Not too close and not too far: enforcing monotonicity requires penalizing the right points

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this work, we propose a practical scheme to enforce monotonicity in neural networks with respect to a given subset of the dimensions of the input space. The proposed approach focuses on the setting where point-wise gradient penalties are used as a soft constraint alongside the empirical risk during training. Our results indicate that the choice of the points employed for computing such a penalty defines the regions of the input space where the desired property is satisfied. As such, previous methods result in models that are monotonic either only at the boundaries of the input space or in the small volume where training data lies. Given this, we propose an alternative approach that uses pairs of training instances and random points to create mixtures of points that lie inside and outside of the convex hull of the training sample. Empirical evaluation carried out using different datasets show that the proposed approach yields predictors that are monotonic in a larger volume of the space compared to previous methods. Our approach does not introduce relevant computational overhead leading to an efficient procedure that consistently achieves the best performance amongst all alternatives.

1 Introduction

Highly expressive model classes such as artificial neural networks, trained under the empirical risk minimization framework, have achieved impressive prediction performance across a broad range of supervised learning tasks and domains such as object recognition [12], speech recognition [9], machine translation [2], among many other examples. However, finding predictors attaining low risk on unseen data is often not enough to enable the use of such models in practice. In fact, practical applications usually have more requirements beside the prediction accuracy. As such, devising approaches that search risk minimizers satisfying practical needs led to several research threads seeking to enable the use of neural networks in *real-life* scenarios. Examples of such requirements include:

- *Robustness*, where low risk is expected even if the model is evaluated under distribution shifts or adversarial perturbations.
- *Fairness*, where the performance of the model is expected to not significantly change across different sub-populations of the data.
- *Explainability/Interpretability*, where models are expected to indicate how the features of the data affect the predictions of the model.

In addition to the requirements mentioned above, a property commonly expected in trained models in certain applications is *monotonicity* with respect to some subset of the input dimensions, i.e., an increase (or decrease) along some particular dimensions strictly imply the function value will not

Submitted to the eXplainable AI approaches for debugging and diagnosis workshop (NeurIPS 2021).

decrease (or will not increase), provided that all other dimensions are kept fixed. As a result, the behavior of monotonic models will be more aligned with properties that the data under consideration is believed to satisfy.

For example, in the case of models used to accept/reject job applications, we expect acceptance scores to be monotonically non-decreasing with respect to features such as past years of experience of a candidate. Thus, given two applicants with exactly the same features, if one of them has more years of experience, then they will have higher, or at least the same, chance of getting accepted. Similar observations can be made when applying machine learning models to other areas such as loan applications. For applications where monotonicity is expected, models failing to satisfy this requirement would damage the user's confidence. In addition, monotonicity was also observed to be a good inductive bias in order to improve generalization in cases where prior knowledge indicates the data generation process satisfies such property [5].

Given the importance of enforcing monotonicity properties in numerous applications, different strategies have been devised in order to enable learning of monotonic predictors. These approaches can be divided into two main categories:

- Monotonicity by construction: This approach focuses on defining a model class that guarantees monotonicity in all of its elements [3, 17, 14, 18, 7, 1]. However, this approach can not be used with general architectures in the case of neural networks. Additionally, the model class might be so constrained that it might affect the prediction performance.

- Encouraging monotonicity during model training: This approach searches for monotonic candidates within a general class of models [13, 15, 10]. Such group of methods is more generally applicable and can be used, for instance, with any neural network architecture. However, they are not guaranteed to yield monotonic predictors unless the monotonicity of the trained model is verified/certified, which can be computationally expensive. Moreover, several retraining cycles might be required to enforce monotonicity.

In this work, we focus on the latter and tackle the problem of performing empirical risk minimization in large classes of models such as neural networks, while simultaneously searching for monotonic predictors within the set of risk minimizing solutions. Specifically, we revisit methods that introduce monotonicity-enforcing regularization penalties. Our contributions can be summarized as follows:

1. We show that previous methods can only satisfy monotonicity either near the training data or near the boundaries of the input domain. To the best of our knowledge, this is the first work that raises the issue of the effect of the points used in calculating the monotonicity constraint and proposes an efficient algorithm that tackles the problem.

2. We propose an efficient approach that, given the same budget, can enforce monotonicity in a larger volume of the input space compared to the previous methods in literature.

The remainder of this paper is organized as follows: in Section 2, we provide background and formally introduce the notion of partial monotonicity under consideration. The related literature is also discussed in this section. The proposed approach is defined in Section 3, where we also discuss and explain the issues in the prior methods upon which we improve. Empirical evaluation is performed in Section 4, while conclusions are summarized in Section 5 along with a discussion on follow-up work.

2 Background and related work

We start by defining the notion of *partial monotonicity* used throughout the paper. Consider the standard supervised learning setting where data instances are observed in pairs $x, y \sim \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}$ correspond to the input and output spaces, respectively. Further consider the function $f : \mathcal{X} \mapsto \mathcal{Y}$, and let M indicate some subset of the input dimensions, i.e. $M \subset \{1, ...d\}$, such that $x = \text{concat}(x_M, x_{\bar{M}})$, where $\bar{M} = \{1, ..., d\} \setminus M$. We further overload the notation indicating function calls to f such that $f(x) = f(x_M, x_{\bar{M}})$.

Definition *Partially monotonic functions relative to M:* We say f is monotonically non-decreasing relative to M[1], denoted f_M, if $f(x_M, x_{\bar{M}}) \leq f(x'_M, x_{\bar{M}}), \forall\, x_M \leq x'_M, \forall\, x_{\bar{M}}$, where the comparison $x_M \leq x'_M$ is performed for every dimension.

The above definition covers functions that do not decrease in value given increasing changes along a subset of the input dimensions, provided that all other dimensions are kept unchanged.

Several approaches were introduced for defining model classes that have such property. The simplest approach restricts the weights of the network to be non-negative [1]. However, doing so affects the prediction performance. Another approach corresponds to using the lattice regression models proposed in [7] and recently extended in [3, 18]. In such case, models are given by interpolations in a grid defined by training data. While such class of models can be made monotonic via the choice of the interpolation strategy, it scales poorly with the dimension of the input space.

For neural networks, approaches such as [14] reparameterizes fully connected layers such that gradients with respect to parameters can only be non-negative. In [17], on the other hand, the class of predictors $H : \mathcal{X} \mapsto \mathcal{Y}$ of the form $H(x) = \int_0^x h(t)dt + H(0)$ is defined, where $h(t)$ is parameterized by a neural network. Given that $h(t)$ is strictly positive, which can be easily enforced via the choice of the activation function, the resulting $H(x)$ will be montonically non-decreasing. While such approaches guarantee monotonicity by design, they can be too restrictive or render overly complicated learning procedures. For example, the approach in [17] requires performing backpropagation through the integration operator.

An alternative approach to that of constraining the model class is to search over general classes of models while assigning higher importance to candidate predictors that are observed to be monotonic in certain parts of the space. Similar to the case of adversarial training [8], Sivaraman et al. [15] proposed an approach that finds counterexamples, i.e. pair of points where the monotonicity constraint is violated. The counterexamples are then included in the training data with adjustments to their target values so as to enforce next iterates of the model will satisfy the monotonicity conditions for them. However, this approach only supports fully-connected ReLU networks. Moreover, the procedure for finding the counterexamples is costly. Alternatively, a regularization penalty is introduced in both Liu et al. [13] and Gupta et al. [10] so that monotonicity is point-wise enforced during training. This requires selecting the points where the penalty is computed during training. While Liu et al. [13] use random draws from a uniform distribution over $\mathcal{X}$, Gupta et al. [10] apply the regularization penalty over the training instances only. As will be further discussed, both approaches have shortcomings that we seek to address in this contribution.

3 Proposed approach

Given the standard supervised learning setting where $\ell\ : \mathcal{Y}^2 \mapsto \mathbb{R}^+$ is a loss function indicating the goodness of the predictions relative to ground truth targets, the goal is to find a predictor $h \in \mathcal{H}$ such that its expected loss – or the so-called *risk* – over the input space is minimized. Such an approach yields the empirical risk minimization framework once a finite sample is used to estimate the risk. However, given the extra monotonicity requirement, we consider an augmented framework where such property is further enforced. We thus seek the optimal monotonic predictors relative to M – denoted h^*_M – such that:

$$h^*_M \in \arg\min_{h \in \mathcal{H}}\ \mathbb{E}_{x,y \sim \mathcal{X} \times \mathcal{Y}}[\ell(h(x), y)] + \gamma \Omega(h, M), \tag{1}$$

where γ is a hyperparameter weighing the importance of the penalty $\Omega(h, M)$ which, in turn, is a measure of *how monotonic* the predictor h is relative to the dimensions indicated by M. $\Omega(h, M)$ can be defined by the following gradient penalty [10, 13]:

$$\Omega(h, M) = \mathbb{E}_{x \sim \mathcal{D}} \left[\sum_{i \in M} \max\left(0, -\frac{\partial h(x)}{\partial x_i}\right)^2 \right], \tag{2}$$

where $\frac{\partial h(x)}{\partial x_i}$ indicates the gradients of h relative to the input dimensions $i \in M$, which are constrained to be non-negative, rendering h monotonically non-decreasing relative to M.

[1] Monotonically non-increasing f can be defined analogously.

At this point, the only missing ingredient to define a practical algorithm to estimate h^*_M in Eq. 1 is how to define the distribution $\mathcal{D}$ over which the expectation in Eq. 2 is computed. In the following, first we discuss the choices of $\mathcal{D}$ in prior work and the consequential issues of such choices. Next, we propose alternatives to alleviate these issues.

3.1 Choosing distributions over which to compute Ω

Past choices for $\mathcal{D}$ are such that:

1. *Define $\mathcal{D}$ as the empirical distribution of the training sample*: In [10], given a training dataset of size N, in addition to using the observed data to estimate the risk, the same data is used to compute the monotonicity penalty so that:

$$\Omega_{train}(h, M) = \frac{1}{N}\sum_{k=1}^{N}\sum_{i\in M}\max\left(0, -\frac{\partial h(x^k)}{\partial x_i^k}\right)^2,$$

   where x^k indicates the k-th instance within the training sample. While this choice seems natural and can be easily implemented, it only enforces monotonicity in the region where the training samples lie, which can be problematic. For example, in case of covariate-shift, the test data might lie in parts of the space different from that of the training data so monotonicity cannot be guaranteed. We thus argue that one needs to enforce the monotonicity property in a region larger than what is defined by the training data. In Section 4.1, we conduct an evaluation under domain shift and show the issue to become more and more relevant with the increase in the dimension d of the input space $\mathcal{X}$.

2. *Define $\mathcal{D} = \text{Uniform}(\mathcal{X})$*: In [13], a simple strategy is defined so that Ω is computed over the random points drawn uniformly across the entire input space $\mathcal{X}$; i.e.:

$$\Omega_{random}(h, M) = \mathbb{E}_{x\sim\text{Uniform}(\mathcal{X})}\left[\sum_{i\in M}\max\left(0, -\frac{\partial h(x)}{\partial x_i}\right)^2\right].$$

   Despite its simplicity and ease of use, this approach has some flaws. In high-dimensional spaces, random draws from any distribution of bounded variance will likely lie in the boundaries of the space, hence far from the regions where data actually lie. Moreover, it is commonly observed that naturally occurring high-dimensional data is structured in lower-dimensional manifolds (c.f. [6] for an in-depth discussion on the manifold hypothesis). It is thus likely that random draws from the uniform distribution will lie nowhere near regions of space where training/testing data will be observed. We further illustrate the issue with an example in Section 3.2, which can be summarized as follows: consider the cases of uniform distributions over the unit n-sphere. In such case, the probability of a random draw lying closer to the sphere's surface than to its center is $P(||x||_2 > \frac{1}{2}) = \frac{2^n-1}{2^n}$, as given by the volume ratio of the two regions of interest. Note that $P(||x||_2 > \frac{1}{2}) \to 1$ as $n \to \infty$, which suggests the approach in [13] will only enforce monotonicity at the boundaries of the space.

In summary, the previous approaches are either too focused in enforcing monotonicity where the training data lie, or too loose such that the monotonicity property is uniformly enforced across a large space, and the actual data manifold may be neglected. We thus propose an alternative to such approaches where we can have some control over the volume of the input space where the monotonicity property will be enforced. Our approach uses the idea of data mixup [19], where auxiliary data is created via interpolations of pairs of data points.

3.2 Mixtures of points as a means to enlarge the region where Ω is computed

Mixup was introduced in [19] with the goal of training classifiers whose outputs are smooth across trajectories in the input space from instances of different classes. Given a pair of data points (x', y'), (x'', y''), the method augments the training data using interpolations given by $(\lambda x' + (1-\lambda)x'', \lambda y' + (1-\lambda)y'')$, where $\lambda \sim \text{Uniform}([0, 1])$. The approach was later extended in [16] so that the smoothness property is further imposed in high-level learned representations. Moreover, mixup is used to enforce fairness constraints in [4] by forcing training algorithms to give preference to predictors that behave smoothly in the space in between sub-populations of the data.

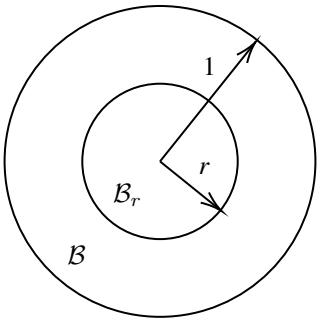


Figure 1: Illustration of unit spheres $\mathcal{B}$ and $\mathcal{B}_r$ on the plane.

In this work, we propose using mixup to generate points where the monotonicity penalty Ω can be computed and highlight the following motivations for doing so:

1. *Interpolation:* this is obtained by mixing-up pairs of data instances (i.e. when $0 < \lambda < 1$), which more densely populates the convex hull of the training data. We argue doing so is beneficial in helping to achieve monotonicity in the entirety of the hull rather than only on the training data itself as in [10].
2. *Extrapolation*: mixing up data points with boundary instances obtained at random from Uniform($\mathcal{X}$) enables extrapolation in the sense that resulting mixtures of points can lie anywhere between the data manifold and the boundaries of the space. This encourages monotonicity on points that are outside the hull made of the training sample.

We thus claim – and report empirical evidence – that performing mixup enables the computation of Ω on parts of the space that are disregarded if one focus only on either observed data or random draws from uninformed choices of distributions such as the uniform.

To further illustrate the issue discussed in the item 2 of Section 3.1 as well the effect of our proposal, we discuss a simple example considering random draws from the unit n-sphere, shown in Figure 1, i.e., the set of points $\mathcal{B} = \{x \in \mathbb{R}^n : ||x||_2 < 1\}$. We further consider a concentric sphere of radius $0 < r < 1$ given by $\mathcal{B}_r = \{x \in \mathbb{R}^n : ||x||_2 < r\}$. We are interested in the probability of a random draw from $\mathcal{B}$ to lie outside of $\mathcal{B}_r$, i.e.: $P(||x||_2 > r), x \sim \mathcal{D}(\mathcal{B})$, for some distribution $\mathcal{D}$. We start by defining $\mathcal{D}$ as the Uniform($\mathcal{B}$), which results in $P(||x||_2 > r) = 1 - r^n$. In Figure 2a, we can see that for growing n, $P(||x||_2 > r)$ is very large even if $r \approx 1$, which suggests most random draws will lie close to $\mathcal{B}$'s boundary.

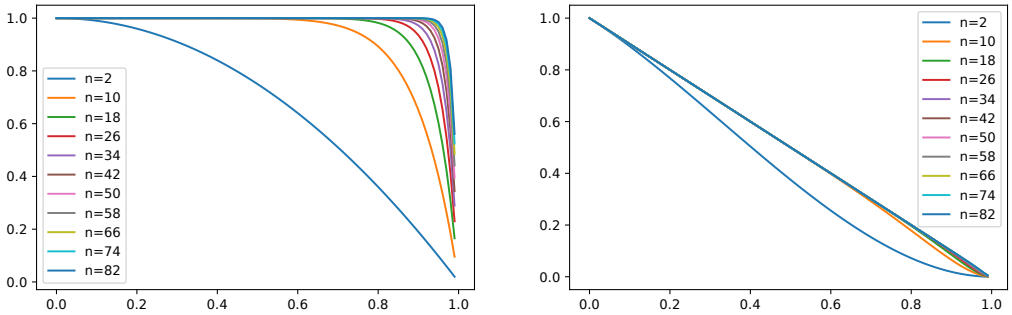
We now evaluate the case where mixup is applied and random draws are taken in two steps: we first observe $y \sim$ Uniform($\mathcal{B}$), and then we perform mixup between y and the origin[2], i.e. $x = \lambda y$, $\lambda \sim$ Uniform($[0, 1]$). In this case, $P(||x||_2 > r) = (1 - r^n)(1 - r)$, which is shown in Figure 2b as a function of r for increasing n. We can then observe that even for large n, $P(||x||_2 > r)$ decays linearly with r, i.e., we populate the interior of $\mathcal{B}$ and x in this case follows a non-uniform distribution such that its norms histogram is uniform.

In the next section, we perform an empirical evaluation where we compare the predictors resulting from different choices for computing Ω during training. Moreover, we evaluate the strategy described above and observe that combining random and data points yields monotonicity in a larger volume of $\mathcal{X}$ compared to existing algorithms.

4 Evaluation

In order to evaluate the effect of the choices of Ω as discussed in Section 3, we carry out an empirical study which is divided into two parts: we start by analyzing proof-of-concept experiments using synthetic data, generated in such a way that one can control the dimension of $\mathcal{X}$ as well as the number of dimensions with respect to which monotonicity is expected. After that, we report results on 3 commonly used datasets covering classification and regression settings with input spaces of different

[2]Similar conclusions hold for any fixed point within $\mathcal{B}$. The origin is chosen for convenience.



(a) $P(||x||_2 > r)$ as a function of r for various n and $x \sim$ Uniform($\mathcal{B}$).

(b) $P(||x||_2 > r)$ as a function of r for various n. In this case, $x = \lambda y$, $\lambda \sim$ Uniform([0, 1]), $y \sim$ Uniform($\mathcal{B}$).

Figure 2: Illustrative example showing that uniformly distributed draws on a unit sphere in $\mathbb{R}^n$ concentrate on its boundary for large n. Applying mixup populates the interior of the space.

dimensions. In both cases, models are implemented using the same architecture as in [13], i.e., using dense layers whose weights are kept separate in early layers for the input dimensions with respect to which monotonicity is to be enforced. We set the depth of all networks to 3, and use a bottleneck of size 10 for two datasets (Compas and Loan Lending Club), and 100 for the case of the Blog Feedback dataset and the experiments on generated data. Training is carried out with the Adam optimizer [11] with a global learning rate of $5\text{e}{-3}$, and γ is set to $1\text{e}4$. The training batch size is set to 256 throughout experiments.

For all evaluation cases, we consider the baseline where training is carried out without any monotonicity enforcing penalty. For the regularized cases, the different approaches used for computing Ω are as follows:

1. Ω_{random} ([13]): uses random points drawn from Uniform($\mathcal{X}$). In this case, the sample observed at each training iteration is set to a size of 1024 throughout all experiments.
2. Ω_{train} ([10]): uses the actual data observed at each training iteration; i.e., the observed mini-batch itself is used to compute Ω.
3. Ω_{mixup}: in this case, the penalty is computed on points generated by mixing-up points from the training data and random points. In further details, for each mini-batch of size $N > 1$, we augment it with complementary random data and obtain a final mini-batch of size $2N$. Out of the $\frac{2N(2N-1)}{2}$ possible pairs of points, we take a random subsample of 1024 pairs to compute mixtures of instances. In this case, we use $\lambda \sim$ Uniform([0,1]) and λ is independently drawn for each pair of points.

Results are reported in terms of both prediction performance and level of monotonicity. The latter is assessed via the fraction ρ of points within a sample where the monotonicity constraint is violated; i.e., given a set of N data points, we compute:

$$\rho = \frac{\sum_{k=1}^{N} \mathbb{1}[\min_{i \in M} \frac{\partial h(x)}{\partial x_i^k} < 0]}{N}, \tag{3}$$

such that $\rho = 0$ corresponds to monotonic models over the considered points. In order to quantify the degree of monotonicity in different parts of the space, we compute ρ for 3 different sets of points:

1. ρ_{random}: computed on a sample drawn according to Uniform($\mathcal{X}$). We used a sample of 10,000 points throughout the experiments.
2. ρ_{train}: computed on the training data.
3. ρ_{test}: computed on the test data.

4.1 Proof-of-concept evaluation

We start by describing the approach we employ to generate data containing the properties required by our evaluation. Denote a design matrix by $X_{N\times D}$ such that each of its N rows corresponds to a feature vector within $\mathbb{R}^D$. In order to ensure the data lies in some manifold, we first obtain a low-dimensional synthetic design matrix given by $X'_{N\times d}$, where each entry is sampled randomly from Uniform($[-10, 10]$). We then expand it to $\mathbb{R}^D$ by applying the following transformation:

$$X = X'A, \tag{4}$$

where the expansion matrix given by $A_{d\times D}$ is such that each of its entries are independently drawn from Uniform($[0, 1]$). Throughout our experiments, $d = \lfloor 0.3D \rfloor$ was employed.

Target values for the function f to be approximated are defined as sums of functions of scalar arguments applied independently over each dimension. We thus select a set of dimensions $M \in [D]$ with respect to which f is to be monotonic, i.e.:

$$f(x) = \sum_{i\in M} g_i(x_i) + \sum_{j\in \bar{M}} h_j(x_j), \tag{5}$$

and every $g_i : \mathbb{R} \mapsto \mathbb{R}$ is increasing monotonic, while every $h_i : \mathbb{R} \mapsto \mathbb{R}$ is not monotonic.

We then create two evaluation datasets. One of them, referred to as the validation set, is identically distributed with respect to X since it is obtained following the same procedure discussed above. In order to simulate covariate-shift, we create a test set by changing the expansion matrix A to a different one.

$$X_{val} = X'_{val}A, \quad X_{test} = X'_{test}A_{test}, \tag{6}$$

where A_{test} will be given by entry-wise linear interpolations between A, used to generate the training data, and a newly sampled expansion matrix A': $A_{test} = \alpha A' + (1 - \alpha)A$. The parameter $\alpha \in [0, 1]$, set to 0.8 in the reported evaluation, controls the shift between A and A_{test} in terms of the Frobenius norm, which, in turn, enables the control of how much the test set shifts relative to the training data.

We thus trained models to approximate f for spaces of increasing dimensions as well as for an increasing number of dimensions with respect to which f is monotonic. Results are reported in Table 1 in terms of RMSE on the two evaluation datasets, and in terms of monotonicity in Table 2 where ρ is computed both on random points and on the shifted test set. Entries in the tables correspond to the centers of 95% confidence intervals resulting from 20 independent training runs.

We highlight the two following observations regarding the prediction performances shown in table 1: different models present consistent performances across evaluations, which suggests different monotonicity-enforcing penalties do not significantly affect prediction accuracy. Moreover, the proposed approach used to generate test data under covariate-shift is effective given the gap in performance consistently observed between the validation and the test partitions. In terms of monotonicity, results in Table 2 suggest that Ω_{random} and Ω_{train} are only effective on either random or data points, which seems to aggravate when the dimension D grows. Ω_{mixup}, on the other hand, is effective on both sets of points, and continues to work well for growing D. Furthermore, covariate-shift significantly affects Ω_{train} for higher-dimensional cases, while Ω_{mixup} performs well in such case.

$\lvert M\rvert/D$	*20/100*		*40/200*		*80/400*		*100/500*	
	Valid. RMSE	Test RMSE	Valid. RMSE	Test RMSE	Valid. RMSE	Test RMSE	Valid. RMSE	Test RMSE
Non-mon.	0.007	0.107	0.006	0.082	0.007	0.087	0.011	0.146
Ω_{random}	0.008	0.117	0.006	0.081	0.007	0.093	0.012	0.125
Ω_{train}	0.008	0.115	0.006	0.086	0.007	0.089	0.012	0.134
Ω_{mixup}	0.008	0.114	0.007	0.084	0.008	0.088	0.012	0.134

Table 1: Prediction performance of models trained on generated data in spaces of growing dimension (D) and number of monotonic dimensions ($|M|$). Different regularization strategies do not affect prediction performance. The performance gap consistently observed across the evaluation sets highlights the shift between the two sets of points. The lower the values of RMSE the better.

$\|M\|/D$	*20/100*		*40/200*		*80/400*		*100/500*	
	ρ_{random}	ρ_{test}	ρ_{random}	ρ_{test}	ρ_{random}	ρ_{test}	ρ_{random}	ρ_{test}
Non-mon.	99.90%	99.99%	97.92%	94.96%	98.47%	96.56%	93.98%	90.01%
Ω_{random}	0.00%	3.49%	0.00%	4.62%	0.01%	11.36%	0.02%	19.90%
Ω_{train}	1.30%	0.36%	4.00%	0.58%	9.67%	0.25%	9.25%	5.57%
Ω_{mixup}	0.00%	0.35%	0.00%	0.44%	0.00%	0.26%	0.00%	0.42%

Table 2: Fraction of monotonic points ρ for models trained on generated data in spaces of growing dimension (D) and number of monotonic dimensions ($|M|$). Different regularization strategies is effective on only one of ρ_{random} or ρ_{test}, while Ω_{mixup} seems effective throughout conditions. The lower the values of ρ the better.

4.2 Benchmarks

We now proceed and evaluate our models on real data. Three often used benchmarks are employed for this set of experiments, and the datasets are described in further details in Table 3.

Dataset	Dim$[\mathcal{X}]$	$\|M\|$	# Train	# Test	Task
Compas[3]	13	4	4937	1235	*Classification*
Loan Lending Club[4]	33	11	8500	1500	*Regression*
Blog Feedback[5]	280	8	47287	6904	*Regression*

Table 3: Description of datasets used for empirical evaluation.

Results are summarized in Tables 4, 5, and 6 in terms of both prediction performance along with the metric indicating the *degree of monotonicity* of the predictor for each regularization strategy. Prediction performance is measured in terms of accuracy for classification tasks, and RMSE for the case of regression. Monotonicity, on the other hand, is quantified via the fraction ρ of points within a sample where the monotonicity constraint is violated. Results reported in the tables correspond to 95% confidence intervals corresponding to multiple independent training runs.

	Non-mon.	Ω_{random}	Ω_{train}	Ω_{mixup}
Validation acc.	69.1%±0.2%	68.5%±0.1%	68.5%±0.1%	68.4%±0.1%
Test acc.	68.5%±0.2%	68.1%±0.2%	68.0%±0.2%	68.3%±0.2%
ρ_{random}	55.45%±12.26%	0.01%±0.01%	6.41%±4.54%	0.00%±0.00%
ρ_{train}	92.98%±2.70%	2.08%±2.21%	0.00%±0.00%	0.00%±0.00%
ρ_{test}	92.84%±2.75%	2.16%±2.35%	0.00%±0.00%	0.00%±0.00%

Table 4: Evaluation results for the COMPAS dataset in terms of 95% confidence intervals resulting from 20 independent training runs. Results correspond to the checkpoint that obtained the best prediction performance on validation data throughout training. The lower the values of ρ the better.

We start the analysis of the results by comparing the prediction performances of the models obtained under the different regularization strategies with the unregularized baseline. In this case, we observe that across all datasets, the different penalties do not result in significant variations in the performance of the final predictors. This indicates that the class of predictors corresponding to the subset of $\mathcal{H}$ that is monotonic relative to M, denoted by $\mathcal{H}_M$, has enough capacity so as to be able to match the performance of the best canditates within $\mathcal{H}$.

In terms of monotonicity, we observe a clear pattern leading to the following intuition: *monotonicity is achieved in the regions where it is enforced*. This is evidenced by the observation that ρ_{random} is consistently lower for Ω_{random} relative to Ω_{train} and Ω_{mixup} while, on the other hand, ρ_{train} and ρ_{test} are consistently lower for Ω_{train} and Ω_{mixup} compared to Ω_{random}. Moreover, a comparison between Ω_{train} and Ω_{mixup} shows what we anticipated: enforcing monotonicity in points resulting from mixup yields predictors that are as monotonic as those given by the use of Ω_{train} in actual data,

[3] `https://www.kaggle.com/danofer/compass`

[4] `https://www.openintro.org/data/index.php?data=loans_full_schema`

[5] `https://archive.ics.uci.edu/ml/datasets/BlogFeedback`

but significantly better at the boundaries of $\mathcal{X}$. Finally, the results demonstrate that our proposed approach $\Omega_{mixup-TR}$ achieves the best results in terms of monotonicity for all the sets of points that we considered. Moreover, our approach introduces no significant computation overhead.

	Non-mon.	Ω_{random}	Ω_{train}	Ω_{mixup}
Validation RMSE	0.213±0.000	0.223±0.002	0.222±0.002	0.235±0.001
Test RMSE	0.221±0.001	0.230±0.001	0.229±0.002	0.228±0.001
ρ_{random}	99.11%±1.70%	0.00%±0.00%	14.47%±7.55%	0.00%±0.00%
ρ_{train}	100.00%±0.00%	7.23%±7.76%	0.01%±0.01%	0.00%±0.00%
ρ_{test}	100.00%±0.00%	6.94%±7.43%	0.04%±0.03%	0.00%±0.00%

Table 5: Evaluation results for the Loan Lending Club dataset in terms of 95% confidence intervals resulting from 20 independent training runs. Results correspond to the checkpoint that obtained the best prediction performance on validation data throughout training. The lower the values of ρ the better.

	Non-mon.	Ω_{random}	Ω_{train}	Ω_{mixup}
Validation RMSE	0.174±0.000	0.175±0.001	0.177±0.000	0.168±0.000
Test RMSE	0.139±0.001	0.139±0.001	0.142±0.001	0.143±0.001
ρ_{random}	76.17%±12.37%	0.05%±0.08%	3.86%±4.19%	0.00%±0.01%
ρ_{train}	78.67%±5.28%	78.59%±6.37%	0.01%±0.01%	0.01%±0.01%
ρ_{test}	76.29%±6.47%	78.99%±7.20%	0.02%±0.02%	0.02%±0.02%

Table 6: Evaluation results for the Blog Feedback dataset in terms of 95% confidence intervals resulting from 20 independent training runs. Results correspond to the checkpoint that obtained the best prediction performance on validation data throughout training. The lower the values of ρ the better.

5 Conclusion

We studied the supervised learning setting where the outputs of the model are expected to be monotonic with respect to some subset of the input space dimensions. Specifically, we focused on the case where general classes of models are used but the learning scheme is expected to yield risk minimizers satisfying monotonicity requirements. We thus focused on point-wise regularization penalties that bias the learning algorithm towards predictors that have non-negative gradients relative to the dimensions with respect to which monotonicity is desired.

Our empirical evaluation shows that the monotonicity property is achieved in the parts of the space where it is imposed. Such finding suggests that past approaches will only enforce monotonicity too far from the data [13] or too close to the data [10]. In order to alleviate this issue, we propose a novel approach that computes the regularization penalty on mixup points. Our method generates the penalty data via linear combinations between training instances and random points. In doing so, we can interpolate the data points and populate the convex hull of the training sample. At the same time, we can extrapolate and create points further away from the manifold where data lies. Empirical evaluation confirmed that our proposed gradient penalty Ω_{mixup}, using combinations between random samples and data instances, results in the most effective regularization strategy across all evaluated schemes, yielding models observed to satisfy the monotonicity requirement in a larger volume of the input space when compared to alternative strategies.

Given that we observed point-wise gradient penalties to be effective in yielding monotonicity, we envision extensions of this work towards larger neural networks applied in structured data such as image and text. In that case, however, we hypothesize monotonicity might be enforced between the model's outputs and high-level representations so as to enable interpretability of predictions.

References

[1] N. P. Archer and S. 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] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] W. T. Bakst, N. Morioka, and E. Louidor. Monotonic kronecker-factored lattice. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=0pxiMpCyBtr`.

[4] C.-Y. Chuang and Y. Mroueh. Fair mixup: Fairness via interpolation. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=DNl5s5BXeBn`.

[5] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems*, pages 472–478, 2001.

[6] C. Fefferman, S. Mitter, and H. Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

[7] E. Garcia and M. Gupta. Lattice regression. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL `https://proceedings.neurips.cc/paper/2009/file/4b0250793549726d5c1ea3906726ebfe-Paper.pdf`.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[9] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pages 1764–1772. PMLR, 2014.

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

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[13] X. Liu, X. Han, N. Zhang, and Q. Liu. Certified monotonic neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15427–15438. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/b139aeda1c2914e3b579aafd3ceeb1bd-Paper.pdf`.

[14] A.-p. Nguyen and M. R. Martínez. Mononet: towards interpretable models by learning monotonic features. *arXiv preprint arXiv:1909.13611*, 2019.

[15] A. Sivaraman, G. Farnadi, T. Millstein, and G. Van den Broeck. Counterexample-guided learning of monotonic neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11936–11948. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/8ab70731b1553f17c11a3bbc87e0b605-Paper.pdf`.

[16] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR, 2019.

[17] A. Wehenkel and G. Louppe. Unconstrained monotonic neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/2a084e55c87b1ebcdaad1f62fdbbac8e-Paper.pdf`.

[18] S. You, D. Ding, K. Canini, J. Pfeifer, and M. Gupta. Deep lattice networks and partial monotonic functions. *arXiv preprint arXiv:1709.06680*, 2017.

[19] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR (Poster)*, 2018. URL `https://openreview.net/forum?id=r1Ddp1-Rb`.