Monotonicity Regularization: Improved Penalties and Novel Applications to Disentangled Representation Learning and Robust Classification

João Monteiro [*] **Mohamed Osama Ahmed**[1] **Hossein Hajimirsadeghi**[1] **Greg Mori**[1,2]

[1]Borealis AI
[2]Simon Fraser University

Abstract

We study settings where gradient penalties are used alongside risk minimization with the goal of obtaining predictors satisfying different notions of monotonicity. Specifically, we present two sets of contributions. In the first part of the paper, 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 set of contributions, we introduce regularization strategies that enforce other notions of monotonicity in different settings. 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 inducing monotonicity can be beneficial in applications such as: (1) allowing for controllable data generation, (2) defining strategies to detect anomalous data, and (3) generating explanations for predictions. 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 neural networks have achieved impressive prediction performance across a broad range of supervised learning tasks [Krizhevsky et al., 2012, Graves and 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. Hence, 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 data sub-populations, and (3) *Explainability/Interpretability*, where models are expected to indicate how the features of the data imply their predictions.

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 the 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, the more experienced candidate should be assigned an equal or higher chance of getting accepted. For applications where monotonicity is expected, having a predictor failing to satisfy this requirement would damage the user's confidence. As such, different strategies have been devised in order to enable training monotonic predictors. These approaches can be divided into two main categories:

Monotonicity by construction: In this case, focus lies on defining a model class that guarantees monotonicity in all of its elements Bakst et al. [2021], Wehenkel and Louppe [2019], Nguyen and Martínez [2019], You et al. [2017], Gar-

[*]Work done while interning at Borealis AI. Currently at ServiceNow

cia and Gupta [2009], Archer and Wang [1993]. However, this approach can not be used with general architectures. Additionally, the model class can be constrained to the extent that it might affect the prediction performance.

Monotonicity via regularization: This approach is based on 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 costly.

In addition to being a *requirement* as in the examples discussed above, monotonicity has been also observed to be a useful feature in certain cases. For example, it can define an effective inductive bias and improve generalization in cases where prior knowledge indicates the data generating process satisfies such property [Dugas et al., 2001]. In such cases, however, it is not necessary to satisfy the property 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.

This work comprises two complementary sets of contributions, and in both cases 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 further detail, our contributions can be summarized as follows:

1. In Section 3, we identify a limitation in previous methods and show they only enforce monotonicity either near the training data or near the boundaries of the input space. Then, we propose an efficient algorithm that tackles this problem. In particular, we modify Mixup [Zhang et al., 2018] and use it to mix data with random noise. We show that doing so helps populate the interior of the input space. With extensive evaluation on synthetic data and benchmarks, we show that the proposed strategy enforces monotonicity in a larger volume relative to previous methods in the literature.
2. In Section 4, we define different notions of monotonicity along with regularization penalties aimed at enforcing them. We show that doing so introduces useful properties in models used for applications such as generative modeling or object recognition, and does not compromise the original performance obtained without the penalties. Contrary to the discussion on the first part of the paper in Section 3, the monotonicity property is not required to be satisfied everywhere and, as such, constraints that focus only on the actual data points are proposed.

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 differentiable functions $f : \mathcal{X} \mapsto \mathcal{Y}$, 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 $\bar{M} = \{1, ..., d\} \backslash M$.

Definition 1 *Partially monotonic functions relative to M:* We say f is monotonically non-decreasing relative to M, denoted f_M, if $\min_{i \in M} \frac{\partial f(x)}{\partial x_i} \geq 0, \forall\, x \in \mathcal{X}$.

This 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 a property. The simplest approach restricts the weights of the network to be non-negative [Archer and Wang, 1993]. However, doing so affects the prediction performance. Another approach corresponds to using lattice models [Garcia and Gupta, 2009, You et al., 2017]. In this case, models are given by interpolations in a grid defined by training data. Such a class of models can be made monotonic via the choice of the interpolation strategy and recently introduced variations [Bakst et al., 2021] scale efficiently with the dimension of the input space, but downstream applications might still require different classes of models to satisfy this type of property. For neural networks, approaches such as [Nguyen and Martínez, 2019] reparameterize fully connected layers such that the gradients with respect to parameters can only be non-negative. Wehenkel and 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 mapping parameterized by a neural network. While such approaches guarantee monotonicity by design, they can be too restrictive or give overly complicated learning procedures. For example, the approach in [Wehenkel and Louppe, 2019] requires backpropagating through the integral. An alternative approach is based on 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 to find counterexamples, i.e., pairs of points where the monotonicity constraint is violated, which are included in the training data to enforce monotonicity conditions in the next iterations of the model. However, this approach only supports fully-connected ReLU networks. Moreover, the procedure for finding the counterexamples is costly. Alternatively, Liu et al. [2020], Gupta et al. [2019] introduced point-wise regularization penalties for enforcing monotonicity, where the penalties are estimated via sampling. 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 AN EFFICIENT FIX FOR MONOTONICITY PENALTIES

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:

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

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]:

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

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 PENALTY

In the following, we present and discuss two past choices for $\mathcal{D}$:

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 B, 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.:

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

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. [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 a 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.

In summary, the previous approaches are either too focused on 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 approach 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 and Mroueh, 2021], where auxiliary data is created via interpolations of pairs of data points, to populate areas of the space that are otherwise disregarded. Mixup was introduced by Zhang et al. [2018] with the goal of training classifiers with smooth outputs 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 a variation of this approach where data-data and noise-data pairs are mixed to define points where Ω can be estimated. Algorithm 1 describes a procedure used to compute the proposed regularization Ω_{mixup}.

We 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 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.

Algorithm 1 Procedure to compute Ω_{mixup}.

```
*Input* mini-batch $X_{[N\times d]}$, model $h$, monotonic dimensions $M$
$X_\Omega = \{\}$ # Initialize set of points used to compute regularizer.
$\tilde{X}_{[N\times d]} \sim \text{Uniform}(\mathcal{X}^N)$ # Sample random mini-batch with size $N$.
$\hat{X} = \text{concat}(X, \tilde{X})$ # Concatenate data and random batches.
**repeat**
    $i, j \sim \text{Uniform}(\{1, 2, ..., 2N\}^2)$ # Sample random pair of points.
    $\lambda \sim \text{Uniform}([0,1])$
    $x = \lambda\hat{X}^i + (1-\lambda)\hat{X}^j$ # Mix random pair.
    $X_\Omega$.add($x$) # Add $x$ to set of regularization points.
**until** Maximum number of pairs reached
$\Omega_{mixup}(h, M) = \frac{1}{|X_\Omega|}\sum_{x\in X_\Omega}\sum_{i\in M}\max\left(0, -\frac{\partial h(x)}{\partial x_i}\right)^2$
**return** $\Omega_{mixup}$
```

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 report results for the following datasets: *Compas*[1], *Loan Lending Club*[2], and *Blog Feedback*[3]. In Table 1, we list details on the three datasets used to evaluate our proposals as reported in Section 3.2.

Models follow the architecture in [Liu et al., 2020] 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 reproted in appendix B. Training is carried out with the Adam optimizer [Kingma and Ba, 2014] 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} [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.
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 probability ρ of a model to *not satisfy* definition 1, which we estimate via the fraction $\hat{\rho}$ of points within a sample where the monotonicity constraint is violated; i.e., given a set of N data points, we compute:

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

such that $\hat{\rho} = 0$ corresponds to monotonic models over the considered points. Moreover, in order to quantify the degree of monotonicity in different parts of the space, we estimate ρ for 3 different sets of points: (1) $\hat{\rho}_{random}$, computed on a sample drawn according to Uniform($\mathcal{X}$). We used a sample of 10,000 points throughout the experiments. (2) $\hat{\rho}_{train}$, computed on the training data. And (3) $\hat{\rho}_{test}$: computed on the test data. Results are summarized in Table 2 in terms of both prediction performance along with the metric $\hat{\rho}$ indicating the *degree of monotonicity* of the predictor for each regularization strategy. Prediction performance is measured in terms of accuracy for classification tasks, and

[1]https://www.kaggle.com/danofer/compass

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

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

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

Table 1: Description of datasets used for empirical evaluation.

RMSE for the case of regression. Results reported in the tables represent 95% confidence intervals corresponding to 20 independent training runs. Across evaluations, different penalties do not result in significant variations in terms of prediction, but affect how monotonic trained models are.

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 $\hat{\rho}_{random}$ is consistently lower for Ω_{random} relative to Ω_{train} and Ω_{mixup} while, on the other hand, $\hat{\rho}_{train}$ and $\hat{\rho}_{test}$ 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.

4 APPLICATIONS OF MONOTONICITY PENALTIES

In Section 3, we presented an efficient approach to enforce monotonicity when it is a requirement. We now consider a different perspective and show that adding monotonicity constraints during training can yield extra benefits to trained models. In these cases, monotonicity is not a requirement, and hence it is not necessary for it to be satisfied everywhere. As such, the penalties we discuss from now on are computed considering only data points, and no random draws are utilized. In the following sections, we introduce notions of monotonicity that will be enforced in our models, and discuss advantages of using monotonicity for different applications such as controllable generative modelling and for the detection of anomalous data. In Appendix E, we consider a further application for cases where one's interest is to obtain explanations from observed predictions.

4.1 DISENTANGLED REPRESENTATION LEARNING UNDER MONOTONICITY

We first consider the case of disentangled representation learning. In this case, generative approaches often assume that the 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 specific latent variable. However, we argue that *disentanglement is necessary but not sufficient* to enable controllable data generation. That is, one needs latent variables that satisfy some notion of monotonicity to be able to decide their values resulting in desired properties.

For example, assume we are interested in generating images of simple geometric forms, and desire to control factors such as shape and size. In this example, even if a disentangled set of latent variables is available, we cannot decide how to change the value of the latent variable to get a bigger or a smaller object if there is no monotonic relationship between the size and the value of the corresponding latent variable. We address this issue and build upon the weakly supervised framework introduced by Locatello et al. [2020]. This work extends the popular β-VAE setting [Higgins et al., 2016] by introducing weak supervision such that the training instances are presented to the model in pairs (x^1, x^2) where only one or a few generative factors are changing between each pair.

Here, we propose to apply a notion of monotonocity over the activations of the corresponding latent variables to have more controlable factors. In the VAE setting, data is assumed to be generated according to $p(x|z)p(z)$ given the latent variables z. Approximation is then performed by introducing $p_\theta(x|z)$ and $q_\phi(z|x)$, both parameterized by neural networks. Our goal is to have z fully factorizable in its dimensions, i.e.:

$$p(z) = \prod_{i=1}^{Dim[z]} p(z_i), \tag{4}$$

which needs to be captured by the approximate posterior distribution $q_\phi(z|x)$. Training is performed by maximization of the following lower-bound on the data likelihood:

$$\begin{aligned}\mathcal{L}_{ELBO} = \mathbb{E}_{x^1,x^2} \sum_{i\in\{1,2\}} \mathbb{E}_{\tilde{q}_\phi(\hat{z}|x^i)} \log(p_\theta(x^i|\hat{z})) \\ - \beta D_{KL}(\tilde{q}_\phi(\hat{z}|x^i), p(\hat{z})),\end{aligned} \tag{5}$$

	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%
$\hat{\rho}_{random}$	55.45%±12.26%	0.01%±0.01%	6.41%±4.54%	0.00%±0.00%
$\hat{\rho}_{train}$	92.98%±2.70%	2.08%±2.21%	0.00%±0.00%	0.00%±0.00%
$\hat{\rho}_{test}$	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
$\hat{\rho}_{random}$	99.11%±1.70%	0.00%±0.00%	14.47%±7.55%	0.00%±0.00%
$\hat{\rho}_{train}$	100.00%±0.00%	7.23%±7.76%	0.01%±0.01%	0.00%±0.00%
$\hat{\rho}_{test}$	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
$\hat{\rho}_{random}$	76.17%±12.37%	0.05%±0.08%	3.86%±4.19%	0.00%±0.01%
$\hat{\rho}_{train}$	78.67%±5.28%	78.59%±6.37%	0.01%±0.01%	0.01%±0.01%
$\hat{\rho}_{test}$	76.29%±6.47%	78.99%±7.20%	0.02%±0.02%	0.02%±0.02%

Table 2: 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 $\hat{\rho}$ the better.

where $\tilde{q}_\phi(\hat{z}_j|x^i) = q_\phi(z_j|x^i)$ for the latent dimensions z_i that change across x^1 and x^2, and $\tilde{q}_\phi(\hat{z}_j|x^i) = \frac{1}{2}(q_\phi(\hat{z}_j|x^1) + q_\phi(\hat{z}_j|x^2))$ for those that are common (*i.e.*, the approximate posterior of the shared latent variables are forced to be the same for x^1 and x^2).

The outer expectation is estimated by sampling pairs of data instances (x^1, x^2) where only a number of generative factors vary. In our experiments, we consider the case where exactly one generative factor changes across inputs. Moreover, we follow Locatello et al. [2020] and assign the changing factor, denoted by y, to the dimension j of z such that:

$$y = \underset{j \in Dim[z]}{\arg\max}\, D_{KL}(z_j^1, z_j^2). \tag{6}$$

While the above objective enforces disentanglement, controllable generation requires some regularity in z so that users can decide values of z resulting in desired properties in the generated samples.

To account for that, we then introduce Ω_{VAE} to enforce such a regularity. In this case, a monotonic relationship is enforced for the *distance between data pairs where only a particular generative factor vary and a corresponding latent variable*. In other words, an increasing trend in the value of each dimension of z should yield a greater change in the output along a generative factor. Formally, Ω_{VAE} is defined as the following symmetric cross-entropy estimate:

$$\begin{aligned}\Omega_{VAE} = &-\frac{1}{2m}\sum_{i=1}^{m}\log\frac{e^{\frac{L(x^{i,1},x^{i,2},y^i)}{\mu}}}{\sum_{k=1}^{K} e^{\frac{L(x^{i,1},x^{i,2},k)}{\mu}}} \\ &+\log\frac{e^{\frac{L(x^{i,2},x^{i,1},y^i)}{\mu}}}{\sum_{k=1}^{K} e^{\frac{L(x^{i,2},x^{i,1},k)}{\mu}}},\end{aligned} \tag{7}$$

where L is given by the gradient of the mean squared error (MSE) between images that are 1-factor away along the dimension y of z, assigned to the changing factor, i.e., for the pair x^i and x^j varying only across factor y, we have:

$$L(x^i, x^j, y) = \frac{\partial \text{MSE}(\hat{x}^i, x^j)}{\partial \tilde{z}_y}. \tag{8}$$

In this case, $\hat{x}^i$ indicates the reconstruction of x^i. We evaluate such an approach by training the same 4-layered convolutional VAEs described in [Higgins et al., 2016] using the 3d-shapes dataset[4]. 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 over current approaches in terms of *how disentangled* the learned representations are. Rather, we seek to achieve similar results in that sense, but impose extra regularity and structure in the

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

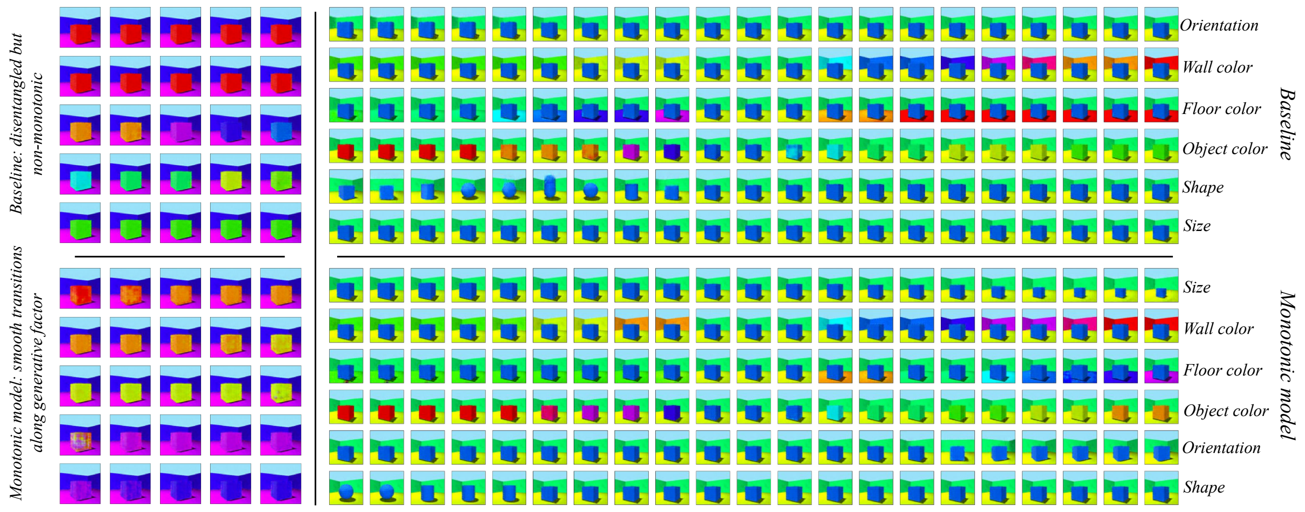

Figure 1: Comparisons between data generated by standard and monotonic models. On the two panels on the left, we compare generations from a linear combination of the latent code of 2 images which only differs in the object color. On the two panels vertically stacked on the right, we start from the same image but change one latent dimension at a time.

relationship between the generated images and the values of z so that the generative process is more easily *controllable*.

Qualitative analysis is performed and shown in Figure 1. The two panels on the left represent the data generated by a linear combination of the latent code corresponding to two images that only vary in the factor *object color*. The panels stacked on the right present a per-dimension traversal of the latent space starting from a common image. It can be observed that disentanglement is indeed achieved in both cases. The monotonic model presents much smoother transitions between colors while the base model gives long sequences of very close images followed by very sharp transitions where the colors sometimes repeat (e.g., green-yellow-green transitions in the fourth row).

As for the results per factor, the monotonic model provides more structure in the latent space compared to the base model. This can be observed in the shape factor. The monotonic model provides a certain order: sphere, cylinder, and then cube. Visually inspecting many samples, the monotonic model is following this order for the generated shapes. This pattern is even more pronounced in the color factors. We have found that the colors generated by the monotonic model follows the order of the colours in *the HUE cycle*. So our model has ordered the latent space and we know how to navigate it to generate a desired image. On the other hand, the baseline has no clear order of the latent space. For example, the baseline generates cubes at different ranges of z. Similarly, the colors generated by the baseline model do not have a clear order.

To further support the claim that Ω_{VAE} induces regularity in the latent space, we introduce the analysis shown in Table 3. We started by increasing z_3 (associated to *floor color* for both models), and recorded the sequence of the generated colors. We observed that for a large fraction of the data, the monotonic models yield sequences of images where the color of the floor is ordered according to its corresponding HUE angle. Further details are available in Appendix G along with detailed plots of color transitions and a comparison with the HUE cycle.

Model	*HUE structured rate*
Base model	0.00%
Mon. model	89.44%

Table 3: rate of examples where colors are sorted according to hue. A large amount of the sequences generated by monotonic VAEs result in interpretable ordering.

4.2 GROUP MONOTONIC CLASSIFIERS

We now consider the case of K-way classifiers realized through convolutional neural networks. In this case, data examples correspond to pairs $x, y \sim \mathcal{X} \times \mathcal{Y}$, and $\mathcal{Y} = \{1, 2, 3, ..., K\}$, $K \in \mathbb{N}$. 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 the models h such that the outputs corresponding to each class satisfy a monotonic relationship with a specific subset of high-level representations, given by some inner convolutional layer. Intuitively, our goal is to “reserve" groups of high-level features to activate more intensely than the remainder depending on the underlying class. Imposing such a structure can benefit the learned models via, for instance, more accurate anomaly detection.

Let the outputs of a specific layer within a convolutional model be 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 consider the rather common case of convolutional layers where each feature map a_w is 2-dimensional. We then partition such a set of representations into disjoint subsets, or *slices*, of uniform sizes. Each subset is then paired with a particular output or class, and hence denoted by S_k, $k \in \mathcal{Y}$. An illustration is provided in Figure 2, where a generic convolutional model has the outputs of a specific layer partitioned into slices S_k, which are then used to define output units over $\mathcal{Y}$.

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

For training, we perform monotonic risk minimization as described in Eq. 1, and the risk is given by the negative log-likelihood over training points. Moreover, we design a penalty Ω that focuses only on observed data points during training and penalizes the slices of the Jacobian corresponding to a given class, i.e., a cross-entropy criterion enforces larger gradients on the specific class slice. We highlight that in this case, unlike the discussion in Section 3, monotonicity *is not* an application requirement, and it does not need to be satisfied everywhere.

In order to formally introduce such a penalty, denoted by 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, a batch of size m, and inverse temperature μ, Ω_{group} will be:

$$\Omega_{group} = -\frac{1}{m} \sum_{i=1}^{m} \log \frac{e^{\frac{O^i_{y^i}(x^i)}{\mu}}}{\sum_{k=1}^{K} e^{\frac{O^i_k(x^i)}{\mu}}}. \quad (9)$$

4.2.1 Assessing performance of group monotonic classifiers

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, and classifiers in each case correspond to WideResNets [Zagoruyko and Komodakis, 2016] and ResNet-50 [He et al., 2016], respectively. Training details are presented in Appendix C.

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 4: Top-1 accuracy of standard and group monotonic models.

Results are reported in Table 4 in terms of the top-1 prediction accuracy measured on the test data. We use standard classifiers as the baselines where no monotonicity penalty is applied in order to isolate the effect of the penalty. In both datasets, the total activation classifiers for group monotonic models (indicated by the prefix *mono*) are able to approximate the performance of 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.

Considering performances obtained at the output layer, there were small variations in accuracy when we included monotonicity penalties, which should be considered in practical uses of group monotonicity. Nonetheless, results suggest that one can perform closely to unconstrained models while focusing on the set of group monotonic candidates.

Additional experiments are reported on Table 3 on Appendix D for cases with small sample sizes, where we show that the performance of the classifier defined at the output layer upper bounds that of the total activation classifier, i.e., *the better the underlying classifier the more group monotonic it can be made*.

4.2.2 Using group monotonicity to detect anomalies

After showing that group monotonicity can be enforced successfully without significantly affecting the prediction performance, we discuss approaches to leverage it and introduce applications of the models satisfying such a property. In particular, we consider the application of detecting anomalous data instances, i.e., those where the model may have made a mistake. For example, consider the case where a classifier is deployed to production and, due to some problem external to the model, it is queried to do prediction for an input consisting of white noise. Standard classifiers would provide a prediction even for such a clearly anomalous input. However, a more desirable behavior is to somehow indicate that the instance is problematic. We claim that imposing structure in the features, e.g., by enforcing group monotonicity, can help in deciding when not to predict.

To evaluate the proposed method, we implement anomalous test instances using 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 $T_\mathcal{Y}(x)$:

$$H^*(x) = \frac{\sum_{k \in \mathcal{Y}} p_k(x) \log p_k}{\log K}, \tag{10}$$

where $K = |\mathcal{Y}|$ and the set $p_\mathcal{Y}(x)$ corresponds to the parameters of a categorical distribution defined by:

$$p_\mathcal{Y}(x) = \text{softmax}(T_\mathcal{Y}(x)). \tag{11}$$

Decisions can then be made by comparing $H^*(x)$ with a threshold $\tau \in [0, 1]$, defining the detector $\mathbb{1}_{\{H^* > \tau\}}$.

We evaluate the detection performance of this approach on both MNIST and CIFAR-10. Training for the case of CIFAR-10 follows the same setup discussed on Section 4.2.1. 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. 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 $\frac{8}{255}$ for the cases of MNIST and CIFAR-10, respectively. Detection performance is reported in Table 5 for the considered cases in terms of the area under the operating curve (AUC-ROC).

The baselines are the models for which the monotonicity penalty is not enforced. They are trained under the same conditions and the same computation budget as the models where the penalty is enforced. The results are as expected, i.e., for monotonic models, test examples for which the total activations are not structured very often correspond to anomalous inputs.

Finally, due to space constraints, we discuss the application of group monotonicity to explainability in appendix E. The implementation of our empirical evaluation is available at: `https://github.com/BorealisAI/monotonicity-mixup`.

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

Table 5: AUC-ROC (the higher the better) for the detection of adversarially perturbed data instances.

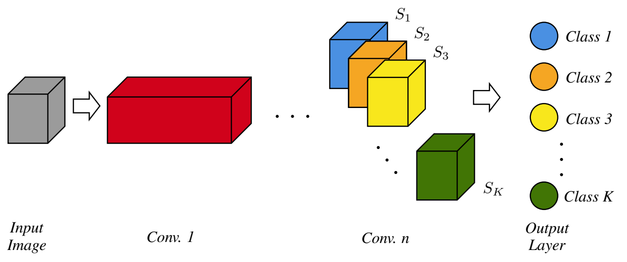

Figure 2: Group monotonic convolutional model splits representations into disjoint subsets.

5 CONCLUSION

We proposed approaches that 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 prior work that resulted in models satisfying the property only in very specific parts of the space.

We then introduced an efficient 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, we further argued that, even when not required, *models satisfying monotonicity present useful properties*. We studied the case of image classifiers and generative models and showed that imposing structure in learned representations via group monotonicity is beneficial and can be done efficiently. In particular, monotonic variational autoencoders were shown to yield latent spaces that are easier to navigate since those present more regular transitions when compared to the standard generative models under the same setting.

Acknowledgements

We would like to thank Amir Abdi and Gavin W. Ding for their help and insightful discussions. We also thank Masoud Hashemi for his support in preparing our code for open-source release.

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`.

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*, pages 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, editors, *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*, pages 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.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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.

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, 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`.

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, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 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.

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, 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`.

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*, pages 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, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/2a084e55c87b1ebcdaad1f62fdbbac8e-Paper.pdf`.

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.

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`.