# Monotonicity in Deep Learning: State of the Art

Varun Nandkumar Golani

University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany
vngolani@mail.uni-paderborn.de

**Abstract.** The performance of Machine learning (ML) models can be improved by incorporating some additional information related to data. This report explains in detail one such property which is monotonicity. Firstly, it categorizes the techniques to enforce monotonicity into two categories Configuration of the Network Architecture and Penalizing Non-Monotonicity. Secondly, it discusses some tests for detecting monotonicity between input features and output variable and some state-of-the-art techniques to enforce monotonicity categorized based on the two defined categories. Finally, this report concludes the topic with a comparison table summarizing the discussed techniques on different factors.

**Keywords:** Deep Neural Networks · Monotonicity · Neural Networks · Heuristic Test · Global Monotonicity Verification · Monotone MIN-MAX Networks · Deep Lattice Networks · Domain Adapted Neural Networks · Point-wise Loss · Certified Monotonic Neural Networks.

## 1 Introduction

Neural Network(s) (NN) have been in the literature for quite some time now [2] and recently Deep Neural Network(s) (DNN) are gaining a lot of attention because of the new techniques to train them which produce state-of-the-art results as compared to other baseline models on various tasks like image recognition [11], speech recognition [12] etc. ML models usually perform the best when they exploit certain known properties/characteristics of data beforehand. It can also be thought of as giving a learning direction to a ML model such that the probability of finding the right model is much higher. The property that this report will focus on is monotonicity.

Monotonicity is a relationship in which the increase in the input variable results in increase (decrease) of an output variable for a monotonically increasing (decreasing) input feature [1]. For instance, there is a monotonic decreasing trend between SCHUFA score (German credit score) and the loan interest rates. If all (the subset of) the input features and the output follow a monotonic trend then this is known as total (partial) monotonicity [4,5].

Monotonicity is of critical importance in systems where fairness or security is necessary, and it also helps with interpretability of the black-box NN/DNN models [9]. Monotonicity constraints can also be applied to NN/DNN models

in situations involving datasets with limited data, data with poor quality and noisy training data [10]. It is important to note that monotonicity should only be enforced if the input features and the output variable have a monotone relationship and not all input features should be assumed as monotone. Here [5] introduces a heuristic monotonicity test and considers those features as monotone for training, which results in a better model than fully monotonic network.

## 2 Related Work

The most common methods for enforcing monotonicity are by Configuration of the Network Architecture and by Penalizing Non-Monotonicity. The former method enforces monotonicity by constraining the weights to be positive (or negative) based on the problem, and it involves special layers in the network architecture which ensures monotonicity. The latter method introduces a term in the loss function which can be considered as a penalty for non-monotonicity and the NN/DNN optimizes this new loss function (training error + non-monotonicity error) which ensures monotonicity. A summary of the these methods used in the literature can be found in the following subsections.

### 2.1 Configuration of the Network Architecture (CNA)

Sill (1998) [3] uses a feed forward three layer (linear unit, hidden and output layer) NN architecture and enforces a positive weight constraint for links from the input layer to the linear unit layer. Daniels and Kamp (1999) [8] propose a monotonic index and using this train a monotonic NN for datasets having total monotonicity with respect to the input features and the output variable. Extending the network architecture of [3], Daniels and Velikova (2010) [5] use a Heuristic Test for Monotonicity to determine the monotonic trend between the independent input features and the output variable which is then used to train Partially Monotone Networks and these perform better than [3]. You et al. (2017) [6] propose a DLN with three types of layers (calibrators, linear embedding, ensemble of lattices) to learn a partially monotonic function which performs better than [5].

### 2.2 Penalizing Non-Monotonicity (PNM)

Sill and Abu-Mostafa (1996) [7] define a new term known as monotonicity error (apart from the training error) in the objective function and uses a NN to maximize it for classification and regression problems. Muralidhar et al. (2018) [10] incorporate monotonicity constraints by introducing a domain loss term along with the standard NN/DNN loss and the regularization term (to avoid overfitting) in the loss function and the model trained from this method is called Domain Adapted Neural Network(s) (DANN). Gupta et al. (2019) [4] introduce a monotonicity term in the loss function also known as Point-wise Loss (PWL). This error term can be used with any existing DNN architecture and is able to

enforce partial monotonicity. Liu et al. (2020) [9] propose a two-step method to train NN/DNN and the networks trained from this method are known as Certified Monotonic Neural Network(s) (CMNN). The algorithm alternates between two steps which are calculating the loss function (NN/DNN loss + monotonicity regularization term) and global monotonicity verification. If monotonicity constraints are not satisfied in step 2, weight for regularization $\lambda$ is increased and the algorithm continues, else the algorithm returns a CMNN.

## 3 Tests for Monotonicity

Now it is known that monotonicity is a desirable property to have, the question that needs to be answered is how to know which input features are monotonic with respect to the output variable. For some datasets it is known which input features are monotonic with respect to the output variable, but it cannot be known in advance for datasets with unknown features [7] or the lack of domain knowledge. The following are some methods from the literature to determine monotonicity/type of monotonicity given the dataset.

### 3.1 Heuristic Test

This method [5] defines a degree of monotonicity DgrMon of a dataset D which is as follows:

$$DgrMon(D) = \frac{\#Monotone\ Pairs(D)}{\#Comparable\ Pairs(D)} \tag{1}$$

The value of this measure is between 0 and 1. A value closer to or equal to 1 imply an increasing, a value closer to 0 imply a decreasing and a value closer to or equal to 0.5 implies no (for e.g. randomly distributed labels) monotonic trend between the input features and the output variable. This test cannot show that an input feature is monotone theoretically because not all data samples are available in the dataset.

To determine the effect of a single input feature on the output variable the measure is compared for cases with and without the input feature in D. If the measure without the input feature in D decreases as compared to with the input feature in D, this means that the removed input feature is monotone with respect to the output variable. The same procedure is carried out for each input feature and in the result we know which all input features are monotonic with respect to the output variable.

### 3.2 Global Monotonicity Verification

As the name suggests, global monotonicity verification [9] checks for all the data points in the input domain $\mathcal{X}$. Let $f$ be a differentiable function, $x_\alpha$ be a set of input features that are monotonic. To check if $f$ is monotone with respect to $x_\alpha$ on $\mathcal{X}$, the following equation is used:

$$U_\alpha := \min_{x,\, l \in \alpha} \{\partial_{x_l} f(x), x \in \mathcal{X}\} \tag{2}$$

To verify global monotonicity in 2, it is sufficient to show that $U_\alpha \geq 0$. This problem can be transformed into a Mixed Integer Linear Programming (MILP) problem and the new problem is as follows:

$$U_\alpha = \min_{x,\, l \in \alpha} \left\{ \sum_{i=1}^{n} a_i w_{i,l} z_i \;\; s.t. \;\; z_i \in G(x, w_i, b_i), \;\; x \in \mathcal{X} \right\} \tag{3}$$

where $G(x, w_i, b_i) = \left\{ z_i \mid z_i \in \{0,1\},\ w_i^T x + b_i \leq u_i z_i,\ w_i^T x + b_i \geq l_i(1 - z_i) \right\}$, $u_i = sup_{x \in \mathcal{X}} \{w_i^T x + b_i\}$, $l_i = inf_{x \in \mathcal{X}} \{w_i^T x + b_i\}$, $z_i \in \{0,1\}$ which indicates the ReLU activation.

The optimization problem in 3 can be solved using MILP solvers as discussed in [9]. For global monotonicity verification, the solution obtained after solving 3 should be greater than or equal to zero or some MILP solvers can stop early under an assigned budget and return a lower bound of the optimal value i.e. lower bound of $U_\alpha$ (sufficient to show the bound $U_\alpha \geq 0$). The detailed reformulation of the MILP problem from equation 2 to 3 is given in [9].

## 4 Enforcing Monotonicity by Configuration of the Network Architecture

### 4.1 Monotone MIN-MAX Networks (MMMN)

**Total Monotone MIN-MAX Networks (TMMMN)** TMMMN [3] (See Fig. 1.) have three layers (1 hidden layer, 1 MAX hidden layer, 1 MIN output layer) and for inputs where increasing (decreasing) monotonicity is required then the weights going from that input to the next hidden layer are constrained to be positive (or negative). The units from the first hidden layer are grouped into K groups by the MAX operator in the second layer. The third layer receives input from all the K MAX groups and computes the MIN over those K groups which is then the output of the network.

Formally, consider the K groups as $g_1$, $g_2$,....., $g_K$ and if each group k consists of $h_k$ hyperplanes $w^{(k,1)}$, $w^{(k,2)}$,....., $w^{(k,h_k)}$ then $g_k(x)$ is defined as:

$$g_k(x) = \max_j w^{(k,j)}.x - t^{(k,j)}, 1 \leq j \leq h_k \tag{4}$$

The final output of the network y is defined as:

$$y = \min_k g_k(x) \; (For\ Regression\ Problems) \tag{5}$$

$$y = \sigma(\min_k g_k(x)) \; (For\ Classification\ Problems) \tag{6}$$

where $\sigma(u) = e.g.\ \frac{1}{1+e^{-u}}$

These three layer networks can approximate any continuous, differentiable monotonic function given sufficiently many groups and sufficiently many hyperplanes within each group [3]. The MAX hidden layer over the K groups can approximate convex parts and the MIN output layer over all the MAX units can approximate concave parts of the target function.

**Partial Monotone MIN-MAX Networks (PMMMN)** PMMMN [5] are an extension of [3] with some minor changes in the architecture which are swapping the MIN and the MAX layer (i.e. second layer is a MIN hidden layer, final layer is a MAX output layer), monotonicity is only enforced with respect to monotonic inputs only. All the equations are similar to above and can be adapted by considering the changes as described here. Similar to TMMMN, PMMMN also have universal approximation capabilities which is also proved in [5].

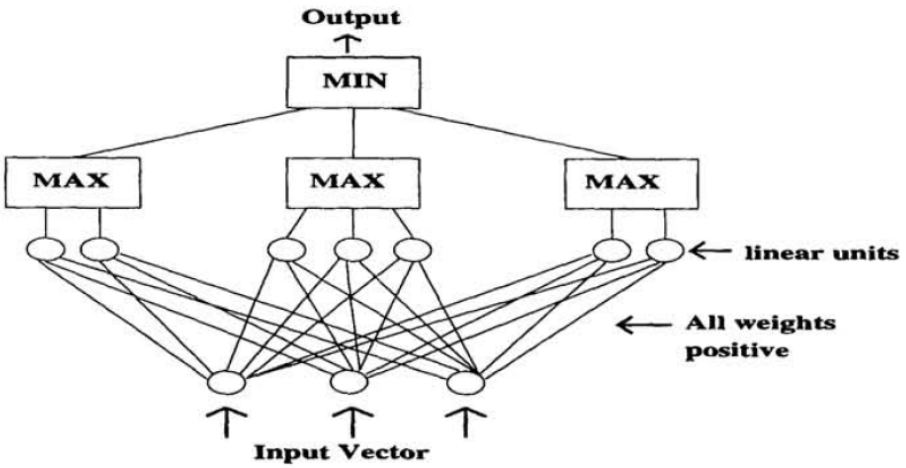


**Fig. 1.** Architecture of a 3 layer Total Monotonic MIN-MAX Network [3], all the connections from the input layer to the first hidden layer are constrained to be positive. Source: Sill, J.: Monotonic Networks. In: Advances in Neural Information Processing Systems 10, pages 661–667. MIT Press (1998)

### 4.2 Deep Lattice Networks (DLN)

A Deep Lattice Network [6] enforces partial monotonicity by incorporating the three layers into its architecture. Consider $x_{\mathrm{t}} \in \mathbb{R}^{\mathrm{D_t}}$ to be the input vector to the $\mathrm{t^{th}}$ layer and $x_{\mathrm{t}}[d]$ to be the $\mathrm{d^{th}}$ input for $d = 1, ..., D_{\mathrm{t}}$. Let $x_t^m$ be a subset of monotonic features from $x_{\mathrm{t}}$ and $x_t^n$ be a subset of non-monotonic features from $x_{\mathrm{t}}$. The layers for DLN are as follows:

**Linear Embedding Layer** A Linear Embedding consists of weight matrices $W_t^m \in \mathbb{R}^{D_{t+1}^m \times D_t^m}$ for $x_t^m$ and $W_t^n \in \mathbb{R}^{(D_{t+1} - D_{t+1}^m) \times (D_t - D_t^m)}$ for $x_t^n$ and a bias vector $b_{\mathrm{t}}$. To preserve monotonicity for the embedded vector $W_t^m x_t^m$, the following constraints are imposed:

$$W_t^m[i,j] \geq 0 \ \ \forall \ i,j \tag{7}$$

The final output of the linear embedding layer is:

$$x_{\mathrm{t}+1} = \begin{bmatrix} x_{t+1}^{m} \\ x_{t+1}^{n} \end{bmatrix} = \begin{bmatrix} W_t^m x_t^m \\ W_t^n x_t^n \end{bmatrix} + b_{\mathrm{t}} \tag{8}$$

The weight matrices $W_t^m$, $W_t^n$ and the bias $b_{\mathrm{t}}$ are discriminatively trained.

**Calibration Layer** This layer is a one dimensional piecewise linear transform for each input that will map $\mathbb{R}$ to a unit interval [0,1]. The input vector at the $(\mathrm{t}+1)^{\mathrm{th}}$ layer can be written as:

$$x_{\mathrm{t}+1} := \left[ c_{\mathrm{t},1}(x_{\mathrm{t}}[1]) \;\; c_{\mathrm{t},2}(x_{\mathrm{t}}[2]) \;\; ..... \;\; c_{\mathrm{t},\mathrm{D_t}}(x_{\mathrm{t}}[D_{\mathrm{t}}]) \right]^{\mathrm{T}} \tag{9}$$

where $c_{\mathrm{t,d}}$ is a 1D lattice with $K$ key-value pairs. Each 1D lattice is a look-up table with $K$ key-value pairs and the output of the lattice is linearly interpolated between the two b values corresponding to the two closest a values in the look-up table ($a \in \mathbb{R}^{\mathrm{K}}$, $b \in \mathbb{R}^{\mathrm{K}}$). The calibration function is given as follows:

$$c(x[d]; a, b) = \sum_{k=1}^{K} \alpha[k] ReLU(x - a[k]) + b[1] \tag{10}$$

where

$$\alpha[k] := \begin{cases} \frac{b[2]-b[1]}{a[2]-a[1]} & \text{if } k = 1 \\ \frac{b[k+1]-b[k]}{a[k+1]-a[k]} - \frac{b[k]-b[k-1]}{a[k]-a[k-1]} & \text{if } k = 2, ...., K-1 \\ -\frac{b[K]-b[K-1]}{a[K]-a[K-1]} & \text{if } k = K \end{cases}$$

The output parameters $b \in [0,1]^{\mathrm{K}}$ are discriminatively trained and monotonicity constraints over b can be incorporated by the following:

$$b[k] \leq b[k+1] \;\; for \; k = 1, 2, ..., K-1 \tag{11}$$

**Ensemble of Lattice Layers** An ensemble lattice layer is a collection of G lattices with S inputs for each lattice. Each S dimensional lattice (look-up table) has inputs in the range $[0,1]^{\mathrm{S}}$ and $2^{\mathrm{S}}$ parameters ($\theta \in \mathbb{R}^{2^{\mathrm{S}}}$), each parameter represents the lattice's output for each of the $2^{\mathrm{S}}$ vertices of the lattice. Two interpolation methods Multilinear Interpolation and Simplex Interpolation can be used. Multilinear Interpolation can be expressed as $\psi(x)^{\mathrm{T}}\theta$ where $\psi(x)$ is a non-linear feature transformation $\psi(x) : [0,1]^{\mathrm{S}} \rightarrow [0,1]^{2^{\mathrm{S}}}$. If the next layer of the network is also an ensemble lattice layer then the inputs from the previous layer are randomly rearranged and then are assigned to the $G_{\mathrm{t}+1} \times S_{\mathrm{t}+1}$ inputs of the next ensemble lattice layer.

The above layers ensure end-to-end monotonicity which means that if a component has a monotonic input then the corresponding output i.e. input to the next layer will also be a monotonic input. An example of a DLN can be seen in Fig. 2.

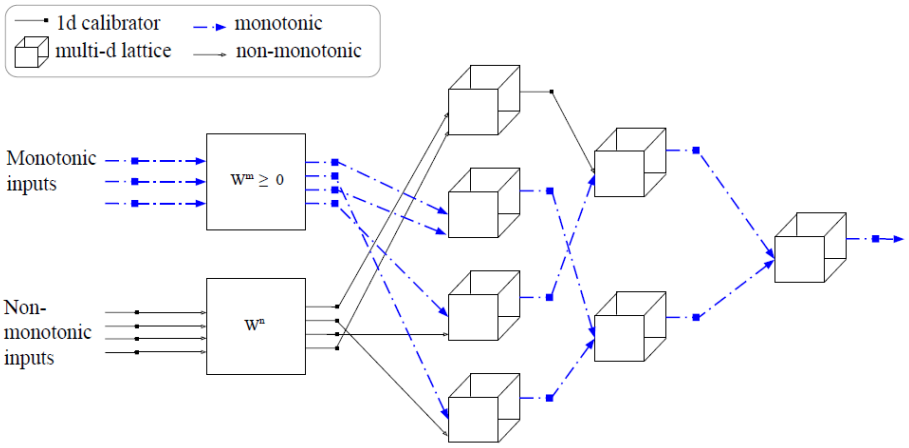


**Fig. 2.** An example of a 9 layer DLN [6]: calibrators, linear embedding, calibrators, ensemble of lattices, calibrators, ensemble of lattices, calibrators, lattice, calibrator. Source: You S. et al.: Deep lattice networks and partial monotonic functions. In: Advances in Neural Information Processing Systems 30, pages 2981–2989. Curran Associates, Inc. (2017)

## 5 Enforcing Monotonicity by Penalizing Non-Monotonicity

### 5.1 Domain Adapted Neural Network(s) (DANN)

A DANN [10] is a NN/DNN which incorporates domain knowledge in the training process. The domain knowledge that this section focuses on is monotonicity. This is achieved by modifying the loss function and the general form of the loss function is given as:

$$\underset{f}{\operatorname{argmin}}\ Loss(Y, \hat{Y}) + \lambda_D Loss_D(\hat{Y}_1, \hat{Y}_2) + \lambda R(f) \tag{12}$$

where $Y$ is the actual label/value, $\hat{Y}, \hat{Y}_1, \hat{Y}_2$ are the predicted labels/values, $Loss(Y, \hat{Y})$ is the NN/DNN loss, $Loss_D(\hat{Y}_1, \hat{Y}_2)$ is the calculated domain loss by enforcing the monotonicity constraint $\hat{Y}_1 < \hat{Y}_2$, $\lambda_D$ is the weight assigned to the domain loss, $R(f)$ is the L2 regularization term used to prevent the model $f$ from overfitting and $\lambda$ is the weight assigned to the regularization term.

The loss $Loss_D(\hat{Y}_1, \hat{Y}_2)$ in equation 12 can be expressed as:

$$Loss_D(\hat{Y}_1, \hat{Y}_2) = \sum_{i=1}^{m} \mathbb{I}\Big((x_1^i < x_2^i) \wedge (\hat{y}_1^i > \hat{y}_2^i)\Big).ReLU(\hat{y}_1^i - \hat{y}_2^i) \tag{13}$$

where $x_1, x_2$ are the same input feature measurements in different conditions, $\hat{y}_1^i, \hat{y}_2^i$ are the predicted values for input vectors $x_1, x_2$, $ReLU(z) = max(0, z)$, $\wedge$ is the logical and operator, $\mathbb{I}(a) = 1$ if $a$ is true else $\mathbb{I}(a) = 0$ if $a$ is false.

The equation 13 has a simple interpretation, if $x_1^i < x_2^i$ and $\hat{y}_1^i < \hat{y}_2^i$ which means monotonic behaviour is followed and the loss is zero, else if $x_1^i < x_2^i$ and $\hat{y}_1^i > \hat{y}_2^i$ which means there is a non-monotonic relationship, and it is penalized and the ReLU part of the equation can be thought of as a weight i.e. magnitude of the error.

### 5.2 Point-wise Loss (PWL)

This method [4] enforces monotonicity by adding a term in the objective function to incorporate a priori knowledge regarding monotonicity. The minimax objective function is defined as follows:

$$min\ \mathcal{L}_{\text{mono}} = min\left\{\sum_{i=1}^{n} max(0, -\nabla_{\cdot \text{M}} f(x_{\text{i}}; \theta)) + \mathcal{L}_{\text{NN}}\right\} \tag{14}$$

where $f(x)$ is an estimator function, D is the feature set, $\nabla_{\cdot \text{M}} = \sum_j \frac{\partial f(x_{\text{i}};\theta)}{\partial \text{x[M]}_{\text{i}}^j}\ \forall j \in M$ is the divergence with respect to the feature set x[M] (M $\subseteq$ D), $\theta$ are the parameters that can be trained, $\mathcal{L}_{\text{NN}}$ is the loss function for NN, $\text{x}_\text{i}$ is the $\text{i}^{\text{th}}$ input vector in the training set $\{(\text{x}_\text{i}, \text{y}_\text{i})\}$ and n is the number of training examples in that set.

This method provides flexibility i.e. it can be used on any NN, DNN architecture and with any loss function $\mathcal{L}_{\text{NN}}$ which makes it easier to implement it on existing architectures and thus incorporating monotonicity. Note that no monotonicity test is used to choose the subset of monotonic features M from D, rather monotonic input features are chosen from D considering the domain knowledge.

### 5.3 Certified Monotonic Neural Network(s) (CMNN)

A Certified Monotonic Neural Network(s) (CMNN) [9] is a NN/DNN which is guaranteed to be monotonic in contrast to other methods like PWL [4] as discussed in [9]. This method uses a monotonicity regularization (penalty for violating monotonicity) with the NN loss in the loss function which provides flexibility and guaranteed monotonicity. The algorithm alternates between the following two steps until it satisfies the global monotonicity verification as in Subsection 3.2.

**Step 1** The first step is to train a NN $f$ with the following loss function:

$$\min_f \mathcal{L}(f) + \lambda R(f)\ \ where\ \ R(f) = \mathbb{E}_{x \sim Uni(\mathcal{X})}\left[\sum_{l \in \alpha} max(0, -\partial_{x_l} f(x))^2\right] \tag{15}$$

where $\mathcal{L}(f)$ is the NN training loss, $R(f)$ is a penalty for violating monotonicity, $\lambda$ is the weight for regularization, $Uni(\mathcal{X})$ is a uniform distribution on $\mathcal{X}$. $R(f) = 0$ means that $f$ is monotonic with respect to the monotonic features $x_\alpha$ in the dataset.

$R(f)$ is approximated with by uniformly drawing samples of size 1024 over the input domain during each iteration of gradient descent and samples drawn differ in every iteration. Practically, this algorithm uses a modified version of equation 15 which introduces a small valued positive constant b instead of 0 in the max function which makes it $R(f) = \mathbb{E}_{x \sim Uni(\mathcal{X})} \left[ \sum_{l \in \alpha} max(b, -\partial x_l f(x))^2 \right]$. This version is used because if equation 15 is used it gives the value of $U_\alpha$ slightly smaller than zero.

**Step 2** This step calculates the value/lower bounds $U_\alpha$ to verify the global monotonicity ($U_\alpha \geq 0$). If this condition is satisfied then the algorithm terminates, else increase $\lambda$ and the algorithm goes back to Step 1.

**Global Monotonicity Verification for DNN** Consider, $f : \mathcal{X} \rightarrow R$ is a deep ReLU network with even number of layers 2K. If the DNN has an odd number of layers, an identity layer can be added at the top of the network, and its weights can be fixed during training. For DNN, the global monotonicity verification can be extended by considering DNN as a combination of two layer networks:

$$f(x) = f_{\text{2K:2K-1}} \circ ..... \circ f_{4:3} \circ f_{2:1}(x) \tag{16}$$

where $f_{\text{2K:2K-1}}$ denotes a two layer network of $2\text{K}^{\text{th}}$ and $(\text{2K-1})^{\text{th}}$ layers of $f$. Therefore, a condition for global monotonicity verification for DNN can be formulated as $f_{\text{2K:2K-1}}, \forall k = 1, ...., K$ in which each block is monotonic.

## 6 Summary & Conclusion

In general, monotonicity increases the interpretability of the black box NN/DNN models and makes the learner biased towards specific hypothesis in the hypothesis space. As discussed before, monotonicity should only be used if there exists a total/partial monotonic relationship between the input features and output variable because it biases the learner which might lead to poor model quality. To determine the monotonic relationships, the tests described in Section 3 or the domain knowledge can be used. The techniques for enforcing monotonicity are summarized in the Table 1. Due to limited horizontal space the table uses vertical space instead and abbreviation is used which is Mono. stands for Monotonicity.

The best state-of-the-art technique to enforce monotonicity is CMNN [9] because of the flexibility of this technique which means it can be applied to any existing NN/DNN architectures by just changing the loss function, global monotonicity verification to certify that the output NN/DNN is monotone. The output NN/DNN from this technique also have the following properties: fairness, interpretability [9]. Finally, Liu et al. (2020) [9] also compare their technique with other state-of-the-art techniques which includes MMMN [3] and DLN [6], and it performs the best on four datasets used for evaluation.

**Table 1.** Summary and comparison of state-of-the-art techniques for enforcing monotonicity as discussed in Sections 4, 5

| | **MMMN [3], [5] 4.1 (1998,2010)** | **DLN [6] 4.2 (2017)** | **PWL [4] 5.2 (2019)** | **CMNN [9] 5.3 (2020)** | **DANN [10] 5.1 (2018)** |
|---|---|---|---|---|---|
| **Category** | CNA 2.1 | CNA 2.1 | PNM 2.2 | PNM 2.2 | PNM 2.2 |
| **How is Mono. enforced?** | Positive weight constraints, MIN and MAX layers | Linear Embedding, Calibration & Ensemble of Lattice layers | Loss function by penalizing negative gradients | 2 steps, Loss Function + Global Mono. Verification | Mono. constraint in the loss function |
| **Test for Mono.** | Yes, Subsection 3.1 | No | No | Yes, Subsection 3.2 | No |
| **Mono. Guarantee** | Yes | Yes | No | Yes | No |
| **Can the technique be applied to existing NN/DNN architectures?** | No | No | Yes | Yes | Yes |

## References


1. Toptal, https://www.toptal.com/machine-learning/monotonic-ai-models
2. Wikipedia, https://en.wikipedia.org/wiki/Artificial_neural_network
3. Sill, J.: Monotonic Networks. In: Jordan M., Kearns M., and Solla S. (eds.), Advances in Neural Information Processing Systems 10, pages 661–667. MIT Press (1998). http://papers.nips.cc/paper/1358-monotonic-networks.pdf
4. Gupta, A., Shukla, N., Marla, L., Kolbeinsson, A., Yellepeddi, K.: How to Incorporate Monotonicity in Deep Networks While Preserving Flexibility?. In: NeurIPS 2019 Workshop on Machine Learning with Guarantees, Vancouver, Canada (2019).
5. Daniels H., Velikova M.: Monotone and partially monotone neural networks. In: IEEE Transactions on Neural Networks, 21(6):906–917 (2010). ISSN 1045-9227. https://doi.org/10.1109/TNN.2010.2044803
6. You S., Ding D., Canini K., Pfeifer J., Gupta M.: Deep lattice networks and partial monotonic functions. In: Guyon I., Luxburg U. V., Bengio S., Wallach H., Fergus R., Vishwanathan S., Garnett R. (eds.), Advances in Neural Information Processing Systems 30, pages 2981–2989. Curran Associates, Inc. (2017). http://papers.nips.cc/paper/6891-deep-lattice-networks-and-partial-monotonic-functions.pdf
7. Sill, J., Abu-Mostafa Y.: Monotonicity Hints. In: Advances in Neural Information Processing Systems. No.9. MIT Press, Cambridge, MA, pp. 634-640. ISBN 0-262-10065-7 (1996).
8. Daniels H., Kamp B.: Application of MLP Networks to Bond Rating and House Pricing. In: Neural Comput. & Applic. 8, 226–234 (1999). https://doi.org/10.1007/s005210050025
9. Liu X., Han X., Zhang N., Liu Q.: Certified Monotonic Neural Networks. In: 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.
10. Muralidhar N., Islam M. R., Marwah M., Karpatne A., Ramakrishnan N.: Incorporating Prior Domain Knowledge into Deep Neural Networks. In: IEEE International Conference on Big Data (2018). https://doi.org/10.1109/BigData.2018.8621955
11. He K., Zhang X., Ren S., Sun J.: Deep Residual Learning for Image Recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016). https://doi.org/10.1109/CVPR.2016.90
12. Heymann J., Drude L., Haeb-Umbach R.: Wide Residual BLSTM Network with Discriminative Speaker Adaptation for Robust Speech Recognition. Computer Speech and Language (2016).