DOI: 10.1111/cgf.15109
Eurographics Conference on Visualization (EuroVis) 2024
W. Aigner, D. Archambault, and R. Bujack
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 3

Exploring Classifiers with Differentiable Decision Boundary Maps

A. Machado[1], M. Behrisch[1], A. Telea[1]

[1]Department of Information and Computing Sciences, Utrecht University, Netherlands

Abstract

Explaining Machine Learning (ML) — and especially Deep Learning (DL) — classifiers' decisions is a subject of interest across fields due to the increasing ubiquity of such models in computing systems. As models get increasingly complex, relying on sophisticated machinery to recognize data patterns, explaining their behavior becomes more difficult. Directly visualizing classifier behavior is in general infeasible, as they create partitions of the data space, which is typically high dimensional. In recent years, Decision Boundary Maps (DBMs) have been developed, taking advantage of projection and inverse projection techniques. By being able to map 2D points back to the data space and subsequently run a classifier, DBMs represent a slice of classifier outputs. However, we recognize that DBMs without additional explanatory views are limited in their applicability. In this work, we propose augmenting the naive DBM generating process with views that provide more in-depth information about classifier behavior, such as whether the training procedure is locally stable. We describe our proposed views — which we term Differentiable Decision Boundary Maps — over a running example, explaining how our work enables drawing new and useful conclusions from these dense maps. We further demonstrate the value of these conclusions by showing how useful they would be in carrying out or preventing a dataset poisoning attack. We thus provide evidence of the ability of our proposed views to make DBMs significantly more trustworthy and interpretable, increasing their utility as a model understanding tool.

CCS Concepts

• ***Human-centered computing*** → *Visualization techniques;* • ***Computing methodologies*** → *Machine learning;* • ***Mathematics of computing*** → *Dimensionality reduction;*

1. Introduction

Communicating how a classifier behaves in all its complexity is a challenging problem, and one that has received increasing attention over recent years. A classifier f is a model trained with the objective of accurately attributing categorical information to some input data. While simple and explainable classifiers exist — *e.g.*, Logistic Regression, Decision Trees — a vast share of current research focuses on techniques more suited to model (increasingly) complex data, such as Deep Learning (DL). These techniques are knowingly hard to dissect, which is directly at odds with the need to understand the classifier's behavior as a crucial way to implement improvements, foster trust [vdEAA*23], and debug failure modes.

This calls for several types of tools and methodologies to shine light on this challenging object of study. While using numeric aggregate metrics — such as accuracy, F-measure, area under ROC curve — is a valid approach to evaluate a classifier, it provides little in the sense of *exploring* the classifier's behavior and how it changes across different regions of the data space. Metrics typically allow an investigator to have either a global view — each metric outputs a single score for the classifier as a whole — or an unaggregated, per data point, view. This introduces a disconnect in the pipeline: classifiers are *total* functions, able to output classifications for any point in the training data space. On the other hand, data points only *sparsely* sample the data space, making any approach that relies solely on them limited by design on the type of insights it can produce: it cannot say what happens in the (quite large) unsampled portions of the data space.

Decision Boundary Maps (DBM [MRHT18, SHH20]) are a family of techniques that partially lifts this limitation, allowing for denser inspection of f's behavior across large portions of the data space. A DBM — see Figure 1 — provides a dense map-like view of a classifier's behavior by mapping points $\mathbf{p} \in \mathbb{R}^2$ back to the original data space $\mathbb{R}^n$ — a process called inverse projection — and next applying f on the inverse-projected data. The categorical output of f determines the color associated with point $\mathbf{p}$. All such techniques generate images similar to the ones in Fig. 1 — a running example we refer to throughout the text. These convey typically the "arg max" output of the classifier (Fig. 1, left) or can be easily augmented with confidence information, here encoded in the luminance channel (Fig. 1, right). We argue that these maps, without additional information, are hardly interpretable: they show *something* about the classifier, evidently, but are generated through a process that might *itself* introduce artifacts in the visualization. In order to be able to derive knowledge from DBMs, we see the need for extra information such that they become more usable and therefore useful. We point out a few shortcomings of vanilla DBMs:

- Even in regions where the DBM shows the classifier is highly confident, it does not show whether f is classifying data too far removed from any training data point, meaning it's *wrongly* confident of its output;
- Different regions in the DBM look qualitatively similar in appearance — possibly even when confidence is encoded in the map — but the classifier and inverse projection might behave quite differently;
- 2D distances in a DBM do not accurately represent n-D distances since the inverse projection is in general nonlinear;
- Neighborhood relations in 2D might not neatly correspond to neighborhoods in the data space.

In summary, we argue that the DBM, although useful, *masks* aspects of the high dimensional and nonlinear nature of both inverse projection and classifier that are interesting for ML practitioners. We propose *augmenting* DBMs with the goal of enabling them to additionally answer the following questions:

Q1: In which regions of the DBM is the confidence of the classifier f *misleading*?
Q2: Where is the model poorly *supported*?
Q3: Where is the DBM *distorting* data space distances?
Q4: Where is the training of f *sensitive*?
Q5: Why is f's confidence low/high in specific regions of space?

We demonstrate that most of these questions can be phrased in terms of specific types of sensitivity analysis. In other words, we can think about them as asking "what if this aspect were to be perturbed?" It is not surprising then that the tool of choice for this problem is the *derivative*. Differentiation gives us precisely that: the amount of change in outcome effected by a small perturbation in input. The majority of the interactive views proposed in section 3 are created in this fashion, leading us to term them **Differentiable** Decision Boundary Maps (∂DBM).

The main contributions of this paper are as follows:

- ∂DBMs, a novel set of interactive views aimed at improving the usability of DBMS, relying on differentiable f and P^{-1}.
- A batch-wise implementation of the adversarial example generation algorithm DeepFool (see section 3.2), allowing its concurrent application to thousands of data points.
- An algorithm for deriving an approximate *direct* projection from P^{-1} without any extra training, described in section 4.1.
- The demonstration of the potential of these new tools in an end-to-end dataset poisoning use case.

We implement ∂DBMs in a Python tool and make our code publicly available at `https://git.science.uu.nl/vig/adversarial-dbm-tool`. We confirm that, in this paper, we have reported all measures, conditions, and data exclusions, and dataset/sample sizes, that may be applicable.

In the next sections, we elaborate on these questions and propose answers, which we believe strongly expand the DBM approach for model explanation. We cover background and related work in section 2 and describe our Differentiable Decision Boundary Maps (∂DBM) in section 3. We also show a possible use case for ∂DBMs in section 4 and discuss avenues for future work in section 6.

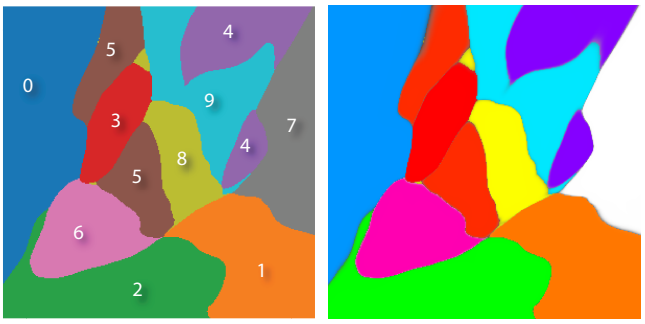


Figure 1: *Left: Plain DBM for a Neural Network classifier over the MNIST dataset, generated with NNInv (from a t-SNE projection). Each pixel* $\mathbf{p}$ *is colored according to* $f(\mathbf{p}^\dagger)$. *Each resulting zone is annotated with its respective class. Right: the same DBM augmented with classifier confidence encoded in luminance. This is our running example for the rest of the text. In section 3 we show how our ∂DBMs augment this shallow visualization with additional information.*

2. Background and Related Work

We introduce a few notations used throughout this work. Let $\mathcal{D} = \{\mathbf{x}_i\}_{i=1,\dots,m}$ be a dataset of samples $\mathbf{x}_i \in \mathbb{R}^n$. When available, a training sample's label y_i can be queried through the function $Y : \mathcal{D} \to \mathcal{C}$, with $|\mathcal{C}| = K$ classes. We denote a classifier by $f_\theta : \mathbb{R}^n \to \mathcal{C}$. A projection is a function $P_\theta : \mathbb{R}^n \to \mathbb{R}^q$ that maps a high-dimensional point in $\mathbb{R}^n$ to a low-dimensional point (in this work, we consider $q = 2$). An inverse projection $P_\theta^{-1} : \mathbb{R}^q \to \mathbb{R}^n$ performs the opposite mapping. Importantly, note that P^{-1} is not strictly speaking the mathematical inverse of P as, in general, P is not injective. We will use $\mathbf{p}, \mathbf{p}_0$ to refer to pixels in a DBM, and $\mathbf{p}^\dagger$ as shorthand for $P^{-1}(\mathbf{p})$. The parameters θ are omitted in our discussion when acceptable. We represent derivatives, gradients, and Jacobian matrices by abusing the $\frac{\partial \cdot}{\partial \cdot}$ notation as

$$\mathcal{J}P^{-1} \doteq \begin{bmatrix} (\nabla_{\mathbf{p}} P_1^{-1})^T \\ (\nabla_{\mathbf{p}} P_2^{-1})^T \\ \vdots \\ (\nabla_{\mathbf{p}} P_n^{-1})^T \end{bmatrix} = \frac{\partial P^{-1}}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial P_1^{-1}}{\partial \mathbf{p}} \\ \frac{\partial P_2^{-1}}{\partial \mathbf{p}} \\ \vdots \\ \frac{\partial P_n^{-1}}{\partial \mathbf{p}} \end{bmatrix} = \begin{bmatrix} \frac{\partial P_1^{-1}}{\partial p_1} & \frac{\partial P_1^{-1}}{\partial p_2} \\ \frac{\partial P_2^{-1}}{\partial p_1} & \frac{\partial P_2^{-1}}{\partial p_2} \\ \vdots & \vdots \\ \frac{\partial P_n^{-1}}{\partial p_1} & \frac{\partial P_n^{-1}}{\partial p_2} \end{bmatrix}.$$

We will also refer to the Euclidean norm of a vector $\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$ and the Frobenius norm of a matrix $\|\mathbf{A}\|_{\text{Fro}} = \sqrt{\sum_{i,j} a_{ij}^2}$.

2.1. Gradient-based Explanations for Classifiers

Developing algorithms and techniques to explain the behavior of classifiers is an active area of research [AS22, RBB*23, AASA21, SMV*19]. This applies especially to so-called Deep Learning techniques which are notorious for their "black-box" aspect: the models thus learned consist of sets of opaque numbers arranged in vectors and matrices and combined through special operations. When deploying such models, practitioners and stakeholders alike can benefit from ways to *probe* it, shining light into their decision-making process, as a way to mitigate risk and assess quality.

Out of all the possible ways to develop such techniques, we devote special attention to those using derivatives (*e.g.*, gradients, Jacobian matrices) to generate explanations for classifications.

In the context of Convolutional Neural Networks (CNNs), which are models of choice when dealing with *e.g.* image data, many techniques focus on providing explanations directly relatable to the data space. Chattopadhyay *et al.*'s Grad-CAM++ [CSHB18] use gradients to derive visualizations of where the "gaze" of the classifier is directed to which are then overlaid atop the classified image. Similarly, Pan *et al.* [PLZ21] create maps of feature activations — *i.e.*, which pixels of an image are important for a classification decision — by integrating gradients along paths starting from adversarial examples. Such approaches aim to add explanations that relate directly to the data *points*. In this work, we argue for the utility of explanations ranging over a large portion of the data *space*. For this, we rely on projections (P) and their inverses (P^{-1}) to create 2-D decision maps, as others have done before us [SHH20, OEHJT22]. Further, we show gradient-based approaches to enrich these decision maps, and demonstrate their increased usefulness.

2.2. Projections, Inverse Projections, and Decision Maps

Projection algorithms are formally functions

$$P_\theta : \mathbb{R}^n \to \mathbb{R}^q \tag{1}$$

where $q \ll n$ is the dimension of the projected space, usually represented as a scatterplot with $q = 2$ or 3. In this work, since we focus on Decision Boundary Maps, we use $q = 2$ throughout. Projection algorithms aim to take a given dataset $\mathcal{D}$ and produce a projection $P(\mathcal{D})$ preserving important data patterns.

Tens of projection algorithms exist which differ in the characteristics of the data $\mathcal{D}$ they aim to preserve in $P(\mathcal{D})$, whether they work locally or globally and linearly or not, and how they optimize their related cost functions. Well-known examples hereof are PCA [Jol02] (global, linear optimization), t-SNE [vdMH08] (local, nonlinear optimization), and UMAP [MHM18] (similar to t-SNE, but using a different cost and optimization scheme). For additional details, we refer to relevant surveys in the area [SVPM14, HFA17, NA18, EMK*19].

Projections enable not only the visualization of high-dimensional data $\mathcal{D}$ but also of several objects that operate on that data. Consider a classifier f trained on some dataset $\mathcal{D}$ for which we also have a projection $P(\mathcal{D})$. One can then visualize the output of f in the same projection plot by coloring each projected point $P(\mathbf{x}_i)|\mathbf{x}_i \in \mathcal{D}$ according to $f(\mathbf{x}_i)$. This yields a *sparse* 2D visualization of the behavior of f at the samples $\mathbf{x}_i$. However, this gives no information for points outside $\mathcal{D}$. Differently put, we do not know what f is like for the white space areas in $P(\mathcal{D})$.

Different techniques of *inverse projection* have been proposed to bridge this gap. An inverse projection P^{-1} is a function

$$P_\theta^{-1} : \mathbb{R}^q \to \mathbb{R}^n \tag{2}$$

designed to minimize a reconstruction error $\|P^{-1}(P(\mathbf{x})) - \mathbf{x}\|$ for $\mathbf{x} \in \mathcal{D}$. Note that the term "inverse" is abused here: P^{-1} is typically not an exact inverse function in the mathematical sense — since P usually is not injective —, but an approximation thereof. Having such a P^{-1}, one can pick an as fine as desired regular grid of points $\mathbf{p}$ in $\mathbb{R}^2$. Coloring each such pixel $\mathbf{p}$ by the value $f(P^{-1}(\mathbf{p}))$ yields a so-called decision boundary map (DBM). Same-color compact areas in a DBM indicate so-called decision zones where f yields the same output (class); neighbor pixels of different colors indicate decision boundaries, where f changes output. Additionally, color brightness (or saturation) can encode the classifier's confidence.

Several techniques of DBM generation exist. They mainly differ on exactly *how* P^{-1} is derived and used. iLAMP [ABD*12] inverts the LAMP [JCC*11] projection technique by local interpolation using radial basis functions. In DeepView [SHH20], the invertibility of UMAP [MHM18] means P^{-1} is enabled by design. SDBM [OEHJT22] works similarly by deep learning P and P^{-1} by an autoencoder design. NNInv [ERH*19] also uses deep learning, but a different architecture than SDBM, to learn P^{-1} for any user-provided P. We use NNInv for DBM generation throughout this work due to its genericity and differentiability.

The insights obtained from a DBM depend, obviously, on the errors that P and P^{-1} potentially introduce. Several metrics exist to measure projection errors both locally (for point neighborhoods) and globally [VK06, Aup07, LA11, JCC*11, MMT15]. A detailed comparison of projection errors is provided in [EMK*19]. For inverse projections, one typically does not have ground truth data for points outside the dataset $\mathcal{D}$ used to construct P^{-1}. As such, P^{-1} is typically assessed by the mean square or absolute error of the reconstruction error. Errors of P can be used to filter out poorly projected points to construct better P^{-1} mappings [REJT19].

A separate challenge involves the *stability* of direct (and inverse) projections. Simply put, if small changes in a dataset yield large changes in P and/or P^{-1}, then both these mappings and the corresponding DBMs are prone to misinterpretation since the details they show may be artifacts of some small-scale data noise. Stability metrics have been proposed to analyze the behavior of projections of time-dependent data [VGdS*20], respectively the robustness *vs* different noise types for a single deep-learning-based projection technique [BTT22]. However, such analyses have not been used to help the interpretation of DBMs. Oliveira *et al.* [OEJT23] have studied the stability of SDBM by visualizing its changes subject to different noise types (similar to [BTT22]). However, their approach did not consider other P or P^{-1} techniques. More importantly, such visualizations consider artificial data changes (noise). In our work, we consider (and visualize) a more realistic scenario, namely how sensitive a classifier is to mislabeled samples.

3. Differentiable Decision Boundary Maps

We claim that a deeper understanding of the data and classifier can be obtained by augmenting existing DBMs with additional visualizations. We propose and argue for five such views, most of them obtained by using gradient-based methods. The differentiability of both classifier and inverse projection is crucial in our work, hence we call our proposed views *Differentiable* Decision Boundary Maps. These views can be directly overlaid atop a given DBM and also combined with each other to answer Q1-Q5 from Sec. 1. We included numerical scales, aside the color-mapped images, for views where reasoning about the *absolute* values is important; for views which support reasoning about *relative* values, we did omit such

scales. Our tool always provides tooltips to investigate the exact values at any pixel in any view.

3.1. Classifier Confidence

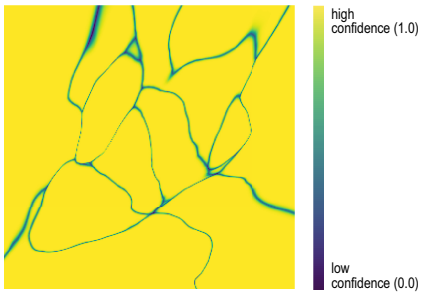


Figure 2: *Each pixel* $\mathbf{p}$ *shows the probability assigned to the best class for* $\mathbf{p}^\dagger$. *We see a classifier that seems uniformly confident of its predictions throughout the represented space.*

As already explained, a plain DBM can be overlaid with the classifier confidence — understood as either the probability assigned to the predicted class or the entropy of the probability distribution over classes — typically encoded into brightness or saturation. Figure 2 shows the confidence of the classifier whose DBM was already shown in Fig. 1 using color coding. If we are to trust this figure, the classifier appears extremely confident for the vast majority of the points in all decision zones (yellow) except in the direct vicinity of the decision boundaries (green bands). As we shall see, using our additional views tells a different story.

3.2. Distance to Decision Boundary

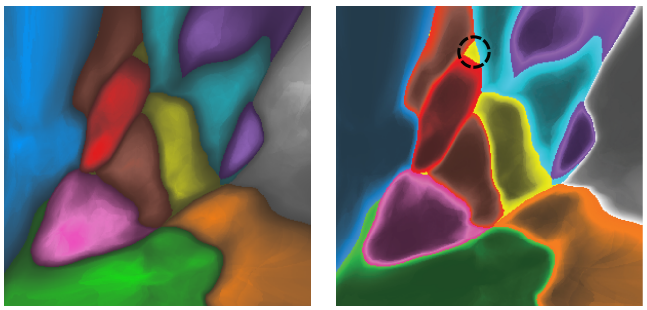
Our first DBM enhancement is to show, for each pixel, the distance of the sample $\mathbf{p}^\dagger$ it represents to the actual classifier decision boundaries in data space. This, combined with the confidence map (Figure 2) aims to answer Q1. We compute this distance as

$$\mathrm{Dist}_{\mathrm{Adv}}(\mathbf{p}) = \min_{\Delta\mathbf{x}\in\mathbb{R}^n} \left\{ \|\Delta\mathbf{x}\|_2 \;\middle|\; f(\mathbf{p}^\dagger + \Delta\mathbf{x}) \neq f(\mathbf{p}^\dagger) \right\}. \quad (3)$$

Computing $\mathrm{Dist}_{\mathrm{Adv}}$ exactly is complex. However, techniques for Adversarial Example Generation can approximate it well. In our work, we use DeepFool [MFF16] for this goal. Alternative approaches to compute this distance have been proposed in [EAS*23] and [REJT19]. However, these approaches use iterative numerical approximation techniques to search for the closest decision boundary point in the data space, which are sensitive to parameter setting and very slow. DeepFool runs considerably faster and is a more principled approach for the same goal. We also extend the original DeepFool implementation to work on batches of data, concurrently generating adversarial examples for thousands of data points at a time. Overall, we can compute maps of 300^2 pixels in less than a second on a commodity PC (for more detailed performance data, see supplemental material).

Figure 3a shows the distance to decision boundary computed by Eqn. 3 for our running example with luminance encoding. Dark areas correspond to low distances; bright areas indicate points that are far away from the decision boundaries. This map focuses attention on points far away from the boundaries. Alternatively, to focus on points close to the boundaries, which have a higher likelihood to be misclassified upon small data changes or classifier hyperparameter tuning, we can use an inverse luminance mapping (Fig. 3b). Pixels close to boundaries are now bright. Comparing these images with Fig. 1 or the separately-visualized confidence in Fig. 2, it becomes clear that the distance from a pixel $\mathbf{p}$ to the closest boundary in the DBM, *i.e.*, closest DBM pixel of a different color than $\mathbf{p}$, is *not* reflective of the actual distance between the data point $\mathbf{p}^\dagger$ and its closest decision boundary *in data space*. Our maps in Fig. 3a show that the data-space distance varies within each DBM region in complex ways. Moreover, while the confidence visualization (Fig. 2) suggests that all pixels away from the decision boundaries are qualitatively equal (high classifier-assigned confidence), our distance-to-boundary visualization (Fig. 3a) exposes hidden qualitative differences. Some entire regions in the map appear to be completely 'brittle' in the sense that they have quite low distances to decision boundaries in data space (see circled region in Fig. 3b).

These observations immediately lead us to test what would happen if we introduced mislabeled data points in such fragile regions. For this, we select 10 points $\mathbf{p}_i$ in a given fragile — *i.e.*, low $\mathrm{Dist}_{\mathrm{Adv}}$ — region, assign wrong labels to them, and add their inverse-projected $\mathbf{p}_i^\dagger$ mislabeled samples to the original training set of our classifier, which contains 5 thousand data points. We keep the number of mislabeled points low to avoid introducing class imbalance to the data. Figures 4(b,c) and (d,e) show two such experiments, each using another fragile region. In both cases, we see that it is *easy* to visibly modify the classifier's decision zones and, thus, behavior. Hence, visualizing such brittle regions is useful to inform classifier designers of areas in data space where the classifier is easily influenced.



(a) *Highlighting safe regions (high* Dist_{Adv}*)*

(b) *Highlighting brittle regions (low* Dist_{Adv}*).*

Figure 3: *Visualizations of approximate distance to the closest decision boundary in data space. Highlights (bright areas) show points that are (a) far away from, respectively (b) close to boundaries.* Dist_{Adv} *values are scaled to* $[0,1]$ *before visual encoding.*

3.3. Distance to Training Data

Any classifier is subject to generalization problems [ZLQ*23] — the further a sample is from the training set, the higher is the likelihood that that sample will be misclassified. This is fundamentally

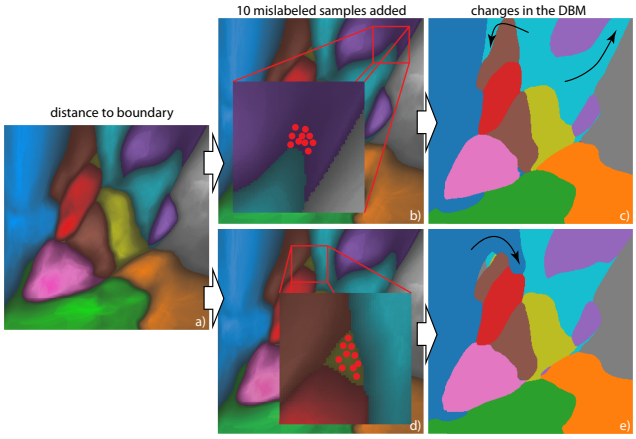


Figure 4: *Finding brittle regions with low $Dist_{Adv}$ values (a). Adding mislabeled data points in two such regions (b,d) has a strong effect on the DBM. In case (b), the cyan region, class 9, expands so that it fully separates the purple and gray top-right regions (c). In case (d), the blue region of class 0 wraps around to form a single connected decision zone with the new data points (e).*

a question of *support* (Q2) that current DBMs do not depict. We address this by visualizing the actual distance from the sample corresponding to each DBM pixel $\mathbf{p}$ and the closest point in the training set $\mathcal{D}_t$ as

$$\mathrm{Dist}_{\mathrm{train}}(\mathbf{p}) = \min_{\mathbf{x}\in\mathcal{D}_t} \|\mathbf{p}^\dagger - \mathbf{x}\|_2. \tag{4}$$

Figure 5a shows $\mathrm{Dist}_{\mathrm{train}}$ for our running example encoded on a blue-to-yellow colormap (blue=small, yellow=large, distances). This view exposes regions of the DBM where, although the classifier might have high confidence values (see Fig. 2), it is operating on samples quite different from those on which it was trained. Especially salient is the dark blue area (Figure 5a, bottom right): the values of $\mathrm{Dist}_{\mathrm{train}}$ are consistently low for that region which corresponds to class 1 (orange in Fig. 1 left). This is because all pixels $\mathbf{p}$ in that region back-project close to the training set, meaning that P^{-1} "captures" that region well. Class 1 represents the digit 1 in MNIST, which is the simplest and least-varying shape in the dataset, which in turn is easy to capture. There is further evidence to the above observations. We see that the training samples do not cover the entire region — there are no white dots to the extreme right of the dark blue zone in Figure 5a. Still, this area is overall dark, meaning that P^{-1} "reaches" close to the training data for the entire region, even from points distant from $P(\mathcal{D}_t)$.

We can be further more specific and ask if a given data region has enough support to be classified as a given class $c \in \mathcal{C}$. We expect good support if a given pixel $\mathbf{p}$ is classified as c *and* is close to some training points of class c in the data space. To check this, for a pixel $\mathbf{p}$ that has label (color) c, we compute the distance of the data point corresponding to $\mathbf{p}$ to the closest training-set point of class c as

$$\mathrm{Dist}_{\mathrm{SameClass}}(\mathbf{p}) = \min_{\mathbf{x}\in\mathcal{D}_t} \left\{ \|\mathbf{p}^\dagger - \mathbf{x}\| \;\middle|\; Y(\mathbf{x}) = f(\mathbf{p}^\dagger) \right\} \tag{5}$$

Figure 5b shows $\mathrm{Dist}_{\mathrm{SameClass}}$ for our running example. We see

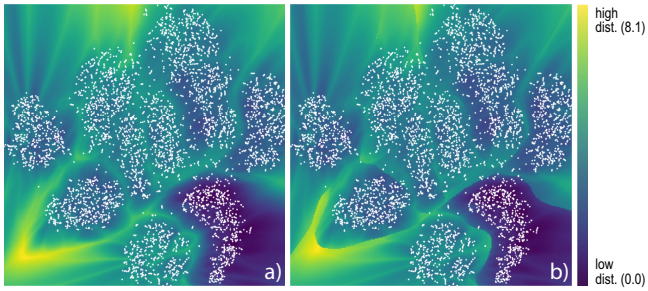


Figure 5: *Distance to any training point $Dist_{train}$ (a) and to training points in the same class as $\mathbf{p}^\dagger$, $Dist_{SameClass}$ (b). White points show the training set $\mathcal{D}_t$.*

that this distance is subtly different from $\mathrm{Dist}_{\mathrm{train}}$ shown in Figure 5a. Again, the bottom-right region is dark blue, indicating strong support in that region for class 1. The delineation of the dark blue region is now, however, very sharp, matching very well the shape of the orange decision zone (class 1) in Figure 1 left. We also see a clearly delineated yellow region bottom-left in Figure 5b. This tells that the respective region, corresponding to the border of classes 2 and 6 (green and pink, Figure 1 left), is likely a too far-away extrapolation of the class-2 training. Again, the plain confidence visualization (Figure 1 right) does not show us such insights.

3.4. Projected Space Expansion

We next propose several views based on metrics computed by differentiating specific components of the DBM generation process. This requires introducing some additional notation. We assume we are dealing with a classifier f that has some *internal* scoring that is then transformed (say, by a softmax function) into a probability distribution over classes.

As mentioned in Sec. 2, it is possible that P^{-1} itself introduces artifacts in the DBM. We propose a way to visualize how much P^{-1} expands the 2D space locally to be able to invert P with low error rate, therefore answering Q3. Espadoto et al. [EAS*23] computed this via an approximate pseudo total derivative called Gradient Maps. In contrast, we use the *exact* derivative obtained through automatic *symbolic* differentiation by

$$\mathrm{SpaceExpansion}(\mathbf{p}_0) = \left\| \frac{\partial P^{-1}}{\partial \mathbf{p}}\bigg|_{\mathbf{p}=\mathbf{p}_0} \right\|_{\mathrm{Fro}}, \tag{6}$$

where we assume that P^{-1} is differentiable with respect to its input. High values of this metric indicate regions where P^{-1} strongly expands the 2D space — that is, close pixels in the DBM map to far-away points in the data space. These are regions where small-scale details in the DBM, *e.g.* the presence of decision boundaries, are very uncertain or even misleading, since zones lying close to each other in the 2D image are *not* neighbors in the data space.

Figure 6 shows this space expansion computed by Gradient Maps (a), respectively our method (b). We note that our method is visibly less blurry than Gradient Maps since the latter uses finite difference approximations which introduce approximation errors that smooth out finer detail, whereas we compute the exact norm of the Jacobian

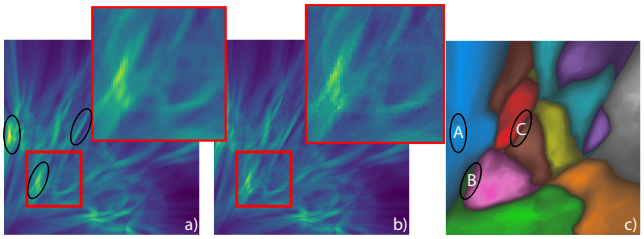


Figure 6: *Space expansion of P^{-1} visualized by (a) Gradient Maps [EAS*23], respectively our method (b). Our technique shows more sharp details than Gradient Maps.*

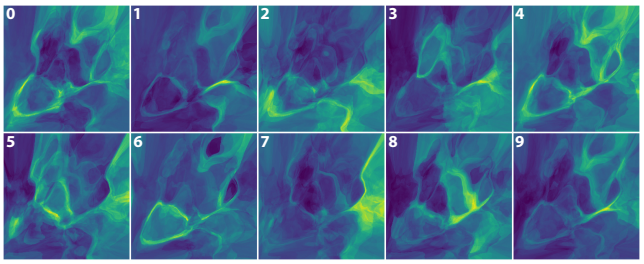


Figure 7: *Sensitivity metric for each class in the MNIST dataset. Yellow (resp. blue) pixels map through P^{-1} to regions in n-D space where the classifier's activations change rapidly (resp. slowly).*

in Eqn. 6. We next outline three regions with high space expansion values (yellow areas inside black ellipses, image (a)). Looking at the actual decision zones (image (c)), we see that region A occurs deep inside the blue zone — hence, the expansion of P^{-1} here is not causing interpretation problems. However, regions B and C occur close to decision boundaries — hence, the exact positions of these boundaries in the DBM can be misleading.

3.5. Sensitivity to Mislabeled Samples

As already mentioned, the nonlinear characteristics of both f and P^{-1} mean that plain DBMs mask the complexity of the decision function and of the data space. We also know that the process of training a classifier can be influenced by several aspects, even under fixed hyperparameters. For instance, the initial weights of a neural network are random; the order in which data samples are processed can affect gradient estimation; and the network itself may have stochastic components such as Dropout layers [SHK*14]. With this in mind, a classifier designer may be interested in *how sensitive* (Q4) each decision zone is to perturbations that can be introduced by retraining. One way to measure this is to determine how easy it is to create new decision zones for each possible class. The easier this is, the more sensitive the training process is in that DBM region.

We measure this sensitivity by

$$\text{ClassifSensitivity}(\mathbf{p}, c) = \left\| \frac{\partial h[c]}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{p}^\dagger} \right\|_2, \tag{7}$$

where $h(\mathbf{x})$ is the pre-softmax activation layer of the classifier f and by $[c]$ we mean the score corresponding to class c. Figure 7 shows this metric computed for all 10 classes of our running example. For class 1, we see that the sensitivity is high only inside class 1's decision zone (see the actual decision zones in Figure 1 left). This means that adding samples wrongly labeled as class 1 should have little to no effect on the classifier. Conversely, the sensitivity for class 4 shows high values throughout a vast portion of the map. that is, telling class 4 apart from the others is problematic for this classifier, as it can be easily convinced that other digits are class 4 by retraining with mislabeled data points.

We further show the predictive power of these maps with two scenarios (Figure 8). We start with the same DBM (a). Next, we add only 10 mislabeled samples in a region of high sensitivity (b) — the total training set being of 5K samples. We see that the DBM visibly changes (c). Conversely, we add 100 mislabeled samples to a low sensitivity region (d). The DBM now barely changes (e).

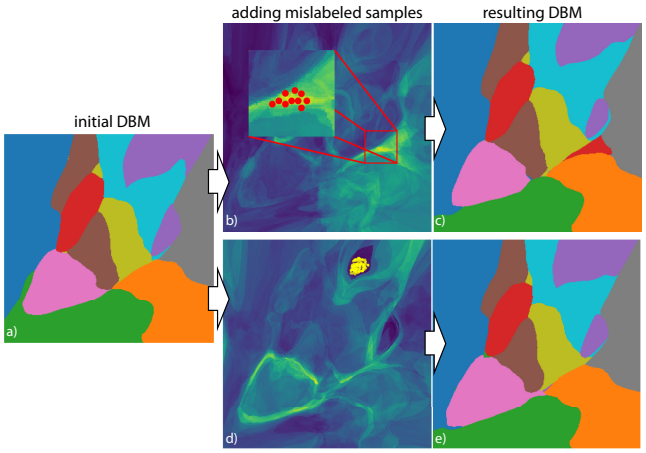


Figure 8: *Initial DBM (a). Adding mislabeled samples to two regions of high, resp. low sensitivities (b,d) leads to very different, resp. almost unchanged, DBMs (c,e).*

3.6. Class Variability in 2D Space

Not all changes in classifier activation patterns result in a different classification output. That is, even if f outputs only one class for a given region of a DBM (a decision zone), it might be the case that f is internally "paying attention" to different aspects of the input. We expect the activation pattern changes to be different from class to class, pointing to different amounts of variability among the elements of each class. This, in turn, can help in judging *why* f's confidence varies (or not) in a given region of the DBM (Q5).

Intuitively, we wish to capture the rates of change in the activation patterns $h(\mathbf{p}^\dagger)$ as we move along the DBM. Since we focus on settings where f and P^{-1} are differentiable, we can compute this as

$$\text{ClassVariability}(\mathbf{p}_0) = \left\| \frac{\partial h \circ P^{-1}}{\partial \mathbf{p}}\Big|_{\mathbf{p}=\mathbf{p}_0} \right\|_{\text{Fro}}, \tag{8}$$

$$\frac{\partial h \circ P^{-1}}{\partial \mathbf{p}} = \frac{\partial h}{\partial \mathbf{x}}\Big|_{\mathbf{x}=P^{-1}(\mathbf{p})} \frac{\partial P^{-1}}{\partial \mathbf{p}}.$$

Computing ClassVariability for every DBM pixel shows the amount of change expected in the activation patterns in a region of the data space corresponding to an infinitesimal movement in the 2D map. This goes one step further than the Gradient Maps [EAS*23]. Differentiating through the classifier f as well as the inverse projection P^{-1} allows for deeper insights into the inner workings of f.

Figure 9a shows ClassVariability for our running example. Bright yellow regions tell that neighbor map pixels correspond to strong changes of the classifier behavior. Not surprisingly, many such changes happen along *borders* between different regions in the DBM. However, our visualization shows that changes also happen *within* those regions — a fact that the original DBM doesn't convey. These within-region changes are not large enough to cause a switch in the classifier's *prediction* — if they were, they would create new decision zones. Still, they tell that the internal activations of the classifier are changing, pointing to *class variability*. We can see this by sampling points in regions with high values of this metric (Fig. 9b) and comparing their variability to points from low value-regions (Fig. 9c). For the former case, we see more variability (images d-h) than for the latter case (images i-m). The class variability shows more activity than the classifier confidence visualization (Figure 2) — the latter spikes to 1 practically everywhere except on the decision boundaries. This shows a uniform confidence in the face of non-uniformly-similar samples (Fig. 9, d-h).

Summarizing the above observations, Table 1 shows how we can combine the class variability and classifier confidence views to draw several conclusion on a classifier's behavior.

Table 1: *Cross-referencing Class Variability with the Confidence map and conclusions that can be drawn.*

		Class Variability	
		Low	High
Confidence	Low	Classifier could not learn the main pattern of this region properly.	Classifier could not learn the patterns of this region properly.
	High	Data pattern is simple enough to be grasped by the classifier.	Data varies considerably, but the classifier can still learn the proper class.

4. Case Study: Data Poisoning

We now describe a use case for our DBM augmentations — assessing whether a data poisoning attack is being carried out when updating a classifier. The need to update a classifier arises from time to time and can be due to a multitude of factors [LLD*19, DRAP15]. For instance, the data distribution might have changed — e.g., user preferences tend to change over time —, or a new source of data may need to be added into a system. In this process, the inclusion of new samples opens up the possibility of an attacker crafting specific training data aimed at *harming* the accuracy of the classifier being updated. These malicious actions are called Data Poisoning attacks, and have been studied extensively [CGD*23, CGD*22]. The particular type of attack we focus on falls under the category of so-called *causative attacks* [BNJT10] — in short, attacks that are carried out by tampering with training data —, which we next briefly describe following Steinhardt *et al.* [SKL17]. The scenario consists of a *defender* learning a model f_θ and an *attacker* who wants the learned f_θ to incur a high test loss (which the defender tries to minimize).

- We establish a "clean" dataset $\mathcal{D}_c$ consisting of n data points.
- The attacker chooses a "poisoned" dataset $\mathcal{D}_p$ of εn data points, where $\varepsilon \in [0,1]$ is the attacker's budget.
- The defender trains on the full dataset $\mathcal{D}_c \cup \mathcal{D}_p$, producing a model $f_{\hat{\theta}}$ and incurring test loss $\mathcal{L}_{\text{test}}(\hat{\theta})$.

We next show that if the attacker chooses points close to decision boundaries in the data space and tampers with their associated labels, they can negatively impact the test loss, fulfilling their goal. Hence, our ∂DBMs can be used as a guide for an attacker to successfully craft poisoned data. Conversely, this also means that using our DBM visualizations can potentially *uncover* this type of attack before it takes place, enabling the defender to filter out the undesirable data.

4.1. Preparation: implicit projection

As already explained, to create our DBM visualizations, we need to be able to run P^{-1} on the projection space. This can be trivially accomplished by using NNInv, as we did for all examples already shown so far. However, to place new data in the ∂DBM, we need to project *unseen* data points via P. Not many projection algorithms support this *out-of-sample* capability — for example, neither t-SNE nor UMAP do. We could solve this by looking for *parametric* projection algorithms, such as PCA, Parametric t-SNE [vdM09], or Auto-Encoder based methods. This would (unnecessarily) impose a limit to the user of ∂DBMs, which goes against our goal of ∂DBMs being as generic as possible.

Another solution would be to use an approximation to the projection, such as NNP [EHT20] — a neural approach that generalizes any projection to unseen data. However, NNP has three disadvantages: (1) it does not always perform well in generalizing the projection it is trained on; (2) using it means introducing *yet another* component in our pipeline that must itself be trained and evaluated; (3) there is no guarantee that the NNP-learned approximation of P is compatible with the P^{-1} already present in the ∂DBM pipeline. That is, the round-trip error $\|P_{\text{NNP}}(P^{-1}(\mathbf{p})) - \mathbf{p}\|_2$ and/or $\|P^{-1}(P_{\text{NNP}}(\mathbf{x})) - \mathbf{x}\|_2$ can be very high.

To address the above issues, without the need for a parametric P, we propose a way to perform a direct projection P *implicitly* from a P^{-1} that is differentiable with respect to its input (a requirement we had earlier for some of our techniques). For each point $\mathbf{x}_{\text{new}} \in \mathbb{R}^n$ we wish to project, we solve the optimization problem

$$\hat{P}(\mathbf{x}_{\text{new}}) = \text{argmin}_{\mathbf{z}} \|P^{-1}(\mathbf{z}) - \mathbf{x}_{\text{new}}\|_2^2. \tag{9}$$

This is easily done with standard optimizers such as Adam [KB15]. Since the optimization goal is non-convex because of P^{-1}, we use a few tricks to ensure convergence to a sensible projected point — see Algorithm 1. We perform this optimization batch-wise, which makes it is reasonably fast: on a commodity PC, projecting $m = 100$ points takes about 0.3 seconds; for reference, projecting the same points with t-SNE takes equal time.

Algorithm 1 starts by generating κ candidate points for each input data point $\mathbf{x}'_i$ that should be projected. We can either generate

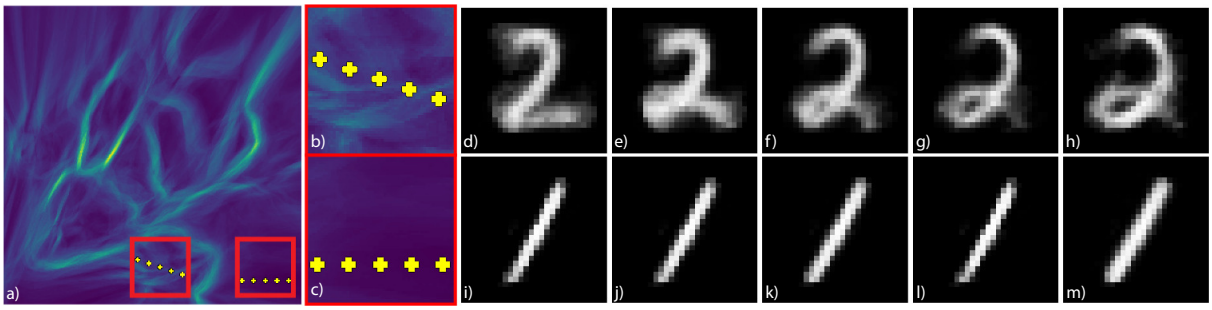


Figure 9: *Comparing equally-spaced points in the projected space, we notice that in regions where ClassVariability is high (b), the elements obtained through P^{-1} change visibly — be it within a given decision zones or across them (d-h). Conversely, when the metric is low (c), we are in a region where there is less variation in the data (i-m).*

those uniformly at random in the projected space (lines 9-11), or use a "warm start". In the latter case, the candidate set for each $\mathbf{x}'_i$ is generated by first finding the data point $\mathbf{n} \in \mathcal{D}_t$ most similar to $\mathbf{x}'_i$ (line 2), then generating κ points by adding Gaussian noise to $\mathbf{n}$ (line 6, $\sigma = 0.01$ in our tests). The generation of κ candidate points per input point to Algorithm 1 is a way to increase the chances of finding a good minimum to the reconstruction error. These points are generated in 2D, not in n-D, which avoids the curse of dimensionality; still, κ should not be set too high since the complexity of the algorithm depends linearly on it ($\kappa = 10$ produces good results in our tests).

Once we have, for each input data point $\mathbf{x}'_i$, a set of candidate projected points $\mathbf{Z}_i \subset \mathbb{R}^2$, Algorithm 1 proceeds by iteratively optimizing the candidate points $\mathbf{z}_k \in \mathbf{Z}_i$ (lines 13-19) so as to minimize the reconstruction error through P^{-1}. We limit the number of times this optimization is done with the hyperparameter T, fixed to 50 in our tests. Finally, we select the element of each candidate set that achieved the smallest reconstruction error (line 20). These are the implicitly projected versions of the input points.

Figure 10 shows our method in use. The left image shows the t-SNE projection P of the MNIST dataset. From this projection, we construct P^{-1} using NNInv, as mentioned earlier. The right image shows the result of $\hat{P}$ (learned from P^{-1}) on unseen data points. As the colors show, the unseen points are projected as expected, given the projection P, within their respective point clusters.

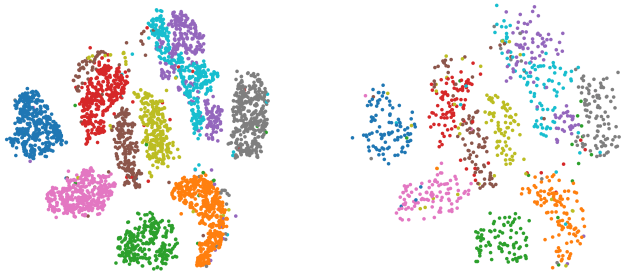


Figure 10: *Left: t-SNE projection of the MNIST dataset, colored by ground truth labels. Right: implicit projection $\hat{P}$ of unseen data obtained through Alg. 1.*

Algorithm 1 Implicit Direct Projection ($\hat{P}$)

Require: $P^{-1}, \mathbf{X}_{\text{new}} = \{\mathbf{x}'_i, \dots\}$, WarmStart $\in \{\mathbb{T}, \mathbb{F}\}$, # seeds $\kappa \geq 1$, # iterations $T > 0$.
Ensure: $\mathbf{O} = \{\mathbf{o}_i, \dots\}$, the projected points from $\mathbf{X}_{\text{new}}$.

1: **if** WarmStart **then**
2: $\quad \mathbf{N} \leftarrow \{\arg\min_{\mathbf{x} \in \mathcal{D}_t} \|\mathbf{x}'_i - \mathbf{x}\|_2^2 \mid \mathbf{x}'_i \in \mathbf{X}_{\text{new}}\}$
3: $\quad \mathbf{n}_i^{(q)} \leftarrow P(\mathbf{n}_i) \; \forall \; \mathbf{n}_i \in \mathbf{N}$ ▷ Precalculated since $\mathbf{N} \subseteq \mathcal{D}_t$.
4: $\quad$ **for all** $\mathbf{x}'_i \in \mathbf{X}_{\text{new}}$ **do**
5: $\quad\quad$ ▷ $\mathbf{Z}_i$ is the candidate set for point $\mathbf{x}'_i$
6: $\quad\quad \mathbf{Z}_i \leftarrow \{\mathbf{n}_i^{(q)} + \varepsilon_k \sim \mathcal{N}(0, \sigma^2) \mid k \in \{1, \dots, \kappa\}\}$
7: $\quad$ **end for**
8: **else**
9: $\quad$ **for all** $\mathbf{x}'_i \in \mathbf{X}_{\text{new}}$ **do**
10: $\quad\quad \mathbf{Z}_i \leftarrow \kappa$ uniformly distributed points in $[0,1]^q$
11: $\quad$ **end for**
12: **end if**
13: **for** $j = 1, \dots, T$ **do**
14: $\quad$ **for all** $\mathbf{x}'_i \in \mathbf{X}_{\text{new}}$ **do** ▷ implemented batch-wise parallel
15: $\quad\quad$ **for all** $\mathbf{z}_k \in \mathbf{Z}_i$ **do**
16: $\quad\quad\quad \mathbf{z}_k \leftarrow \text{OptimizerStep}(\mathbf{z}_k, \nabla_{\mathbf{z}_k} \|P^{-1}(\mathbf{z}_k) - \mathbf{x}'_i\|_2^2)$
17: $\quad\quad$ **end for**
18: $\quad$ **end for**
19: **end for**
20: $\mathbf{o}_i \leftarrow \arg\min_{\mathbf{z}_k \in \mathbf{Z}_i} \|P^{-1}(\mathbf{z}_k) - \mathbf{x}'_i\|_2^2 \; \forall i$ ▷ best candidates
21: **return O**

4.2. The attack setting

As a dataset, we use ciFAIR [BD20], a variant of the well-known CIFAR dataset with duplicate images removed. We use a pretrained ResNet [HZRS16] neural network as a feature extractor, mapping each 32 x 32 RGB image to a feature vector in $\mathbb{R}^{4096}$. As classifier instance, we next train simple 3-layer feed-forward neural networks that take these 4096-dimensional vectors and aim to predict the correct image class.

To carry out the attack in a way that allows for analysis, we first train a model f^{clean} using $m = 5000$ data points from the training set ($\mathcal{D}_c$). The test accuracy is measured on a held-out set $\mathcal{D}_{\text{test}}$. The attacker then chooses εm new data points to form $\mathcal{D}_p$. A new model f^{att} with the same architecture and hyperparameters as f^{clean} is

trained on $\mathcal{D}_c \cup \mathcal{D}_p$. The test accuracy of f^{att} is measured on $\mathcal{D}_{\text{test}}$ to demonstrate the effect of the attack.

4.3. Carrying out the attack

We next discuss how we generate $\mathcal{D}_p$ and the effect that training on the extended dataset has on test accuracy.

Generating poisoned data. In our simplified scenario, we generate the set of εm samples that have not been seen by any of our models by back-projecting (P^{-1}) points that are *close to a decision boundary* (see section 3.2). We choose such data points because, as explained earlier in Sec. 3, decision boundaries are the regions where effecting changes on the classifier's outputs should be easiest. Figure 11(left) shows the t-SNE projection of the ciFAIR dataset created using t-SNE and, also, the DBM of the trained classifier. The right image shows the selected samples close to decision boundaries.

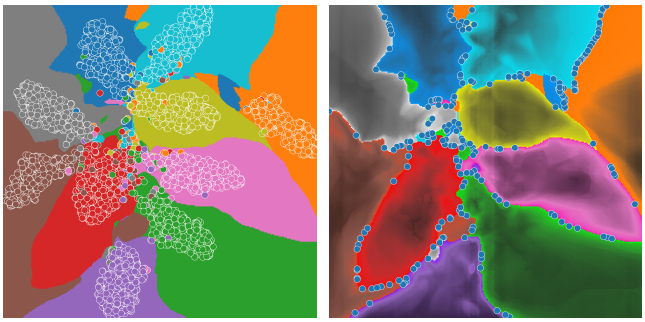


Figure 11: *Left: the ciFAIR dataset projected with t-SNE. Right: generated poisoned points $\mathcal{D}_p$ using $\varepsilon = 0.05$. These all lie close to the decision boundaries of the classifier.*

Figure 12: *The DBMs for the ciFAIR dataset before (left) and after (right) the data poisoning attack. We see that clear differences appear, in the form of new decision zones for class 0, but also in the changes in the shapes of other classes' zones. This reconfiguration of the decision surfaces is directly reflected in the change in test accuracies (Table 2).*

To poison the dataset in a simple way, we add the label 0 to these selected samples. A classifier that is able to predict these *incorrect* labels should intuitively perform worse on the held-out $\mathcal{D}_{\text{test}}$, since we train it with misleading data.

Table 2 shows the results of the attack for two different ε values. We are able to dramatically lower per-class test accuracy by 3.7% (for $\varepsilon = 0.02$) and 5.8% ($\varepsilon = 0.05$) by this simple attack. The overall test accuracy drops by 1.3% ($\varepsilon = 0.02$) and 2.2% ($\varepsilon = 0.05$), taking into account the different relative amounts of points from each class.

Interestingly, the accuracy for class 0 — used for our added poisoned points — *rose* for $\varepsilon = 0.05$. We explain this by observing that, indeed, some of the new points are similar to true members of class 0, hence introducing a correct signal in training; also, the training process itself is prone to perturbations, so the classifier may have learned better from the original training samples of class 0 when performing this experiment.

We conclude that ∂DBMs are simple but effective tools that provide starting guidance to an attacker wishing to negatively impact the test accuracy of a classifier by directly pointing them to regions that reliably affect classification outcomes.

4.4. Defending using ∂DBMs

In turn, a defender can also use ∂DBMs to thwart a poisoning attack. Say the defender has received a dataset $\mathcal{D}_{\text{mixed}} = \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}}$, potentially poisoned as described in Sec. 4.3. We assume the defender has the needed information to decide whether poisoning is suspected or not, *e.g.*, can tell such attacks apart from concept shifts in the data-generating distribution; this type of decision is out of our scope. The attacker may have crafted points in $\mathcal{D}_{\text{new}}$ in the manner of Fig. 11, but the defender has no way of knowing that. The defender can however inspect these new training points by projecting them onto the ∂DBMs of the existing classifier. This can be directly done using Alg. 1.

The defender can now use ∂DBM visualizations to explore $\mathcal{D}_{\text{mixed}}$. Figure 13(left) shows the dataset projected atop the DBM of the classifier. For clarity, we only show here the $\mathcal{D}_{\text{new}}$ points — these can be easily isolated from $\mathcal{D}_{\text{mixed}}$ since we have $\mathcal{D}_{\text{old}}$. In our image, the new points appear close to the decision boundaries, which raises suspicions. The right image explores further by showing the distance to decision boundaries (Sec. 3.2). We now see that all points are very close to the *actual* decision boundaries — even some points which, in the left image, appeared to be further away from the decision boundaries in the *image*. The defender can now conclude that a dataset poisoning attack is underway; the attack can next be thwarted by either filtering out the suspicious samples or choosing not to update the classifier altogether.

5. Discussion

We discuss our method along several aspects, as follows.

Simplicity vs interpretability: ∂DBMs can be created automatically, with zero parameter-setting efforts, for any differentiable classification model, that is, with very little effort even for inexperienced users. Interpreting the set of views that ∂DBMs provide, however, requires a certain amount of training and effort. We argue that such explanatory views are *necessary* for any DBM algorithm. Indeed, such algorithms (1) use direct and inverse projections, which are subject to errors; and (2) as recently shown by Wang *et al.* [WMT23, WT24], all current DBM techniques only visualize a *surface*-like subset of the full data space the classifier works on.

Table 2: *Variation in test accuracy measured on unseen data before poisoning ($\mathcal{D}_c$) and after poisoning with different budgets ($\varepsilon = 0.02, 0.05$). We highlight the classes for which we were able to drive down test accuracy.*

	Test Accuracy per Class									
	0	1	2	3	4	5	6	7	8	9
$\mathcal{D}_c$	83.1%	90.5%	90.0%	82.0%	86.2%	77.5%	89.9%	93.5%	94.7%	89.4%
$\mathcal{D}_c \cup \mathcal{D}_p^{(0.02)}$	79.4% (**-3.7%**)	88.1% (**-2.4%**)	86.8% (**-3.2%**)	82.5% (+0.5%)	83.8% (**-2.4%**)	78.5% (+1.0%)	91.0% (+1.1%)	91.5% (**-2.0%**)	95.2% (+0.5%)	86.9% (**-2.5%**)
$\mathcal{D}_c \cup \mathcal{D}_p^{(0.05)}$	86.8% (+3.7%)	85.7% (**-4.8%**)	84.1% (**-5.8%**)	80.7% (**-1.4%**)	83.3% (**-2.9%**)	77.5% (+0.0%)	87.8% (**-2.1%**)	91.0% (**-2.5%**)	91.8% (**-2.9%**)	86.4% (**-3.0%**)

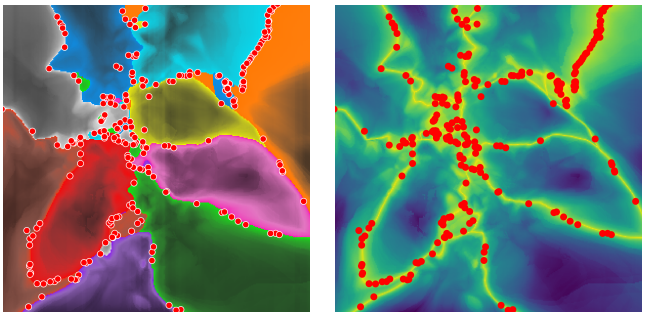


Figure 13: *A defender needs to project the new dataset using $\hat{P}$. The new data points $\mathcal{D}_{new}$ are shown here for the same scenario as above atop the distance to decision boundary view. On the left, we embed it on the DBM (section 3.2) and on the right, we display it independently.*

Having the *option* to visualize ∂DBMs, either as separate views, or overlaid atop a plain DBM, aids in answering specific questions (Q1-5) which, as we have shown, cannot be answered by a plain DBM. We argue that this is valuable even for inexperienced users.

DBM distortions: As explained earlier, all current DBM methods have limitations, either in terms of which part of the data space they show, or the distortions they introduce due to the (inverse) projections they use. We do not aim to *correct* these distortions (or limitations), as this would imply fundamentally changing the DBM construction algorithm. Rather, our explanations work generically for any DBM technique which uses a differentiable inverse projection. As such, we focus on *highlighting* the place, nature, and extent of problems caused by DBM distortions, rather than aiming to fix these. Similar approaches are well known for visualizing (and not correcting) distortions caused by direct projections [HFA17, MCMT14, Aup07, LA11].

6. Conclusion

We have proposed ∂DBMs, a set of visualizations that, independently or as a whole, enable a deeper analysis of Decision Boundary Map (DBM) depictions of classification models. Our views are mainly based on one simple concept — measuring the sensitivity of different elements of the DBM pipeline (inverse projection, classifier) with respect to their inputs, and visualizing how this changes across the DBM. Our techniques can be generically applied to any data dimensionality and to DBMs constructed using any direct projection technique. We show the additional information that can be derived from our visualizations, linking them to concrete uses and demonstrating their explanatory power for classification. We also illustrate our ∂DBMs for the task of creating, but also defending against, an end-to-end dataset poisoning attack. As a side contribution related to this use-case, we also show how to construct an implicit projection function based on a given inverse projection.

∂DBMs, however, also have some limitations. Our techniques require the differentiability of the inverse projection used by the DBM and the classification model itself. However, most ML models (and importantly, deep neural networks), and most inverse projection techniques we know of, fall in this class. Visualizing the *exact* distance to decision boundaries of a classifier (section 3.2) would be computationally intensive, limiting the interactivity of the created maps. Moreover, such exact maps could be very noisy due to the complex hypersurface determined by a classifier, so approximating that distance both increases speed and provides some regularization to the maps. More importantly, ∂DBMs do not overcome an important limitation of their predecessors — they heavily rely on the quality of the generated (direct) projection of the data. Poor projections will lead to poor maps. As such, one still requires exploration to choose a suitable projection technique. An alternative to this that is worth studying is using techniques that provide some regularization of the projected space, such as the recently-proposed ShaRP [MTB23].

We see several immediate avenues for future work. ∂DBMs can be extended by additional, more specialized, views that aim to answer more complex, specific, questions. Adding the support for interactive querying of the maps would allow ML engineers to pose more complex queries on parts of the data and/or the map, thereby narrowing down problems of existing classifiers. Finally, using ∂DBMs in a controlled study to demonstrate their effective added-value in an end-to-end ML engineering task is an important step towards practical validation.

References

[AASA21] ANGELOV P., ARNOLD N., SOARES E., ATKINSON P.: Explainable artificial intelligence: an analytical review. *WIREs Data Mining Knowl Discov 11*, e1424 (2021). 2

[ABD*12] AMORIM E., BRAZIL E., DANIELS J., JOIA P., NONATO L., SOUSA M.: iLAMP: Exploring high-dimensional spacing through backward multidimensional projection. In *Proc. IEEE VAST* (2012). 3

[AS22] ALICIOGLU G., SUN B.: A survey of visual analytics for explainable artificial intelligence methods. *Computers & Graphics 102* (2022), 502–520. 2

[Aup07] AUPETIT M.: Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing 10*, 7–9 (2007), 1304–1330. 3, 10

[BD20] BARZ B., DENZLER J.: Do We Train on Test Data? Purging CIFAR of Near-Duplicates. *Journal of Imaging 6*, 6 (June 2020), 41. URL: https://www.mdpi.com/2313-433X/6/6/41, doi:10.3390/jimaging6060041. 8

[BNJT10] BARRENO M., NELSON B., JOSEPH A. D., TYGAR J. D.: The security of machine learning. *Machine Learning 81*, 2 (Nov. 2010), 121–148. URL: http://link.springer.com/10.1007/s10994-010-5188-5, doi:10.1007/s10994-010-5188-5. 7

[BTT22] BREDIUS C., TIAN Z., TELEA A.: Visual exploration of neural network projection stability. In *Proc. MLVis* (2022). 3

[CGD*22] CINÀ A. E., GROSSE K., DEMONTIS A., BIGGIO B., ROLI F., PELILLO M.: Machine learning security against data poisoning: Are we there yet? *CoRR abs/2204.05986* (2022). URL: https://doi.org/10.48550/arXiv.2204.05986, arXiv:2204.05986, doi:10.48550/ARXIV.2204.05986. 7

[CGD*23] CINÀ A. E., GROSSE K., DEMONTIS A., VASCON S., ZELLINGER W., MOSER B. A., OPREA A., BIGGIO B., PELILLO M., ROLI F.: Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM Comput. Surv. 55*, 13s (jul 2023). URL: https://doi.org/10.1145/3585385, doi:10.1145/3585385. 7

[CSHB18] CHATTOPADHYAY A., SARKAR A., HOWLADER P., BALASUBRAMANIAN V. N.: Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018, Lake Tahoe, NV, USA, March 12-15, 2018* (2018), IEEE Computer Society, pp. 839–847. URL: https://doi.org/10.1109/WACV.2018.00097, doi:10.1109/WACV.2018.00097. 3

[DRAP15] DITZLER G., ROVERI M., ALIPPI C., POLIKAR R.: Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine 10*, 4 (2015), 12–25. doi:10.1109/MCI.2015.2471196. 7

[EAS*23] ESPADOTO M., APPLEBY G., SUH A., CASHMAN D., LI M., SCHEIDEGGER C., ANDERSON E. W., CHANG R., TELEA A. C.: UnProjection: Leveraging Inverse-Projections for Visual Analytics of High-Dimensional Data. *IEEE Transactions on Visualization and Computer Graphics 29*, 02 (Feb. 2023), 1559–1572. Publisher: IEEE Computer Society. URL: https://www.computer.org/csdl/journal/tg/2023/02/09606529/1ymF2Tydh3G, doi:10.1109/TVCG.2021.3125576. 4, 5, 6, 7

[EHT20] ESPADOTO M., HIRATA N. S. T., TELEA A. C.: Deep learning multidimensional projections. *Inf. Vis. 19*, 3 (2020), 247–269. URL: https://doi.org/10.1177/1473871620909485, doi:10.1177/1473871620909485. 7

[EMK*19] ESPADOTO M., MARTINS R., KERREN A., HIRATA N., TELEA A.: Toward a quantitative survey of dimension reduction techniques. *IEEE TVCG 27*, 3 (2019), 2153–2173. 3

[ERH*19] ESPADOTO M., RODRIGUES F. C. M., HIRATA N. S. T., HIRATA JR. R., TELEA A. C.: *Deep Learning Inverse Multidimensional Projections*. The Eurographics Association, 2019. Accepted: 2019-06-02T18:19:20Z. URL: https://diglib.eg.org:443/xmlui/handle/10.2312/eurova20191118, doi:10.2312/eurova.20191118. 3

[HFA17] HEULOT N., FEKETE J.-D., AUPETIT M.: Visualizing dimensionality reduction artifacts: An evaluation, 2017. arXiv:1705.05283v1 [cs.HC]. 3, 10

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016* (2016), IEEE Computer Society, pp. 770–778. URL: https://doi.org/10.1109/CVPR.2016.90, doi:10.1109/CVPR.2016.90. 8

[JCC*11] JOIA P., COIMBRA D., CUMINATO J. A., PAULOVICH F. V., NONATO L. G.: Local affine multidimensional projection. *IEEE TVCG 17*, 12 (2011), 2563–2571. 3

[Jol02] JOLLIFFE I. T.: *Principal Component Analysis*. Springer, 2002. 3

[KB15] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Bengio Y., LeCun Y., (Eds.). URL: http://arxiv.org/abs/1412.6980. 7

[LA11] LESPINATS S., AUPETIT M.: CheckViz: Sanity check and topological clues for linear and nonlinear mappings. *Computer Graphics Forum 30*, 1 (2011), 113–125. 3, 10

[LLD*19] LU J., LIU A., DONG F., GU F., GAMA J., ZHANG G.: Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering 31*, 12 (2019), 2346–2363. doi:10.1109/TKDE.2018.2876857. 7

[MCMT14] MARTINS R., COIMBRA D., MINGHIM R., TELEA A. C.: Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics 41* (2014), 26–42. 10

[MFF16] MOOSAVI-DEZFOOLI S., FAWZI A., FROSSARD P.: Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016* (2016), IEEE Computer Society, pp. 2574–2582. URL: https://doi.org/10.1109/CVPR.2016.282, doi:10.1109/CVPR.2016.282. 4

[MHM18] MCINNES L., HEALY J., MELVILLE J.: Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018). 3

[MMT15] MARTINS R., MINGHIM R., TELEA A. C.: Explaining neighborhood preservation for multidimensional projections. In *Proc. CGVC* (2015), Eurographics, pp. 121–128. 3

[MRHT18] M. RODRIGUES F. C., HIRATA R., TELEA A. C.: Image-Based Visualization of Classifier Decision Boundaries. In *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)* (Oct. 2018), pp. 353–360. ISSN: 2377-5416. doi:10.1109/SIBGRAPI.2018.00052. 1

[MTB23] MACHADO A., TELEA A., BEHRISCH M.: ShaRP: Shape-Regularized Multidimensional Projections. In *EuroVis Workshop on Visual Analytics (EuroVA)* (2023), Angelini M., El-Assady M., (Eds.), The Eurographics Association. doi:10.2312/eurova.20231088. 10

[NA18] NONATO L., AUPETIT M.: Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG 25*, 8 (2018), 2650–2673. 3

[OEHJT22] OLIVEIRA A. A., ESPADOTO M., HIRATA JR R., TELEA A. C.: Sdbm: Supervised decision boundary maps for machine learning classifiers. In *VISIGRAPP (3: IVAPP)* (2022), pp. 77–87. 3

[OEJT23] OLIVEIRA A., ESPADOTO M., JR R. H., TELEA A.: Stability analysis of supervised decision boundary maps. *SN Computer Science 4*, 226 (2023). 3

[PLZ21] PAN D., LI X., ZHU D.: Explaining deep neural network models with adversarial gradient integration. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021* (2021), Zhou Z., (Ed.), ijcai.org, pp. 2876–2883. URL: https://doi.org/10.24963/ijcai.2021/396, doi:10.24963/IJCAI.2021/396. 3

[RBB*23] ROSA B. L., BLASILLI G., BOURQUI R., AUBER D., SANTUCCI G., CAPOBIANCO R., BERTINI E., GIOT R., ANGELINI M.: State of the Art of Visual Analytics for eXplainable Deep Learning. *Comp Graph Forum 42*, 1 (2023), 319–355. 2

[REJT19] RODRIGUES F. C. M., ESPADOTO M., JR R. H., TELEA A.: Constructing and visualizing high-quality classifier decision boundary maps. *Information 10*, 9 (2019), 280–297. 3, 4

[SHH20] SCHULZ A., HINDER F., HAMMER B.: DeepView: Visualizing Classification Boundaries of Deep Neural Networks as Scatter Plots Using Discriminative Dimensionality Reduction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (July 2020), pp. 2305–2311. arXiv:1909.09154 [cs, stat]. URL: http://arxiv.org/abs/1909.09154, doi:10.24963/ijcai.2020/319. 1, 3

[SHK*14] SRIVASTAVA N., HINTON G. E., KRIZHEVSKY A., SUTSKEVER I., SALAKHUTDINOV R.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res. 15*, 1 (2014), 1929–1958. URL: https://dl.acm.org/doi/10.5555/2627435.2670313, doi:10.5555/2627435.2670313. 6

[SKL17] STEINHARDT J., KOH P. W. W., LIANG P. S.: Certified Defenses for Data Poisoning Attacks. In *Advances in Neural Information Processing Systems* (2017), Guyon I., Luxburg U. V., Bengio S., Wallach H., Fergus R., Vishwanathan S., Garnett R., (Eds.), vol. 30, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/9d7311ba459f9e45ed746755a32dcd11-Paper.pdf. 7

[SMV*19] SAMEK W., MONTAVON G., VEDALDI A., HANSEN L. K., MÜLLER K.-R.: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer LNCS, 2019. 2

[SVPM14] SORZANO C., VARGAS J., PASCUAL-MONTANO A.: A survey of dimensionality reduction techniques, 2014. arXiv:1403.2877 [stat.ML]. 3

[vdEAA*23] VAN DEN ELZEN S., ANDRIENKO G., ANDRIENKO N., FISHER B., MARTINS R., PELTONEN J., TELEA A., VERLEYSEN M.: The flow of trust: A visualization framework for externalizing, exploring and explaining trust in ml applications. *IEEE CG & A 43*, 2 (2023), 78–88. 1

[vdM09] VAN DER MAATEN L.: Learning a parametric embedding by preserving local structure. In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics* (Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009), van Dyk D., Welling M., (Eds.), vol. 5 of *Proceedings of Machine Learning Research*, PMLR, pp. 384–391. URL: https://proceedings.mlr.press/v5/maaten09a.html. 7

[vdMH08] VAN DER MAATEN L., HINTON G. E.: Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research 9* (2008), 2579–2605. 3

[VGdS*20] VERNIER E., GARCIA R., DA SILVA I., COMBA J., TELEA A.: Quantitative evaluation of time-dependent multidimensional projection techniques. *Computer Graphics Forum 39*, 3 (2020), 241–252. 3

[VK06] VENNA J., KASKI S.: Visualizing gene interaction graphs with local multidimensional scaling. In *Proc. ESANN* (2006), pp. 557–562. 3

[WMT23] WANG Y., MACHADO A., TELEA A.: Quantitative and qualitative comparison of decision map techniques for explaining classification models. *Algorithms 16*, 9 (2023). 9

[WT24] WANG Y., TELEA A.: Fundamental limitations of inverse projections and decision maps. In *Proc. IVAPP* (2024). 9

[ZLQ*23] ZHOU K., LIU Z., QIAO Y., XIANG T., LOY C. C.: Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 45*, 4 (2023), 4396–4415. doi:10.1109/TPAMI.2022.3195549. 4