EXPRESSIVE MONOTONIC NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

The monotonic dependence of the outputs of a neural network on some of its inputs is a crucial inductive bias in many scenarios where domain knowledge dictates such behavior. This is especially important for interpretability and fairness considerations. In a broader context, scenarios in which monotonicity is important can be found in finance, medicine, physics, and other disciplines. It is thus desirable to build neural network architectures that implement this inductive bias provably. In this work, we propose a weight-constrained architecture with a single residual connection to achieve exact monotonic dependence in any subset of the inputs. The weight constraint scheme directly controls the Lipschitz constant of the neural network and thus provides the additional benefit of robustness. Compared to currently existing techniques used for monotonicity, our method is simpler in implementation and in theory foundations, has negligible computational overhead, is guaranteed to produce monotonic dependence, and is highly expressive. We show how the algorithm is used to train powerful, robust, and interpretable discriminators that achieve competitive performance compared to current state-of-the-art methods across various benchmarks, from social applications to the classification of the decays of subatomic particles produced at the CERN Large Hadron Collider.

1 INTRODUCTION

The need to model functions that are monotonic in a subset of their inputs is prevalent in many ML applications. Enforcing monotonic behaviour can help improve generalization capabilities (Milani Fard et al., 2016; You et al., 2017) and assist with interpretation of the decision-making process of the neural network (Nguyen & Martínez, 2019). Real world scenarios include various applications with fairness, interpretability, and security aspects. Examples can be found in the natural sciences and in many social applications. Monotonic dependence of a model output on a certain feature in the input can be informative of how an algorithm works—and in some cases is essential for real-word usage. For instance, a good recommender engine will favor the product with a high number of reviews over another with fewer but otherwise identical reviews (*ceteris paribus*). The same applies for systems that assess health risk, evaluate the likelihood of recidivism, rank applicants, filter inappropriate content, *etc*.

In addition, robustness to small perturbations in the input is a desirable property for models deployed in real world applications. In particular, when they are used to inform decisions that directly affect human actors—or where the consequences of making an unexpected and unwanted decision could be extremely costly. The continued existence of adversarial methods is a good example for the possibility of malicious attacks on current algorithms (Akhtar et al., 2021). A natural way of ensuring the robustness of a model is to constrain its Lipschitz constant. To this end, we recently developed an architecture whose Lipschitz constant is constrained by design using layer-wise normalization which allows the architecture to be more expressive than the current state-of-the-art with stable and fast training (Redacted, 2021). Our algorithm has been adopted to classify the decays of subatomic particles produced at the CERN Large Hadron Collider in the real-time data-processing system of the LHCb experiment, which was our original motivation for developing this novel architecture.

In this article, we present expressive monotonic Lipschitz networks. This new class of architectures employs the Lipschitz bounded networks from Redacted (2021) along with residual connections to implement monotonic dependence in any subset of the inputs by construction. It also provides exact robustness guarantees while keeping the constraints minimal such that it remains a universal approx-

imator of Lipschitz continuous monotonic functions. We show how the algorithm is used to train powerful, robust, and interpretable discriminators that achieve competitive performance compared to current state-of-the-art methods across various benchmarks, from social applications to its original target application: the classification of the decays of subatomic particles produced at the CERN Large Hadron Collider.

2 Related Work

Prior work in the field of monotonic models can be split into two major categories.

- **Built-in and constrained monotonic architectures**: Examples of this category include Deep Lattice Networks (You et al., 2017) and networks in which all weights are constrained to have the same sign (Sill, 1998). The major drawbacks of most implementations of constrained architectures are a lack of expressiveness or poor performance due to superfluous complexity.
- **Heuristic and regularized architectures (with or without certification)**: Examples of such methods include Sill & Abu-Mostafa (1996) and Gupta et al., which penalizes point-wise negative gradients on the training sample. This method works on arbitrary architectures and retains much expressive power but offers no guarantees as to the monotonicity of the trained model. Another similar method is Liu et al. (2020), which relies on Mixed Integer Linear Programming to certify the monotonicity of piece-wise linear architectures. The method uses a heuristic regularization to penalize the non-monotonicty of the model on points sampled uniformly in the domain during training. The procedure is repeated with increasing regularization strength until the model passes the certification. This iteration can be expensive and while this method is more flexible than the constrained architectures (valid for MLPs with piece-wise linear activations), the computational overhead of the certification process can be prohibitively expensive.

Our architecture falls into the first category. However, we overcome both main drawbacks: lack of expressiveness and impractical complexity. Other related works appear in the context of monotonic functions for normalizing flows, where monotonicity is a key ingredient to enforce invertibility (De Cao et al., 2020; Huang et al., 2018; Behrmann et al., 2019; Wehenkel & Louppe, 2019).

3 Methods

The goal is to develop a neural network architecture representing a vector-valued function

$$f : \mathbb{R}^d \to \mathbb{R}^n, \quad d, n \in \mathbb{N} \tag{1}$$

that is provably monotonic in any subset of its inputs. We first define a few ingredients.

Definition 3.1 (Monotonicity)**.** Let $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{x}_\mathbb{S} \equiv \mathbf{1}_\mathbb{S} \odot \boldsymbol{x}$, and the Hadamard product of $\boldsymbol{x}$ with the indicator vector $\mathbf{1}_\mathbb{S}(i) = [i \in \mathbb{S} \subseteq \{1, \cdots, d\}]$.

We say that outputs $\mathbb{Q} \subseteq \{1, \cdots, n\}$ of f are monotonically increasing in features $\mathbb{S}$ if

$$f(\boldsymbol{x}'_\mathbb{S} + \boldsymbol{x}_{\bar{\mathbb{S}}})_i \leq f(\boldsymbol{x}_\mathbb{S} + \boldsymbol{x}_{\bar{\mathbb{S}}})_i \quad \forall i \in \mathbb{Q} \text{ and } \forall \boldsymbol{x}'_\mathbb{S} \leq \boldsymbol{x}_\mathbb{S} \tag{2}$$

where $\bar{\mathbb{S}}$ denotes the complement of $\mathbb{S}$ and the inequality on the right uses the product (or component-wise) order.

Definition 3.2 (Lip^p function)**.** $g : \mathbb{R}^d \to \mathbb{R}^n$ is Lip^p if it is Lipschitz continuous with respect to the L^p norm in every output dimension, *i.e.*,

$$||g(\boldsymbol{x}) - g(\boldsymbol{y})||_\infty \leq \lambda \|\boldsymbol{x} - \boldsymbol{y}\|_p \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \,. \tag{3}$$

3.1 Lipschitz Monotonic Networks (LMN)

We will henceforth and without loss of generality only consider scalar-valued functions ($n = 1$). We start with a model $g(\boldsymbol{x})$ that is Lip^1 with Lipschitz constant λ. Note the choice of $p = 1$ for the p-norm. It is crucial for universal approximation as it allows a well defined maximum directional derivative for each input regardless of the gradient direction. More details on this can be found below

and in Fig. **??**. $p = 1$ has the convenient side effect that we can tune the robustness requirement for each input individually.

With a model $g(\boldsymbol{x})$ we can define an architecture with built-in monotonicity by adding a term that has gradient λ with respect to $\boldsymbol{x}_s$:

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) + \lambda(\mathbf{1}_s \cdot \boldsymbol{x}) = g(\boldsymbol{x}) + \lambda \sum_{i \in s} x_i. \tag{4}$$

This residual connection $\lambda(\mathbf{1}_s \cdot \boldsymbol{x})$ enforces monotonicity in the input subset $\boldsymbol{x}_s$:

$$\frac{\partial g}{\partial x_i} \in [-\lambda, \lambda], \quad \forall\, i \in \mathbb{N}_{1:n} \tag{5}$$

$$\Rightarrow \frac{\partial f}{\partial x_i} = \frac{\partial g}{\partial x_i} + \lambda \geq 0 \;\; \forall\, \boldsymbol{x} \in \mathbb{R}^n, i \in s\,. \tag{6}$$

The importance of the norm choice The construction presented here only works with Lip^1 constraints as $\text{Lip}^{p\neq 1}$ functions introduce dependencies between the partial derivatives, see Fig. **??**. Therefore, $p = 1$ is the only norm that bounds the gradient within the green square. That might seem counter-intuitive, as $\|v\|_1 \geq \|v\|_2 \geq \|v\|_\infty$ for column vectors. However, we consider gradients with respect to individual inputs to be row vectors, since the first dimension is usually a batching dimension. The row vector $p = 1$ norm is equivalent to the column vector $p = \infty$ norm. When shifting the constraints by introducing the linear term, the green square allows for all possible gradient configurations, given that we can choose λ freely. As a counter example, the red circle, corresponding to $p = 2$ constraints, prohibits important areas in the configuration space.

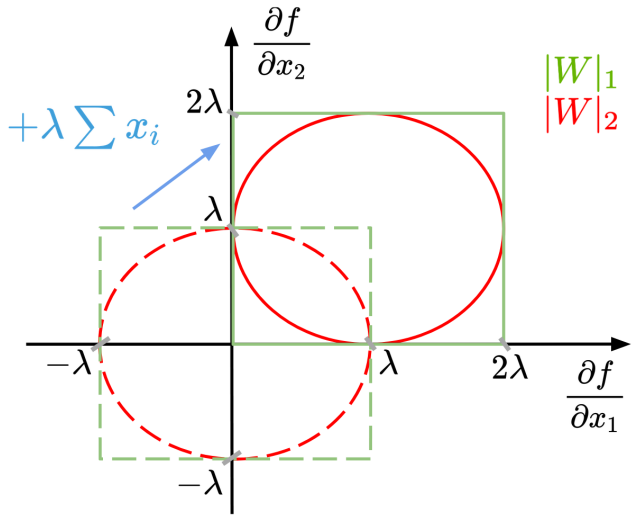


Figure 1: p-norm constrained gradients showing (red) $p = 2$ and (green) $p = 1$. The gradient of a function $g(\boldsymbol{x})$ that is $\text{Lip}^{p=2}$ resides within the dashed red line. For a $\text{Lip}^{p=1}$ function, the boundary is the green dashed line. Note that $\boldsymbol{x}$ is a row vector in this definition. The residual connection (in blue) effectively shifts the possible gradients to strictly positive values and thus enforces monotonicity. Note how the red solid circle does not include all possible gradient configurations. It does for instance not allow for very small gradients in both inputs, whereas the green square includes all configurations, up to an element-wise maximum of 2λ . A $\text{Lip}^{p\neq 1}$ constrained architecture is not suitable for monotonic networks.

To be able to represent all monotonic Lip^1 functions with 2λ Lipschitz constant, the construction of $g(\boldsymbol{x})$ needs to be a universal approximator of Lip^1 functions. In the next section, we will discuss possible architectures for this task.

3.2 LIP$^{p=1}$ APPROXIMATORS

Our goal is to construct a universal approximator of Lip^1 functions, *i.e.*, we would like the hypothesis class to have two properties:

1. It always satisfies Eq. 3, *i.e.*, be Lip^1.
2. It is able to fit all possible Lip^1 functions. In particular, the bound in Eq. 3 needs to be attainable $\forall\, \boldsymbol{x}, \boldsymbol{y}$.

Lip^1 constrained models To satisfy the first requirement, fully connected networks can be Lipschitz bounded by constraining the matrix norm of all weight matrices (Redacted, 2021; Gouk et al., 2020; Miyato et al., 2018). We recursively define the layer l of the fully connected network of depth D with activation σ as

$$\boldsymbol{z}^l = \sigma(\boldsymbol{z}^{l-1})\boldsymbol{W}^l + \boldsymbol{b}^l, \tag{7}$$

where $\boldsymbol{z}^0 = \boldsymbol{x}$ is the input and $f(\boldsymbol{x}) = z^D$ is the output of the neural network. It follows that $g(\boldsymbol{x})$ satisfies Eq. 3 if

$$\prod_{i=1}^{D} \|\boldsymbol{W}^i\|_1 \leq \lambda \tag{8}$$

and σ has a Lipschitz constant less than or equal to 1. There are multiple ways to enforce Eq. 8. Two existing possibilities that involve scaling by the operator norm of the weight matrix (Gouk et al., 2020) are:

$$\boldsymbol{W}^i \to \boldsymbol{W}'^i = \lambda^{1/D} \frac{\boldsymbol{W}^i}{\max(1, \|\boldsymbol{W}^i\|_1)} \qquad \text{or} \qquad W^i \to W'^i = \frac{\boldsymbol{W}^i}{\max(1, \lambda^{-1/D} \cdot \|\boldsymbol{W}^i\|_1)}\,. \tag{9}$$

In our studies, the latter variant seems to train slightly better. However, in some cases it might be useful to use the former to avoid the scale imbalance between the neural network's output and the residual connection used to induce monotonicity. We note that in order to satisfy Eq. 8, it is not necessary to divide the entire matrix by its 1-norm. It is sufficient to ensure that the absolute sum over each column is constrained:

$$\boldsymbol{W}^i \to \boldsymbol{W}'^i = \boldsymbol{W}^i \text{diag}\left(\frac{1}{\max\left(1, \lambda^{-1/D} \cdot \sum_j |W^i_{jk}|\right)}\right)\,. \tag{10}$$

This novel normalization scheme tends to give even better training results in practice, because the constraint is applied in each column individually. This reduces correlations of constraints, in particular, if a column saturates the bound on the norm, the other columns are not impacted. While Eq. 10 may not be suitable as a general-purpose scheme, *e.g.* it would not work in convolutional networks, its performance in training in our analysis motivates its use in fully connected architectures and further study of this approach in future work.

In addition, the constraints in Eqs. Eq. 9 and Eq. 10 can be applied in different ways. For example, one could normalize the weights directly before each call such that the induced gradients are propagated through the network like in Miyato et al. (2018). While one could come up with toy examples for which propagating the gradients in this way hurts training, it appears that this approach is what usually is implemented for spectral norm in PyTorch and TensorFlow (Miyato et al., 2018) . Alternatively, the constraint could be applied by projecting any infeasible parameter values back into the set of feasible matrices after each gradient update as in Algorithm 2 of Gouk et al. (2020).

Constraining according to Eq. 8 is not the only way to enforce Lip^1. Anil et al. (2019) provide an alternative normalization scheme:

$$\|\boldsymbol{W}^1\|_{1,\infty} \cdot \prod_{i=2}^{m} \|\boldsymbol{W}^i\|_\infty \leq \lambda \tag{11}$$

Similarly to how the 1-norm of a matrix is a column-wise maximum, the ∞-norm of a matrix is determined by the maximum 1-norm of all rows and $\|W\|_{1,\infty}$ simply equals the maximum absolute value of an element in the matrix. Therefore, normalization schemes similarly to Eq. 10, in this case row- or element-wise, can be employed to enforce the constraints in Eq. 11.

Preserving expressive power Guaranteeing that the model is Lipschitz bounded is not sufficient, it must also able to saturate the bound to be able to model all possible Lip^1 functions. Some Lipschitz network architectures, *e.g.* Miyato et al. (2018), tend to over constrain the model such that it cannot fit all Lip^1 functions due to *gradient attenuation*. For many problems this is a rather theoretical issue. However, it becomes a practical problem for the monotonic architecture since it often works on the edges of its constraints, for instance when partial derivatives close to zero are required, see Fig. **??**. As a simple example, the authors of Huster et al. (2018) showed that ReLU networks are unable to fit the function $f(x) = |x|$ if the layers are norm-constrained with $\lambda = 1$. The reason lies in fact that ReLU, and most other commonly used activations, do not have unit gradient with respect to the inputs over their entire domain. While element-wise activations like ReLU cannot have unit gradient over the whole domain without being exactly linear, the authors of Anil et al. (2019) explore activations that introduce non-linearities by reordering elements of the input vector. They propose **GroupSort** as an alternative to point-wise activations, and it is defined as follows:

$$\begin{aligned}\sigma_G(\boldsymbol{x}) &= \text{sort}_{1:G}(\boldsymbol{x}_{1:G}) + \text{sort}_{G+1:2G}(\boldsymbol{x}_{G+1:2G}) + \dots \\ &= \sum_{i=0}^{n/G-1} \text{sort}_{iG+1:(i+1)G}(\boldsymbol{x}_{iG+1:(i+1)G}),\end{aligned} \tag{12}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x}_{i:j} = \mathbf{1}_{i:j} \odot \boldsymbol{x}$, and $\text{sort}_{i:j}$ orders the elements of a vector from indices i to j and leaves the other elements in place. This activation sorts an input vector in chunks (groups) of a fixed size G. The GroupSort operation has a gradient of unity with respect to every input, giving architectures constrained with Eq. 8 greatly increased expressive power. In fact, Anil et al. (2019) prove that GroupSort networks with the normalization scheme in Eq. 11 are universal approximators of Lip^1 functions. Therefore, these networks fulfill the two requirements outlined in the beginning of this section.

For universal approximation to be possible, the activation function used needs to be gradient norm preserving (GNP), *i.e.,* have gradient 1 almost everywhere. Householder activations are another instance of GNP activations of which GroupSort-2 is a special case (Singla et al., 2021). The Householder activation is defined as follows:

$$\sigma(\boldsymbol{z}) = \begin{cases} \boldsymbol{z} & \boldsymbol{z}\boldsymbol{v} > 0 \\ \boldsymbol{z}(\mathbf{I} - 2\boldsymbol{v}\boldsymbol{v}^T) & \boldsymbol{z}\boldsymbol{v} \leq 0 \end{cases} \tag{13}$$

Here, $\boldsymbol{z}$ is the preactivation row vector and $\boldsymbol{v}$ is any column unit vector. Householder Lipschitz Networks naturally inherit the universal approximation property.

In summary, we have constructed a neural network architecture $f(\boldsymbol{x})$ via Eq. 4 that can provably approximate all monotonic Lipschitz bounded functions. The Lipschitz constant of the model can be increased arbitrarily by controlling the parameter λ in our construction.

4 Experiments

"Beware of bugs in the above code, I have only proved it correct, not tried it" (Knuth). In the spirit of Donald Knuth, in this section we test our algorithm on many different domains to show that it works well in practice and gives competitive results, as should be expected from a universal approximator.

4.1 Toy Example

Figure 2 shows a toy example where both a monotonic and an unconstrained network are trained to regress on a noisy one-dimensional dataset. The true underlying model used here is monotonic, though an added Gaussian noise term can obscure this in any realization. As can be seen in Fig. 2, no matter how the data are distributed at the edge of the support, the monotonic Lipschitz network is always non-decreasing outside of the support as guaranteed by our architecture. Such out-of-distribution guarantees can be extremely valuable in cases where domain knowledge dictates monotonic behavior is either required or desirable.

4.2 Real-Time Decision-Making at 40 MHz at the LHC

Because many physical systems are modeled with well-known theoretical frameworks that dictate the properties of the system, monotonicity can be a crucial inductive bias in the physical sciences.

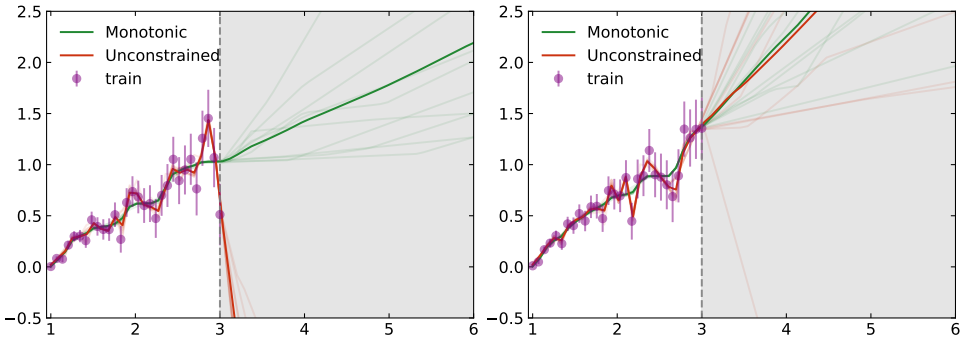


Figure 2: Our monotonic architecture (green) and an unconstrained network (red) trained on two realizations (purple data points) of a one dimensional dataset. The shaded regions are where training data were absent. Each model is trained using 10 random initialization seeds. The dark lines are averages over the seeds, which are each shown as light lines. The unconstrained models exhibit overfitting of the noise, non-monotonic behavior, and highly undesirable and unpredictable results when extrapolating beyond the region occupied by the training data. Conversely, the monotonic Lipschitz models are always monotonic, even in scenarios where the noise is strongly suggestive of non-monotonic behavior. In addition, the Lipschitz constraint produces much smoother models.

For instance, modeling enthalpy, a thermodynamic quantity measuring the total heat content of a system, in a simulator requires a monotonic function of temperature for fixed pressure (as is known from basic physical principles).

In this section, we describe a real-world physics application which requires monotonicity in certain features—and robustness in all of them. The algorithm described here has, in fact, been implemented by a high-energy particle physics experiment at the European Center for Nuclear Research (CERN), and is actively being used to collect data at the Large Hadron Collider (LHC) in 2022, where high-energy proton-proton collisions occur at 40 MHz.

The sensor arrays of the LHC experiments produce data at a rate of over 100 TB/s. Drastic data-reduction is performed by custom-built read-out electronics; however, the annual data volumes are still $O(100)$ exabytes, which cannot be put into permanent storage. Therefore, each LHC experiment processes its data in real time, deciding which proton-proton collision events should be kept and which should be discarded permanently; this is referred to as *triggering* in particle physics. To be suitable for use in trigger systems, classification algorithms must be robust against the impact of experimental instabilities that occur during data taking—and deficiencies in simulated training samples. Our training samples cannot possibly account for the unknown new physics that we hope to learn by performing the experiments!

A ubiquitous inductive bias at the LHC is that outlier collision events are more interesting, since we are looking for physics that has never been observed before. However, uninteresting outliers are frequently caused by experimental imperfections, many of which are included and labeled as background in training. Conversely, it is not possible to include the set of all possible interesting outliers *a priori* in the training. A solution to this problem is to implement *outliers are better* directly using our expressive monotonic Lipschitz architecture from Sec. 3.

Our architecture was originally developed for the task of classifying the decays of heavy-flavor particles produced at the LHC. These are bound states containing a beauty or charm quark that travel an observable distance $\mathcal{O}(1\,\mathrm{cm})$ before decaying due to their (relatively) long lifetimes. This example uses a dataset of simulated proton-proton (pp) collisions in the LHCb detector. Charged particles recorded by LHCb are combined pairwise into decay-vertex (DV) candidates. The task concerns discriminating DV candidates corresponding to heavy-flavor decays from all other sources. Heavy-flavor DVs typically have substantial separation from the pp collision point, due to the relatively long heavy-flavor particle lifetimes, and large transverse momenta, p_{T}, of the component particles, due to the large heavy-flavor particle masses. The main sources of background DVs, described in Redacted (2021), mostly have small displacement and small p_{T}, though unfortunately they can also have extremely large values of both displacement and momentum.

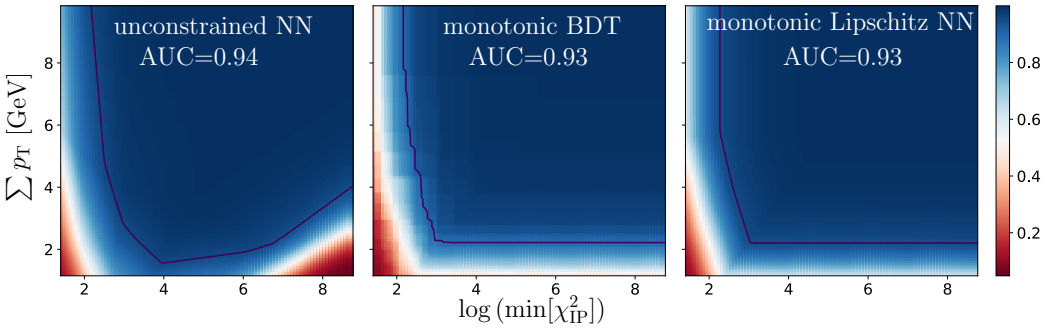


Figure 3: From Redacted (2021): Simplified version of the heavy-quark selection problem using only two inputs, which permits displaying the response everywhere in the feature space; shown here as a heat map with more signal-like (background-like) regions colored blue (red). The dark solid line shows the decision boundary (upper right regions are selected). Shown are (left) a standard fully connected neural network, (middle) a monotonic BDT, and (right) our architecture. The quantities shown on the horiztonal and vertical axes are related to how long the particle lived before decaying and how massive the particle was, respectively.

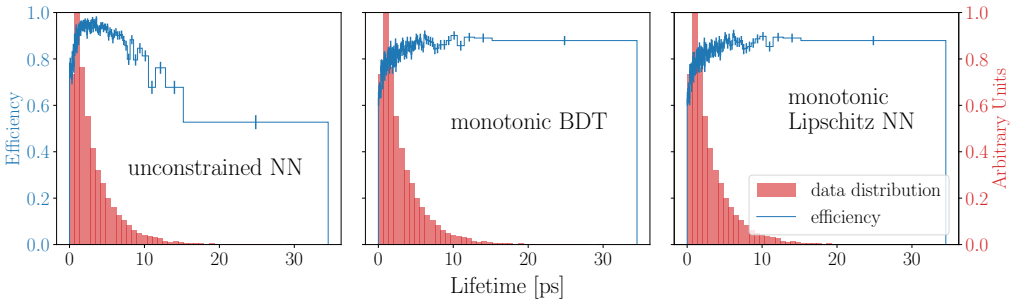


Figure 4: From Redacted (2021): True positive rate (efficiency) of each model shown in Fig. 3 versus the proper lifetime of the decaying heavy-quark particle selected. The monotonic models produce a nearly uniform efficiency above a few picoseconds at the expense of a few percent lifetime-integrated efficiency. Such a trade off is desirable as explained in the text.

Figure 3 shows a simplified version of this problem using only the two most-powerful inputs. Our inductive bias requires a monotonic increasing response in both features (detailed discussion motivating this bias can be found in Redacted (2021)). We see that an unconstrained neural network rejects DVs with increasing larger displacements (lower right corner), and that this leads to a decrease of the signal efficiency (true positive rate) for large lifetimes. The unconstrained model violates our inductive bias. Figures 3 and 4 show that a monotonic BDT (Auguste et al., 2020) approach works here. However, the jagged decision boundary can cause problems in subsequent analysis of the data. Figure 3 also shows that our novel approach from Sec. 3 successfully produces a smooth and monotonic response, and Fig. 4 shows that this provides the monotonic lifetime dependence we desire in the efficiency.

In addition, we note that the added benefit of guaranteed Lipschitz robustness is a major advantage for many real world applications. Specifically for particle physicists, this kind of robustness directly translates to important guarantees when considering experimental instabilities.

Due to the simplicity and practicality of our method, the LHCb experiment is now using the proposed architecture for real-time data selection at a data rate of about 40 Tbit/s.

4.3 PUBLIC DATASETS WITH MONOTONIC DEPENDENCE

In this section, we follow as closely as possible the experiments done in Liu et al. (2020), to be able to directly compare to state-of-the-art monotonic architectures. Liu et al. (2020) studied monotonic architectures on four different datasets: COMPAS (Larson & Kirchner, 2016), BlogFeedback (Buza, 2014), LoanDefaulter (Kaggle, 2015), and ChestXRay (Wang et al., 2017).

COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) refers to a commercial algorithm used by judges and police officers to determine the likelihood of reoffense. Larson & Kirchner (2016) discusses that the algorithm is racially biased and provides a dataset from a two-year study of the real-world performance of COMPAS. The task here is to predict the reoffense probability within the next two years. The dataset has 13 features, 4 of which have a monotonic inductive bias, and contains a total of 6172 data points.

BlogFeedBack This dataset contains 54270 data points with 276 dimensions describing blog posts. The task is to predict the number of comments following the post publication within 24 hours. 8 of the features have a monotonic inductive bias. Just like Liu et al. (2020), we also only consider the 90% of the data points with the smallest targets so as to not let the RMSE be dominated by outliers.

LoanDefaulter The version of this dataset available on Kaggle was updated on a yearly basis up to 2015. Kaggle (2015) contains a link that is, we believe, a superset of the data used in Liu et al. (2020). Luckily, the authors have shared with us the exact version of the dataset they used in their studies for an appropriate comparison. The data is organized in 28 features and the task is to determine loan defaulters. The classification score should be monotonic in 5 features: non-decreasing in number of public record bankruptcies and Debt-to-Income ratio, non-increasing in credit score, length of employment and annual income.

ChestXRay This dataset contains tabular data and images of patients with diseases that are visible in a chest x-ray. The task is to predict whether or not the patient has such a disease. Just like Liu et al. (2020), we send the image through an ImageNet-pretrained ResNet18 He et al. (2016). The penultimate layer output concatenated with tabular data acts as input to the monotonic architecture. Two of the four tabular features are monotonic. In the bottom right table in **??**, there are two entries for our architecture. The *E-E* entry refers to end-to-end training with ResNet18, whereas the other experiment fixes the ResNet weights.

Liu et al. (2020) compare their state-of-the-art certifiably monotonic architecture on those four datasets with existing methods that can enforce monotonicity. They present those results in Tables 1-4 of their paper. We show those tables also here, see Table **??**, extended with results produced by employing our architecture.

As can be seen in Table **??**, our Lipschitz monotonic networks outperform the state-of-the-art in all four problems.

It is also immediately apparent that our architecture is highly expressive. We manage to train tiny networks with few parameters while still achieving competitive performance. Given that some of these datasets have a significant number of features compared to our chosen network width, most parameters are in the weights of the first layer. We manage to build and train even smaller networks when taking only a few important features and ignoring the rest. These networks are denoted with **mini** in Table **??**.

5 LIMITATIONS

We are working on improving the architecture as follows: First, common initialization techniques are not optimal for weight-normed networks (Arpit et al., 2019). Simple modifications to the weight initialization might aid convergence, especially for large Lipschitz parameters. Secondly, we are currently constrained to activation functions that have a gradient norm of 1 over their entire domain, such as **GroupSort**, to ensure universal approximation, see Anil et al. (2019). We will explore other options in the future. Lastly, there is not yet a proof for universal approximation for the

COMPAS

Method	Parameters	Test Acc
Isotonic	N.A.	67.6%
XGBoost	N.A.	$(68.5 \pm 0.1)\%$
Crystal	25840	$(66.3 \pm 0.1)\%$
DLN	31403	$(67.9 \pm 0.3)\%$
Min-Max Net	42000	$(67.8 \pm 0.1)\%$
Non-Neg-DNN	23112	$(67.3 \pm 0.9)\%$
Certified	23112	$(68.8 \pm 0.2)\%$
LMN	**37**	$(\mathbf{69.3 \pm 0.1})\%$

BlogFeedback

Method	Parameters	RMSE
Isotonic	N.A.	$.203$
XGBoost	N.A.	$.176 \pm .005$
Crystal	15840	$.164 \pm .002$
DLN	27903	$.161 \pm .001$
Min-Max Net	27700	$.163 \pm .001$
Non-Neg-DNN	8492	$.168 \pm .001$
Certified	8492	$.158 \pm .001$
LMN	**2225**	$\mathbf{.160 \pm .001}$
LMN mini	**177**	$\mathbf{.155 \pm .001}$

LoanDefaulter

Method	Parameters	Test Acc
Isotonic	N.A.	62.1%
XGBoost	N.A.	$(63.7 \pm 0.1)\%$
Crystal	16940	$(65.0 \pm 0.1)\%$
DLN	29949	$(65.1 \pm 0.2)\%$
Min-Max Net	29000	$(64.9 \pm 0.1)\%$
Non-Neg-DNN	8502	$(65.1 \pm 0.1)\%$
Certified	8502	$(65.2 \pm 0.1)\%$
LMN	**753**	$(\mathbf{65.44 \pm 0.03})\%$
LMN mini	**69**	$(\mathbf{65.28 \pm 0.01})\%$

ChestXRay

Method	Parameters	Test Acc
XGBoost	N.A.	$(64.4 \pm 0.4)\%$
Crystal	26540	$(65.3 \pm 0.1)\%$
DLN	39949	$(65.4 \pm 0.1)\%$
Min-Max Net	35130	$(64.3 \pm 0.6)\%$
Non-Neg-DNN	12792	$(64.7 \pm 1.6)\%$
Certified	12792	$(62.3 \pm 0.2)\%$
Certified E-E	12792	$(66.3 \pm 1.0)\%$
LMN	**1043**	$(\mathbf{67.6 \pm 0.6})\%$
LMN E-E	**1043**	$(\mathbf{70.0 \pm 1.4})\%$

Table 1: We compare our method (in bold) against state-of-the-art monotonic models on a variety of benchmarks. The performance numbers for other techniques were taken from Liu et al. (2020). In the ChestXRay experiment, we train one model with frozen ResNet18 weights (second to last) and another with end-to-end training (last). While our models can generally get quite small, we can achieve even smaller models when only taking a subset of all the features. These models are denoted with "mini."

architecture described in Eq. 8. However, it appears from empirical investigation that the networks do approximate universally, as we have yet to find a function that could not be approximated well enough with a deep enough network. We do not consider this a major drawback, as the construction in Eq. 11 does approximate universally, see Anil et al. (2019). Note that neither of these limitations has any visible impact on the performance of the experiments in Section 4.

6 Conclusion and Future Work

We presented an architecture that provably approximates Lipschitz continuous and partially monotonic functions. Monotonic dependence is enforced via an end-to-end residual connection to a minimally Lip^1 constrained fully connected neural network. This method is simple to implement, has negligible computational overhead, and gives stronger guarantees than regularized models. Our architecture achieves competitive results with respect to current state-of-the-art monotonic architectures, even when using a tiny number of parameters, and has the additional benefit of guaranteed robustness due to its known Lipschitz constant. For future directions of this line of research, we plan to tackle the problems outlined in the limitation section, especially improving initialization of weight-normed networks.

7 Reproducibility Statement

All experiments with public datasets are reproducible with the code provided at `https://github.com/redacted`. The experiments in Section 4.2 were made with data that is not publicly available. For people with access to the data, the code to reproduce those experiments can be found under `https://github.com/redacted`. Those data will be made available in later years at the discretion of the LHCb collaboration.

ACKNOWLEDGMENTS

Redacted.

REFERENCES

Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. Advances in adversarial attacks and defenses in computer vision: A survey. *IEEE Access*, 9:155161–155196, 2021.

Cem Anil, James Lucas, and Roger Grosse. Sorting out Lipschitz function approximation. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 291–301. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/anil19a.html.

Devansh Arpit, Victor Campos, and Yoshua Bengio. How to initialize your network? robust initialization for weightnorm & resnets, 2019.

Charles Auguste, Sean Malory, and Ivan Smirnov. A better method to enforce monotonic constraints in regression and classification trees, 2020.

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks, 2019.

Krisztian Buza. Feedback prediction for blogs. In *Data analysis, machine learning and knowledge discovery*, pp. 145–152. Springer, 2014.

Nicola De Cao, Wilker Aziz, and Ivan Titov. Block neural autoregressive flow. In *Uncertainty in artificial intelligence*, pp. 1263–1273. PMLR, 2020.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. Regularisation of neural networks by enforcing lipschitz continuity, 2020.

Akhil Gupta, Naman Shukla, Lavanya Marla, Arinbjörn Kolbeinsson, and Kartik Yellepeddi. How to incorporate monotonicity in deep networks while preserving flexibility? URL https://arxiv.org/abs/1909.10662.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In *International Conference on Machine Learning*, pp. 2078–2087. PMLR, 2018.

Todd Huster, Cho-Yu Jason Chiang, and Ritu Chadha. Limitations of the lipschitz constant as a defense against adversarial examples, 2018.

Kaggle. Lending club loan data. 2015. URL https://www.kaggle.com/datasets/wordsforthewise/lending-club.

Donald Knuth. Donald Knuth's homepage. https://www-cs-faculty.stanford.edu/~knuth/faq.html. Accessed: 2022-09-28.

SM Angwin J Larson and Lauren Kirchner. There's software used across the country to predict future criminals. and it's biased against blacks, 2016.

Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. Certified monotonic neural networks, 2020. URL https://arxiv.org/abs/2011.10219.

Mahdi Milani Fard, Kevin Canini, Andrew Cotter, Jan Pfeifer, and Maya Gupta. Fast and flexible monotonic functions with ensembles of lattices. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/c913303f392ffc643f7240b180602652-Paper.pdf.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. *arXiv e-prints*, art. arXiv:1802.05957, February 2018.

A. Nguyen and M. Martínez. Mononet: Towards interpretable models by learning monotonic features. 9 2019. doi: 10.48550/arxiv.1909.13611. URL https://arxiv.org/abs/1909.13611v1.

Redacted. Redacted. *Redacted*, 2021.

Joseph Sill. Monotonic networks. In M. Jordan, M. Kearns, and S. Solla (eds.), *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998. URL https://proceedings.neurips.cc/paper/1997/file/83adc9225e4deb67d7ce42d58fe5157c-Paper.pdf.

Joseph Sill and Yaser Abu-Mostafa. Monotonicity hints. In M.C. Mozer, M. Jordan, and T. Petsche (eds.), *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1996. URL https://proceedings.neurips.cc/paper/1996/file/c850371fda6892fbfd1c5a5b457e5777-Paper.pdf.

Sahil Singla, Surbhi Singla, and Soheil Feizi. Improved deterministic l2 robustness on cifar-10 and cifar-100. In *International Conference on Learning Representations*, 2021.

Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, M Bagheri, and R Summers. Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *IEEE CVPR*, volume 7, 2017. URL https://www.kaggle.com/datasets/nih-chest-xrays/sample.

Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. *Advances in neural information processing systems*, 32, 2019.

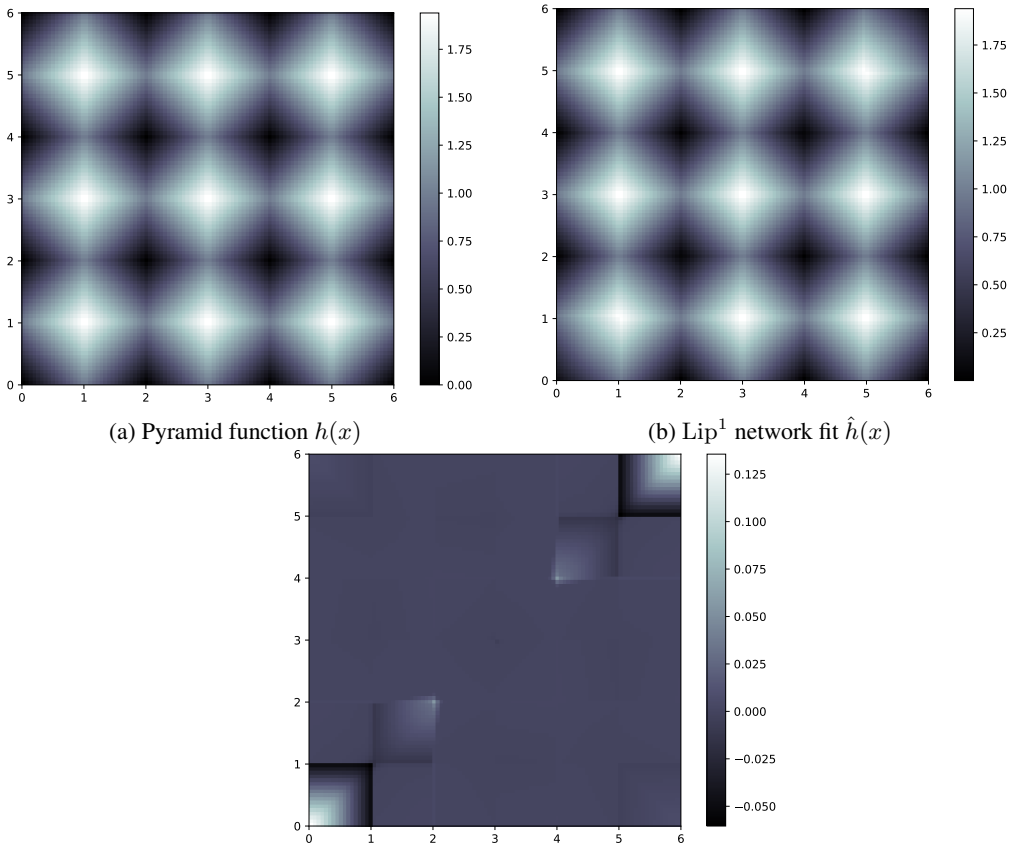
Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta. Deep lattice networks and partial monotonic functions, 2017.

A EXAMPLE FIT OF A FUNCTION AT THE LIPSCHITZ BOUNDARY

We showcase the expressiveness of our implementation from Section 3.2 in practice by fitting a function $h(\boldsymbol{x})$ with maximum gradient norm almost everywhere:

$$h(\boldsymbol{x}) = \sum_i (-1)^{\lfloor x_i \rfloor}(x_i - \lfloor x_i \rfloor) + (\lfloor x_i \rfloor \bmod 2) \tag{14}$$

Note that $\|\nabla h\|_1 = 1$ for all but a measure zero set of the domain. The absolute value function (restricted to $[-1, 1]$) is a special case of the function described above. Lipschitz-constrained ReLU networks are unable to fit the absolute value function let alone the more general case (Huster et al., 2018). Only a GroupSort Lip^p network for $p = 1$ is able to fit this function. See empirical results in Figure 5. The trained network is 6 layers deep with a width of 512 and a total of 1,052,673 parameters. This shows that the operator norm constraint as implemented in Section 3.2 does not restrict expressiveness in practice, even for more moderately-sized networks.



(a) Pyramid function $h(x)$

(b) Lip^1 network fit $\hat{h}(x)$

(c) Error $\hat{h}(x) - h(x)$

Figure 5: Pyramid-function in 2D (a) and a fit to it (b) with the architecture from Section 3.2. The difference $\hat{h}(x) - h(x)$ is shown on the bottom (c). This function maximizes the gradient norm within the constraint. It is therefore a good candidate to demonstrate expressiveness of Lip$^{p=1}$ approximator implementations.

B EXAMPLE FIT OF COMPLEX DECISION BOUNDARY

Figure 6 shows an example where this complicated decision boundary is learned by a Lipschitz network (as defined in Sec. 3.1) trained on the boundary while achieving zero loss, demonstrating the expressiveness that is possible to obtain in these models.

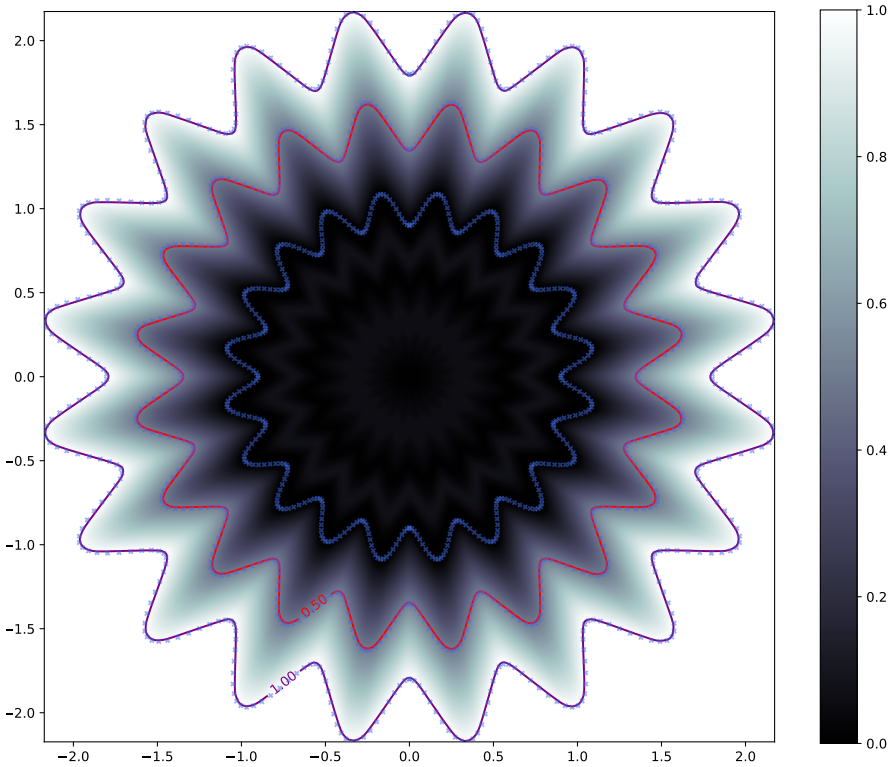


Figure 6: Regression example to emulate a complex decision boundary in two dimensional classification. The training data points are shown in blue (the inner radius is labeled 0, the middle is labeled 0.5, and the outer radius is labeled 1), while the output of the network is shown in color. The contour lines of the network output are shown in purple and red for the values of 1.0 and 0.5, respectively, which properly trace out the curves populated by the outer and middle sets of data points.