Constrained Monotonic Neural Networks

Davor Runje [1 2] **Sharath M Shankaranarayana** [1]

Abstract

Wider adoption of neural networks in many critical domains such as finance and healthcare is being hindered by the need to explain their predictions and to impose additional constraints on them. Monotonicity constraint is one of the most requested properties in real-world scenarios and is the focus of this paper. One of the oldest ways to construct a monotonic fully connected neural network is to constrain signs on its weights. Unfortunately, this construction does not work with popular non-saturated activation functions as it can only approximate convex functions. We show this shortcoming can be fixed by constructing two additional activation functions from a typical unsaturated monotonic activation function and employing each of them on the part of neurons. Our experiments show this approach of building monotonic neural networks has better accuracy when compared to other state-of-the-art methods, while being the simplest one in the sense of having the least number of parameters, and not requiring any modifications to the learning procedure or post-learning steps. Finally, we prove it can approximate any continuous monotone function on a compact subset of $\mathbb{R}^n$.

1. Introduction

Deep Learning has witnessed widespread adoption in many critical real-world domains such as finance, healthcare, etc (LeCun et al., 2015). Incorporating prior knowledge such as monotonicity in trained models helps in improving the performance and generalization ability of the trained models (Mitchell, 1980; Dugas et al., 2000). The introduction of structural biases such as monotonicity makes models also more data-efficient, enabling a leap in predictive power on smaller datasets (Veličković, 2019). Apart from the requirements of having models with high accuracy, there is also a need for transparency and interpretability, and monotonicity helps in partially achieving the above requirements (Gupta et al., 2016). Due to legal, ethical and/or safety concerns, monotonicity of predictive models with respect to some input or all the inputs is required in numerous domains such as financial (house pricing, credit scoring, insurance risk), healthcare (medical diagnosis, patient medication) and legal (criminal sentencing) to list just a few. For example, when using machine learning to predict admission decisions, it may seem unfair to select student X over student Y, if Y has a higher score than X, while all other aspects of the two are identical (Liu et al., 2020). In another example, one would expect an individual with a higher salary to have a higher loan amount approved, all else being equal (Sivaraman et al., 2020). A model without such a monotonic property would not, and certainly should not, be trusted by society to provide a basis for such important decisions.

Monotonicity has been an active area of research and the existing methods on the subject can be broadly categorized into two types:

1. Monotonic architectures by construction: neural architectures guaranteeing monotonicity by construction (Archer & Wang, 1993; Sill, 1997; Daniels & Velikova, 2010; Milani Fard et al., 2016; You et al., 2017).
2. Monotonicity by regularization: enforcing monotonicity in neural networks during training by employing a modified loss function or a heuristic regularization term (Sill & Abu-Mostafa, 1996; Gupta et al., 2019).

The simplest method to achieve monotonicity by construction is to constrain the weights of the fully connected neural network to have only non-negative (for non-decreasing variables) or only non-positive values (for non-ascending) variables when used in conjunction with a monotonic activation function, a technique known for 30 years (Archer & Wang, 1993). When used in conjunction with saturated (bounded) activation functions such as the sigmoid and hyperbolic tangent, these models are difficult to train, i.e. they do not converge to a good solution. On the other hand, when used with non-saturated (unbounded) convex activation functions such as ReLU (Nair & Hinton, 2010), the resulting models

[1]Airt Research, Zagreb, Croatia [2]Algebra University College, Zagreb, Croatia. Correspondence to: Davor Runje <davor@airt.ai>, Sharath M Shankaranarayana <sharathms@alumni.iitm.ac.in>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

are always convex (Liu et al., 2020), severely limiting the applicability of the method in practice.

Our main contribution is a modification of the method above which, in conjunction with non-saturated activation functions, is capable of approximating non-convex functions as well: when the original activation function is used with additional two monotonic activation functions constructed from it in a neural network with constrained weights, it can approximate any monotone continuous functions.

The resulting model is guaranteed to be monotonic, can be used in conjunction with popular convex monotonic non-saturated activation function, doesn't have any additional parameters compared to a non-monotonic fully-connected network for the same task, and can be trained without any additional requirements on the learning procedure. Experimental results show it is exceeding the performance of all other state-of-the-art methods, all while being both simpler (in the number of parameters) and easier to train.

Our contributions can be summarized as follows:

1. A modification to an existing constrained neural network layer enabling it to model arbitrary monotonic function when used with non-saturated monotone convex activation functions such as ReLU, ELU, SELU, and alike.
2. Experimental comparisons with other recent works showing that the proposed architecture can yield equal or better results than the previous state-of-the-art and with significantly fewer parameters.
3. A proof showing that the proposed architecture can approximate any monotone continuous function on a compact subset of $\mathbb{R}^n$ for a large class of non-saturated activation functions.

2. Related work

2.1. Activation functions

Right from its inception in perceptron (Rosenblatt, 1958), non-linear activation functions have historically been one of the most important components of neural networks. Previously, the saturated functions such as the sigmoid (Rumelhart et al., 1986), the hyperbolic tangent (Neal, 1992), and its variants were the most common choice of activation functions. Currently, one of the most important factors for state-of-the-art results accomplished by modern neural networks is the use of non-saturated activation functions. The use of *Rectified Linear Unit* (ReLU) (Nair & Hinton, 2010; Glorot et al., 2011) as activation function was instrumental in achieving good performance in newer architectures. The ReLU has since become a de facto choice of activation in most practical implementations and continues to be widely used because of its advantages such as simple computation, representational sparsity, and linearity. Later, a number of activation functions were proposed to deal with solving problems of dead neurons and aid in faster convergence (Maas et al., 2013), (Clevert et al., 2016) (He et al., 2015), (Zheng et al., 2015), (Hendrycks & Gimpel, 2016), (Ramachandran et al., 2017), (Klambauer et al., 2017).

The idea of using both the original activation function and its point reflection in the same layer has been proposed in (Shang et al., 2016) where both outputs of ReLU and the negative value of its point reflection were used in the construction of *concatenated ReLU* (CReLU) activation function. The proposed modification outputs two values instead of one and therefore increases the number of parameters. In (Zagoruyko & Komodakis, 2017), the authors propose *negative concatenated ReLU* (NCReLU) flip the sign and use the point reflection directly. In (Eidnes & Nøkland, 2018), the authors propose *bipolar ReLU* which consists of using ReLU on the half of the neurons in the layer and the point reflection of ReLU on the other half.

2.2. Monotonicity by construction

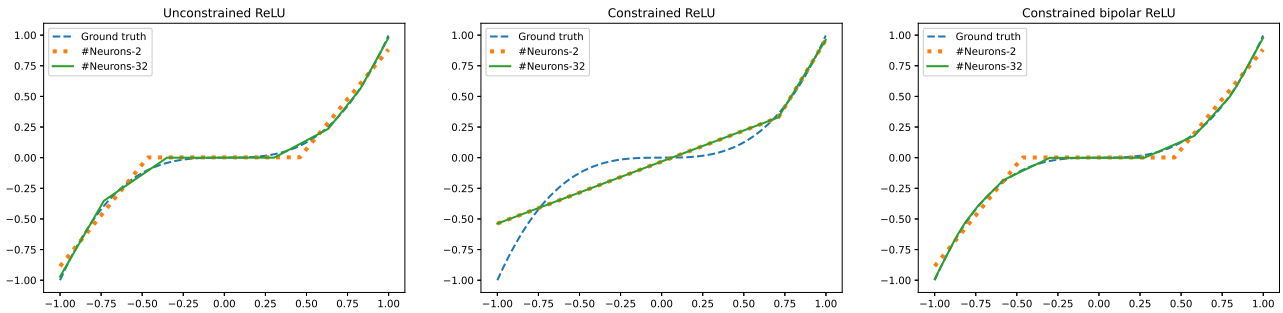
Apart from the approaches mentioned in the introduction, another approach to building monotonic neural architecture is Min-Max networks where monotonic linear embedding and max-min-pooling are used (Sill, 1997). In (Daniels & Velikova, 2010), authors generalized this approach to handle functions that are partially monotonic and proved that the resulting networks have the universal approximation property. However, such networks are very difficult to train and not used in practice.

Deep lattice networks (DLN) (You et al., 2017) use a combination of linear calibrators and lattices (Milani Fard et al., 2016) for learning monotonic functions. This is the most widely used method in practice today, but not without its limits. Lattices are structurally rigid thereby restricting the hypothesis space significantly. They also require a very large number of parameters to obtain good performance.

Given a model with a convex output function, it is possible to use backpropagation (Rumelhart et al., 1986) to make a monotonic model by computing the derivation of the output function. One simple way to construct a convex function is to use an unsaturated monotonic activation function in a fully connected layer as mentioned above or a more elaborate architecture such as the input convex neural networks (Amos et al., 2017). These constructions are computationally more complex than the simple solution proposed here.

2.3. Monotonicity by regularization

Monotonicity can be enforced during the training by modifying the loss function or adding a regularization term.



(a) Unconstrained ReLU (b) Constrained ReLU (c) Constrained ReLU based activations

Figure 1: Approximations of the cubic function $f(x) = x^3$

In (Sill & Abu-Mostafa, 1996), the authors propose a modified loss function that penalizes the non-monotonicity of the model. The algorithm models the input distribution as a joint Gaussian estimated from the training data and samples random pairs of monotonic points that are added to the training data. In (Gupta et al., 2019), the authors propose a point-wise loss function that acts as a soft monotonicity constraint. These methods are straightforward to implement and can be used with any neural network architecture, but they do not guarantee the monotonicity of the trained model.

Recently, there is an increasing number of proposed methods to certify or verify monotonicity obtained by regularization methods. In (Liu et al., 2020), the authors propose an optimization-based technique for mathematically verifying, or rejecting, the monotonicity of an arbitrary piece-wise linear (e.g., ReLU) neural network. The method consists of transforming the monotonicity verification problem into a mixed integer linear programming (MILP) problem that can be solved using an off-the-shelf MILP solver.

In (Sivaraman et al., 2020), the authors propose an approach that finds counterexamples (defined as the pair of points where the monotonicity constraint is violated) by employing satisfiability modulo theories (SMT) solver (Barrett & Tinelli, 2018). To satisfy the monotonicity constraints, these counterexamples are included in the training data with adjustments to their target values to enforce the next iterations of the model to be monotonic.

Both methods (Liu et al., 2020; Sivaraman et al., 2020) have been shown to support ReLU as the activation function only and there is no obvious way how to extend them to other activation functions. More precisely, they rely on piecewise linearity of ReLU to work, the property not satisfied by other variants such as ELU, SELU, GELU, etc. Last but not least, the procedure for certifying/verifying using MILP or SMT solvers is computationally very costly. These approaches also require multiple reruns or iterations to arrive at certified/verified monotonic networks.

3. Constrained neural networks

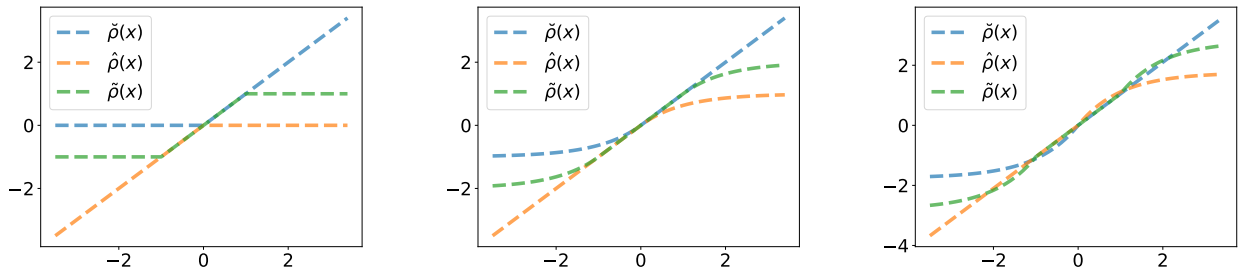
Most of the commonly used activation functions such as ReLU, ELU, SELU, etc. are monotonically increasing zero-centred, convex, lower-bounded non-polynomial functions. When used in a fully-connected, feed-forward neural network with at least one hidden layer and with unconstrained weights, they can approximate any continuous function on a compact subset. The simplest way to construct a monotonic neural network is to constrain its weights when used in conjunction with a monotone activation function. However, when the activation function is convex as well, the constrained neural network is not able to approximate non-convex functions.

To better illustrate this, and to propose a simple solution in this particular example, we refer the readers to Figure 1 where the goal is to approximate a simple cubic function x^3 using a neural network with a single hidden layer with either 2 or 32 neurons and with ReLU activation. A cubic function is apt for our illustration since it is concave in the considered interval $[-1, 0]$ and convex in the interval $[0, 1]$:

1a. An unconstrained ReLU network with n neurons can approximate both concave and convex segments of the cubic function using at most $n + 1$ piecewise linear segments. Increasing the number of neurons will provide a better fit with the function being approximated. Notice that even though the cubic function is monotone, there is no guarantee that the trained model will be monotone as well.

1b. If we constrain the weights of the network to be non-negative while still employing ReLU activation, the resulting model is monotone and convex. We can no longer approximate non-convex segments such as the cubic function on $[-1, 0]$ in the figure, and increasing the number of neurons from 2 to 32 does not yield any significant improvement in the approximation.

1c. Our proposed solution uses a combination of three ac-



(a) ReLU based activations (b) ELU based activations (c) SELU based activations

Figure 2: Activation functions construction

tivation functions in the hidden layer in order to gain the ability to model non-convex, monotone continuous functions. Notice that increasing the number of neurons increases the number of piecewise linear segments to approximate the cubic function. The resulting network is monotone by construction even when trained on noisy data.

The schematic block diagram of our proposed solution (which we refer to as Constrained Monotone Fully Connected Layer or Monotonic Dense Unit interchangeably) is shown in the figure Fig. 3. The individual components of the proposed solution are defined and described in the subsequent subsection.

3.1. Constrained monotone fully connected layer

We say that a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$ is *partially monotonically increasing with respect to* x_i if

$$x_i^0 > x_i^1 \Rightarrow f\left(x_1, \ldots, x_i^0, \ldots x_n\right) \geq f\left(x_1, \ldots, x_i^1, \ldots x_n\right).$$

Similarly, f is *partially monotonically decreasing with respect to* x_i if

$$x_i^0 > x_i^1 \Rightarrow f\left(x_1, \ldots, x_i^0, \ldots x_n\right) \leq f\left(x_1, \ldots, x_i^1, \ldots x_n\right).$$

A set $S \subseteq R$ is *compact* if every sequence in S has a subsequence that converges to a point in S. One can easily show that closed intervals $[a, b]$ are compact, and compact sets can be thought of as generalizations of such closed bounded intervals.

Our construction is preconditioned on a priori knowledge of (partial) monotonicity of a multivariate, multidimensional function f. Let $f : K \mapsto \mathbb{R}^m$ be defined on a compact segment $K \subseteq \mathbb{R}^n$. Then we define its n-dimensional *monotonicity indicator vector* $\mathbf{t} = [t_1, \ldots, t_n]$ element-wise as follows:

$$t_j = \begin{cases} 1 & \text{if } \dfrac{\partial f(\mathbf{x})_i}{\partial x_j} \geq 0 \text{ for each } i \in \{1, \ldots, m\} \\ -1 & \text{if } \dfrac{\partial f(\mathbf{x})_i}{\partial x_j} \leq 0 \text{ for each } i \in \{1, \ldots, m\} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Given an $(m \times n)$-dimensional matrix $\mathbf{M}$ and n-dimensional monotonicity indicator vector $\mathbf{t}$, we define the operation $|.|_t$ assigning an $(m \times n)$-dimensional matrix $\mathbf{M}' = |\mathbf{M}|_t$ to $\mathbf{M}$ element-wise as follows:

$$m'_{j,i} = \begin{cases} |m_{j,i}| & \text{if } t_i = 1 \\ -|m_{j,i}| & \text{if } t_i = -1 \\ m_{j,i} & \text{otherwise} \end{cases} \tag{2}$$

Definition 1 (Constrained linear layer)**.** *Let* $\mathbf{W} \in \mathbb{R}^{n \times m}$, $\mathbf{t} \in \{-1, 0, 1\}^n$, $\mathbf{x} \in \mathbb{R}^n$ *and* $\mathbf{b} \in \mathbb{R}^m$. *The output* $\mathbf{h} \in \mathbb{R}^m$ *of the* constrained linear layer *with monotonicity indicator vector* $\mathbf{t}$, *weights* $\mathbf{W}$, *biases* $\mathbf{b}$ *and input* $\mathbf{x}$ *is:*

$$\mathbf{h} = |\mathbf{W}^{\mathbf{T}}|_{\mathbf{t}} \cdot \mathbf{x} + \mathbf{b} \tag{3}$$

Lemma 1. *For each* $i \in \{1, \ldots, n\}$ *and* $j \in \{1, \ldots, m\}$ *we have:*

- *if* $t_i = 1$, *then* $\dfrac{\partial h_j}{\partial x_i} \geq 0$, *and*
- *if* $t_i = -1$, *then* $\dfrac{\partial h_j}{\partial x_i} \leq 0$.

We use $\breve{\mathcal{A}}$ to denote the set of all zero-centred, monotonically increasing, convex, lower-bounded functions.

Definition 2. *Let* $\breve{\rho} \in \breve{\mathcal{A}}$. *Then*

$$\hat{\rho}(x) = -\breve{\rho}(-x) \tag{4}$$

$$\tilde{\rho}(x) = \begin{cases} \breve{\rho}(x+1) - \breve{\rho}(1) & \textit{if } x < 0 \\ \hat{\rho}(x-1) + \breve{\rho}(1) & \textit{otherwise} \end{cases} \tag{5}$$

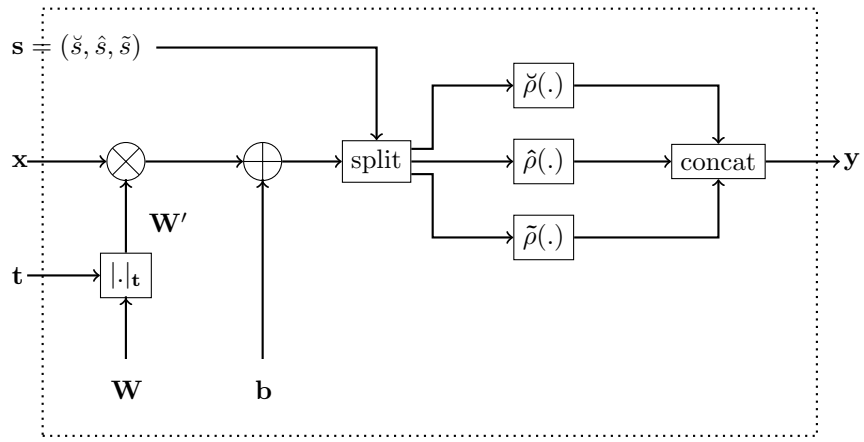


Figure 3: Proposed Monotonic Dense Unit or Constrained Monotone Fully Connected Layer

The main idea of our construction is use of three zero-centred, monotonically increasing activation functions, each applied to a part of neurons in a layer:

- the original activation function $\breve{\rho} \in \mathcal{A}$,
- *concave upper-bounded* function $\breve{\rho}$, and
- *bounded function* $\tilde{\rho}$.

Plots of such constructed activation functions for popular activation functions are given in Figure 2. Using only convex and concave activation functions is sufficient for approximating many functions such as the cubic function in Figure 1, but not for e.g. the sigmoid function. As we will show below, the saturated activation function is crucial for the universal property of our construction.

Definition 3 (Combined activation function)**.** *Let* $\breve{\rho} \in \breve{\mathcal{A}}$, $\mathbf{h} \in \mathbb{R}^m$ *and* $\mathbf{s} = (\breve{s}, \hat{s}, \tilde{s}) \in \mathbb{N}^3$ *such that* $\breve{s} + \hat{s} + \tilde{s} = m$. *Then the output of the combined activation function* $\rho^{\mathbf{s}} : \mathbb{R}^m \to \mathbb{R}^m$ *is defined element-wise as follows:*

$$\rho^{\mathbf{s}}(\mathbf{h})_j = \begin{cases} \breve{\rho}(h_j) & \text{if } j \leq \breve{s} \\ \hat{\rho}(h_j) & \text{if } \breve{s} < j \leq \breve{s} + \hat{s} \\ \tilde{\rho}(h_j) & \text{otherwise} \end{cases} \tag{6}$$

Lemma 2. *Let* $\mathbf{y} = \rho^{\mathbf{s}}(\mathbf{h})$. *Then for each* $j \in \{1, \ldots, m\}$ *we have* $\frac{\partial y_j}{\partial h_j} \geq 0$. *Moreover*

- *if* $\mathbf{s} = (m, 0, 0)$, *then* $\rho^{\mathbf{s}}_j$ *is convex; and*
- *if* $\mathbf{s} = (0, m, 0)$, *then* $\rho^{\mathbf{s}}_j$ *is concave.*

Definition 4 (Monotone constrained fully connected layer)**.** *Let* $n, m \in \mathbb{N}$, $\breve{\rho} \in \breve{\mathcal{A}}$, $\mathbf{t} \in \{-1, 0, 1\}^n$, $\mathbf{s} = (\breve{s}, \hat{s}, \tilde{s}) \in \mathbb{N}^3$ *such that* $\breve{s} + \hat{s} + \tilde{s} = m$, $\mathbf{W} \in \mathbb{R}^{n \times m}$, $\mathbf{x} \in \mathbb{R}^n$ *and* $\mathbf{b} \in \mathbb{R}^m$. *Then the output* $\mathbf{y}$ *of the* monotone constrained fully connected layer *with monotonicity indicator vector* $\mathbf{t}$, *weights* $\mathbf{W}$, *biases* $\mathbf{b}$ *and input* $\mathbf{x}$ *is*

$$\mathbf{y} = \rho^{\mathbf{s}}\left(|\mathbf{W}^{\mathbf{T}}|_{\mathbf{t}} \cdot \mathbf{x} + \mathbf{b}\right) \tag{7}$$

From Lemma 1 and 2 directly follows:

Corollary 3. *For each* $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$ *we have:*

- *if* $t_i = 1$, *then* $\frac{\partial y_j}{\partial x_i} \geq 0$,
- *if* $t_i = -1$, *then* $\frac{\partial y_j}{\partial x_i} \leq 0$,
- *if* $\mathbf{s} = (m, 0, 0)$, *then* $\mathbf{y}_j$ *is convex; and*
- *if* $\mathbf{s} = (0, m, 0)$, *then* $\mathbf{y}_j$ *is concave.*

On the layer level, we can control both monotonicity, convexity and concavity of the output with respect to chosen input variables. The following section discuss how we can use such layers to build practical neural networks with the same properties.

3.2. Composing monotonic constrained dense layers

As mentioned before, the main advantage of our proposed monotonic dense unit is its simplicity. We can build deep neural nets with different architectures by plugging in our monotonic dense blocks. Figures 4 and 5 show two examples of neural architectures that can be built using the proposed monotonic dense block.

The first example shown in the Figure 4, corresponds to the standard MLP type of neural network architecture used in

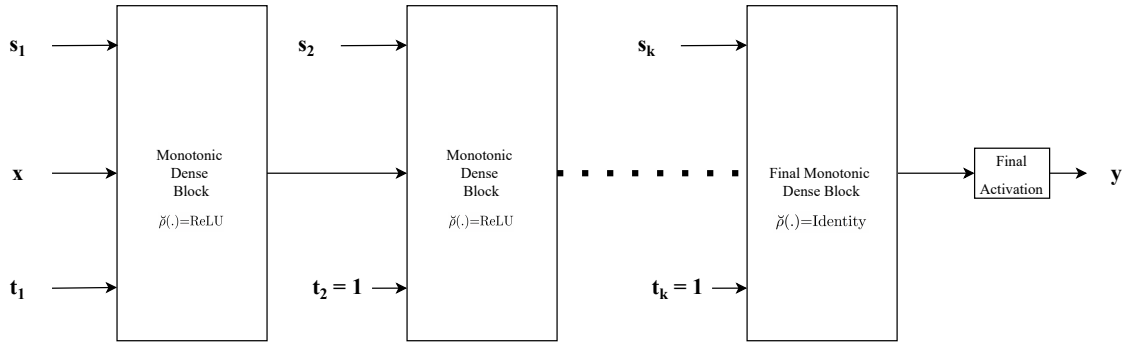


Figure 4: Neural architecture type 1

general, where each of the input features is concatenated to form one single input feature vector $\mathbf{x}$ and fed into the network, with the only difference being that instead of standard fully connected or dense layers, we employ monotonic dense units throughout. For the first (or input layer) layer, the indicator vector $\mathbf{t}$, is used to identify the monotonicity property of the input feature with respect to the output. Specifically, $\mathbf{t}$ is set to 1 for those components in the input feature vector that are monotonically increasing and is set to -1 for those components that are monotonically decreasing and set to 0 if the feature is non-monotonic. For the subsequent hidden layers, monotonic dense units with the indicator vector $\mathbf{t}$ always being set to 1 are used in order to preserve monotonicity. Finally, depending on whether the problem at hand is a regression problem or a classification problem (or even a multi-task problem), an appropriate activation function (such as linear activation or sigmoid or softmax) to obtain the final output.

Figure 5 shows another example of a neural network architecture that can be built employing proposed monotonic dense blocks. The difference when compared to the architecture described above lies in the way input features are fed into the hidden layers of neural network architecture. Instead of concatenating the features directly, this architecture provides flexibility to employ any form of complex feature extractors for the non-monotonic features and use the extracted feature vectors as inputs. Another difference is that each monotonic input is passed through separate monotonic dense units. This provides an advantage since depending on whether the input is completely concave or convex or both, we can adjust the activation selection vector $\mathbf{s}$ appropriately along with an appropriate value for the indicator vector $\mathbf{t}$. Thus, each of the monotonic input features has a separate monotonic dense layer associated with it. Thus as the major difference to the above-mentioned architecture, we concatenate the feature vectors instead of concatenating the inputs directly. The subsequent parts of the network are similar to the architecture described above wherein for the rest of the hidden monotonic dense units, the indicator vector $\mathbf{t}$ is always set to 1 to preserve monotonicity.

3.3. Universal approximation

The classical Universal Approximation Theorem (Cybenko, 1989; Hornik, 1991; Pinkus, 1999) states that any continuous function on a closed interval can be approximated with a feed-forward neural network with one hidden layer if and only if its activation function is nonpolynomial. In (Kidger & Lyons, 2020), authors prove the approximation property holds for arbitrary *deep* neural networks with bounded number of neurons in each layer holds if the activation function is nonaffine and differential at at least one point.

In (Daniels & Velikova, 2010), authors show the universal approximation property for constrained multivariate neural networks using *sigmoid* as the activation functions: any multivariate continuous monotone function on a compact subset of $\mathbb{R}^k$ can be approximated with a constrained neural network with the sigmoid activation function of at most k layers (Theorem 3.1), reproduced here for completeness:

Theorem 4. *For any continuous monotone nondecreasing function $f : K \to \mathbb{R}$, where K is a compact subset of $\mathbb{R}^k$, there exists a feedforward neural network using the sigmoid as the activation function with at most k hidden layers, positive weights, and output O such that $|O_x - f(x)| < \epsilon$, for any $x \in K$ and $\epsilon > 0$.*

However, the proof of the theorem uses only the fact that the Heavyside function $\mathbf{H}$ defined as

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

can be approximated with the sigmoid function on a closed interval (since $\lim_{a \to \infty} \sigma(ax) = H(x)$).

By construction, we have:

Lemma 5. *Let $\breve{\rho} \in \breve{\mathcal{A}}$. Then the Heavyside function can be approximated with $\tilde{\rho}_H$ on $\mathbb{R}$, where*

$$\tilde{\rho}_H(x) = \alpha \tilde{\rho}(x) + \beta$$

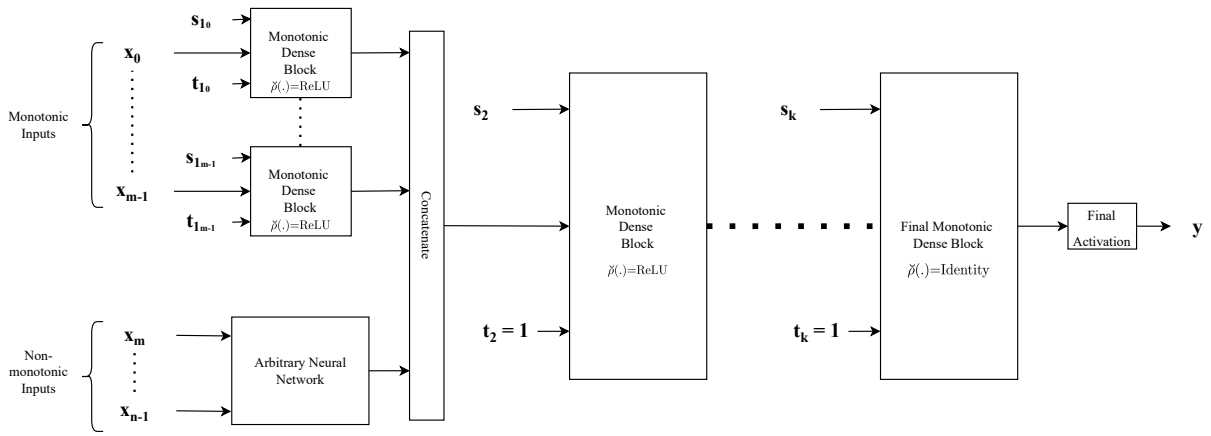


Figure 5: Neural architecture type 2

for some $\alpha, \beta \in \mathbb{R}$ *and* $\alpha > 0$.

Lemma 6. *Let* $\tilde{\rho}_{\alpha,\beta}$ *be an activation function for some* $\alpha, \beta \in \mathbb{R}, \alpha > 0$ *such that for every* $x \in \mathbb{R}$

$$\tilde{\rho}_{\alpha,\beta}(x) = \alpha\tilde{\rho}(x) + \beta.$$

Then for every constrained monotone neural network $\mathcal{N}_{\alpha,\beta}$ *using* $\tilde{\rho}_{\alpha,\beta}$ *as an activation function (*$s = (0, 0, \tilde{s})$*), there is a constrained monotone neural network* $\mathcal{N}$ *using* $\tilde{\rho}$ *as an activation function such that for every* $\mathbf{x} \in \mathbb{R}^n$*:*

$$\mathcal{N}(\mathbf{x}) = \mathcal{N}_{\alpha,\beta}(\mathbf{x}).$$

Finally, from Theorem 4, Lemma 5 and Lemma 6, we have the universal approximation property:

Theorem 7. *Let* $\breve{\rho} \in \breve{\mathcal{A}}$*. Then any multivariate continuous monotone function on a compact subset of* $\mathbb{R}^k$ *can be approximated with a monotone constrained neural network of at most* k *layers using* ρ *as the activation function.*

The Theorem 7 gives us the upper bound on the number of layers needed for approximating an arbitrary function. In practice, the number and width of layers for a given function and given dataset are found by hyperparameter search. The best results in our experiments were achieved by neural networks with a significantly smaller number of layers. The proof shows that the saturated activation functions are sufficient for the universal approximation property. However, experimental results show that the networks with predominately unsaturated activation functions are easier to train and achieve better results.

4. Experiments

In order to analyze the practical utility of the proposed method, we experiment with various datasets and compare them with the recent state-of-the-art. For the first set of experiments, we use the datasets employed by authors in (Liu et al., 2020) and use the exact train and test split for proper comparison. We perform experiments on 3 datasets: COMPAS (J. Angwin & Kirchner, 2016), which is a classification dataset with 13 features of which 4 are monotonic; Blog Feedback Regression (Buza, 2014), which is a regression dataset with 276 features of which 8 are monotonic; Loan Defaulter[1], which is a classification dataset with 28 features of which 5 are monotonic. The dataset contains half a million data points. For comparison with other methods, we compare with *Certified monotonic networks (Certified)* (Liu et al., 2020) and other methods described in it.

For the second set of experiments, we use 2 datasets: *Auto MPG* (which is a regression dataset with 3 monotonic features) and *Heart Disease* (which is a classification dataset with 2 monotonic features) as employed in the work (Sivaraman et al., 2020) and once again use the exact train and test split for proper comparison. We compare with the method *COMET* described in (Sivaraman et al., 2020) along with *Min-Max Net* (Daniels & Velikova, 2010) and *Deep Lattice Network (DLN)* (You et al., 2017) as described in (Sivaraman et al., 2020), and also more details regarding the datasets have been provided in the supplementary.

We use cross-entropy for the classification tasks and we use mean-squared-error for the regression tasks as loss functions. We employ Bayesian optimization tuning with Gaussian process (Snoek et al., 2012) to find the optimal hyperparameters such as the number of neurons, network depth or layers, initial learning rate etc.

The code for experiments was written in the Keras framework (Chollet et al., 2015) and KerasTuner (O'Malley et al., 2019) via integration from the Tensorflow framework, version 2.11 (Abadi et al., 2015). All experiments were performed using a Google Colaboratory instance with NVidia

[1]https://www.kaggle.com/wendykan/lending-club-loan-data

Method	COMPAS		Blog Feedback		Loan Defaulter	
	Parameters	Test Acc ↑	Parameters	RMSE ↓	Parameters	Test Acc ↑
Isotonic	N.A.	67.6%	N.A.	0.203	N.A.	62.1%
XGBoost (Chen & Guestrin, 2016)	N.A.	68.5% $\pm$ 0.1%	N.A.	0.176 $\pm$ 0.005	N.A.	63.7% $\pm$ 0.1%
Crystal (Milani Fard et al., 2016)	25840	66.3% $\pm$ 0.1%	15840	0.164 $\pm$ 0.002	16940	65.0% $\pm$ 0.1%
DLN (You et al., 2017)	31403	67.9% $\pm$ 0.3%	27903	0.161 $\pm$ 0.001	29949	65.1% $\pm$ 0.2%
Min-Max Net (Daniels & Velikova, 2010)	42000	67.8% $\pm$ 0.1%	27700	0.163 $\pm$ 0.001	29000	64.9% $\pm$ 0.1%
Non-Neg-DNN	23112	67.3% $\pm$ 0.9%	8492	0.168 $\pm$ 0.001	8502	65.1% $\pm$ 0.1%
Certified (Liu et al., 2020)	23112	68.8% $\pm$ 0.2%	8492	0.158 $\pm$ 0.001	8502	65.2% $\pm$ 0.1%
Ours	**2317**	**69.2% $\pm$ 0.2%**	**1101**	**0.156 $\pm$ 0.001**	**177**	**65.3% $\pm$ 0.01%**

Table 1: Comparison of our method with other methods described in (Liu et al., 2020)

Method	Auto MPG	Heart Disease
	MSE ↓	Test Acc ↑
Min-Max Net (Daniels & Velikova, 2010)	10.14 $\pm$ 1.54	0.75 $\pm$ 0.04
DLN (You et al., 2017)	13.34 $\pm$ 2.42	0.86 $\pm$ 0.02
COMET (Sivaraman et al., 2020)	8.81 $\pm$ 1.81	0.86 $\pm$ 0.03
Ours	**8.37 $\pm$ 0.08**	**0.89 $\pm$ 0.00**

Table 2: Comparison of our method with other methods described in (Sivaraman et al., 2020)

Tesla T4 GPU (Bisong, 2019). The code is publicly available at (Runje & Shankaranarayana, 2023a), while the preprocessed datasets for experiments are available at (Runje & Shankaranarayana, 2023b).

4.1. Results

The results for the datasets above are summarized in Tables 1 and 2. It shows that our method outperforms other methods in terms of test accuracy for classification tasks and mean squared error and root mean squared error (MSE and RMSE) for regression tasks. For each of the datasets, we run the experiments ten times after finding the optimal hyperparameters and report the mean and standard deviation of the best five results. Experiment results show that networks learned by our method can achieve better results with fewer parameters, than the best-known algorithms for monotonic neural networks, such as Min-Max Network (Daniels & Velikova, 2010) and Deep Lattice Network (You et al., 2017). It should be noted that the recent state-of-the-art works- *Certified* (Liu et al., 2020) and *COMET* (Sivaraman et al., 2020) require multiple runs in order to even satisfy monotonic constraints whereas monotonicity is guaranteed by simply employing the proposed monotonic dense units.

The most important advantage of our solution is simplicity and computational complexity. Our models have slightly better performance on all datasets we tested them on, but it is important to note they have significantly fewer parameters and the simplest training procedure. As such, they have the potential to significantly reduce the carbon footprint when used at scale:

- The number of parameters of our model is an order or even two orders of magnitude smaller than alternatives. At prediction time, this translates roughly into 1-2 orders of magnitude less computation (most parameters are used in multiplications)
- At training time, we do not use any additional computationally expensive procedures apart from gradient descent, unlike alternative approaches such as *COMET* (Sivaraman et al., 2020) and *Certified* (Liu et al., 2020).

5. Conclusion

In this paper, we proposed a simple and elegant solution to build constrained monotonic networks which can approximate any continuous partially monotonic function. Specifically, we introduced a constrained monotone fully connected layer which can be used as a drop-in replacement for a fully connected layer to enforce monotonicity. We then employed our constrained monotone fully connected layer to build neural network models and showed that we can achieve better results to the recent state-of-the-art (Sivaraman et al., 2020; Liu et al., 2020) in addition to the well-known works such as Min-Max networks (Daniels & Velikova, 2010) and DLNs (You et al., 2017). However, the main advantage of the proposed solution is not higher accuracy but its computational and memory complexity: we use orders of magnitude

fewer parameters and computation which makes the resulting neural networks more energy efficient. Last but not least, we proved such networks can approximate any multivariate monotonic function.

Extending these results to other types of neural network layers is straightforward, but we leave it for future work. E.g. convolutional layer can be made monotonic with respect to a subset of features by replacing its fully-connected filters with monotone constrained fully connected layers. As long as such layers are combined with other monotonic convolutional and fully connected layers and monotone activation functions, the resulting neural network would be monotone.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from https://tensorflow.org.

Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 146–155. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/amos17b.html.

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

Barrett, C. and Tinelli, C. Satisfiability modulo theories. In *Handbook of model checking*, pp. 305–343. Springer, 2018.

Bisong, E. *Google Colaboratory*, pp. 59–64. Apress, Berkeley, CA, 2019. ISBN 978-1-4842-4470-8. doi:10.1007/978-1-4842-4470-8_7. URL https://doi.org/10.1007/978-1-4842-4470-8_7.

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

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi:10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

Chollet, F. et al. Keras. https://keras.io, 2015.

Clevert, D., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (ELUs). In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989. ISSN 0932-4194. doi:10.1007/BF02551274. URL http://dx.doi.org/10.1007/BF02551274.

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

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., and Garcia, R. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems*, 13, 2000.

Eidnes, L. H. and Nøkland, A. Shifting mean activation towards zero with bipolar activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

Gennari, J. H., Langley, P., and Fisher, D. Models of incremental concept formation. *Artificial intelligence*, 40(1-3): 11–61, 1989.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323. JMLR Workshop and Conference Proceedings, 2011.

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

Gupta, M., Cotter, A., Pfeifer, J., Voevodski, K., Canini, K., Mangylov, A., Moczydlowski, W., and Van Esbroeck, A. Monotonic calibrated interpolated look-up tables. *The Journal of Machine Learning Research*, 17(1):3790–3836, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Hendrycks, D. and Gimpel, K. Bridging nonlinearities and stochastic regularizers with Gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, Mar 1991. doi:10.1016/0893-6080(91)90009-T. URL https://doi.org/10.1016/0893-6080(91)90009-T.

J. Angwin, J. Larson, S. M. and Kirchner, L. Machine bias: There's software used across the country to predict future criminals. and it's biased against blacks. *ProPublica*, 2016.

Kidger, P. and Lyons, T. Universal approximation with deep narrow networks. In *Proceedings of the 33rd Annual Conference on Learning Theory (COLT 2020)*, pp. 2306–2327, 2020.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. *Advances in neural information processing systems*, 30, 2017.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 1476-4687. doi:10.1038/nature14539.

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

Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. Citeseer, 2013.

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

Mitchell, T. M. The need for biases in learning generalizations. Technical report, Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. In *ICML*, pp. 807–814, 2010.

Neal, R. M. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113, 1992.

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al. Kerastuner. https://github.com/keras-team/keras-tuner, 2019.

Pinkus, A. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi:10.1017/S0962492900002919.

Quinlan, J. R. Combining instance-based and model-based learning. In *Proceedings of the tenth international conference on machine learning*, pp. 236–243, 1993.

Ramachandran, P., Zoph, B., and Le, Q. V. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7(1):5, 2017.

Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Rudin, C., Wang, C., and Coker, B. The age of secrecy and unfairness in recidivism prediction. *Harvard Data Science Review*, 2(1), 3 2020.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Runje, D. and Shankaranarayana, S. M. Code of the experiments in the paper "Constrained Monotonic Neural Networks", May 2023a. URL https://github.com/airtai/mono-dense-keras.

Runje, D. and Shankaranarayana, S. M. Preprocessed datasets for experiments in the paper "Constrained Monotonic Neural Networks", May 2023b. URL https://doi.org/10.5281/zenodo.7968969.

Shang, W., Sohn, K., Almeida, D., and Lee, H. Understanding and improving convolutional neural networks via concatenated rectified linear units. *CoRR*, abs/1603.05201, 2016. URL http://arxiv.org/abs/1603.05201.

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

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

Sivaraman, A., Farnadi, G., Millstein, T., and Van den Broeck, G. Counterexample-guided learning of monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:11936–11948, 2020.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In Pereira, F., Burges, C., Bottou, L., and Weinberger,

K. (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

Veličković, P. *The resurgence of structure in deep neural networks*. PhD thesis, University of Cambridge, 2019.

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

Zagoruyko, S. and Komodakis, N. Diracnets: Training very deep neural networks without skip-connections. *CoRR*, abs/1706.00388, 2017. URL https://arxiv.org/abs/1706.00388.

Zheng, H., Yang, Z., Liu, W., Liang, J., and Li, Y. Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–4. IEEE, 2015.

A. Detailed proofs

We restate all lemmas from the main text here are give detailed proofs of them.

The following is well known result, proved here for completeness only:

Lemma 1. *For each $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$ we have:*

- *if $t_i = 1$, then $\dfrac{\partial h_j}{\partial x_i} \geq 0$, and*
- *if $t_i = -1$, then $\dfrac{\partial h_j}{\partial x_i} \leq 0$.*

Proof. From equation 3, we have

$$\mathbf{h} = |\mathbf{W}^T|_{\mathbf{t}} \cdot \mathbf{x} + \mathbf{b} \tag{8}$$

$$h_j = \sum_i w'_{i,j} x_i + b_j \tag{9}$$

$$\frac{\partial h_j}{\partial x_i} = w'_{i,j} \tag{10}$$

Finally, from equation 2 we have

$$\frac{\partial h_j}{\partial x_i} = \begin{cases} |w_{i,j}| \geq 0 & \text{if } t_i = 1 \\ -|w_{i,j}| \leq 0 & \text{if } t_i = -1 \end{cases}$$

□

Lemma 2. *Let $\mathbf{y} = \rho^{\mathbf{s}}(\mathbf{h})$. Then for each $j \in \{1, \dots, m\}$ we have $\dfrac{\partial y_j}{\partial h_j} \geq 0$. Moreover*

- *if $\mathbf{s} = (m, 0, 0)$, then $\rho^{\mathbf{s}}_j$ is convex; and*
- *if $\mathbf{s} = (0, m, 0)$, then $\rho^{\mathbf{s}}_j$ is concave.*

Proof. From equation 4,5 and 6:

$$\hat{\rho}(x) = -\breve{\rho}(-x)$$

$$\tilde{\rho}(x) = \begin{cases} \breve{\rho}(x+1) - \breve{\rho}(1) & \text{if } x < 0 \\ \hat{\rho}(x-1) - \hat{\rho}(1) & \text{otherwise} \end{cases}$$

$$\rho^{\mathbf{s}}(h_j) = \begin{cases} \breve{\rho}(h_j) & \text{if } j \leq \breve{s} \\ \hat{\rho}(h_j) & \text{if } \breve{s} < j \leq \hat{s} \\ \tilde{\rho}(h_j) & \text{otherwise} \end{cases}$$

we have:

$$\frac{\partial y_j}{\partial h_j} = \begin{cases} \breve{\rho}'(h_j) \geq 0 & \text{if } j \leq \breve{s} \\ \breve{\rho}'(-h_j) \geq 0 & \text{if } \breve{s} < j \leq \hat{s} \\ \breve{\rho}'(h_j + 1) \geq 0 & \text{if } \breve{s} + \hat{s} < j \text{ and } h_j < 0 \\ \breve{\rho}'(1 - h_j) \geq 0 & \text{if } \breve{s} + \hat{s} < j \text{ and } h_j \geq 0 \end{cases}$$

if $\mathbf{s} = (m, 0, 0)$, we have:

$$\rho^{\mathbf{s}}(h_j) = \breve{\rho}(h_j)$$

which is a convex function.

Similarly, if $\mathbf{s} = (0, m, 0)$, we have:

$$\rho^{\mathbf{s}}(h_j) = \hat{\rho}(h_j)$$

which is a concave function. □

Corollary 3. *For each $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$ we have:*

- *if $t_i = 1$, then $\dfrac{\partial y_j}{\partial x_i} \geq 0$,*
- *if $t_i = -1$, then $\dfrac{\partial y_j}{\partial x_i} \leq 0$,*
- *if $\mathbf{s} = (m, 0, 0)$, then $\mathbf{y}_j$ is convex; and*
- *if $\mathbf{s} = (0, m, 0)$, then $\mathbf{y}_j$ is concave.*

For completeness, we repeat the Theorem 3.1 from (Daniels & Velikova, 2010) and its proof here:

Theorem 4. *For any continuous monotone nondecreasing function $f : K \to \mathbb{R}$, where K is a compact subset of $\mathbb{R}^k$, there exists a feedforward neural network using the sigmoid as the activation function with at most k hidden layers, positive weights, and output O such that $|O_x - f(x)| < \epsilon$, for any $x \in K$ and $\epsilon > 0$.*

Proof. The proof is derived by induction on the number of input variables k. Without loss of generality, we may assume that $f > 0$ (otherwise, we add a constant C and approximate $f + C$ with the network output O, then modify O with a negative bias at the output node). First, we assume that f is strictly increasing and C^∞. In case of $k = 1$, we write

$$f(x) = \int_0^\infty \mathbf{H}\left(f(x) - u\right) du \tag{11}$$

where $\mathbf{H}$ is the Heavyside function:

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Since f is continuous and increasing, it is invertible and therefore the right-hand side of 11 can be written as

$$f(x) = \int_0^\infty \mathbf{H}\left(x - f^{-1}(v)\right) dv \tag{12}$$

The integral can be approximated arbitrarily well by a Riemann sum

$$\sum_{i=1}^{N} (v_{i+1} - vi)\, \mathbf{H}\left(x - f^{-1}(v_i)\right) \tag{13}$$

where $[v_i]_{i=1}^N$ is a partition of the interval $[f(a), f(b)]$. This expression corresponds to a neural network with input x, one hidden layer with N neurons all connected to the input with weight of 1, bias term in the hidden neurons $f^{-1}(v_i)$, and the weights connecting the hidden layer with the output $v_{i+1} - vi > 0$. Note that the Heavyside function $\mathbf{H}$ can be replaced by a sigmoid activation function using a standard approximation argument.

Assume that Theorem 3.1 holds for $k - 1$ input variables. We now combine the integral representation in 11 with the induction assumption. For a given v, we may solve the equation of the level set corresponding to to v for x_k

$$f(x_1, \ldots, x_k) = v.$$

By the implicit function theorem, there exists a function g_v such that

$$f(x_1, \ldots, g_v(x_1, \ldots, x_{k-1})) = v. \tag{14}$$

Note that g_v is decreasing in all arguments x_i. This can be seen by taking the partial derivative of 14 with respect to x_i. We will now show that

$$\mathbf{H}\left(f(x) - v\right) = \mathbf{H}\left(x_k - g_v(x_1, \ldots, x_{k-1})\right)$$

analogously to 12 for the 1-D case. Note that it is sufficient to show that

$$f(x) < v \text{ if and only if } x_k < g_v(x_1, \ldots, x_{k-1})$$

and

$$f(x) > v \text{ if and only if } x_k > g_v(x_1, \ldots, x_{k-1}).$$

But this follows easily from 14 and the fact that f is increasing in all its arguments. We now approximate the integral in 11 with a Riemann sum, leading to the following equation analogously to 13:

$$R = \sum_{i=1}^{N} (v_{i-1} - v_i)\, \mathbf{H}\left(x_k - g_{v_i}\left(x_1, \ldots, x_{k-1}\right)\right) \tag{15}$$

Since g_{v_i} is decreasing in all its arguments $-g_{v_i}$ is increasing. By the induction assumption, we can approximate $-g_{v_i}$ with a feedforward neural network O_i with $x_1, \ldots, x_{k-1}$ as inputs, $k-1$ hidden layers, and nonnegative weights, such that

$$\left| \sum_{i=1}^{N} (v_{i-1} - v_i)\, \mathbf{H}\left(x_k - O_i\left(x_1, \ldots, x_{k-1}\right)\right) - R \right| < \epsilon$$

because the sum is finite. Expression 15 corresponds to a feedforward neural network with k inputs and k hidden layers. Here $k-1$ hidden layers are needed to represent $-g_{v_i}$ and the k-th hidden layer is needed to combine N neural networks with outputs O_i and the input x_k. The weights on the connections between the last hidden layer and the final output are $(v_{i+1} - v_i) > 0$. The input x_k is directly (skip-layer) connected to the k-th hidden layer.

We can now easily generalize the proof to continuous nondecreasing functions. For continuous functions, we define the convolution of f with a mollifier K_δ by

$$f_\delta = f \otimes K_\delta$$

Then, f_δ is C^∞ and $f_\delta \longrightarrow f$ as $\delta \downarrow 0$ uniform on compact subsets. Furthermore, f_δ is also increasing since $K_\delta > 0$. Now choose δ such that $|f - f_\delta| < \frac{\epsilon}{2}$ and approximate f_δ with a feedforward neural network O such that $|f_\delta - O| < \frac{\epsilon}{2}$. Then, $|f - O| < \epsilon$.

If f is nondecreasing, then approximate f by f_δ

$$f_\delta = f + \delta(x_1 + \cdots + x_k)$$

which is strictly increasing and let $\delta \downarrow 0$.

□

Lemma 5. *Let $\breve{\rho} \in \breve{\mathcal{A}}$. Then the Heavyside function can be approximated with $\tilde{\rho}_H$ on $\mathbb{R}$, where*

$$\tilde{\rho}_H(x) = \alpha\tilde{\rho}(x) + \beta$$

for some $\alpha, \beta \in \mathbb{R}$ and $\alpha > 0$.

Proof. Since $\breve{\rho}$ is bounded from below, then it has a limit

$$c = \lim_{x \to -\infty} \breve{\rho}(x) = -\lim_{x \to \infty} \hat{\rho}(x) < 0$$

From equation 5, we have

$$\lim_{x \to -\infty} \tilde{\rho}(x) = \lim_{x \to -\infty} \breve{\rho}(x) - \breve{\rho}(1) = c - \breve{\rho}(1)$$
$$\lim_{x \to +\infty} \tilde{\rho}(x) = \lim_{x \to \infty} \hat{\rho}(x) + \breve{\rho}(1) = -(c - \breve{\rho}(1))$$

Let $\tilde{\rho}_H$ be defined as follows:

$$\tilde{\rho}_H(x) = \frac{\tilde{\rho}(x) - c + \breve{\rho}(1)}{2\left(-c + \breve{\rho}(1)\right)}$$

Then

$$\lim_{a \to \infty} \tilde{\rho}_H(a \cdot x) = H(x)$$

□

Lemma 6. *Let $\tilde{\rho}_{\alpha,\beta}$ be an activation function for some $\alpha, \beta \in \mathbb{R}, \alpha > 0$ such that for every $x \in \mathbb{R}$*

$$\tilde{\rho}_{\alpha,\beta}(x) = \alpha\tilde{\rho}(x) + \beta.$$

Then for every constrained monotone neural network $\mathcal{N}_{\alpha,\beta}$ using $\tilde{\rho}_{\alpha,\beta}$ as an activation function ($s = (0, 0, \tilde{s})$), there is a constrained monotone neural network $\mathcal{N}$ using $\tilde{\rho}$ as an activation function such that for every $\mathbf{x} \in \mathbb{R}^n$:

$$\mathcal{N}(\mathbf{x}) = \mathcal{N}_{\alpha,\beta}(\mathbf{x}).$$

Proof. Let $\mathbf{h}_k$ be the output of the k-th constrained linear layer of $\mathcal{N}_{\alpha,\beta}$ evaluated on input $\mathbf{x}$. For every $k > 1$, we have

$$\begin{aligned}
\mathbf{h}_1 &= |\mathbf{W}_1|_{\mathbf{t}_1} \cdot \mathbf{x} + \mathbf{b}_1 \\
\mathbf{h}_k &= |\mathbf{W}_k|_{\mathbf{t}_k} \cdot \mathbf{y}_{k-1} + \mathbf{b}_k \\
\mathbf{y}_k &= \tilde{\rho}_{\alpha,\beta}(\mathbf{h}_k) \\
\mathbf{y} &= \mathbf{h}_l
\end{aligned}$$

where l is the total number of layers of $\mathcal{N}_{\alpha,\beta}$. We can rewrite $\mathbf{h}_k$ as follows:

$$\begin{aligned}
\mathbf{h}_k &= |\mathbf{W}_k|_{\mathbf{t}_k} \cdot \mathbf{y}_{k-1} + \mathbf{b}_k \\
&= |\mathbf{W}_k|_{\mathbf{t}_k} \cdot \tilde{\rho}_{\alpha,\beta}(\mathbf{h}_{k-1}) + \mathbf{b}_k \\
&= |\mathbf{W}_k|_{\mathbf{t}_k} \cdot (\alpha\tilde{\rho}(\mathbf{h}_{k-1}) + \beta\mathbf{1}) + \mathbf{b}_k && \text{(with } |\mathbf{1}| = |\mathbf{h}_{k-1}|\text{)} \\
&= \alpha|\mathbf{W}_k|_{\mathbf{t}_k} \cdot \tilde{\rho}(\mathbf{h}_{k-1}) + \beta|\mathbf{W}_k|_{\mathbf{t}_k}\mathbf{1} + \mathbf{b}_k \\
&= |\alpha\mathbf{W}_k|_{\mathbf{t}_k} \cdot \tilde{\rho}(\mathbf{h}_{k-1}) + \beta|\mathbf{W}_k|_{t_k}\mathbf{1} + \mathbf{b}_k && \text{(from } \alpha > 0\text{)} \\
&= |\mathbf{W}'_k|_{t_k} \cdot \tilde{\rho}(\mathbf{h}_{k-1}) + \mathbf{b}'_k
\end{aligned}$$

for $\mathbf{W}'_k = \alpha\mathbf{W}_k$ and $\mathbf{b}'_k = \beta|\mathbf{W}_k|_{t_k}\mathbf{1} + \mathbf{b}_k$.

Hence, for every $\mathbf{x} \in \mathbb{R}^n$, the output of the neural network $\mathcal{N}$ with weights $\mathbf{W}_1, \mathbf{W}'_2, \ldots, \mathbf{W}'_l$ and biases $\mathbf{b}_1, \mathbf{b}'_2, \ldots, \mathbf{b}'_l$ is:

$$\mathcal{N}(\mathbf{x}) = \mathcal{N}_{\alpha,\beta}(\mathbf{x})$$

□

Theorem 7. *Let $\breve{\rho} \in \breve{\mathcal{A}}$. Then any multivariate continuous monotone function f on a compact subset of $\mathbb{R}^k$ can be approximated with a monotone constrained neural network of at most k layers using ρ as the activation function.*

Proof. The proof of the Theorem 4 uses only the fact that the Heavyside function $\mathbf{H}$ defined as

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

can be approximated with the sigmoid function on a closed interval (since $\lim_{a \to \infty} \sigma(ax) = H(x)$).

From the Theorem 4 and the Lemma 5, any continuous monotone function f on a compact subset of $\mathbb{R}^k$ can be approximated with a monotone constrained neural network with at most k layers using $\tilde{\rho}_H$ as the activation function.

From Lemma 6, we can replace $\tilde{\rho}_H$ with $\tilde{\rho}$. Hence, any continuous monotone function f on a compact subset of $\mathbb{R}^k$ can be approximated with a monotone constrained neural network with at most k layers using ρ as the activation function and $\mathbf{s}_i = (0, 0, \tilde{s}_i)$.

□

B. Datasets Description

The descriptions of datasets used for comparison are detailed below. As mentioned in the section 4, the datasets are chosen from (Liu et al., 2020) and (Sivaraman et al., 2020) for proper evaluation. The train-test splits of $80\% - 20\%$ are used for all comparison experiments.

1. COMPAS (J. Angwin & Kirchner, 2016) is a binary classification dataset, where the task is to predict risk score of an individual committing crime again two years, based on the criminal records of individuals arrested in Florida. The risk score needs to be monotonically increasing with respect to the following attributes `number of prior adult convictions`, `number of juvenile felony`, `number of juvenile misdemeanor`, and `number of other convictions`. It should be noted that there have been ethical concerns with the dataset (J. Angwin & Kirchner, 2016; Rudin et al., 2020)

2. Blog Feedback (Buza, 2014) is a regression dataset where the task is to predict the number of comments in the upcoming 24 hours from a feature set containing 276 features of which 8 (A51, A52, A53, A54, A56, A57, A58, A59) are monotonic features. The readers are suggested to refer to link [2] for more details. As mentioned by the authors of (Liu et al., 2020), only the data points with targets smaller than the 90th percentile are used since the outliers could dominate the mean-squared-error metric.

3. Lending club loan data[3] is a classification dataset, where the task is to predict whether the individual would default on loan, from a feature set having 28 features containing data such as the current loan status, latest payment information etc,. The probability of default should be non-decreasing with respect to `number of public record bankruptcies`, `Debt-to-Income ratio`, and non-increasing with respect to `credit score`, `length of employment`, `annual income`.

4. Auto MPG[4] (Quinlan, 1993) is a regression dataset where the task is to predict city-cycle fuel consumption in miles per gallon (MPG) from a feature set containing 7 features of which the monotonic features are `weight (W)`, `displacement (D)`, and `horse-power (HP)`

5. Heart Disease[5] (Gennari et al., 1989) is a classification dataset, where the task is to predict the presence of heart disease from a feature set containing 13 features of which the risk associated with heart disease needs to be monotonically increasing with respect to the features `trestbps (T)`, `cholestrol (C)`)

[2] https://archive.ics.uci.edu/ml/datasets/BlogFeedback
[3] https://www.kaggle.com/wendykan/lending-club-loan-data
[4] https://archive.ics.uci.edu/ml/datasets/auto+mpg
[5] https://archive.ics.uci.edu/ml/datasets/heart+disease