Partially Monotonic Learning for Neural Networks

Joana Trindade[1], João Vinagre[1,2], Kelwin Fernandes[3], Nuno Paiva[4], and Alípio Jorge[1,2]

[1] Faculdade de Ciências, Universidade do Porto, Portugal
joana_trindade_55@hotmail.com, amjorge@fc.up.pt
[2] LIAAD - INESC TEC, Porto, Portugal
jnsilva@inesctec.pt
[3] NILG.AI, Porto, Portugal
kelwin@nilg.ai
[4] NOS Comunicações, S.A, Portugal
nuno.paiva@nos.pt

Abstract. In the past decade, we have witnessed the widespread adoption of Deep Neural Networks (DNNs) in several Machine Learning tasks. However, in many critical domains, such as healthcare, finance, or law enforcement, transparency is crucial. In particular, the lack of ability to conform with prior knowledge greatly affects the trustworthiness of predictive models. This paper contributes to the trustworthiness of DNNs by promoting monotonicity. We develop a multi-layer learning architecture that handles a subset of features in a dataset that, according to prior knowledge, have a monotonic relation with the response variable. We use two alternative approaches: (i) imposing constraints on the model's parameters, and (ii) applying an additional component to the loss function that penalises non-monotonic gradients. Our method is evaluated on classification and regression tasks using two datasets. Our model is able to conform to known monotonic relations, improving trustworthiness in decision making, while simultaneously maintaining small and controllable degradation in predictive ability.

Keywords: Interpretability · Deep neural networks · Monotonicity

1 Introduction

In the past few years, the Artificial Intelligence (AI) research community has identified the problem of trustworthiness in Machine Learning (ML) models. Black-box models, such as Deep Neural Networks (DNN) and ensemble models, can achieve high predictive performance, but the complexity of their structure and internal computations are hard to interpret and explain. As a response to this problem, multiple lines of research on interpretability, mostly under the explainable AI (XAI) has become central. One obvious way to boost trustworthiness is to use intrinsically more interpretable models, such as rule- or tree-based models. However, black-box models consistently outperform simpler methods in several tasks (e.g., computer vision, language models, and many standard tabular datasets). Besides, simpler models are limited in some level of interpretation and it is not guaranteed that these models always improve interpretability [7].

There are two paths to improve the interpretability of ML models [4]: incorporating prior knowledge in the learning process, also known as *in-model* methods; or using *post-hoc* methods that aim to provide intuitive explanations based on the output generated by the model. The post-hoc approach is the most common for interpreting black-box models. However, according to Rudin [11], the use of post-hoc methods perpetuates a bad practice, which can cause damage to society in high-risk scenarios and, therefore, the in-model approach is more adequate to ensure trustworthy AI [14]. The in-model approach imposes constraints based on domain knowledge, using different types of reasoning: rules, cases, sparsity, or monotonicity. The best technique, or combination of techniques, depends on the application and all of them have some form of trade-off with predictive ability.

In this work, we address interpretability in DNNs by promoting monotonicity, while minimising degradation of predictive ability. We know that monotonic relations (i.e., only vary in one direction) between independent variables and a learned objective function exist in many ML problems, particularly in business contexts. For example, we know that increasing the price of some product or service, without any other change, will very likely reduce sales. Doctors know that higher bad cholesterol levels increase the risk of stroke. However, ML algorithms tend to create non-linear, non-monotonic, and even non-continuous functions to approximate the relations between variables. Even if ML models that do not consider monotonic properties are accurate in most predictions, often provide others that are quite obviously wrong. In critical applications, the consequences can be disastrous. By imposing monotonicity, we can leverage knowledge to obtains more accurate, robust, and trustworthy ML models of the data considered [3].

Our main contributions are the following:

- We propose a generally applicable learning framework to train semi-monotonic neural networks with a loss function that induces monotonicity;
- We conduct an evaluation of existing approaches to monotonic neural networks on two distinct problems: classification and regression.

The remainder of the paper is structured as follows: Section 2 presents related work. Section 3 focuses on fundamental concepts and Section 4 describes the proposed approach. Section 5 provides details about the datasets, our experimental methodology, the obtained results, and discusses these results. Finally, Section 6 concludes this work and addresses future research directions.

2 Related Work

According to [3], one of the taxonomies for monotonic algorithms is based on the generation of predictive models satisfying the monotonic constraints partially or totally. There are several families of predictors depending on the type of model they build, for instance, decision trees or rule-based models, ensembles, support vector machines, and DNN. In the latter, we can find two approaches in the literature to impose monotonicity: (1) by imposing hard constraints on the model's structure and/or parameters, or (2) by applying soft constraints in the training process, e.g. by penalising non-monotonic behaviour. Structural modification in DNNs was first proposed by Archer and Wang [1], who used positive weight constraints. Sill [12] introduced these constraints into a three-layer neural network, performing maximum and minimum operations on groups of hyperplanes. The signals of the hyperplane weights are limited to simulate total monotonicity. Daniels and Velikova [5] extended some results obtained on MIN-MAX networks [12], using partially monotonic functions in low-dimension spaces. Zhu et al. [16] proposed a generalisation of extreme learning machines for monotonic classification. The proposal involves a quadratic programming problem. You et al. [15] focused on the challenge of learning partially monotonic flexible functions. For this, they developed Deep Lattice Networks, that alternate between three types of layers: linear embeddings, calibrators, and lattice ensembles. Silva et al. [13] developed an interpretable DNN able to generate explanations in different styles and granularities, according to the preferences of the decision-maker. Recently, Nguyen et al. [9] developed a DNN architecture called MonoNet. It learns high-level arbitrary features that are monotonically related to the target variable. For each feature, MonoNet learns a score of importance obtained by exploiting "local explainability". Márquez-Neila et al. [8] compare soft with hard constraints and argue that soft constraints perform better. Pathak et al. [10] formulate the loss function to optimise convolutional networks with arbitrary linear constraints on the structured output space of image pixel labels. Gupta et al. [6] introduce a gradient-based point-wise loss function to impose partial monotonicity on any DNN without changing the network architecture.

Our approach combines the model architecture similar to [13] to promote partial monotonicity, and a learning method to train semi-monotonic neural networks, based on [6]. We further develop this approach for dealing with arbitrary scales when choosing the desired trade-off. We also compare hard and soft constraints in real-world datasets with partially monotonic feature spaces.

3 Monotonicity

As in [9], we assume that f is a predictor that operates on vectors $\mathbf{x} \in \mathbb{R}^n$ of $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ features in the dataset such that $\mathbf{y} = f(\mathbf{x})$. The monotonicity analysis for a given independent feature x_i is easily performed: we generate a new vector **x'** by introducing a disturbance x'_i, where $x'_i = x_i + \Delta$ and $\Delta \geq 0$, in the independent feature's domain, and fixing the remaining independent features. We observe its behaviour with the target feature.

Definition 1. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is called* ***monotonically increasing*** *w.r.t. x_i if $x_i \leq x'_i$ implies $f(\boldsymbol{x}) \leq f(\boldsymbol{x'})$. Equivalently, if $x_i \leq x'_i$, it is required to check, for all $i = \{1, ..., n\}$, the (univariate) constraint $f|_i : x_i \mapsto y$:*

$$f(x_1, x_2, ..., x_i, ..., x_n) \leq f(x_1, x_2, ..., x'_i, ..., x_n) \ . \tag{1}$$

by fixing all the components except the x_i, in the usual sense of monotonicity for univariate functions.

The definition is similar for *monotonically decreasing functions*. If the function does not respect the conditions, then it is called *non-monotonic function*. According to [2], there are two main classes to distinguish monotonic problems. The contrast is based on the set of independent features on which the function f depends monotonically. Thus, monotonicity can be classified as *total* or *partial*.

Total Monotonicity. *The function f depends monotonically on all independent features of the dataset. We assume that y in the dataset is generated by:*

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon, \tag{2}$$

where f is a monotonic function and ϵ is a random error.

The total monotonicity constraint of f on $\mathbf{x}$ is defined according to Definition 1, for all independent features x_i, for any i.

Partial Monotonicity. *The function f depends monotonically on a subset of features of the dataset. In partially monotonic problems, the total set of features $\boldsymbol{x}$ is separated into $\boldsymbol{x}^m$ and $\boldsymbol{x}^n$ subsets to represent, respectively, the monotonic and non-monotonic feature subsets. Therefore, the dataset $\mathcal{D} = (\boldsymbol{x}^m, \boldsymbol{x}^n, \boldsymbol{y})$, where $\boldsymbol{x} = (\boldsymbol{x}^m, \boldsymbol{x}^n)$. We assume that the target $\boldsymbol{y}$ in the dataset is generated by:*

$$\boldsymbol{y} = f(\boldsymbol{x}^m, \boldsymbol{x}^n) + \epsilon, \tag{3}$$

where f is a monotonic function in $\boldsymbol{x}^m$ and ϵ is a random error.

The partial monotonicity constraint of f on $\mathbf{x}$ is defined according to Definition 1, for independent features in $\mathbf{x}^m$. Note that although f is monotonic, the data generated by Eq. (2) and (3) are not necessarily monotonic due to the random effect of ϵ.

4 Partially Monotonic Learning

In most real-world datasets, the target feature depends monotonically on a subset of features, but not on all of them. However, fully monotonic models cannot model non-monotonic features. To enable partial monotonicity, we build an architecture based on the proposal by [13]. This architecture consists of two independent neuronal subnets that process separately the monotonic and non-monotonic features. The monotonicity constraints are applied on the monotonic stream in which, without loss of generality, we assume that the probability of observing the positive class increases with the the value of the monotonic features. The sign of the input monotonic features that have a monotonically decreasing behaviour is inverted to admit monotonically increasing relations. Then, both subnets are concatenated and processed again by a sequence dense layers with monotonicity constraints (Fig. 1). The unconstrained subnet maps its feature space into a latent monotonic space, requiring additional parameters to learn complex patterns.

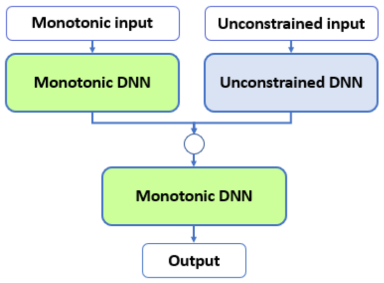


Fig. 1. Proposed architecture. Schema based on [13].

4.1 Loss Function

Using the architecture in Fig. 1, we can easily model monotonicity using hard constraints. Our proposal is an approach that uses this architecture with soft-constraints, using a modified loss function that penalises non-monotonic behaviour.

As in [6], we assume the general configuration of a supervised learning problem with a training set with k instances composed by $\mathcal{D} = (\mathbf{x}, \mathbf{y})$, where $\mathbf{x} = (\mathbf{x}^n, \mathbf{x}^m)$. The label could either be real-valued, $\mathbf{y} \in \mathbb{R}$, or binary, $\mathbf{y} \in \{0, 1\}$. The objective is to determine an estimator function f which is differentiable and monotonic w.r.t. $\mathbf{x}^m = \mathbf{x}[M]$, where M is a subset of monotonic features defined by $M \subseteq \mathcal{D}$ in $\mathbf{x} \in \mathbb{R}^{\mathcal{D}}$. We only consider increasing monotonicity. Thus, the objective function $\mathcal{L}$ combines a monotonic loss component, $\mathcal{L}_{mono}$, and a standard empirical loss component, $\mathcal{L}_{NN}$. Computed on each monotonic feature x_i^m, for $i = \{1, ..., k\}$, Eq. (4) describes a objective function through the sum of these components in the following form:

$$\mathcal{L} = p \cdot s \cdot \underbrace{\left(\sum_{i=1}^{k} max\left(0, -\nabla \cdot_M f(x_i^m; \theta)\right) \right)}_{\mathcal{L}_{mono}} + (1 - p) \cdot \mathcal{L}_{NN} \ . \tag{4}$$

The hyperparameter p is a monotonicity weight, s is a scale adjustment factor, $\nabla \cdot_M$ is divergence w.r.t feature set $\mathbf{x}[M]$ i.e., $\sum_j \frac{\partial f(x_i^m; \theta)}{\partial x[M]_i^j} \forall j \in M$, θ are trainable parameters and $\mathcal{L}_{NN}$ refers to the empirical risk minimisation (ERM) for neural networks.

During each gradient descent step, the maximum of monotonic component penalises only trends of monotonically decreasing gradients. In this case, the minus sign of a negative partial derivative of $f(x_i^m; \theta)$ w.r.t. the monotonic features of $\mathbf{x}[M]$ results in a positive value, indicating that the maximum between zero and that value, results a gradient penalty. As for gradients that obey monotonicity, the maximum in the first component is zero and, hence, there are no penalties on gradients.

Hyperparameter p is a simple weight factor to control the relative contribution of each of the loss components. With $p = 1$, the model adjusts for monotonicity only, whereas with $p = 0$, the model does not care about monotonic relations at all.

The scale adjustment factor s aims to balance between the scales of the two optimisation components. A dominant loss component induces its dominance through gradients with large magnitudes, which prevent the harmonisation of the training stage, making the definition of p harder. We propose multiplying the magnitude order of the majority component by the minority component. To do this, we calculate the quotient r between the averages of the loss components $\mathcal{L}_{NN}$ and $\mathcal{L}_{mono}$ in each epoch. The result r denotes the amount of necessary adjustment to balance the components. To prevent both tasks from making exactly equal contributions – that would artificially flatten differences through successive epochs –, we assume the quotient magnitude order r as a re-scaling factor s. We calculate s using 10 to the power of $\lfloor \log_{10}(r) \rfloor$. All the necessary quantities above are calculated according to (5).

$$m_{NN} = \overline{\mathcal{L}_{NN}} \qquad m_{mono} = \overline{\mathcal{L}_{mono}} \qquad r = \frac{m_{NN}}{m_{mono}} \qquad s = 10^{\lfloor \log_{10}(r) \rfloor} \,. \tag{5}$$

5 Evaluation

To evaluate our proposal, we assess the effect of monotonic component on the predictive ability of classification and regression problems. The first problem arises from the recommendation of service upgrades to customers of a large telecom service provider. The problem consists of predicting whether a specific customer will accept or refuse a personalised upgrade offer. We refer to this dataset as *Dataset 1*. The second problem consists of predicting the price sales of used cars. We refer to this dataset as *Dataset 2*.

In both problems, monotonic features are previously known. In the classification case, we know, for instance, that between two offers A and B with the exact same conditions, but where the price of A is higher than the price of B, it is very unlikely that the customer will prefer A over B. The same applies to other not so obvious features, such as internet speed or call limits. In the car price prediction task, features such as mileage and cylinders are also likely to follow a monotonic relation with the target feature (i.e., sale price).

5.1 Datasets

Table 1 summarises the details of the two datasets.

For the first task, we use a proprietary dataset, collected between January 8th and November 30th, 2018, which contains the history of offers made to customers, as well as whether the customer accepted the offer. Each transaction contains the features of the current subscription, as well as the features of the offered upgrade, plus some previously engineered features. For the second task

Table 1. Characteristics of each dataset.

Characteristics	Dataset 1	Dataset 2
Nº of instances	> 574 000	401 204
Nº of features	328	17
Nº of monotonic features	28	6
Data accessibility	Proprietary	Public
Data Context	Telecom operator	Car sales
Task	Classification	Regression

– used car price prediction –, we use the available dataset from Kaggle[5]. It contains information about the condition of use, manufacturer and model, year of manufacture, odometer, and other categories.

5.2 Methodology

We randomly split both datasets into 80%–20% for training and testing, respectively. The training set is further divided into 80%–20% for training and validation. Dataset 1 is ready to use, however, Dataset 2 requires some preprocessing tasks. First, several categorical features, such as links to web pages, manufacturer and model are ignored, and some other categorical features (e.g. size, condition) are transformed into numerical features. For features with missing values, we apply the iterative imputation strategy[6]. Some noisy instances are also manually removed, e.g., data whose year of manufacture is greater than 2020 or fictitious prices below 100 and over 300,000 dollars. The `year` feature – the manufacturing year – requires special attention, given that the monotonicity signal of the feature is inverted once a car becomes a "classic". We assume that a vehicle becomes a classic when it turns 25 years from its manufacturing year. We translate `year` into two new features to reflect this: `classic_years` and `modern_years`, which means, respectively, the number of years after becoming a classic and the vehicle's age until it reaches 25 years. The reference year is 2020. Thus, the `classic_years` and `modern_years` features are monotonically increasing and decreasing with the sale price, respectively.

For weight initialisation, we consider a uniform distribution, whose lower limit takes non-negative values. The activation function for intermediate layers is LeakyReLU, to avoid the "Dying ReLU" problem. The loss function for the regression problem is Mean Squared Logarithmic Error (MSLE), defined as $\mathcal{L}_{NN}(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}\left(log(y_i + 1) - log(\hat{y}_i + 1)\right)^2$. Unlike MSE and MAE, MSLE minimises the penalising effect of high differences in the predicted values. To improve and stabilise the training of the models, we use regularisation with dropout and batch normalisation (BN) layers. The BN layers are fundamental for the hard constraints model because it allows to soften the constraints and guarantee the power of representation of the objective function. The optimal

[5] https://kaggle.com/austinreese/craigslist-carstrucks-data
[6] https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html

set of hyperparameters for the network architecture were obtained using Grid Search.

5.3 Monotonic Features Extraction

In Dataset 1, we define as monotonic features those referring to the recommended service. As for Dataset 2, we select the subset of features that keep a monotonic relation with the sales price. The features for monotonicity analysis are grouped as follows[7]:

– Monotonically increasing features: `condition`, `size`, `classic_years` and `cylinders`.
– Monotonically decreasing features: `modern_years` and `odometer`.
– Without monotonic constraints: remaining features in the dataset.

5.4 Models

We simulate the two approaches to enforce monotonicity presented in Section 1. For the first approach, we rely on Silva et al. [13] to implement the model's parameter constraints. The second approach is based on Gupta et al. [6] to change the learning process. Both models are compared against an baseline *Unconstrained Model* (UM), whose model architecture is described by Fig. 1. The monotonic models are, respectively:

– *Monotonic Model* (MM) with two input layers with constraints on the network weights.
– *Partial Monotonic Model* (PMM) with two input layers with an adapted loss function.

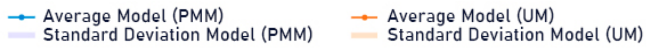
5.5 Monotonicity Analysis

This section presents the results of three experiments: monotonicity evaluation, predictive performance evaluation and impact of hyperparameter p.

Monotonicity Evaluation. To evaluate features monotonicity, we present a comparative analysis between Models UM and PMM in Figures 2-5. Recall that the only difference between these two models is the modified loss function. For each dataset, we chose a feature from the monotonic subset to perform the monotonicity analysis of the models: `rsp_tens` feature represents the difference in ten units in the RSP (retail selling price) for Dataset 1; and `classic_years` for Dataset 2.

Figures 2 and 4 show the average and standard deviation (SD) of the predictions of each model, after adding a Δ value. It allows checking which direction the

[7] In most real-world problems, including the ones illustrated in this paper, domain expertise is essential to distinguish between true and spurious monotonic relations.



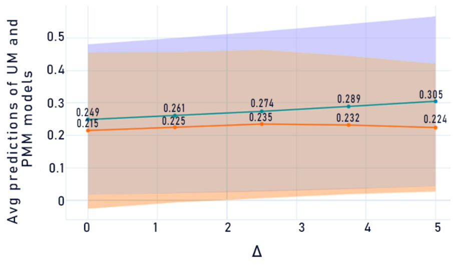


Fig. 2. Relation of Δ in average and SD predictions of feature `rsp_tens` (Dataset 1).

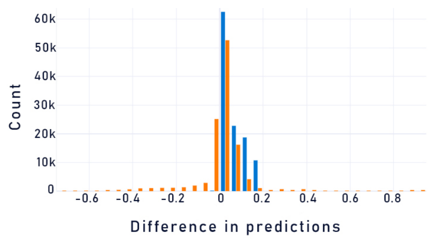


Fig. 3. Distribution of average predictions of feature `rsp_tens` (Dataset 1).

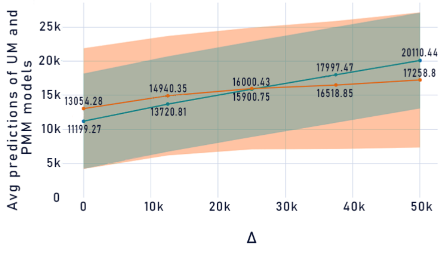


Fig. 4. Relation of Δ in average and SD predictions of `classic_years` (Dataset 2).

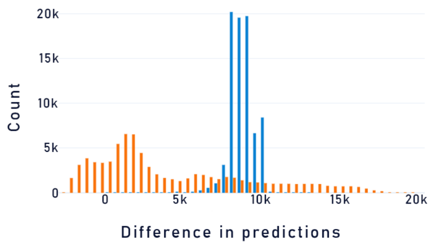


Fig. 5. Distribution of average predictions of `classic_years` (Dataset 2).

target feature takes as a gradual increase in the values of a given feature occurs. Figures 3 and 5 indicate the predictions distribution of each model, after adding a Δ value. In Model PMM, `rsp_tens` has an increasing monotonic function with propensity (see Fig. 2). It means that propensity tends to increase when offer's RSP is discounted one or more tens. By contrast, we can see an inflexion point ($\Delta = 2.5$) in curve of Model UM. Thus, this model cannot recognise a monotonic function with propensity. Model UM contains around 43.6% of non-monotonic cases, while Model PMM includes only 0.02% (Fig. 3). In Dataset 2, both models learn that `classic_years` and sales price relation is a monotonically increasing function (Fig. 4). This means that the older a classic car is, the more the car is valued. Model UM records 19% of non-monotonic examples, while Model PMM does not cover any non-monotonic example (Fig. 5).

Predictive Performance Evaluation. The evaluation metrics for the classification problem are AUC-ROC (Area Under the ROC Curve), AUC-PR (Area Under the Precision-Recall Curve) and global lift (value obtained in the first decile) – higher values are better. For regression problem, the evaluation metrics are MSE (Mean Squared Error) and MAE (Mean Absolute Error) – lower values are better. For Model PMM, we assumed a monotonicity weight equal to 10%. Table 2 shows the results of the predictive performance obtained for each model, regarding Datasets 1 and 2, described in section 5.1.

Table 2. Summary of the values obtained from used evaluation metrics to each model.

	Dataset 1 (classification)			Dataset 2 (Regression)	
	AUC-ROC	Global lift	AUC-PR	MAE	MSE
UM	**0.849**	**3.70**	**0.689**	4802	7.68×10^7
MM	0.834	3.468	0.651	**4462**	$\mathbf{6.95 \times 10^7}$
PMM ($p = 10\%$)	0.837	3.430	0.653	4853	7.85×10^7

In Dataset 1, Model UM has the best performance but, in Dataset 2, it is second worse. In contrast, Model MM has the best predictive performance values achieved in Dataset 2. However, in Dataset 1, it is the worst performing model in AUC-ROC and AUC-PR. Model PMM is the worst performing model according to all metrics, but nevertheless with relatively small degradation. In the classification task, Models MM and PMM have very similar results, while with the regression task, Model PMM is closer to Model UM.

Impact of Hyperparameter *p*. Hyperparameter p controls the relative importance of the empirical and monotonic loss components in Model PMM. We observe that imposing monotonicity causes a performance degradation (see comparison between Models UM and PMM in Table 2). Thus, we study the influence of p weight on model's predictive ability, in Dataset 1. Figure 6 describes the results of the global lift metric, assuming a set of monotonicity weights with spaced values of 10%. Figure 7 shows, for a subset of features, the proportion of instances that do not respect the monotonicity constraints with increasing p.

In general, the increase in the weight of the monotonicity component leads to a degradation of global lift (see Fig. 6). When p reaches 100%, the empirical loss minimisation is cancelled, and we are basically training a random model with rigid monotonic constraints. All features indicate that very few instances have non-monotonic cases when $p \geq 10\%$ (see Fig. 7). We also note that $p = 10\%$ already allows to obtain, simultaneously, accurate and monotonic results.

Discussion. The trade-off between monotonicity and predictive performance depends essentially on hyperparameter p. In Fig. 6, we see that p can influence the learning curve $\mathcal{L}_{NN}$ and hence, the learning curve $\mathcal{L}$. The higher the weighting of the monotonicity component, the more evident this relation becomes, showing that there is an obvious trade-off between the loss components.

Results in section 5.5 confirm that isolating the monotonic features from the others to enforce monotonicity are not enough to limit the model learning

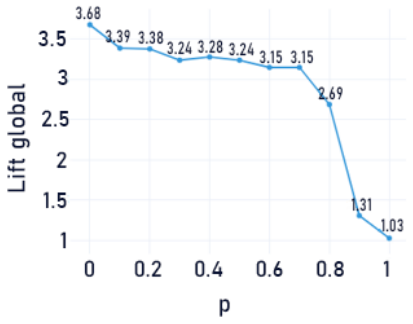


Fig. 6. Relation between monotonicity weight and global lift in Model PMM (Dataset 1).

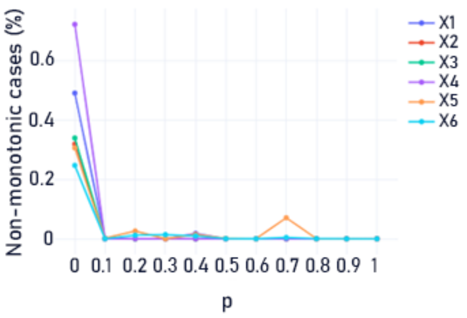


Fig. 7. Relation between monotonicity weight and proportion of non-monotonic cases in Model PMM (Dataset 1).

(see Fig. 2 and 4). Although the usage of monotonicity constraints confirms a substantial refinement in the integration of domain knowledge in black-box models. Besides, we found that the imposition of these constraints does not considerably deteriorate the predictive performance.

6 Conclusion and Future Work

In this work, we present a ML model architecture that integrates an additional penalty for learning monotonic relations from a predefined subset of features. The results show that monotonicity can promote or ensure, in the best case, the monotonic function of relevant features to the predictions, without considerably hurting predictive ability. Thus, the application of monotonicity constraints is a solution to connect domain knowledge in the learning process. Steps for future work include: testing for more use-cases, especially on smaller datasets; the research of scale adjustment techniques for a more robust balance between loss components; and the addition of non-compensatory decision-making strategy to our method.

Acknowledgements. This work is partially developed within project AIDA - Adaptive, Intelligent and Distributed Assurance Platform (reference POCI-01-0247-FEDER-045907) co-financed by the ERDF - European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 and by the Portuguese Foundation for Science and Technology – FCT, under CMU Portugal.

References

1. Archer, N.P., 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)
2. Bartolj, T.: Testing Monotonicity of Variables. Master's thesis, Faculty of Economics and Business Administration, Tilburg University (2010)
3. Cano, J.R., Gutiérrez, P.A., Krawczyk, B., Woźniak, M., García, S.: Monotonic classification: An overview on algorithms, performance measures and data sets. Neurocomputing **341**, 168–182 (2019)
4. Carvalho, D.V., Pereira, E.M., Cardoso, J.S.: Machine learning interpretability: A survey on methods and metrics. Electronics **8**(8), 832 (2019)
5. Daniels, H., Velikova, M.: Monotone and partially monotone neural networks. IEEE Transactions on Neural Networks **21**(6), 906–917 (2010)
6. Gupta, A., Shukla, N., Marla, L., Kolbeinsson, A., Yellepeddi, K.: How to incorporate monotonicity in deep networks while preserving flexibility? arXiv preprint arXiv:1909.10662 (2019)
7. Lipton, Z.C.: The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery. Queue **16**(3), 31–57 (2018)
8. Márquez-Neila, P., Salzmann, M., Fua, P.: Imposing hard constraints on deep networks: Promises and limitations. arXiv preprint arXiv:1706.02025 (2017)
9. Nguyen, A.p., Martínez, M.R.: Mononet: Towards interpretable models by learning monotonic features. arXiv preprint arXiv:1909.13611 (2019)
10. Pathak, D., Krahenbuhl, P., Darrell, T.: Constrained convolutional neural networks for weakly supervised segmentation. In: Proceedings of the IEEE international conference on computer vision. pp. 1796–1804 (2015)
11. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence **1**(5), 206–215 (2019)
12. Sill, J.: Monotonic networks. Advances in neural information processing systems **10**, 661–667 (1997)
13. Silva, W., Fernandes, K., Cardoso, M.J., Cardoso, J.S.: Towards complementary explanations using deep neural networks. In: Understanding and Interpreting Machine Learning in Medical Image Computing Applications, pp. 133–140. Springer (2018)
14. Toreini, E., Aitken, M., Coopamootoo, K., Elliott, K., Zelaya, C.G., van Moorsel, A.: The relationship between trust in ai and trustworthy machine learning technologies. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency. pp. 272–283 (2020)
15. You, S., Ding, D., Canini, K., Pfeifer, J., Gupta, M.: Deep lattice networks and partial monotonic functions. In: Advances in neural information processing systems. pp. 2981–2989 (2017)
16. Zhu, H., Tsang, E.C., Wang, X.Z., Ashfaq, R.A.R.: Monotonic classification extreme learning machine. Neurocomputing **225**, 205–213 (2017)