MONOTONICITY AS A REQUIREMENT AND AS A REGULARIZER: EFFICIENT METHODS AND APPLICATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We study the setting where risk minimization is performed over general classes of models and consider two cases where monotonicity is treated as either a requirement to be satisfied everywhere or a useful property. We specifically consider cases where point-wise gradient penalties are used alongside the empirical risk during training. In our first contribution, we show that different choices of penalties define the regions of the input space where the property is observed. As such, previous methods result in models that are monotonic only in a small volume of the input space. We thus propose an approach that uses mixtures of training instances and random points to populate the space and enforce the penalty in a much larger region. As a second contribution, we introduce the notion of group monotonicity as a regularization bias for convolutional models. In this case, we consider applications, such as image classification and generative modeling, where monotonicity is not a hard constraint but can help improve some aspects of the model. Namely, we show that using group monotonicity can be beneficial in several applications such as: (1) generating explanations for predictions, (2) defining strategies to detect anomalous data, and (3) allowing for controllable data generation. Our proposed approaches do not introduce relevant computational overhead while leading to efficient procedures that provide extra benefits over baseline models.

1 INTRODUCTION

Highly expressive model classes such as artificial neural networks have achieved impressive prediction performance across a broad range of supervised learning tasks and domains (Krizhevsky et al., 2012; Graves & Jaitly, 2014; Bahdanau et al., 2014). 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 other than 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: (1) *Robustness*, where low risk is expected even if the model is evaluated under distribution shifts, (2) *Fairness*, where the performance of the model is expected to not significantly change across different sub-populations of the data, and (3) *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 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 except their years of experience, then the more experienced candidate should be assigned an equal or higher chance of getting accepted. For applications where monotonicity is expected, predictor failing to satisfy this requirement would damage the user's confidence. As such, 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*, in which case focus lies on defining a model class that guarantees monotonicity in all of its elements Bakst et al. (2021); Wehenkel

& Louppe (2019); Nguyen & Martínez (2019); You et al. (2017); Garcia & Gupta (2009); Archer & Wang (1993). 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. Alternatively, a second approach consists of *encouraging monotonicity during training*, i.e., searching for monotonic candidates within a general class of models (Liu et al., 2020; Sivaraman et al., 2020; Gupta et al., 2019). 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 extra verification/certification steps are performed, which can be computationally very expensive. In addition to being a *requirement* as in the examples discussed above, monotonicity was also observed to be a useful feature in certain cases. For instance, it defined an effective inductive bias and improved generalization in cases where prior knowledge indicates the data generation process satisfies such property, as discussed by Dugas et al. (2001). In such a case, however, the property needs not to be satisfied everywhere (i.e., in the bulk of the input space), since it is enforced simply as a desirable *feature* of trained models rather than a design specification.

In this work, we tackle the problem of performing empirical risk minimization over rich classes of models such as neural networks, while simultaneously searching for monotonic predictors within the set of risk minimizing solutions. In summary, our contributions are two-fold:

1. *monotonicity as a requirement*: 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. We then propose an efficient algorithm that tackles the problem. In details, we apply Mixup (Zhang et al., 2018) between data and random points to populate the input space, which is shown to enforce monotonicity in a larger volume relative to previous methods in literature.

2. *monotonicity as a feature*: We define a new notion of monotonicity which we observed to be useful when enforced in larger models in cases such as object recognition or generative modeling of images. In these cases, there is no need to enforce the property everywhere and, as such, penalties that focus only on actual data points are discussed (i.e., mixup is not needed). Models satisfying the property are compared with standard predictors across three applications, where we observed the property to offer benefits without compromising the original performance.

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 1 *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.

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 (Archer & Wang, 1993). However, doing so affects the prediction performance. Another approach corresponds to using lattice regression models proposed by Garcia & Gupta (2009) and recently extended by Bakst et al. (2021); You et al. (2017). 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 (Nguyen & Martínez, 2019) reparameterizes fully connected layers such that gradients with respect to parameters can only be non-negative. Wehenkel & Louppe (2019), on the other hand, consider the class of predictors $H : \mathcal{X} \mapsto \mathcal{Y}$ of the form $H(x) = \int_0^x h(t)dt + H(0)$, where $h(t)$ is a strictly positive

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

mapping parameterized by a neural network. While such approaches guarantee monotonicity by design, they can be too restrictive or render overly complicated learning procedures. E.g., the approach in (Wehenkel & Louppe, 2019) requires backpropagating through the integral. An alternative approachcorresponds to searching over general classes of models while assigning higher importance to predictors observed to be monotonic. Similar to the case of adversarial training (Goodfellow et al., 2014), Sivaraman et al. (2020) proposed an approach that finds counterexamples, i.e. pairs of points where the monotonicity constraint is violated, which are included in the training data to enforce next iterates of the model to satisfy the monotonicity conditions. However, this approach only supports fully-connected ReLU networks. Moreover, the procedure for finding the counterexamples is costly. Alternatively, point-wise regularization penalties are introduced by Liu et al. (2020); Gupta et al. (2019), which requires selecting points where the penalty is computed. While Liu et al. (2020) use uniform random draws, Gupta et al. (2019) apply the regularization penalty over the training instances. Both approaches have shortcomings that we seek to address.

3 Monotonicity as a requirement

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 seek the optimal monotonic predictors relative to M, h^*_M:

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

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 (Gupta et al., 2019; Liu et al., 2020):

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

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. At this point, the only missing ingredient to define algorithms to estimate h^*_M is how to define the distribution $\mathcal{D}$ over which the expectation in Eq. 2 is computed, discussed in the following sections.

3.1 Choosing distributions over which to compute the monotonicity penalty

Past choices for $\mathcal{D}$ are such that: 1. *Define $\mathcal{D}$ as the empirical distribution of the training sample*: In (Gupta et al., 2019), 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:

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

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 Appendix A.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 Liu et al. (2020), 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.:

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

	Non-mon.	Ω_{random}	Ω_{train}	Ω_{mixup}
		COMPAS		
Validation accuracy	69.1%±0.2%	68.5%±0.1%	68.5%±0.1%	68.4%±0.1%
Test accuracy	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%
		Loan Lending Club		
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%
		Blog feedback		
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 1: Evaluation results 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.

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. (Fefferman et al., 2016) 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 examples in Appendix A, 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 (Liu et al., 2020) 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 (Zhang et al., 2018; Verma et al., 2019; Chuang & Mroueh, 2021), where auxiliary data is created via interpolations of pairs of data points, to populate ares of the space that are otherwise disregarded. Mixup was introduced by Zhang et al. (2018) 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 $\lambda \sim \text{Uniform}([0, 1])$. 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* of data points more densely populates the convex hull of the training data. (2) *Extrapolation* cases where mixup is performed between data points and instances obtained at random from results in points that lie anywhere between the data manifold and the boundaries of the space. We thus claim 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.

3.2 Evaluation

In order to evaluate the effect of different choices of Ω, we report results on three commonly used datasets covering classification and regression settings with input spaces of different dimensions. Namely, we reporte results for the following datasets: *Compas*, *Loan Lending Club*, and *Blog Feedback*. Further details on the data can be found in Table 6 in the appendix. Models are implemented using the same architecture as in (Liu et al., 2020), 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 (Kingma & Ba, 2014) with a global learning rate of $5\mathrm{e}{-3}$, and γ is set to $1\mathrm{e}4$. 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} (Liu et al., 2020) which 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} (Gupta et al., 2019) which uses the actual data observed at each training iteration; i.e., the observed mini-batch itself is used to compute Ω. And (3) Ω_{mixup} (*ours*), in which case the penalty is computed on points generated by mixing-up points from the training data and random points. In 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: $\rho = \frac{\sum_{k=1}^{N} \mathbb{1}[\min_{i \in M} \frac{\partial h(x)}{\partial x_i^k} < 0]}{N}$, such that $\rho = 0$ corresponds to monotonic models over the considered points. Moreover, 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. And (3) ρ_{test}: computed on the test data. Results are summarized in Table 1 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. 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 $\mathcal{H}_M$, 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}. 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, but significantly better at the boundaries of $\mathcal{X}$. Finally, the results demonstrate that our proposed approach Ω_{mixup} 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.

Model	$\arg\max_{k \in \mathcal{Y}} h(x)_k$	$\arg\max_{k \in \mathcal{Y}} T_k(x)$
CIFAR-10		
WideResNet	95.46%	16.35%
*Mono*WideResNet	95.64%	94.95%
ImageNet		
ResNet-50	75.85%	0.10%
*Mono*ResNet-50	76.50%	72.52%

Table 2: Top-1 accuracy of standard and group monotonic models. Total activation classifiers on par with the output layer for monotonic models.

4 Monotonicity as a feature

We now consider the case where some notion of monotonicity is to be enforced in more general classes of models. In what follows, we will discuss such a notion and how it can be useful once successfully achieved. we remark that, in this case, monotonicity is not a requirement and, as such, needs not to hold everywhere. The penalties we discuss from now on are computed considering only data points and no random draws are utilized. In particular, we are interested in the case of

K-way classifiers realized through convolutional neural networks. In such a case, data examples once more correspond to pairs $x, y \sim \mathcal{X} \times \mathcal{Y}$, but in this case, $\mathcal{Y} = \{1, 2, 3, ..., K\}$, $K \in \mathbb{N}$. Models parameterize a data-conditional categorical distribution over $\mathcal{Y}$, i.e., for a given model h, $h(x)_{\mathcal{Y}}$ will yield likelihoods for each class indexed in $\mathcal{Y}$. Under this setting, we introduce the notion of *Group Monotonicity*: we aim to find models h such that outputs corresponding to particular classes satisfy a monotonic relationship with a subset of high-level representations, output by some inner convolutional layer. More precisely, consider the outputs of a specific layer within a convolutional model is represented by a_w, $w \in [1, 2, 3, ..., W]$, where W indicates the width of the chosen layer given by its number of output feature maps. For simplicity of exposition, we will consider the rather common case where each feature map a_w is 2-dimensional. Given that, we partition such set of representations into disjoint subsets, or *groups*, of uniform sizes. Each subset is then paired with a particular output or class, and hence denoted S_k, $k \in \mathcal{Y}$. An illustration is provided in Figure 1, where a generic convolutional model has the outputs of a specific layer partitioned into slices S_k which are paired with corresponding output units.

Model	AUC-ROC
MNIST	
WideLeNet	54.47%
*Mono*WideLeNet	100.00%
CIFAR-10	
WideResNet	67.35%
*Mono*WideResNet	79.33%

Table 3: AUC-ROC (the higher the better) for the detection of adversarially perturbed data instances.

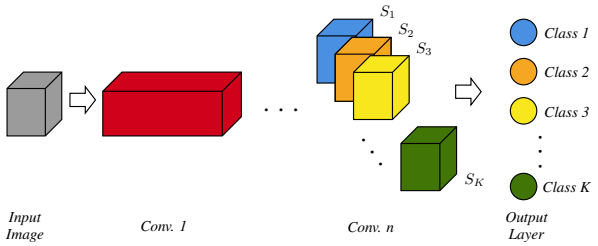

Figure 1: Illustration of a group monotonic convolutional model. Representations output by a certain convolutional layer are partitioned into disjoint subsets.

Definition 2 *Group monotonic convolutional classifiers:* We say h is group monotonic for input x and its corresponding class label y if $h(x)_y$ is partially monotonic relative to all elements in S_y.

We highlight that, in this case, unlike the discussion presented on earlier sections, monotonicity *is not* a design requirement and, as such, needs not be satisfied everywhere. Intuitively, our goal is to "reserve" groups of high-level features to be monotonic with respect to different classes since, as will be discussed, imposing such structure in the learned representations can be beneficial. For training, we perform monotonic risk minimization as described in 1, and the risk is given by the negative log-likelihood over training points. Moreover, we design a penalty Ω that focus only on observed data points during training and penalize only the slices of the Jacobian that correspond to a given class. In order to formally introduce such a penalty, denoted Ω_{group}, we first define the total gradient O_k, $k \in \mathcal{Y}$, of a slice S_k as follows: $O_y(x) = \sum_{a_w \in S_y} \sum_{i,j} \frac{\partial h(x)_y}{\partial a_{w,i,j}}$, where the inner sum accounts for spatial dimensions of a_w. Given the set of total gradients, one such value per class, Ω_{group} will be given as follows for a batch of size m and inverse temperature μ:

$$\Omega_{group} = -\frac{1}{m} \sum_{i=1}^{m} \log \frac{e^{\frac{O_y^i(x_i)}{\mu}}}{\sum_{k=1}^{K} e^{\frac{O_k^i(x_i)}{\mu}}}. \tag{3}$$

4.1 Evaluation

4.1.1 Group monotonic classifiers perform on par with their non-monotonic counterparts

We start our evaluation by verifying whether the group monotonicity property can be effectively enforced into classifiers trained on standard object recognition benchmarks. In order to do so, we verify the performance of the *total activation classifier*, as defined by: $\arg\max_{k \in \mathcal{Y}} T_k(x)$, where T_k indicates the total activation on slice S_k: $T_k(x) = \sum_{a_w \in S_k} \sum_{i,j} a_{w,i,j}(x)$.

A good prediction performance of such a classifier serves as evidence that the group monotonicity property is satisfied by the model over the test data under consideration since it indicates the slice relative to the underlying class of test instances has the highest total activation. We thus run evaluations for both CIFAR-10 and ImageNet. For the case of CIFAR-10, trained models correspond to WideResNets (Zagoruyko & Komodakis, 2016). Models are initialized randomly and trained both with and without the monotonicity penalty. Standard stochastic gradient descent (SGD) implements the parameters update rule with a learning rate starting at 0.1, being decreased by a factor of 10 on epochs 10, 150, 250, and 350. Training is carried out for a total of 600 epochs with a batch size of 64. For ImageNet, on the other, training consists of fine tuning a pre-trained ResNet-50, where the fine-tuning phase included the monotonicity penalty. We do so by training the model for 30 epochs on the full ImageNet training partition. In this case, given that the label set $\mathcal{Y}$ is relative large, using the standard ResNet-50 would result in small slices S_k. To avoid that, we add an extra final convolution layer with $W = 15K$. Training is once more carried out with SGD using a learning rate set to 0.001 in this case, and reduced by a factor of 5 at epoch 20. In both cases, the group monotonicity property is enforced at the last convolutional layer. Other hyperparameters such as the strength γ of the monotonicity penalty as well as the inverse temperature μ used to compute Ω_{group} are set to 1 and 50 for the case of CIFAR-10, and to 5 and 10 for the case of ImageNet. Both momentum and weight decay are further employed and their corresponding parameters are set to 0.9 and 0.0001. Results are reported in Table 2 and correspond to the top-1 prediction accuracy measured on the test data. We use standard classifiers as baselines, in which case no monotonicity penalty is applied, so as to be able to isolate the effect of the penalty. In both datasets, total activation classifiers for group monotonic models, are able to closely match the performance obtained by the classifier defined at the output layer, $\arg\max_{k \in \mathcal{Y}} h(x)_k$. This suggests that the higher total activation generally matches the predicted class for group monotonic models, which indicates the property is successfully enforced in both cases. Moreover, the prediction performance obtained at the output layer was not affected by the introduction of the monotonicity property, i.e. one can match the risk of unconstrained models while focusing on the set of group monotonic candidates. Additional experiments are reported on Appendix C for cases with small sample sizes.

4.1.2 Using features structure to detect anomalies

Since we observed that group monotonicity can be enforced without affecting prediction performance, we discuss approaches to leverage it and define applications of models satisfying such a property. In particular, we consider the case where one is interested in detecting anomalous instances, i.e. those where it is likely the model made a mistake. Consider, for instance, the case where a classifier is deployed to production and, due to some problem external to the model, it is queried for a prediction for an input consisting of white noise. Standard classifiers would provide a prediction even in such a clearly anomalous input. A more desirable behavior in such a case would for the model to somehow indicate that instance is problematic. We claim that imposing structure in the features learned by large models, e.g. by enforcing group monotonicity, can help in deciding when not to predict. That is, if the structure that was imposed during training is not observed during testing, then it's likely that the input is anomalous in some way, and a model can then refuse to predict test instances or request human support. In order to evaluate such application, we implement anomalous test instances by the means of adversarial perturbations. Namely, we create L_∞ PGD attackers (Madry et al., 2017) and detect anomalies based on simple statistics of the features. In details, for a given input x, we compute the normalized entropy $H^*(x)$ of the categorical distribution defined by the application of the softmax operator over the set of total activations $T_{\mathcal{Y}}(x)$: $H^*(x) = \frac{\sum_{k \in \mathcal{Y}} p_k(x) \log p_k}{\log K}$, where $K = |\mathcal{Y}|$ and the set $p_{\mathcal{Y}}(x)$ corresponds to the parameters of a categorical distribution defined by: $p_{\mathcal{Y}}(x) = \text{softmax}(T_{\mathcal{Y}}(x))$. Decisions can then be made by comparing $H^*(x)$ with a threshold $\tau \in [0, 1]$, defining the detector $\mathbb{1}_{\{H^* > \tau\}}$.

Detection performance of such an approach is evaluated on both MNIST and CIFAR-10. Training for the case of CIFAR-10 follows the same setup discussed on Section 4.1.1. For MNIST on the other hand, we modify the standard LeNet architecture by increasing the width of the second convolutional layer from 64 to 150. This layer is then used to enforce the group monotonicity property. The resulting model is referred to as WideLeNet. Training is performed for 20 epochs using a batch size of 64 and the Adadelta updating rule (Zeiler, 2012), with a learning rate of 1. Moreover, γ and μ are set to $1e10$ and 1, respectively. Adversarial attacks are created under the white-box setting, i.e. by exposing the full model to the attacker. The perturbation budget in terms of L_∞ distance is set to 0.3

and and $\frac{8}{255}$ for the cases of MNIST and CIFAR-10, respectively. Detection performance is reported in Table 3 for the considered cases in terms of the area under the operating curve (AUC-ROC). Baselines correspond to models for which the monotonicity penalty is not enforced. Those are trained under the same conditions and given the same computation budget as the models where the penalty is enforced. Results indicate what we anticipated, i.e., the cases where the total activations are not structured as enforced during training very often correspond to anomalous inputs.

5 DISENTANGLED REPRESENTATION LEARNING UNDER MONOTONICITY CONSTRAINTS

We now discuss an application of some notion of monotonicity to a generative setting. In particular, we consider the case where one's interest is to learn a disentangled set of representations. In such a case, it's often assumed that latent variables are independent, and hence control over generative factors can be achieved, e.g. one can modify a specific aspect of the data by modifying the value of a corresponding latent variable. However, we argue that disentanglement is necessary but not sufficient to enable controllable data generation, and one needs latent variables that satisfy some notion of monotonicity to be able to decide theirs values resulting in desired properties. We specifically build upon the weakly supervised framework introduced by Locatello et al. (2020), which extends the β-VAE setting (Higgins et al., 2016) by introducing weak supervision, given that training instances are presented to the model in pairs (x_1, x_2) between which it's known only a few of the set of possible generative factors have changed. We then include group monotonocity over the activations of corresponding latent variables. In this case, data is modeled according to $p(x|z)p(z)$ for latent variables z. Approximation is then performed by introducing $p_\theta(x|z)$ and $q_\phi(z|x)$, both parameterized by neural networks. Our interest is that z fully factorizes in its dimensions, i.e. $p(z) = \prod_{i=1}^{Dim[z]} p(z_i)$, which needs to be captured by the approximate posterior distribution $q_\phi(z|x)$. Training is then performed by means of maximization of a lower-bound on the likelihood of the training data under the assumed model, which yields the following training loss:

$$\mathcal{L}_{ELBO} = \mathbb{E}_{x^1,x^2} \sum_{i\in\{1,2\}} \mathbb{E}_{\tilde{q}_\phi(\hat{z}|x^i)} \log(p_\theta(x^i|\hat{z})) - \beta D_{KL}(\tilde{q}_\phi(\hat{z}|x^i), p(\hat{z})), \tag{4}$$

where $\tilde{q}_\phi(\hat{z}_j|x^1, x^2) = q_\phi(z_j|x^i)$ for the latent dimensions z_i that change across x^1 and x^2, and $\tilde{q}_\phi(\hat{z}_j|x^1, x^2) = \frac{1}{2}(q_\phi(\hat{z}_j|x^1) + q_\phi(\hat{z}_j|x^2))$ for those that are common, the approximate posterior of the shared latent variable are forced to be the same for x^1 and x^2. The outer expectation is estimated by observing pairs of data instances (x^1, x^2) across which it is known only a number of generative factors vary. In our experiments, we consider the specific case where exactly one generative factor changes across inputs. Moreover, we follow Locatello et al. (2020) and assign the changing factor, denoted y, to the dimension j of z such that $y = \arg\max_{j\in Dim[z]} D_{KL}(z_j^1, z_j^2)$.

While the above objective enforces disentangled latent variables, controllable generation requires some regularity in z so that users are able to decide its values resulting in desired features in generated samples. We introduce Ω_{VAE} to enforce latent representations are such that latent activations are monotonic with respect to the distance between pairs of images between which only a particular generative factor vary. In other words, an increasing trend in the value of each dimension of z should yield a greater change along one of the generative factors. Formally, Ω_{VAE} is defined as follows:

$$\Omega_{VAE} = -\frac{1}{2m}\sum_{i=1}^{m} \log \frac{e^{\frac{L^i(x^{i,1},x^{i,2},y^i)}{\mu}}}{\sum_{k=1}^{K} e^{\frac{L^i(x^{i,1},x^{i,2},k)}{\mu}}} + \log \frac{e^{\frac{L^i(x^{i,2},x^{i,1},y^i)}{\mu}}}{\sum_{k=1}^{K} e^{\frac{L^i(x^{i,2},x^{i,1},k)}{\mu}}}, \tag{5}$$

where L is given by the gradient of the mean squared error (MSE) between pairs of images that are 1-factor away with respect to the dimension y of z assigned to the changing factor, i.e.:

$$L(x^i, x^j, y) = \frac{\partial \text{MSE}(\hat{x}^i, x^j)}{\partial \tilde{z}_y}. \tag{6}$$

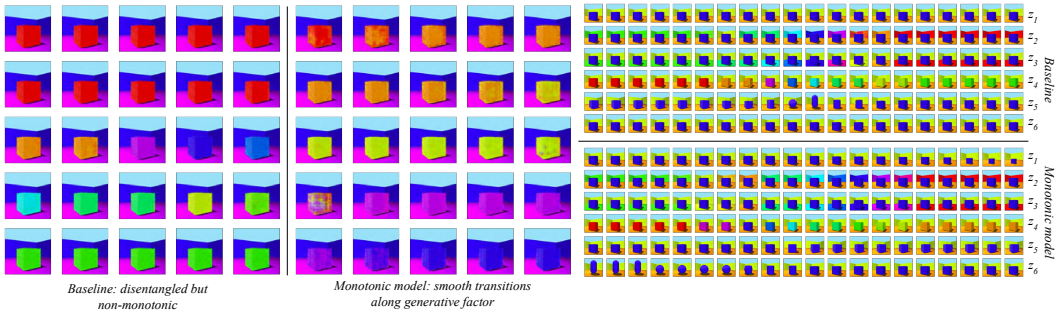

Figure 2: Comparisons between data generated by standard and monotonic models. On the two panels on the left, we compare generations along the line passing through latent variables for inputs for which only object color changes. On the two panels vertically stacked on the right, we start from the same image but change one latent dimension at a time.

In this case, $\hat{x}$ indicates the reconstruction of x. We evaluate such an approach by training the same 4-layered convolutional VAEs described in (Higgins et al., 2016) using the 3d-shapes dataset[2]. The dataset is composed of images containing shapes generated from 6 independent generative factors: *floor color*, *wall color*, *object color*, *scale*, *shape* and *orientation*. All combinations of these factors are present exactly once, resulting in $m = 480000$. We compared VAEs trained with and without the inclusion of the monotonicity penalty given by Ω_{VAE}. We highlight that the goal of the proposed framework is not to improve on top of current approaches in terms of *how disentangled* learned representations are. Rather, we seek to achieve similar results in that sense, but impose extra regularity in the relationship between generated images and the values of z so that the generative process is more easily *controllable*. A qualitative analysis is performed on top of generated data shown in Figure 2. On the two panels on the left, we represent data generated by latent variables obtained from traversals on the line in the latent space that intercepts points corresponding to images that only vary in terms of the factor *object color*. By comparing the two cases, one can notice that disentanglement is indeed achieved in both cases, but the monotonic model present much smoother transitions between colors. The base case, on the other hand, presents long sequences of very close images followed by very sharp transitions where colors sometimes repeat (e.g., green-yellow-green transitions in the fourth row). The panels stacked on the right present a per-dimension traversal of the latent space starting from a common image. In that case, it can be once again observed that both models do seem disentangled. However, transitions seem more regular for models where monotonicity was enforced. For example, transitions in shape (z_5 and z_6 for base and monotonic models, respectively) are coherent in the baseline case since there is an ordering corresponding to capsule, sphere, cylinder, and cube. The baseline instead generates cubes at different ranges of z.

6 Conclusion

We proposed approaches aiming to enable learning algorithms based on risk minimization to find solutions that satisfy some notion of monotonicity. First, we discussed the case where monotonicity is a *design requirement* that needs to be satisfied everywhere. In this case, we identified limitations in past work that resulted in models that satisfy the property only in very specific parts of the space. We then introduced a simple and effective procedure that was observed to significantly improve the solutions in terms of the volume of the space where the monotonicity requirement is achieved. In addition to that, we further argued that, even when not required, *models satisfying monotonicity present useful properties*. For instance, we considered the case of large image classifiers and showed that imposing structure in learned representations via a property we called group monotonicity, which can be done efficiently via gradient penalties, results in effective approaches to detect anomalous test data. Moreover, a similar property applied to variational autoencoders was observed to yield latent spaces that are easier to navigate since those presents more regular transitions when compared to standard generative models under the considered setting.

[2] https://github.com/deepmind/3d-shapes

REFERENCES

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

William Taylor Bakst, Nobuyuki Morioka, and Erez Louidor. Monotonic kronecker-factored lattice. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0pxiMpCyBtr.

Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pp. 839–847. IEEE, 2018.

Ching-Yao Chuang and Youssef Mroueh. Fair mixup: Fairness via interpolation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=DNl5s5BXeBn.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems*, pp. 472–478, 2001.

Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

Eric Garcia and Maya Gupta. Lattice regression. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta (eds.), *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL https://proceedings.neurips.cc/paper/2009/file/4b0250793549726d5c1ea3906726ebfe-Paper.pdf.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pp. 1764–1772. PMLR, 2014.

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

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. Certified monotonic neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 15427–15438. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/b139aeda1c2914e3b579aafd3ceeb1bd-Paper.pdf.

Francesco Locatello, Ben Poole, Gunnar Raetsch, Bernhard Schölkopf, Olivier Bachem, and Michael Tschannen. Weakly-supervised disentanglement without compromises. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6348–6359. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/locatello20a.html.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

An-phi Nguyen and María Rodríguez Martínez. Mononet: towards interpretable models by learning monotonic features. *arXiv preprint arXiv:1909.13611*, 2019.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Aishwarya Sivaraman, Golnoosh Farnadi, Todd Millstein, and Guy Van den Broeck. Counterexample-guided learning of monotonic neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 11936–11948. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/8ab70731b1553f17c11a3bbc87e0b605-Paper.pdf.

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pp. 6438–6447. PMLR, 2019.

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

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta. Deep lattice networks and partial monotonic functions. *arXiv preprint arXiv:1709.06680*, 2017.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR (Poster)*, 2018. URL https://openreview.net/forum?id=r1Ddp1-Rb.