Constrained Monotonic Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep neural networks are becoming increasingly popular in approximating arbitrary functions from noisy data. But wider adoption is being hindered by the need to explain such models 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 its weights to be non-negative while employing a monotonic activation function. Unfortunately, this construction does not work with popular non-saturated activation functions such as ReLU, ELU, SELU etc, as it can only approximate convex functions. We show this shortcoming can be fixed by employing the original activation function for a part of the neurons in the layer, and employing its point reflection for the other part. Our experiments show this approach of building monotonic deep neural networks have matching or better accuracy when compared to other state-of-the-art methods such as deep lattice networks or monotonic networks obtained by heuristic regularization. This method is the simplest one in the sense of having the least number of parameters, not requiring any modifications to the learning procedure or steps post-learning steps.

1 Introduction

Deep Learning has witnessed widespread adoption in many critical real-world domains such as finance, healthcare, etc [19]. Predictive models built using deep neural networks have been shown to have high accuracy. Incorporating prior knowledge such as monotonicity in trained models help in improving the performance and generalization ability of the trained models [23, 8]. Introduction of structural biases such as monotonicity make models also more data-efficient, enabling a leap in predictive power on smaller datasets [39]. 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 [12]. 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. All other things being equal, a larger house should be deemed more valuable, a bank's clients with higher income should be eligible for a larger loan [33], an offender with a longer crime history should be predicted as more likely to commit another crime [29], etc. A model without such a monotonic property would not, and certainly should not, be trusted by the society to provide a basis for such important decisions. However, the monotonicity of deep learning models is not a guaranteed property even when trained on monotonic data, let alone when training on noisy data typically used in practice.

Although, monotonicity is an important and increasingly often even required property, there is no simple or easy method to enforce it. It has been an active area of research and the existing methods on the subject can be broadly categorized into two types:

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

1. Monotonic architectures by construction: This line of research focuses on neural architectures guaranteeing monotonicity by construction [1, 34, 7, 22, 41].
2. Monotonicity by regularization: This line of research focuses on enforcing monotonicity in neural networks during training by employing a modified loss function or a heuristic regularization term [35, 11]. However, employing a modified loss function or a regularization term give us no guarantees that the trained model is indeed monotonic. A recent line of research work focuses on verifying/certifying monotonicity of such models [36, 20]. These approaches are computationally very expensive and they require multiple retraining cycles to enforce monotonicity.

We will give a more detail account of the existing methods in the next section.

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 almost 30 years [1]. However, this method does not work well in practice. When used in conjuction 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 [24], the resulting models are always convex [20], severely limiting the applicability of the method in practice.

The main contribution of this paper is a simple modification of the method above which, in conjuction with non-saturated activation functions, is capable of approximating non-convex functions as well: if both concave and convex monotone activation functions are used in a neural network with constrained weights, it regains the ability to approximate monotone continuous functions that are either convex, concave or nothing of the above.

There are a number of possible ways to satisfy this condition, with the simplest one using both an activation function $\breve{\rho}$ and its point reflection with respect to point $(0, 0)$ defined as:

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

Assuming the original activation function a is both monotonic and convex, the properties holding for ReLU, ELU, SELU and Softplus, the proposed modification uses the original activation function a on a part of the neurons in the network and its point reflection $\dot{a}$ on the rest of the neurons.

Depending on whether should the resulting model be convex, concave or neither, the part of the neurons using the original activation function is all, nothing or something in between, respectively.

The resulting model is guaranteed to be monotonic, can be used in conjunction with any 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 number of parameters) and easier to train.

Our contributions can be summarized as follows:

1. We propose a modification to an existing constricted neural network layer enabling it to model non-convex functions when used with non-saturated monotone convex activation functions such as ReLU, ELU, SELU and alike.
2. We perform comparison with other recent works and show that our proposed novel neural network block can yield comparable and in some cases better results than the previous state-of-the-art and with significantly less parameters.

2 Related work

Methods to build monotonic models can be broadly categorized into two broad categories: monotonic by construction and monotonic by regularization. Before dwelling deeper into those methods, we give a short overview of activation functions as they are crucial to understanding our work.

2.1 Activation functions

Right from it's inception in perceptron [28], non linear activation functions have historically been one of the most important components of neural networks. Previously, the saturated functions such as the sigmoid [30], the hyperbolic tangent [25] 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 the modern neural networks is use of non-saturated activation functions. The use of *Rectified Linear Unit* (ReLU) [24, 10] as activation function was instrumental in achieving good performance in newer architectures. The ReLU has since become defacto choice of activation in most of the practical implementations and continues to be widely used because of its advantages such as simple computation, representational sparsity and linearity.

Later, small modifications to ReLU activation function termed *Leaky ReLU* [21] and *Exponential Linear Unit* (ELU) [5] were proposed to address the problem of dead neurons in ReLU. A parametic form of Leaky ReLU termed *Parametric ReLU* (PReLU) was later proposed in [13]. The authors of [13] also introduced a better initialization scheme for these rectifier-like activation functions that helped in exploring powerful deep neural architectures and was one of the first works that surpassed the human performance in Imagenet classification task [31]. The major advantages attributed to the use of non-saturated activation functions such as ReLU, Leaky ReLU, ELU and PReLU over the saturated ones such as *sigmoid* and *hyperbolic tangent* are alleviation of the problem of exploding and vanishing gradient and acceleration of the convergence during training [40].

The more recent works on activation functions such as *Softplus* [43], *Gaussian Error Linear Units* (GELU) [14], *Swish* [27], *Scaled Exponential Linear Unit* (SELU) [18] etc., have broadly focussed on solving problems of dead ReLU and aid in faster convergence. These activation functions have been shown to provide improved results and hence tend to be useful in practice.

ReLU and its variants LeakyReLU, PReLU, ELU, SELU and Softmax have the following important properties for our work: they are non-decreasing, monotonic and convex. Notice that this is not a general rule for activation functions used today, e.g. GELU, Swish are not monotonic and we will not use it in our construction.

The idea of using the both the original activation function and its point reflection in the same layer has been proposed in [32] where both outputs of ReLU and the negative value of its point reflection were used in construction of *concatenated ReLU* (CReLU) activation function. The proposed modification outputs two values instead of one and therefore increases the number of parameters. In [42], the authors propose *negative concatenated ReLU* (NCReLU) flip the sign and use the point reflection directly. Notice that proposed architectural change could be applied to other non-saturated, monotonic activation functions as well, but with unknown impact on their performance.

In [9], 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. The same construction could be used with other ReLU family activation functions as well. However, the focus of their work was to alleviate the need of normalizing layers and improving the performance of deep vanilla recurrent neural networks (RNNs) by using *bipolar ReLU*.

Both NCReLU and bipolar ReLU could have been be used in construction of constrained neural network capable of representing non-convex function, but to the best of our knowledge they have not been so far. The focus of attention when they were introduced was to increase performance of CNN networks on computer vision tasks and that was probably the reason why this, although important application, was overlooked.

2.2 Monotonicity by construction

Provided a monotone, non-decreasing activation function, a straightforward method to enforce non-decreasing monotonicity of a fully connected neural network is to constrain its weights to be non-negative as proposed in [1]. However, when compared to a neural network with a non-saturating activation function, the neural network using a saturating activation function (such as sigmoid and hyperbolic tangent) obtains inferior performance as shown in [24, 10]. This is also true in the case of constrained neural networks as described in the Section 4.

If we replace sigmoid with a non-decreasing monotone convex activation function such as ReLU, resulting networks are easy to train and but can approximate convex functions only [20]. This limitation makes the method highly impractical as in the real-world use cases the functions modelled are rarely convex.

Another approach to build monotonic neural architecture is Min-Max networks where monotonic linear embedding and max-min-pooling are used [34]. In [7], 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. Their construction does not allow for replacement with other activation functions.

Deep lattice networks (DLN) [41] use a combination of linear calibrators and lattices [22] for learning monotonic functions. This is the most widely used method in practice today, but not without its limitation. Lattices are structurally rigid thereby restricting the hypothesis space significantly. Also, DLN requires a very large number of parameters to obtain good performance.

2.3 Monotonicity by regularization

In the second category, the research works focus on enforning monotonicity during the training process by modifying the loss function or by adding additional regularization term.

In [35], the authors propose a modified loss function that penalizes 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 [11], 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. However, these methods do not guarantee monotonicity of trained model.

Recently, there is an increasing number of proposed methods to certify or verify monotonicity obtained by regularization methods. In [20], 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 to a mixed integer linear programming (MILP) problem that can be solved using an off-the-shelf MILP solver.

In [36], 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 [2]. To satisfy the monotonicity constraints, these counterexamples are included in the training data with adjustments to their target values so as to enforce next iterations of the model to be monotonic.

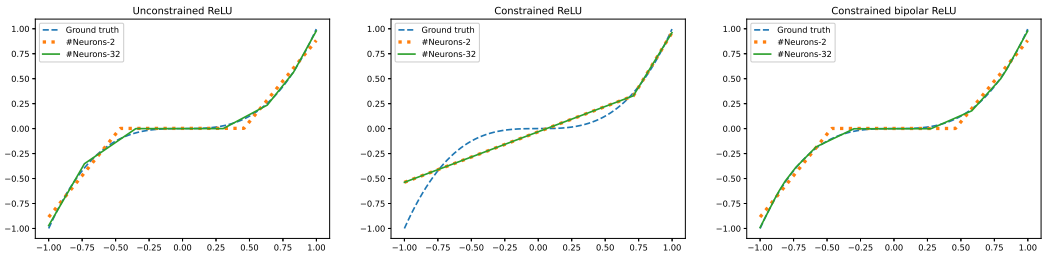
Both methods [20, 36] have been shown to supports ReLU as the activation function only and there is no obvious way how to extend them to other activation functions. More precisely, they rely on piece-wise linearity of ReLU to work, the property not satisfied by other variants such as ELU, SELU, GELU, 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.

2.4 Universal approximation property

The classical Universal Approximation Theorem [6, 15, 26] 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 [17], 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 [7], authors shows 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 of at most k layers.

However, the universal approximation theorem does not hold for constrained neural networks using monotonone convex non-saturated activation functions: the resulting model are convex and cannot



(a) Unconstrained ReLU (b) Constrained ReLU (c) Constrained bipolar ReLU

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

approximate non-convex functions. We hypothesize the universal approximation property with respect to monotonic functions holds if both convex and concave monotone activation functions are used.

3 Constrained neural networks

Most of the commonly used activation functions such as ReLU, ELU, SELU etc. are monotonically increasing, convex, 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 the 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 $[0, -1]$:

Fig. 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.

Fig. 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.

Fig. 1c. The proposed solution uses both convex and concave activation functions in the hidden layer, in this case bipolar ReLU, to gain 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. 2. The individual components of the proposed solution are defined and described in the subsequent subsection.

3.1 Constrained monotone fully connected layer

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

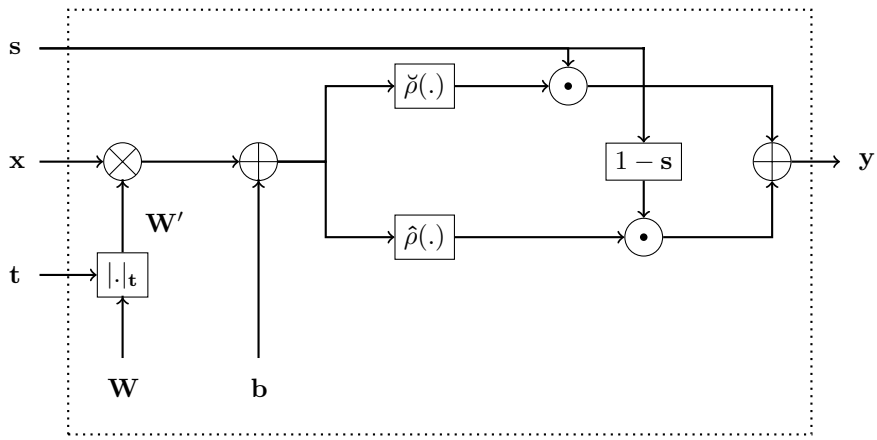


Figure 2: Proposed Monotonic Dense Unit or Constrained Monotone Fully Connected Layer: The input to the layer is one dimensional vector $\mathbf{x}$ with n elements and the output is one dimensional vector $\mathbf{y}$ with m elements. The layer is parameterized with kernel weights $\mathbf{W}$, a two-dimensional matrix with $n \times m$ elements, and one dimensional vector bias $\mathbf{b}$ with m elements. The weights $\mathbf{W}$ are then constrained to obtain $\mathbf{W}'$ using the operation $|.|_t$ with the help of n-dimensional monotonicity indicator vector $\mathbf{t}$. *convex* function $\breve{\rho}$ is monotonically increasing and convex activation function (such as ReLU, ELU) and $\hat{\rho}$ is its point reflection. We employ an m-dimensional *activation selector vector* $\mathbf{s} \in [0,1]^m$ to obtain a *combined activation function* $\rho^{\mathbf{s}}$ is a weighted sum of $\breve{\rho}$ and $\hat{\rho}$. Thus the output $\mathbf{y}$ is then obtained by $\mathbf{y} = \rho^{\mathbf{s}}(|\mathbf{W}|_{\mathbf{t}} \cdot \mathbf{x} + \mathbf{b})$

define its n-dimensional *monotonicity indicator vector* $\mathbf{t}$ element wise as follows:

$$t_i = \begin{cases} 1 & \text{if } \dfrac{\partial f(\mathbf{x})_j}{\partial x_i} \geq 0 \text{ for each } j \in \{1, \ldots, m\} \\ -1 & \text{if } \dfrac{\partial f(\mathbf{x})_j}{\partial x_i} \leq 0 \text{ for each } j \in \{1, \ldots, m\} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

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

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

Definition 1 (Constrained linear layer)**.** *Let* $\mathbf{W}$ *be an* $(n \times m)$*-dimensional matrix,* $\mathbf{x}$ *an* n*-dimensional vector,* $\mathbf{b}$ *an* m*-dimensional vector and* $\mathbf{h}$ *an* m*-dimensional vector. Then the output* $\mathbf{h}$ *of the* constrained linear layer *with monotonicity indicator vector* $\mathbf{t}$*, weights* $\mathbf{W}$*, biases* $\mathbf{b}$ *and input* $\mathbf{x}$ *is:*

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

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* $\dfrac{\partial h_j}{\partial x_i} \leq 0$*.*

Let $\breve{\rho}$ be a monotonically increasing *convex* function such as ReLU, ELU, etc. Then its point reflection $\hat{\rho}$ around the origin is:

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

Notice that $\hat{\rho}$ is a monotonically increasing *concave* function.

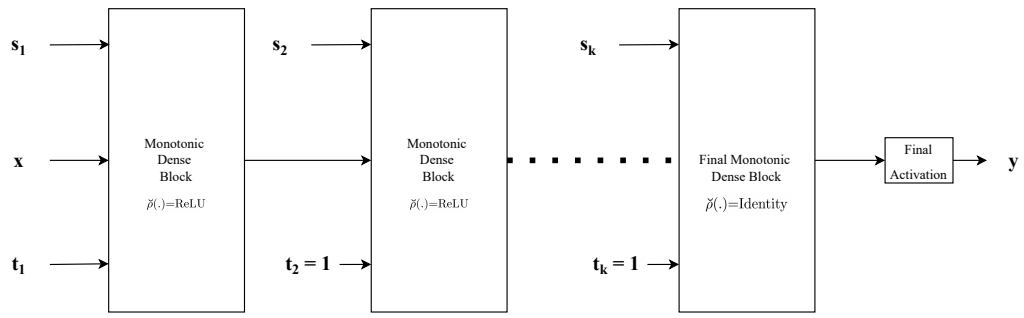


Figure 3: Neural architecture type 1

Definition 2 (Combined activation function)**.** *Given a monotonically increasing* convex *function* $\breve{\rho}$*, its point reflection* $\hat{\rho}$ *and* m*-dimensional* activation selector vector $\mathbf{s} \in [0,1]^m$*, the output of the* combined activation function $\rho^{\mathbf{s}}$ *is a weighted sum of* $\breve{\rho}$ *and* $\hat{\rho}$*:*

$$\rho^{\mathbf{s}}(\mathbf{h}) = \mathbf{s} \odot \breve{\rho}(\mathbf{h}) + (\mathbf{1} - \mathbf{s}) \odot \hat{\rho}(\mathbf{h}) \tag{5}$$

where $\odot$ *is the element-wise (Hadamard) product and* $\mathbf{1}$ *is* m*-dimensional vector with all elements equal to 1.*

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

- *if* $\mathbf{s} = \mathbf{1}$*, then* $\rho^{\mathbf{s}}_j$ *is convex; and*
- *if* $\mathbf{s} = \mathbf{0}$*, then* $\rho^{\mathbf{s}}_j$ *is concave.*

Definition 3 (Monotone constrained fully connected layer)**.** *Let* n *and* m *be positive natural numbers,* $\breve{\rho}$ *a monotonically increasing* convex *function,* $\mathbf{t}$ *an* n*-dimensional monotonicity selector vector and* $\mathbf{s}$ *an* m*-dimensional activation selector vector. Then the* monotone constrained fully connected layer *with* m *neurons is a tuple* $(\breve{\rho}, \mathbf{t}, \mathbf{s})$*, denoted* $\mathcal{MFC}_{\breve{\rho},\mathbf{s},\mathbf{t}}$*.*

Moreover, let weights $\mathbf{W}$ *be an* $(n \times m)$*-dimensional matrix, biases* $\mathbf{b}$ *an* m*-dimensional vector and input* x *an* n*-dimensional vector. Then the* output function *of the monotone constrained fully connected layer, denoted* $\mathcal{MFC}^{\mathbf{W},\mathbf{b}}_{\breve{\rho},\mathbf{s},\mathbf{t}}(\mathbf{x})$*, assigns an* m*-dimensional vector* $\mathbf{y}$ *to* $(\mathbf{W}, \mathbf{b}, \mathbf{x})$ *as follows:*

$$\mathbf{y} = \mathcal{MFC}^{\mathbf{W},\mathbf{b}}_{\breve{\rho},\mathbf{s},\mathbf{t}}(\mathbf{x}) = \rho^{\mathbf{s}}\left(|\mathbf{W}|_{\mathbf{t}} \cdot \mathbf{x} + \mathbf{b}\right) \tag{6}$$

Notice that $\mathcal{MFC}_{\breve{\rho},\mathbf{s},\mathbf{t}}$ determines an architecture of the layer, while $\mathcal{MFC}^{\mathbf{W},\mathbf{b}}_{\breve{\rho},\mathbf{s},\mathbf{t}}$ is a fully instantiated layer with all of its internal parameters $\mathbf{W}$ and $\mathbf{b}$ defining a function from input to output values.

From Lemma 1 and 2 directly follows:

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

$$\begin{aligned} &\text{if } t_i = 1, \text{ then } \frac{\partial y_j}{\partial x_i} \geq 0; && \text{if } \mathbf{s} = \mathbf{1}, \text{ then } \mathbf{y}_j \text{ is convex;} \\ &\text{if } t_i = -1, \text{ then } \frac{\partial y_j}{\partial x_i} \leq 0; && \text{if } \mathbf{s} = \mathbf{0}, \text{ then } \mathbf{y}_j \text{ is concave.} \end{aligned}$$

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. The figures Fig 3 and 4 shows two examples of neural architectures that can be built using the proposed monotonic dense block.

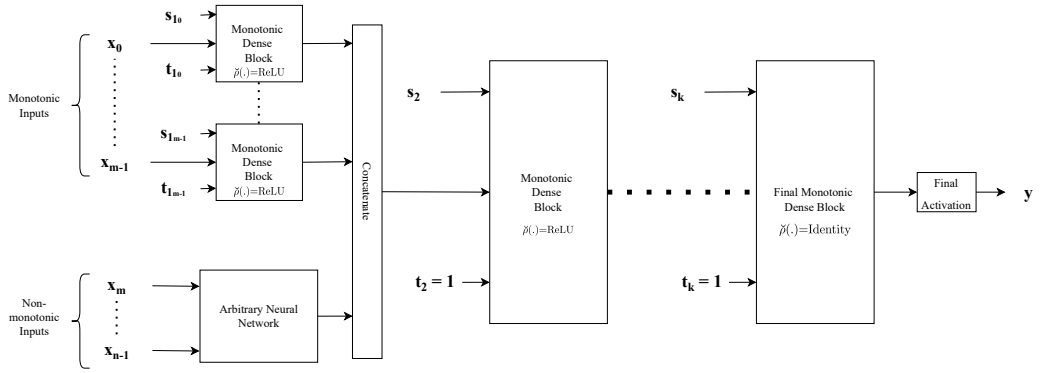


Figure 4: Neural architecture type 2

The figure Fig. 3 shows an example of a model with both monotonic feature and non-monotonic feature in the input. For non-monotonic feature, we can employ any kind of neural network layer such as simple fully connected or dense layer, or embedding neural network layer or even complicated block such as transformer etc. For the sake of simplicity, we show neural architecture example with only dense layer. For monotonic feature, we specifically employ the proposed monotonic dense unit. It should be noted that our proposed monotonic dense unit would act as a regular dense layer without any constraint on weights if the indicator vector $\mathbf{t}$ is 0. In this specific example, there is only one hidden monotonic dense layer and the bipolar activation is used in the hidden layer. For the final output monotonic dense layer we do not employ any activation but only the weights are constrained to be non-negative to preserve monotonicity. Depending on whether the problem at hand is a regression problem or a classification problem or even a multi-task problem we employ appropriate final activation function (such as linear activation or sigmoid or softmax) to obtain the final output.

The figure Fig. 4 shows another example of a model with monotonically increasing and convex feature, monotonically decreasing and concave feature, and non-monotonic feature in the input. Each of the input feature has a separate dense layer associated with it. The weights of the monotonic dense unit associated with the first feature is constrained to be positive and a convex monotonic activation function is used as is. The weights of the monotonic dense unit associated with the first feature is constrained to be negative and point reflection (or mirror) of a convex monotonic activation function is used. The third feature has a dense layer with unconstrained weights associated with it since it is non-monotonic. The weights of the subsequent neural layers are constrained to be non-negative just like the example above to enforce monotonicity in the whole network.

4 Experiments

In order to analyze practical utility of the proposed method, we experiment with various datasets and compare with the recent state-of-the-art. For the first set of experiments, we use the datasets employed by authors in [20] and use the exact train and test split for proper comparison. We perform experiments on 3 datasets: COMPAS [16], which is a classification dataset with 13 features of which 4 are monotonic; Blog Feedback Regression [3], 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)* [20] 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 [36] and once again use the exact train and test split for proper comparison. We compare with the method *COMET* described in [36] along with *Min-Max Net* [7] and *Deep Lattice Network (DLN)* [41] as described in [36].

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

For the classification tasks, we use cross-entropy and for the regression tasks, we use mean-squared-error as loss functions. We employ gridsearch to find the optimal number of neurons, network depth or layers, batch size, activation function and the number of epochs. For training, we first find the optimal learning rate using learning rate finder [37] and then train with one cycle policy [38].

4.1 Results

The results on the dataset above are summarized in Tables 1, 2. It shows that our method tends either match or surpass the other methods in terms of test accuracy for classification tasks and root mean squared error (RMSE) for regression tasks. For each of the datasets, we run the experiments ten times after finding the optimal hyperparameters and report mean and standard deviation of best five results. Experiment results show that networks learned by our method can achieve better results with fewer parameters, than the best-known and state-of-the-art algorithms for monotonic neural networks, including Min-Max Network [7] and Deep Lattice Network [41].

Method	COMPAS [16]		Blog Feedback [3]		Loan Defaulter	
	Parameters	Test Acc ↑	Parameters	RMSE ↓	Parameters	Test Acc ↑
Isotonic	N.A.	67.6%	N.A.	0.203	N.A.	62.1%
XGBoost [4]	N.A.	$68.5\% \pm 0.1\%$	N.A.	0.176 ± 0.005	N.A.	$63.7\% \pm 0.1\%$
Crystal [22]	25840	$66.3\% \pm 0.1\%$	15840	0.164 ± 0.002	16940	$65.0\% \pm 0.1\%$
DLN [41]	31403	$67.9\% \pm 0.3\%$	27903	0.161 ± 0.001	29949	$65.1\% \pm 0.2\%$
Min-Max Net [7]	42000	$67.8\% \pm 0.1\%$	27700	0.163 ± 0.001	29000	$64.9\% \pm 0.1\%$
Non-Neg-DNN	23112	$67.3\% \pm 0.9\%$	8492	0.168 ± 0.001	8502	$65.1\% \pm 0.1\%$
Certified [20]	23112	$68.8\% \pm 0.2\%$	8492	0.158 ± 0.001	8502	$65.2\% \pm 0.1\%$
Ours	**101**	$\mathbf{68.9\% \pm 0.5\%}$	**1101**	$\mathbf{0.156 \pm 0.001}$	**177**	$\mathbf{65.3\% \pm 0.001\%}$

Table 1: Comparison of our method with other methods described in [20]

Method	Auto MPG	Heart Disease
	RMSE ↓	Test Acc ↑
Min-Max Net [7]	10.14 ± 1.54	0.75 ± 0.04
DLN [41]	13.34 ± 2.42	0.86 ± 0.02
COMET [36]	8.81 ± 1.81	0.86 ± 0.03
Ours	$\mathbf{3.04 \pm 0.13}$	$\mathbf{0.86 \pm 0.02}$

Table 2: Comparison of our method with other methods described in [36]

5 Conclusion

In this paper, we proposed a simple and elegant solution to build constrained monotonic networks which can approximate convex as well as concave functions. 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 either similar or better results to the recent state-of-the-art ([36, 20] in addition to the well-known works such as Min-Max networks [7] and DLNs [41]. One drawback of our proposed method is that we are restricted by the choice of activation functions i.e., we can only employ monotonic activation functions. In future, we would like to build simple monotonic counterparts for other standard neural layers such as convolutional neural networks, recurrent neural networks and its variants, and attention models.

References

[1] Norman P Archer and Shouhong Wang. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision*

Sciences, 24(1):60–75, 1993.

[2] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of model checking*, pages 305–343. Springer, 2018.

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

[4] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[5] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[6] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

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

[8] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems*, 13, 2000.

[9] Lars Hiller Eidnes and Arild Nøkland. 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.

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[11] Akhil Gupta, Naman Shukla, Lavanya Marla, Arinbjörn Kolbeinsson, and Kartik Yellepeddi. How to incorporate monotonicity in deep networks while preserving flexibility? *arXiv preprint arXiv:1909.10662*, 2019.

[12] Maya Gupta, Andrew Cotter, Jan Pfeifer, Konstantin Voevodski, Kevin Canini, Alexander Mangylov, Wojciech Moczydlowski, and Alexander Van Esbroeck. Monotonic calibrated interpolated look-up tables. *The Journal of Machine Learning Research*, 17(1):3790–3836, 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[14] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with Gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

[15] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, Mar 1991.

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

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

[18] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in neural information processing systems*, 30, 2017.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[20] Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. Certified monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:15427–15438, 2020.

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

[22] Mahdi Milani Fard, Kevin Canini, Andrew Cotter, Jan Pfeifer, and Maya Gupta. Fast and flexible monotonic functions with ensembles of lattices. *Advances in neural information processing systems*, 29, 2016.

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

[24] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, pages 807–814, 2010.

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

[26] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[27] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7(1):5, 2017.

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

[29] Cynthia Rudin, Caroline Wang, and Beau Coker. The age of secrecy and unfairness in recidivism prediction. *Harvard Data Science Review*, 2(1), 3 2020.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[32] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. *CoRR*, abs/1603.05201, 2016.

[33] Arnab Sharma and Heike Wehrheim. Higher income, larger loan? monotonicity testing of machine learning models. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 200–210, 2020.

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

[35] Joseph Sill and Yaser Abu-Mostafa. Monotonicity hints. *Advances in neural information processing systems*, 9, 1996.

[36] Aishwarya Sivaraman, Golnoosh Farnadi, Todd Millstein, and Guy Van den Broeck. Counterexample-guided learning of monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:11936–11948, 2020.

[37] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[38] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[39] Peter Veličković. *The resurgence of structure in deep neural networks*. PhD thesis, University of Cambridge, 2019.

[40] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[41] Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta. Deep lattice networks and partial monotonic functions. *Advances in neural information processing systems*, 30, 2017.

[42] Sergey Zagoruyko and Nikos Komodakis. Diracnets: Training very deep neural networks without skip-connections. *CoRR*, abs/1706.00388, 2017.

[43] Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, and Yanpeng Li. Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4. IEEE, 2015.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] In Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Provided in supplementary
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Provided as part of code in supplementary. Also, it should be noted that we build our experiments based on two previous state-of-the-arts - Certified Monotonic Network [20] and COMET [36], therefore for fair comparison, we employ the exact splits the authors employed in their repective works
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See 1, 2
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Our code can be run on Google's Collaboratory and with default settings. A GPU instance uses a NVIDIA Tesla T4 GPU with 16GB RAM
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] Our work has also been applied as patent.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The code is included in the supplemental material and will be released on github
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

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 $i \in \{1, \ldots, n\}$ 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 $\dfrac{\partial h_j}{\partial x_i} \leq 0$.*

Proof. From $\mathbf{h} = |\mathbf{W}|_{\mathbf{t}} \cdot \mathbf{x} + \mathbf{b}$ we get $h_j = \sum_i w'_{i,j} x_i + b_j$. Hence $\dfrac{\partial h_j}{\partial x_i} = w'_{i,j}$. 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 $j \in \{1, \ldots, m\}$ we have $\dfrac{\partial y_j}{\partial h_j} \geq 0$. Moreover*

- *if $\mathbf{s} = \mathbf{1}$, then $\rho^{\mathbf{s}}_j$ is convex; and*
- *if $\mathbf{s} = \mathbf{0}$, then $\rho^{\mathbf{s}}_j$ is concave.*

Proof. From equation 5

$$\rho^{\mathbf{s}}(\mathbf{h}) = \mathbf{s} \odot \breve{\rho}(\mathbf{h}) + (\mathbf{1} - \mathbf{s}) \odot \hat{\rho}(\mathbf{h})$$

we have:

$$\begin{aligned} y_j &= s_j \breve{\rho}(h_j) + (1 - s_j)\hat{\rho}(h_j) \\ \frac{\partial y_j}{\partial h_j} &= s_j \breve{\rho}(h_j) + (1 - s_j)\hat{\rho}(h_j) \\ &= s_j \frac{\partial \breve{\rho}(h_j)}{\partial h_j} + (1 - s_j)\frac{\partial \hat{\rho}(h_j)}{\partial h_j} \end{aligned}$$

Since both $\breve{\rho}$ and $\hat{\rho}$ are monotonically increasing and $s_j \in [0, 1]$ we have:

$$\frac{\partial y_j}{\partial h_j} \geq 0$$

if $\mathbf{s} = \mathbf{1}$ or $\mathbf{s} = \mathbf{0}$, we have $\rho^{\mathbf{s}}(\mathbf{h}) = \breve{\rho}(\mathbf{h})$ or $\rho^{\mathbf{s}}(\mathbf{h}) = \hat{\rho}(\mathbf{h})$, which is a convex or a concave function in all components of the output, respectively.

□