input, the class of continuous GONs and the class of unimodal functions are identical. **Right:** For multi-dimensional input spaces, the set of GONs is more expressive than the set of unimodal functions, and unimodal functions more expressive than the set of concave functions.

bijective, a continous one-dimensional c with a convex domain must be either monotonically increasing or monotonically decreasing: otherwise, one can find 3 points $x < y < z$ for which either $f(x) < f(y) > f(x)$ or $f(x) > f(y) < f(z)$ and by the mean value theorem it follows that any point c in between $f(x)$ and $f(y)$ will have at least 2 distinct pre-images, contradicting f being one-to-one. Without loss of generality, assume c is monotonically increasing. Then we have $c(x_1) \leq c(x_2) \leq c(x^*) = 0$. Since u is unimodal w.r.t its input 0, we have $u(c(x_1)) \leq u(c(x_2)) \iff h(x_1) \leq h(x_2)$. Therefore h is increasing for $x \leq x^*$. An analogous argument shows that h is decreasing for $x \geq x^*$. Thus h is unimodal

with respect to x^*. □

D.4. Proof for Linear Inequality Constraints To Make A Lattice Function Unimodal

Some visual intution for this lemma is given in Figure 6.

Throughout this section we use the following notation. For a function $u : \mathbb{R}^D \to \mathbb{R}$ we denote its partial derivative with respect to the ith input variable by $\partial_i u$. If u is univariate we denote its derivative by u'. For $n \in \mathbb{N}$ we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and for $\mathbf{x} \in \mathbb{R}^D$, and $d \in [D]$, we denote by $\mathbf{x}[d]$ the dth entry of $\mathbf{x}$. Finally, we denote by $\mathbf{e}_d \in [0, 1]^D$ the one-hot vector where $\mathbf{e}_d[i] = 1$ iff $i = d$.

Consider a lattice with dimension D, size vector $\mathbf{V}$, and parameters $\{\theta_\mathbf{v}\}_{\mathbf{v}\in\mathcal{M}_\mathbf{V}}$. For $\mathbf{x} \in \mathbb{R}^D$, we define the cell of $\mathbf{x}$ to be the set of its 2^D neighboring grid vertices given by $\mathcal{N}(\mathbf{x})=\big\{\lfloor\mathbf{x}[1]\rfloor, \lfloor\mathbf{x}[1]\rfloor+1\big\} \times \ldots \times \big\{\lfloor\mathbf{x}[D]\rfloor, \lfloor\mathbf{x}[D]\rfloor+1\big\}$. Then the lattice function u is given by

$$u(\mathbf{x}) = \sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x})} \theta_\mathbf{v} \Phi_\mathbf{v}(\mathbf{x}), \tag{11}$$

where $\Phi_\mathbf{v}(\mathbf{x})$ is the linear interpolation weight on vertex $\mathbf{v}$ given by:

$$\Phi_\mathbf{v}(\mathbf{x}) = \prod_{d=1}^{D} \Big(1 + (\mathbf{x}[d] - \mathbf{v}[d])(-1)^{I_{\mathbf{v}[d]=\lfloor\mathbf{x}[d]\rfloor}}\Big), \tag{12}$$

and I is the standard indicator function. See (Gupta et al., 2016) for more details.

To prove Lemma 1, we'll need the following supporting lemma (Lemma 2), which gives a formula for the partial derivative of a lattice function.

Lemma 2: Let $f : \mathbb{R}^D \to \mathbb{R}$ be a lattice function with dimension D, size vector $\mathbf{V}$ and parameters $\{\theta_\mathbf{v}\}_{\mathbf{v}\in\mathcal{M}_\mathbf{V}}$. Then for all $d \in [D]$, and $\mathbf{x} \in \mathcal{M}_\mathbf{v}$ with $\mathbf{x}[d] \notin \mathbb{Z}$ (i.e. $\mathbf{x}$ does not lie on the boundary of two adjacent lattice cells in the dth direction)

$$\partial_d f(\mathbf{x}) = \sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x})} \Phi_\mathbf{v}(\mathbf{x})(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}),$$

where $\lceil\mathbf{v}\rceil_{d,\mathbf{x}}$ is $\mathbf{v}+\mathbf{e}_d$, if $\mathbf{v}[d]=\lfloor\mathbf{x}[d]\rfloor$, or $\mathbf{v}$, otherwise, and $\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}=\lceil\mathbf{v}\rceil_{d,\mathbf{x}}-\mathbf{e}_d$.

Proof. Let $\mathbf{x}$ satisfy the requirements of the lemma. By (11), $\partial_d f(\mathbf{x})=\sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x})} \theta_\mathbf{v} \partial_d \Phi_\mathbf{v}(\mathbf{x})$. Denoting by $\lambda(v, x)=1+(x-v)(-1)^{I_{v=\lfloor x\rfloor}}$, for $x\in\mathbb{R}$ and $v\in\mathbb{N}$, we get

$$\begin{aligned}
\partial_d f(\mathbf{x}) &= \sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x})} \theta_\mathbf{v} \partial_d \prod_{i=1}^{D} \lambda(\mathbf{v}[i], \mathbf{x}[i]) \\
&= \sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x})} \theta_\mathbf{v} (-1)^{I_{\mathbf{v}[d]=\lfloor\mathbf{x}[d]\rfloor}} \prod_{i\neq d} \lambda(\mathbf{v}[i], \mathbf{x}[i]),
\end{aligned}$$

where we used the fact that for $x \in \mathbb{R} \setminus \mathbb{Z}$, $\partial\lambda/\partial x = (-1)^{I_{v=\lfloor x\rfloor}}$. Partitioning the set $\mathcal{N}(\mathbf{x})$ of size 2^D into the 2^{D-1} pairs $\{(\mathbf{v}, \lceil\mathbf{v}\rceil_{d,\mathbf{x}}) : \mathbf{v} \in \mathcal{N}(\mathbf{x}), \mathbf{v} = \lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}\}$, we may regroup the summands to obtain

$$\partial_d f(\mathbf{x}) = \sum_{\substack{\mathbf{v}\in\mathcal{N}(\mathbf{x}) \\ \mathbf{v}=\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}} \big(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}\big) \prod_{i\neq d} \lambda(\mathbf{v}[i], \mathbf{x}[i]) \tag{13}$$

Now, observe that $1 = \lambda(\lfloor\mathbf{x}[d]\rfloor, \mathbf{x}[d]) + \lambda(\lfloor\mathbf{x}[d]\rfloor+1, \mathbf{x}[d])$. Thus, for $\mathbf{v} \in \mathcal{N}(\mathbf{x})$ with $\mathbf{v} = \lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}$, it holds that

$$\begin{aligned}
\prod_{i\neq d} \lambda(\mathbf{v}[i], \mathbf{x}[i]) &= \big(\lambda(\lfloor\mathbf{x}[d]\rfloor, \mathbf{x}[d]) + \lambda(\lfloor\mathbf{x}[d]\rfloor+1, \mathbf{x}[d])\big) \cdot \\
&\qquad \prod_{i\neq d} \lambda(\mathbf{v}[i], \mathbf{x}[i]) \\
&= \Phi_\mathbf{v}(\mathbf{x}) + \Phi_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}(\mathbf{x}).
\end{aligned} \tag{14}$$

Substituting (14) into (13), we get

$$\begin{aligned}\partial_d f(\mathbf{x}) &= \sum_{\substack{\mathbf{v}\in\mathcal{N}(\mathbf{x})\\ \mathbf{v}=\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}} \big(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}\big)\big(\Phi_{\mathbf{v}}(\mathbf{x}) + \Phi_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}(\mathbf{x})\big)\\ &= \sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x})} \Phi_{\mathbf{v}}(\mathbf{x})\big(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}\big).\end{aligned}$$

□

We are now ready to prove Lemma 1.

Lemma 1: Let $u : \mathcal{S}_D \to \mathbb{R}$ be the function of a D-dimensional lattice of size $\mathbf{V} \in \mathbb{N}^D$. For $d = 1, \ldots, D$, denote by $\mathbf{e}_d \in \{0,1\}^D$ the one-hot vector with $\mathbf{e}_d[i] = 1$ iff $i = d$, and for $n \in \mathbb{N}$, denote by $[n]$ the set $\{1, \ldots, n\}$. Let $s \in [D]$. Every restriction of u to a function with s inputs obtained by fixing the last $D - s$ inputs to constants is unimodal with respect to the maximizer $\mathbf{0} \in \mathbb{R}^s$ iff for every $\mathbf{v} \in \mathcal{M}_{\mathbf{V}}$, $\delta_1, \ldots, \delta_s \in \{0,1\}$ such that $\mathbf{v} + \delta_d\mathbf{e}_d, \mathbf{v} - (1-\delta_d)\mathbf{e}_d \in \mathcal{M}_{\mathbf{V}}$ for all $d \in [s]$, it holds that

$$\sum_{d=1}^{s} (\theta_{\mathbf{v}+\delta_d\mathbf{e}_d} - \theta_{\mathbf{v}-(1-\delta_d)\mathbf{e}_d})\mathbf{v}[d] \leq 0. \tag{15}$$

Proof. Every restriction obtained from u by fixing the last $D - s$ features to constants is unimodal w.r.t $\mathbf{0}$ if and only if every such restriction is decreasing along rays originating in $\mathbf{0}$. The latter statement can be equivalently restated as: for each $\mathbf{x} \in \mathbb{R}^D$, the function $f_{\mathbf{x}} : [0,1] \to \mathbb{R}$, given by $f_{\mathbf{x}}(t) = u(\mathbf{r}_{\mathbf{x}}(t))$, with $\mathbf{r}_{\mathbf{x}}(t) = (t\mathbf{x}[1], \ldots, t\mathbf{x}[s], \mathbf{x}[s+1], \mathbf{x}[s+2], \ldots, \mathbf{x}[D])$, is decreasing. Since each such $f_{\mathbf{x}}$ is continuous and piecewise-differentiable with finitely many pieces, the last condition is equivalent to requiring that $f'_{\mathbf{x}}(t) \leq 0$ for all $t \in]0,1]$ where the derivative is defined. Observe that it's sufficient to require that for all $\mathbf{x} \in \mathbb{R}^D$, $f'_{\mathbf{x}}(1) \leq 0$, when it's defined, since $f'_{\mathbf{x}}(t) = f'_{\mathbf{r}_{\mathbf{x}}(t)}(1)/t$. Therefore, statement 1 of the lemma holds if and only if

$$\forall \mathbf{x} \in \mathbb{R}^D, f'_{\mathbf{x}}(1) \leq 0 \tag{16}$$

By the chain rule, $f'_{\mathbf{x}}(1) = \sum_{d=1}^{s} \partial_d u(\mathbf{x}) \cdot \mathbf{x}[d]$ and hence using Lemma **??** we have

$$\begin{aligned}f'_{\mathbf{x}}(1) &= \sum_{d\in[s],\mathbf{v}\in\mathcal{N}(\mathbf{x})} \Phi_{\mathbf{v}}(\mathbf{x})(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\mathbf{x}[d]\\ &= \sum_{d,\mathbf{v}} \Phi_{\mathbf{v}}(\mathbf{x})(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})(\mathbf{x}[d] - \lfloor\mathbf{x}[d]\rfloor)\\ &\quad + \sum_{d,\mathbf{v}} \Phi_{\mathbf{v}}(\mathbf{x})(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\lfloor\mathbf{x}[d]\rfloor,\end{aligned}$$

where to get the last equality we added to and subtracted from each summand the quantity $\Phi_{\mathbf{v}}(\mathbf{x})(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\lfloor\mathbf{x}[d]\rfloor$.

Next, for a fixed $d \in [D]$, partitioning the set $\mathcal{N}(\mathbf{x})$ of size 2^D into the 2^{D-1} pairs $\{(\mathbf{v}, \lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}) : \mathbf{v} \in \mathcal{N}(\mathbf{x}), \mathbf{v} = \lceil\mathbf{v}\rceil_{d,\mathbf{x}}\}$, we regroup the terms in the summation and get

$$\begin{aligned}f'_{\mathbf{x}}(1) = \sum_{d,\mathbf{v}:\mathbf{v}=\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} \Big(&(\Phi_{\mathbf{v}}(\mathbf{x}) + \Phi_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}(\mathbf{x}))(\mathbf{x}[d] - \lfloor\mathbf{x}[d]\rfloor)\\ &\times (\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\Big) \qquad (17)\\ + \sum_{d,\mathbf{v}} \Phi_{\mathbf{v}}(\mathbf{x})&(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}} - \theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\lfloor\mathbf{x}[d]\rfloor. \qquad (18)\end{aligned}$$

Now, using (12) and defining $\lambda(v,x)=1+(x-v)(-1)^{I_{v=\lfloor x \rfloor}}$ for $x\in\mathbb{R}$ and $v\in\mathbb{N}$, we have for each $\mathbf{v}\in\mathcal{N}(\mathbf{x})$, with $\mathbf{v}=\lceil\mathbf{v}\rceil_{d,\mathbf{x}}$

$$\begin{aligned}
&\Big(\Phi_{\mathbf{v}}(\mathbf{x})+\Phi_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}(\mathbf{x})\Big)(\mathbf{x}[d]-\lfloor\mathbf{x}[d]\rfloor)\\
&\quad=(\mathbf{x}[d]-\lfloor\mathbf{x}[d]\rfloor)\sum_{\mathbf{w}\in\{\mathbf{v},\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}\}}\prod_{i=1}^{D}\lambda(\mathbf{w}[i],\mathbf{x}[i])\\
&\quad=(\mathbf{x}[d]-\lfloor\mathbf{x}[d]\rfloor)\Big(\prod_{i\neq d}\lambda(\mathbf{v}[i],\mathbf{x}[i])\Big)\cdot\\
&\qquad\Big(\lambda(\lfloor\mathbf{x}[d]\rfloor+1,\mathbf{x}[d])+\lambda(\lfloor\mathbf{x}[d]\rfloor,\mathbf{x}[d])\Big),
\end{aligned}$$

where to get the last equality, observe that for $i\neq d$, the ith entry of $\mathbf{v}$ and $\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}$ is the same. Noting that $\lambda(\lfloor\mathbf{x}[d]\rfloor+1,\mathbf{x}[d])+\lambda(\lfloor\mathbf{x}[d]\rfloor,\mathbf{x}[d])=1$ and that $\mathbf{x}[d]-\lfloor\mathbf{x}[d]\rfloor=\lambda(\mathbf{x}[d],\mathbf{v}[d])$, we get

$$\begin{aligned}
\Big(\Phi_{\mathbf{v}}(\mathbf{x})+\Phi_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}}(\mathbf{x})\Big)(\mathbf{x}[d]-\lfloor\mathbf{x}[d]\rfloor)&=\prod_{i=1}^{D}\lambda(\mathbf{v}[i],\mathbf{x}[i])\\
&=\Phi_{\mathbf{v}}(\mathbf{x})
\end{aligned}\tag{19}$$

Plugging (19) into (17), we get

$$\begin{aligned}
f'_{\mathbf{x}}(1)&=\sum_{d,\mathbf{v}:\mathbf{v}=\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}\Phi_{\mathbf{v}}(\mathbf{x})(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}-\theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\\
&\quad+\sum_{d,\mathbf{v}}\Phi_{\mathbf{v}}(\mathbf{x})(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}-\theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\lfloor\mathbf{x}[d]\rfloor\\
&=\sum_{d,\mathbf{v}}\Big(\Phi_{\mathbf{v}}(\mathbf{x})(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}-\theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\cdot(I_{\mathbf{v}=\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}+\lfloor\mathbf{x}[d]\rfloor)\Big)\\
&=\sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x})}\Phi_{\mathbf{v}}(\mathbf{x})\sum_{d=1}^{s}(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}-\theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\mathbf{v}[d],
\end{aligned}$$

where the last equality holds since, for each $\mathbf{v}\in\mathcal{N}(\mathbf{x})$, $I_{\mathbf{v}=\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}+\lfloor\mathbf{x}[d]\rfloor=\mathbf{v}[d]$.

Hence $f'_{\mathbf{x}}(1)$ is a multi-linear interpolation of the values $\{\sum_{d=1}^{s}(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}-\theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\mathbf{v}[d]\}_{\mathbf{v}}$ on $\mathcal{N}(\mathbf{x})$. Thus requiring that it would be nonpositive for all $\mathbf{x}\in\mathbb{R}^D$ is equivalent to requiring that

$$\sum_{d=1}^{s}(\theta_{\lceil\mathbf{v}\rceil_{d,\mathbf{x}}}-\theta_{\lfloor\mathbf{v}\rfloor_{d,\mathbf{x}}})\mathbf{v}[d]\leq 0,\ \forall\mathcal{N}(\mathbf{x}),\mathbf{v}\in\mathcal{N}(\mathbf{x}).$$

It's easy to verify that these are precisely the inequalities in Statement 2. □

D.5. Unimodal Lattice Not Sufficient For a GON To Be Unimodal

Prop. 4: Multi-dimensional GONs *generalize* unimodal functions.

Proof. Our proof is by counterexample. Let f be the function of the 2D lattice with size $(3,3)$ and vertex values: $\theta_{(0,0)}=3,\theta_{(-1,0)}=\theta_{(1,0)}=2,\theta_{(0,-1)}=\theta_{(0,1)}=0,\theta_{(-1,-1)}=\theta_{(1,-1)}=\theta_{(-1,1)}=\theta_{(1,1)}=1$. It's easy to verify that equation (15) of Lemma 1 holds for s=2. Thus f satisfies the unimodality shape constraint with maximizer $(0,0)$. Now, let $c_1:[0,3]\to[-1,1]$ and $c_2:[0,3]\to[-1,1]$ be the PLFs given by:

$$c_1(x)=\begin{cases}x-1 & \text{if } 0\leq x<1\\ (x-1)/2 & \text{if } 1\leq x\leq 3\end{cases},$$

and

$$c_2(x)=\begin{cases}x-1 & \text{if } 0\leq x<2\\ 1 & \text{if } 2\leq x\leq 3\end{cases}.$$

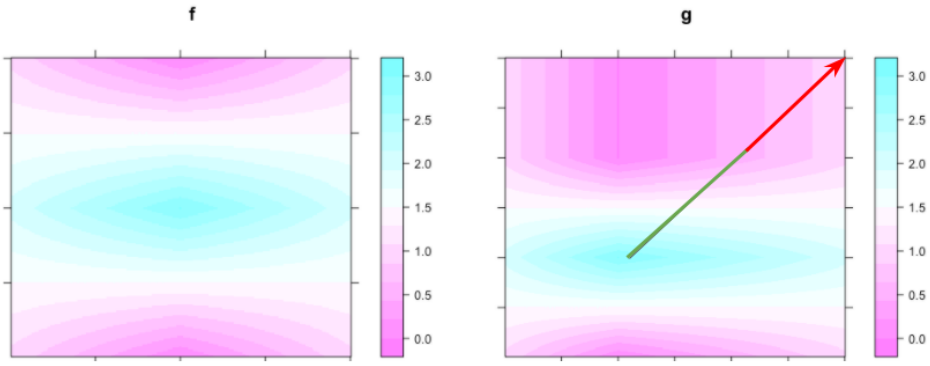

Figure 11. Illustrated counterexample for Prop. 4 with two-dimensional functions f and g shown over the domain $[-1,1]^2$. The function f is unimodal, but the resulting GON function g is not, for example the shown ray starts at the global minimum but the function along that ray is only monotonically decreasing for the green part, the function is decreasing along the red part.

Let $g(x,y) = f(c_1(x), c_2(y))$. Then it can be easily verified that the global maximizer of g is at $(1,1)$ and it is unique. Thus for g to satisfy the unimodal shape constraint, it must do so with maximizer $(1,1)$. However g is not decreasing along the ray $r(t) = (1,1) + t(1,1)$, since $g(r(1)) = g(2,2) = f(1/2,1) = (\theta_{(1,1)} + \theta_{(0,1)})/2 = 1/2$ and $g(r(2)) = g(3,3) = f(1,1) = \theta_{(1,1)} = 1$.

See Figure 11 for the illustration of the f and g functions.

□

D.6. Ensemble of Unimodal Functions is Unimodal

Prop. 5: Let $I \subseteq \mathbb{R}$ be an interval containing 0. For an integer $d > 0$ denote by $\mathcal{S}_d$ the Cartesian product I^d. Fix an integer $Q > 0$, let $u_t : \mathcal{S}_Q \to \mathbb{R}$, $t = 1, \ldots, T$ be unimodal functions with maximizer $\mathbf{0} \in \mathcal{S}_Q$ and let $\pi_t : \mathcal{S}_D \to \mathcal{S}_Q$ be projections given by $\pi_t(\mathbf{x}) = (\mathbf{x}[i_{t,1}], \mathbf{x}[i_{t,2}], \ldots, \mathbf{x}[i_{t,Q}])$. Finally, let $u : \mathcal{S}_D \to \mathbb{R}$, be the ensemble function given by $u(\mathbf{x}) = a_0 + \sum_{t=1}^{T} a_t u_t(\pi_t(\mathbf{x}))$, $a_t \geq 0$. Then $u(\mathbf{x})$ is unimodal with maximizer $\mathbf{0} \in \mathcal{S}_D$.

Proof. Let $\mathbf{z}(r) = r\mathbf{v}$, $r \geq 0$ be a ray in I^D originating in $\mathbf{0}$ for some $\mathbf{v} \in \mathbb{R}^D$. We need to show that $u(\mathbf{z}(r)) = a_0 + \sum_t a_t u_t(\pi_t(\mathbf{z}(r)))$ is decreasing for $r \geq 0$. Since $\pi_t(\mathbf{z}(r)) = r\pi_t(\mathbf{v})$, $r \geq 0$ is a ray in $\mathbb{R}^Q$ in direction $\pi_t(\mathbf{v})$ originating in $\mathbf{0} \subseteq \mathcal{S}_Q$, it follows by the unimodality of each u_t that $u_t(\pi_t(\mathbf{z}(r)))$ is decreasing for $r \geq 0$. The result now follows from the fact that a conical sum of decreasing functions is decreasing. □

E. Details for Monarchs' Reigns Experiments

We provide more details on the data and experimental results.

E.1. Data Details for Monarchs' Reigns Experiments

The data can be downloaded at www.kaggle.com/senzhaogoogle/kingsreign.

All fifty dynasties were sampled from across the globe and from ancient to modern times. The original Monarchs' Reigns dataset (Feldman et al., 2014) (also known as Kings' Reigns, but some of the monarchs were queens or had other titles) consists of 30 royal dynasties, for example the 36 monarchs of the Ottoman Empire from 1299-1922, the 15 monarchs of the Kings of Larsa from 1961 BC to 1674 BC, and the 4 monarchs of the Zulu Dynasty of 1816-1879. We added a test set of 20 additional royal dynasties using the same methodology used for the original dataset based on conversations with the original dataset creator Kyle Stewart (based on conversations about methodology). Example test set dynasties are the 5 monarchs of the 18th century Hotak dynasty in Afghanistan, and the 27 monarch Joseon dynasty of Korea that ended in 1910. All information came from Wikipedia. We will provide a Kaggle notebook for the complete train and test datasets.

The train and test datasets have the following known sampling biases:

Table 7. Longest Reign: Models With Best Validation Scores. Units are years. Bold is best.

Model	Train Set: Root MSE	Test Set: Mean Actual Reign of Model's Arg Max's	Global Arg Max
DNN	14.67	15.05	6th monarch
FICNN	14.89	14.75	1st monarch
GPR	15.54	16.40	7th monarch
GON	14.80	**16.95**	6th monarch

- Dynasties for which there were more complete and well-organized records on Wikipedia were more likely to be sampled. This likely caused under-sampling of pre-Columbian American dynasties, for example.
- An effort was made to sample geographically diverse dynasties, which may have caused under-sampling of some regions and over-sampling of other regions with regards to population.
- An effort was made to sample dynasties across time, which may have caused under-sampling of some timeframes and over-sampling of others with regards to population.
- Current dynasties where the last monarch is still reigning were not sampled.

We note that our use of this data simplifies a number of potentially important factors about the stability of dynasties. For example, in monogamous cultures, it was more difficult to ensure a direct heir than in polygamous cultures (Duindam, 2015). A second issue is simply the definition of dynastic boundaries: what counts as a new dynasty, and has that criteria been sufficiently uniformly applied to the diverse dynasties in this dataset? A third issue is we treated the dynasties as though they were samples drawn IID from the same distribution, but the general reduction in violence over documented history (Pinker, 2011) might imply a shifting distribution towards more stable dynasties, given that many change-overs were due to violence.

E.2. Experimental Details for Monarchs' Reigns

The train set had $N = 30$ dynasties, and the test set had 20 dynasties. For each method, we cross-validated over 18 choices of hyperparameters by leave-one-out cross-validation: we left out one-dynasty at a time and trained a model with each choice of hyperparameters on the other 29 dynasties. For each trained model and left-out dynasty, the predicted maximizer was computed as: $\hat{x} = \arg\max_{x \in \mathcal{X}_{\text{Left-out}}} h(x)$, and we scored $\hat{x}$ by the actual number of years reigned by that monarch in the left-out dynasty. Averaging those scores over the 30 rounds of one-dynasty-left-out formed the overall validation score for that hyperparameter choice. Tables in the Appendix list the 18 hyperparameter choices and corresponding validation scores for each method.

The test metric is the same as the cross-validation metric: for each trained model and each test dynasty, the predicted maximizer was computed as: $\hat{x} = \arg\max_{x \in \mathcal{X}_{\text{Test}}} h(x)$, and we scored $\hat{x}$ by the actual number of years reigned by that monarch in that test dynasty. Averaging those scores over the 20 test dynasties formed the overall test score for that method.

Table 7 shows that the GON model achieved the best test score, followed by the GPR. Note that while both DNN and GON predict a 6th monarch will rule longest, their test scores differ because they made different predictions for the maximizer for test dynasties that have fewer than 6 monarchs, as can be seen in Figure 3.

E.3. Cross-Validation Scores For Different Hyperparameters

The complete cross-validation scores are shown for all the tried hyperparameters in Tables 8, 9 and 10.

F. Details for Puzzles Experiments

The hyperparameter choices were designed to give a range of flexibility. For the FICNN and DNN models, choices were either 3 or 4 layers (3 layers being the default in (Amos et al., 2017)), and either $\{2, 4, 6, 8\}$ hidden nodes. The GPR hyperparameter was the sklearn standard covariance matrix additive smoothing parameter α, ranging from $1e-6$ to 10 in steps of 10. All GON models used a unimodal 3×3 lattice layer for $f(x)$, and varied the number of keypoints in $c(x)$'s PLFs from $K = 2$ to $K = 9$. Because the first and last PLF keypoint are fixed to map to the lattice layer's input domain, the

Table 8. Monarchs' Reigns: GON Validation Scores Over Hyperparameters. Bold is the highest validation score for this model type.

Model	Number Keypoints in f	Number Keypoints in c	Validation Score
GON	3	2	21.23
GON	3	3	20.97
GON	3	5	25.30
GON	3	7	25.26
GON	3	9	**26.00**
GON	3	11	23.8
GON	5	2	22.97
GON	5	3	18.33
GON	5	5	25.27
GON	5	7	20.06
GON	5	9	22.80
GON	5	11	20.93
GON	7	2	22.47
GON	7	3	22.46
GON	7	5	18.33
GON	7	7	18.50
GON	7	9	20.77
GON	7	11	18.73
GON	9	2	22.46
GON	9	3	18.30
GON	9	5	19.73
GON	9	7	19.20
GON	9	9	21.13
GON	9	11	22.10

$K = 2$ case is equivalent to not having a first layer. Any ties were decided in favor of the hyperparameters corresponding to a more-regularized model.

Table 12 and 13 reports actual sales of the highest-predicted validation and test puzzles. The GON was most accurate in predicting the best-selling test puzzle, followed by the DNN and GON. The GPR model chose a test puzzle that was actually a terrible seller.

Our test metric was limited to the test set of puzzles for which there was 2019 sales numbers. In practice though, the business would like to use such a model for guidance as to which new puzzles they should create. For such use, we should ask if the global maximizer is reasonable. As seen in Figure 4, the FICNN and DNN models extrapolated poorly from a popular small puzzle in the train set, leading those models to predict that the global optimizer would be a jigsaw puzzle with zero pieces, which is not reasonable guidance. We questioned whether this was simply bad luck in selecting the hyperparameters, but in fact, 5 of the 8 FICNN models trained predicted the argmax at 0 pieces (see Table 12 in the Appendix). The DNN also only gave reasonable answers for the global maximizer for 3 of its 8 hyperparameter choices.

The five most-flexible GON models consistently predicted a global optimizer would be a puzzle with 190-230 pieces and artwork from around the year 2000. Partners at Artifact said that based on their ten years of sales experience, such puzzles do tend to sell best.

We also note the GON models also generally predicted the best year for art was 2000, which is at the edge of the input domain, which confirms the proposed unimodal shape constraints do not block fitting models with their maximizer on the edge of the input domain.

Table 9. Monarchs' Reigns: FICNN Validation Scores Over Hyperparameters. Bold is the highest validation score for this model type.

Model	Number Layers	Number Hidden Nodes	Validation Score
FICNN	3	2	22.47
FICNN	3	4	22.47
FICNN	3	8	22.20
FICNN	3	16	22.00
FICNN	3	32	21.20
FICNN	3	64	20.70
FICNN	4	2	22.93
FICNN	4	4	**23.17**
FICNN	4	8	21.83
FICNN	4	16	21.63
FICNN	4	32	21.20
FICNN	4	64	20.50
FICNN	5	2	20.36
FICNN	5	4	22.30
FICNN	5	8	20.70
FICNN	5	16	20.96
FICNN	5	32	19.40
FICNN	5	64	20.70
FICNN	6	2	22.06
FICNN	6	4	20.77
FICNN	6	8	21.83
FICNN	6	16	19.73
FICNN	6	32	21.50
FICNN	6	64	19.73

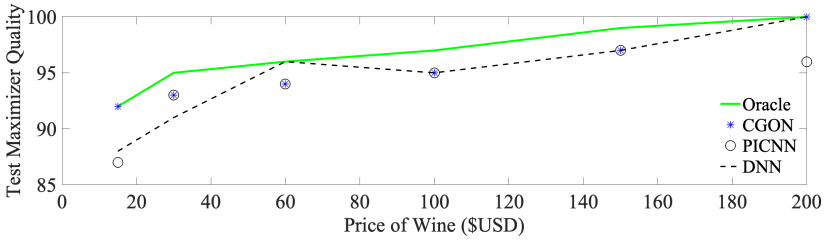

Figure 12. Quality of the predicted highest-quality wine conditioned on price. The oracle marks the true best wine from the test set for each price point.

G. Details for Wine Experiments

Tables 14-16 summarize the validation scores of DNN, FICNN and GON over hyperparameters.

Figure 12 shows the results for the experiments conditioned on price.

H. Details for Hyperparameter Optimization for Image Classifiers

As mentioned in the main paper, image classifers shown in the main paper are trained for $e \in [1, 20]$ epochs. We use ADAM with the default learning rate of 0.001 with a batch size of 128 to train the classifiers.

For GON and CGON, we use an ensemble of D unimodal lattices. All methods are trained for 250 epochs.

Hyperparameters for the optimizers are validated based on 5-fold MSE, which are summarized in Table 17 below.

Table 10. Monarchs' Reigns: DNN Validation Scores Over Hyperparameters. Bold is the highest validation score for this model type.

Model	Number Layers	Number Hidden Nodes	Validation Score
DNN	3	2	21.87
DNN	3	4	18.03
DNN	3	8	17.53
DNN	3	16	19.73
DNN	3	32	21.26
DNN	3	64	22.4
DNN	4	2	19.97
DNN	4	4	20.26
DNN	4	8	20.43
DNN	4	16	19.4
DNN	4	32	20.4
DNN	4	64	21.20
DNN	5	2	22.46
DNN	5	4	18.76
DNN	5	8	22.16
DNN	5	16	21.03
DNN	5	32	22.47
DNN	5	64	24.13
DNN	6	2	22.46
DNN	6	4	19.76
DNN	6	8	21.80
DNN	6	16	24.13
DNN	6	32	**24.70**
DNN	6	64	24.20

I. Details for Simulations with Standard Global Optimization Functions

We ran simulations on two standard benchmark functions, the banana-shaped Rosenbrock function, and the pocked-convex Griewank function, to compare GON against FICNN, DNN, GPR and sample best. For conditional global optimization problems, we compared CGON against PICNN, DNN and GPR.

The multi-dimensional Rosenbrock function has the formula:

$$g(\mathbf{x}) = \sum_{i=1}^{D-1} \left(100 \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2\right). \tag{20}$$

The multi-dimensional Griewank function has the formula:

$$g(\mathbf{x}) = 1 + \frac{1}{4000} \sum_{i=1}^{D} (x_i - 1)^2 - \prod_{i=1}^{D} cos\left(\frac{x_i - 1}{\sqrt{i}}\right). \tag{21}$$

See Figure 13 for a visulization of 2-dimensional Rosenbrock and Griewank functions. For both functions, the true global minimizer is at $\mathbf{x}^* = \mathbf{1}$.

For each function, we randomly generated 50 training sets for each of 60 different experimental set-ups: $D \in \{4, 8, 12, 16\}$ inputs $\times$ $N \in \{100, 1000, 10000\}$ training examples $\times$ $\sigma \in \{0.25, 0.5, 1.0, 2.0, 4.0\}$ noise levels where the training label is $\mathbf{y} = g(\mathbf{x}) + \epsilon$ for $\mathbf{x} \sim Unif(-2, 2)^D, \epsilon \sim \mathcal{N}(0, \sigma g(\mathbf{x}))$. For the conditional global optimization problem, we aim to find $\mathbf{x}^* = \arg\min_{\mathbf{x}} g(\mathbf{x}, \mathbf{z} = \mathbf{0})$, where $\mathbf{x}$ is the first $3D/4$ inputs and $\mathbf{z}$ is the last $D/4$ inputs. Once FICNN/PICNN fit their convex/conditionally-convex functions, their minimizers are found using ADAM with learning rate .001 and 10k steps with projections onto the input domain $[-2, 2]^D$. Details on hyperparameter validation for all methods are in the Appendix.

Table 11. Monarchs' Reigns: GPR Validation Scores Over Hyperparameters. Bold is the highest validation score for this model type.

Model	α	Validation Score
GPR	$\alpha = 1e-12$	18.97
GPR	$\alpha = 1e-11$	17.83
GPR	$\alpha = 1e-10$	17.46
GPR	$\alpha = 1e-9$	16.83
GPR	$\alpha = 1e-8$	18.13
GPR	$\alpha = 1e-7$	20.7
GPR	$\alpha = 1e-6$	21.43
GPR	$\alpha = 1e-5$	21.6
GPR	$\alpha = 1e-4$	14.93
GPR	$\alpha = 1e-3$	18.97
GPR	$\alpha = 1e-2$	21.7
GPR	$\alpha = 1e-1$	22.47
GPR	$\alpha = 1$	22.47
GPR	$\alpha = 10$	22.47
GPR	$\alpha = 100$	**23.27**
GPR	$\alpha = 1e3$	19.63
GPR	$\alpha = 1e4$	19.70
GPR	$\alpha = 1e5$	19.70
GPR	$\alpha = 1e6$	19.70
GPR	$\alpha = 1e7$	19.70
GPR	$\alpha = 1e8$	19.70
GPR	$\alpha = 1e9$	19.70
GPR	$\alpha = 1e10$	19.70
GPR	$\alpha = 1e11$	19.70

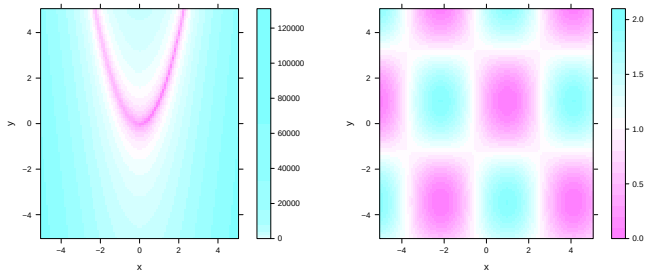

Figure 13. Visualization of 2-dimensional Rosenbrock and Griewank functions.

We found the global maximizer of each response surface as in experiment in Section 5.5. That is, for GON and CGON, we found the global maximizer of the response surface by inverting the PLFs. For FICNN and PICNN, we used ADAM to find their global maximizers. For DNN and GPR, we first generated a finite random set $\mathcal{X}_{\text{candidates}}$ of 100,000 inputs across the domain of and set $\hat{\mathbf{x}} = \arg\max_{\mathbf{x} \in \mathcal{X}_{\text{candidates}}} h(\mathbf{x})$.

Table 18 shows that GON is consistently the best method for all twelve different simulation set-ups. CGON is also consistently best for Rosenbrock. For the globally convex Griewank, CGON is the best or tied for the best in 6 slices, whereas PICNN, DNN and GPR are the best or tied for the best in 0, 5 and 3 slices, respectively. GON and CGON performed especially well in the more challenging cases of large D and high noise σ and few training samples N.

Note that the performance of sample best deteriorates with more training samples, as there is more risk it will overfit a particularly noisy training sample. In fact, in general the performance of the different response surface methods did not

Table 12. New Puzzle Sales: Results for Different Hyperparameters for DNN and FICNN. As marked, the DNN model sometimes came out "flat", that is, it predicted the same sales for all inputs. Ties broken in favor of the smaller/smoother model.

Model	Actual Sales of Highest-Scored Validation Puzzle	Actual Sales of Highest-Scored Test Puzzle	Global Arg Max
DNN 3 layers, 2 hid.	flat model	–	–
DNN 3 layers, 4 hid.	21	30	1200 pieces, year 2000
DNN 3 layers, 6 hid.	74	182	192 pieces, year 2000
DNN 3 layers, 8 hid.	**88**	173	0 pieces, year 2000
DNN 4 layers, 2 hid.	flat model	–	–
DNN 4 layers, 4 hid.	0	7	0 pieces, year 1500
DNN 4 layers, 6 hid.	10	30	192 pieces, year 2000
DNN 4 layers, 8 hid.	16	164	353 pieces, year 2000
FICNN 3 layers, 2 hid.	43	173	0 pieces, year 2000
FICNN 3 layers, 4 hid.	**88**	173	0 pieces, year 2000
FICNN 3 layers, 6 hid.	88	173	0 pieces, year 2000
FICNN 3 layers, 8 hid.	74	182	192 pieces, year 2000
FICNN 4 layers, 2 hid.	88	173	0 pieces, year 2000
FICNN 4 layers, 4 hid.	74	13	0 pieces, year 2000
FICNN 4 layers, 6 hid.	74	182	0 pieces, year 2000
FICNN 4 layers, 8 hid.	88	173	0 pieces, year 2000

Table 13. New Puzzle Sales: Results for Different Hyperparameters for GPR and GON. Ties broken in favor of the smaller/smoother model.

Model	Actual Sales of Highest-Scored Validation Puzzle	Actual Sales of Highest-Scored Test Puzzle	Global Arg Max
GPR $\alpha = 1e-6$	21	3	168 pieces, year 1500
GPR $\alpha = 1e-5$	21	182	242 pieces, year 2000
GPR $\alpha = 1e-4$	74	182	242 pieces, year 2000
GPR $\alpha = 1e-3$	74	182	242 pieces, year 2000
GPR $\alpha = 1e-2$	88	173	68 pieces, year 2000
GPR $\alpha = 1e-1$	43	173	68 pieces, year 2000
GPR $\alpha = 1$	43	173	68 pieces, year 2000
GPR $\alpha = 10$	**88**	2	146 pieces, year 2000
GON 2kp	43	1	600 pieces, year 1700
GON 3kp	31	21	502 pieces, year 1400
GON 4kp	**76**	182	230 pieces, year 2000
GON 5kp	74	182	190 pieces, year 2000
GON 6kp	13	182	212 pieces, year 2000
GON 7kp	74	182	191 pieces, year 2000
GON 8kp	74	182	194 pieces, year 2000
GON 9kp	74	182	213 pieces, year 2000

necessarily get better with more training samples N, which we suspect is due to the fact that as N increases, there is a greater chance of more very noisy samples that confuses the response surface placement of its maximizer.

Table 14. Best Wine: DNN Validation Scores Over Hyperparameters. Bold is the highest validation score for this model type, with ties broken in favor of the smallest model with that validation score. Surprisingly, the DNN consistently chose the same poor test wine as its predicted best. Analysis showed that the DNN's were extrapolating poorly in the high price part of the feature space and putting too much faith in high price as a signal of quality, and that the DNN's prediction is the most expensive test wine.

Model	Val Score	Train MSE	Test Score	Test Maximizer
DNN: 2 layers, 2 nodes	**97**	2.54	88	$3300, acid, juicy, tannin, France
DNN: 2 layers, 4 nodes	97	2.53	88	$3300, acid, juicy, tannin, France
DNN: 2 layers, 8 nodes	97	2.54	88	$3300, acid, juicy, tannin, France
DNN: 2 layers, 16 nodes	97	2.47	88	$3300, acid, juicy, tannin, France
DNN: 2 layers, 32 nodes	97	2.30	88	$3300, acid, juicy, tannin, France
DNN: 2 layers, 64 nodes	92	2.24	94	$1100, complex, earth, lees, tight, Austria
DNN: 3 layers, 2 nodes	97	2.55	88	$3300, acid, juicy, tannin, France
DNN: 3 layers, 4 nodes	97	2.55	88	$3300, acid, juicy, tannin, France
DNN: 3 layers, 8 nodes	97	2.55	88	$3300, acid, juicy, tannin, France
DNN: 3 layers, 16 nodes	97	2.27	88	$3300, acid, juicy, tannin, France
DNN: 3 layers, 32 nodes	97	2.23	88	$3300, acid, juicy, tannin, France
DNN: 3 layers, 64 nodes	97	2.24	88	$3300, acid, juicy, tannin, France
DNN: 4 layers, 2 nodes	97	2.54	88	$3300, acid, juicy, tannin, France
DNN: 4 layers, 4 nodes	97	2.53	88	$3300, acid, juicy, tannin, France
DNN: 4 layers, 8 nodes	97	2.27	88	$3300, acid, juicy, tannin, France
DNN: 4 layers, 16 nodes	97	2.23	88	$3300, acid, juicy, tannin, France
DNN: 4 layers, 32 nodes	97	2.23	88	$3300, acid, juicy, tannin, France
DNN: 4 layers, 64 nodes	97	2.17	88	$3300, acid, juicy, tannin, France

The multi-dimensional Rosenbrock function has formula:

$$g(\mathbf{x}) = \sum_{i=1}^{D-1} \left(100 \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2\right). \tag{22}$$

The multi-dimensional Griewank function has formula:

$$g(\mathbf{x}) = 1 + \frac{1}{4000} \sum_{i=1}^{D} (x_i - 1)^2 - \prod_{i=1}^{D} cos\left(\frac{x_i - 1}{\sqrt{i}}\right). \tag{23}$$

For both functions, the true global minimizer is at $\mathbf{x}^* = (1.0, 1.0, \ldots, 1.0)$.

For both GON and CGON, we first use D PLFs with K keypoints to calibrate the D inputs for optimization. The unimodal function consists of an enesemble of D unimodal lattices, each fuses 3 inputs with V keypoints. For CGON, we let $r : \mathbb{R}^M \to \mathcal{S}_D$ be $r(\mathbf{z})[j] = \sum_{i=1}^{M} PLF_i^j(\mathbf{z}[i]), j = 1, \ldots, D$, where $\mathbf{z}[i]$ and $r(z)[i]$ denote the i-th entry of $\mathbf{z}$ and $r(\mathbf{z})$.

For FICNN and PICNN, we use the formulations in (2) (Figure 1) and (3) (Figure 2), respectively, in Amos, et al. (Amos et al., 2017). All the hidden layers are constructed to have the same hidden dimensions, whenever possible. For DNN, we use fully connected hidden layers with a constant number of hidden nodes across layers. The number of hidden layers and the number of hidden nodes are treated as hyperparameters.

For GPR, we use RBF kernel with $\sigma = 1$, which is the default in the sklearn package. The White Kernel α is treated as a hyperparameter.

For each of the Rosenbrock and Griewank functions, we used grid search to choose hyperparameters for each method that minimize the average $g(\hat{\mathbf{x}})$ over the 600 runs for each function ($5\sigma \times 4D \times 3N \times 10$ repetitions with random seeds), where g denotes the ground truth function. After choosing hyperparameters, we reran the simulation with 50 repetitions, and report the average-50 result in Table 5.6. The hyperprameters of each method are summarized in Table 19.

Table 15. Best Wine: FICNN Validation Scores Over Hyperparameters. Bold is the highest validation score for this model type, with ties broken in favor of the smallest model with that validation score. Like the DNN, analysis showed the FICNN tended to overfit high price as a sign of quality and often chose the most expensive test wine, which actually did not have high points.

Model	Val Score	Train MSE	Test Score	Test Maximizer
FICNN: 2 layers, 2 nodes	97	2.52	88	$3300, acid, juicy, tannin, France
FICNN: 2 layers, 4 nodes	97	2.29	88	$3300, acid, juicy, tannin, France
FICNN: 2 layers, 8 nodes	91	2.33	95	$412, jam, opulent, France
FICNN: 2 layers, 16 nodes	92	2.27	100	$848, acid, hint of, opulent, toast, France
FICNN: 2 layers, 32 nodes	**98**	2.20	94	$1100, complex, earth, lees, tight, Austria
FICNN: 2 layers, 64 nodes	96	2.19	94	$1100, complex, earth, lees, tight, Austria
FICNN: 3 layers, 2 nodes	97	2.53	88	$3300, acid, juicy, tannin, France
FICNN: 3 layers, 4 nodes	96	2.30	94	$1100, complex, earth, lees, tight, Austria
FICNN: 3 layers, 8 nodes	96	2.25	94	$1100, complex, earth, lees, tight, Austria
FICNN: 3 layers, 16 nodes	96	2.23	94	$1100, complex, earth, lees, tight, Austria
FICNN: 3 layers, 32 nodes	91	2.24	96	$351 oak, tannin, tight, toast, Spain
FICNN: 3 layers, 64 nodes	92	2.30	94	$1100, complex, earth, lees, tight, Austria
FICNN: 4 layers, 2 nodes	97	2.40	94	$1100, complex, earth, lees, tight, Austria
FICNN: 4 layers, 4 nodes	97	2.24	94	$900, elegant, Italy
FICNN: 4 layers, 8 nodes	97	2.26	85	$320, acid, crisp, Romania
FICNN: 4 layers, 16 nodes	97	2.28	88	$3300, acid, juicy, tannin, France
FICNN: 4 layers, 32 nodes	97	2.47	88	$3300, acid, juicy, tannin, France
FICNN: 4 layers, 64 nodes	97	2.47	88	$3300, acid, juicy, tannin, France

J. Some Open Questions

We defined GONs (and CGONs) by the shape constraints they must obey: a composition of invertible layers and unimodal layers. We showed how to construct such models using the piece-wise linear functions and lattice layers of DLNs, which are arbitrarily flexible models that are particularly amenable to shape constraints (Gupta et al., 2020; Cotter et al., 2019), but other functions could be used for the invertible layers (Behrmann et al., 2019), and one could use convex networks for the needed unimodal layers (at the cost of some flexibility) (Amos et al., 2017).

Another open question is the choice of loss function when training GONs or other flexible response surfaces. In our experiments, all models were fit using standard mean-squared error. Since the goal of fitting the GON is to predict the maximizer only, it seems intuitive that one should worry more about fitting the examples closer to the (unknown) maximizer. We experimented with loss functions that up-weighted training examples with bigger label values, but, perhaps due to the flexibility of the GONs, did not find they helped much, and eschewed their extra complexity and hyperparameters.

We focused here on the setting where one makes only one prediction. However GONs could also be used as a response surface function within a global optimization algorithm that is able to make a series of guesses. In such a context it might make sense to evolve the neighborhood fit by the GON, or fit many GONs in parallel to different evolving neighborhoods for a multi-agent search like particle-swarm optimization (Kennedy & Eberhart, 1995; Shi & Eberhart, 1998).

Lastly, this work is part of a recent wave of research into shape constraints showing that shape constraints can provide sensible regularization while not hurting useful expressability of AI models (e.g. (Pya & Wood, 2015; Chen & Samworth, 2016; Gupta et al., 2016; Cannon, 2018; Chetverikov et al., 2018; Cotter et al., 2019; Wehenkel & Louppe, 2019; Gasthaus et al., 2019; Wang & Gupta, 2020)). We hope this work will inspire other useful shape constraint regularization strategies for AI.

Table 16. Best Wine: GON Validation Scores Over Hyperparameters. Bold is the highest validation score for this model type, with ties broken in favor of the smallest model with that validation score.

Model	Val Score	Train MSE	Test Score	Test Maximizer
GON 100 2D lattices, 5kp	92	2.31	95	$100 acid, cassis, complex, refined structure, tannin, velvet, US
GON 100 2D lattices, 9kp	93	2.29	97	$375, acid, bright, complex, elegant, refined, structure, tannin, Italy
GON 100 2D lattices, 13kp	93	2.31	97	$375, acid, bright, complex, elegant refined, structure, tannin, Italy
GON 200 2D lattices, 5kp	97	2.30	97	$165 acid, cassis, complex, mineral oak, refined, structure, tannin, US
GON 200 2D lattices, 9kp	**98**	2.28	97	$375, acid, bright, complex, elegant refined, structure, tannin, Italy
GON 200 2D lattices, 13kp	98	2.26	97	$375, acid, bright, complex, elegant refined, structure, tannin, Italy
GON 400 2D lattices, 5kp	96	2.32	95	$100, acid, cassis, complex refined, structure, tannin, velvet, US
GON 400 2D lattices, 9kp	97	2.27	97	$375, acid, bright, complex, elegant refined, structure, tannin, Italy
GON 400 2D lattices, 13kp	97	2.25	94	$1100, complex, earth, lees tight, Austria
GON 800 2D lattices, 5kp	97	2.28	97	$375, acid, bright, complex, elegant refined, structure, tannin, Italy
GON 800 2D lattices, 9kp	98	2.26	94	$1100, complex, earth, lees tight, Austria
GON 800 2D lattices, 13kp	93	2.24	97	$375, acid, bright, complex, elegant refined, structure, tannin, Italy
GON 1600 2D lattices, 5kp	97	2.26	96	$180 butter, complex, lees, mineral US
GON 1600 2D lattices, 9kp	97	2.37	94	$1100, complex, earth, lees tight, Austria
GON 1600 2D lattices, 13kp	94	2.22	96	$450, cream, dense, mineral tight, France

Table 17. Hyperparameters of optimizers.

	Global Optimization				Conditional Global Optimization			
Rosenbrock	GON	FICNN	DNN	GPR	CGON	PICNN	DNN	GPR
PLF kps per input K	5	-	-	-	5	-	-	-
Lattice kps per input V	3	-	-	-	3	-	-	-
Inputs each lattice fuses	2	-	-	-	2	-	-	-
Num hidden layers	-	1	1	-	-	1	1	-
Num hidden nodes	-	256	32	-	-	32	32	-
α in GPR	-	-	-	0.01	-	-	-	0.01

Table 18. Rosenbrock and Griewank simulation results with 95% confidence intervals. We report $g(\hat{x})$ for each predicted minimizer $\hat{x}$, averaged over the slice that fit each row's description (e.g. all 750 runs = 50 random seeds $\times 3N \times 5\sigma$ where $D = 4$). Lower is better for all metrics. Bold means the method is statistically significantly the best or tied for best at 95% level. In the global optimization problem, for DNN and GPR, we take the predicted minimizer as the smallest prediction over 100,000 random sampled points from the domain. In the conditional optimization problem, for DNN and GPR, the 100k sampled inputs are restricted to have the conditional inputs **0**. For sample best, we use x_{i^*} where $i^* = \arg\min_{i=1\ldots N} y_i$. Hyperperameters are chosen based on $g(\hat{x})$ of independent runs.

	Global Optimization					Conditional Global Optimization			
Rosenbrock	GON	FICNN	DNN	GPR	Sample Best	CGON	PICNN	DNN	GPR
$D = 4$	**213** ± *24*	833 ± *92*	2259 ± *151*	2310 ± *186*	3271 ± *156*	**903** ± *68*	1473 ± *132*	1769 ± *136*	1558 ± *157*
$D = 8$	**492** ± *37*	2370 ± *188*	5019 ± *209*	4791 ± *241*	5463 ± *237*	**1340** ± *76*	5799 ± *283*	4334 ± *201*	3822 ± *226*
$D = 12$	**734** ± *47*	3575 ± *278*	7407 ± *220*	7022 ± *257*	7559 ± *273*	**1762** ± *86*	10650 ± *401*	6475 ± *213*	5430 ± *256*
$D = 16$	**1004** ± *21*	5750 ± *128*	9466 ± *91*	9133 ± *107*	9571 ± *113*	**2164** ± *36*	15682 ± *177*	8085 ± *82*	7855 ± *91*
$\sigma = 0.25$	**282** ± *8*	818 ± *34*	4064 ± *110*	6582 ± *201*	1576 ± *59*	**943** ± *22*	5478 ± *178*	3269 ± *102*	5054 ± *165*
$\sigma = 0.5$	**419** ± *13*	1273 ± *52*	5183 ± *116*	3830 ± *117*	6102 ± *97*	**1093** ± *23*	6825 ± *193*	4260 ± *101*	3763 ± *126*
$\sigma = 1.0$	**557** ± *16*	2805 ± *97*	6216 ± *113*	5737 ± *95*	7685 ± *102*	**1500** ± *32*	8622 ± *205*	5405 ± *104*	3772 ± *96*
$\sigma = 2.0$	**797** ± *22*	4382 ± *118*	7075 ± *105*	6445 ± *77*	8385 ± *108*	**1986** ± *40*	10151 ± *217*	6184 ± *98*	5143 ± *76*
$\sigma = 4.0$	**999** ± *26*	6383 ± *139*	7651 ± *105*	6478 ± *70*	8583 ± *109*	**2188** ± *44*	10929 ± *217*	6710 ± *91*	5599 ± *65*
$N = 100$	**473** ± *9*	2983 ± *78*	6462 ± *83*	5820 ± *90*	5042 ± *80*	**1583** ± *30*	11407 ± *173*	5736 ± *74*	5077 ± *80*
$N = 1000$	**897** ± *19*	4281 ± *104*	6237 ± *87*	5923 ± *97*	6481 ± *93*	**1665** ± *29*	8451 ± *152*	5196 ± *77*	4777 ± *87*
$N = 10000$	**463** ± *13*	2133 ± *71*	5414 ± *97*	5700 ± *103*	7875 ± *101*	**1379** ± *25*	5345 ± *133*	4564 ± *91*	4145 ± *95*
Griewank	GON	FICNN	DNN	GPR	Sample Best	CGON	PICNN	DNN	GPR
$D = 4$	**0.45** ± *0.007*	0.49 ± *0.009*	0.82 ± *0.012*	0.71 ± *0.014*	1.12 ± *0.015*	0.81 ± *0.007*	0.79 ± *0.010*	0.79 ± *0.012*	**0.70** ± *0.013*
$D = 8$	**0.51** ± *0.006*	0.83 ± *0.006*	0.91 ± *0.007*	0.99 ± *0.007*	1.03 ± *0.008*	0.93 ± *0.004*	0.96 ± *0.004*	**0.91** ± *0.006*	0.94 ± *0.008*
$D = 12$	**0.59** ± *0.005*	0.90 ± *0.004*	0.97 ± *0.004*	1.02 ± *0.002*	1.04 ± *0.004*	0.97 ± *0.003*	1.00 ± *0.002*	**0.96** ± *0.004*	1.01 ± *0.003*
$D = 16$	**0.64** ± *0.005*	0.94 ± *0.003*	1.00 ± *0.002*	1.01 ± *0.002*	1.02 ± *0.003*	**0.97** ± *0.002*	1.01 ± *0.001*	0.99 ± *0.003*	1.01 ± *0.002*
$\sigma = 0.25$	**0.53** ± *0.006*	0.65 ± *0.008*	0.73 ± *0.010*	0.74 ± *0.012*	0.68 ± *0.012*	0.87 ± *0.005*	0.86 ± *0.008*	**0.72** ± *0.010*	0.76 ± *0.011*
$\sigma = 0.5$	**0.52** ± *0.006*	0.71 ± *0.008*	0.86 ± *0.009*	0.80 ± *0.011*	1.07 ± *0.006*	0.91 ± *0.005*	0.91 ± *0.007*	0.86 ± *0.008*	**0.78** ± *0.010*
$\sigma = 1.0$	**0.52** ± *0.006*	0.79 ± *0.008*	0.95 ± *0.007*	0.95 ± *0.009*	1.13 ± *0.007*	**0.93** ± *0.005*	0.95 ± *0.006*	**0.93** ± *0.007*	**0.92** ± *0.009*
$\sigma = 2.0$	**0.58** ± *0.008*	0.86 ± *0.008*	1.03 ± *0.005*	1.06 ± *0.006*	1.17 ± *0.008*	**0.94** ± *0.005*	0.98 ± *0.006*	1.00 ± *0.006*	1.03 ± *0.007*
$\sigma = 4.0$	**0.60** ± *0.008*	0.93 ± *0.007*	1.05 ± *0.005*	1.10 ± *0.006*	1.19 ± *0.008*	**0.95** ± *0.006*	1.00 ± *0.005*	1.05 ± *0.006*	1.09 ± *0.006*
$N = 100$	**0.59** ± *0.006*	0.91 ± *0.006*	0.99 ± *0.004*	1.00 ± *0.005*	0.98 ± *0.006*	**0.95** ± *0.005*	0.98 ± *0.004*	0.98 ± *0.004*	0.99 ± *0.005*
$N = 1000$	**0.52** ± *0.005*	0.79 ± *0.007*	0.98 ± *0.005*	0.94 ± *0.008*	1.04 ± *0.008*	**0.91** ± *0.004*	0.97 ± *0.005*	0.97 ± *0.006*	0.92 ± *0.007*
$N = 10000$	**0.53** ± *0.004*	0.66 ± *0.006*	0.81 ± *0.008*	0.85 ± *0.010*	1.13 ± *0.008*	0.90 ± *0.004*	0.87 ± *0.006*	**0.79** ± *0.008*	0.84 ± *0.009*

Table 19. Simulation: hyperparameters.

	Global Optimization				Conditional Global Optimization			
Rosenbrock	GON	FICNN	DNN	GPR	CGON	PICNN	DNN	GPR
PLF kps per input K	10	-	-	-	10	-	-	-
Lattice kps per input V	3	-	-	-	3	-	-	-
Num hidden layers	-	2	2	-	-	2	2	-
Num hidden nodes	-	16	32	-	-	16	16	-
α in GPR	-	-	-	1.0	-	-	-	1.0
Griewank	GON	FICNN	DNN	GPR	CGON	PICNN	DNN	GPR
PLF kps per input K	10	-	-	-	10	-	-	-
Lattice kps per input V	3	-	-	-	3	-	-	-
Num hidden layers	-	4	2	-	-	2	2	-
Num hidden nodes	-	32	16	-	-	16	16	-
α in GPR	-	-	-	1.0	-	-	-	1.0