Reconstruction of Univariate Functions from Directional Persistence Diagrams

Aina Ferrà [1,2], Carles Casacuberta [1,2], Oriol Pujol [1,2]

Abstract

We describe a method for approximating a single-variable function f using persistence diagrams of sublevel sets of f from height functions in different directions. We provide algorithms for the piecewise linear case and for the smooth case. Three directions suffice to locate all local maxima and minima of a piecewise linear continuous function from its collection of directional persistence diagrams, while five directions are needed in the case of smooth functions with non-degenerate critical points.

Our approximation of functions by means of persistence diagrams is motivated by a study of importance attribution in machine learning, where one seeks to reduce the number of critical points of signals without a significant loss of information for a neural network classifier.

Introduction

For a finite geometric simplicial complex M in Euclidean space $\mathbb{R}^d$ with $d \geq 2$, the *persistent homology transform* (PHT), defined in [18], is a function on the sphere S^{d-1} that associates to each unit vector v the persistence diagram of sublevel sets of M in the direction v. More precisely, one defines

$$M_t(v) = \{x \in M \mid x \cdot v \leq t\}$$

for $t \in \mathbb{R}$, where the dot denotes scalar product and persistent homology generators of the filtered space $M(v)$ are computed until dimension $d-1$. The corresponding persistence module has the singular homology $H_*(M_t(v); \mathbb{R})$ at level t, together with translation operators induced by inclusions of sublevel sets. In this paper, we use coefficients in $\mathbb{R}$ and we only focus on zero-dimensional homology H_0. The PHT is continuous with respect to any Wasserstein distance on the set of persistence diagrams [5].

It was proved in [18, Theorem 3.1] that the PHT is injective if $d = 2$ or $d = 3$, and this result was extended over all dimensions in [5] and [11]. That is, for finite simplicial complexes embedded in $\mathbb{R}^d$, the collection of persistence diagrams of sublevel sets in all possible directions $v \in S^{d-1}$ uniquely determines the given geometric simplicial complex. See [16] for extensive information about inverse problems in homological persistence.

In order to produce efficient reconstruction algorithms, one seeks to use a small number of directions to reconstruct any given simplicial complex under suitable general position assumptions. In the case of graphs, three directions suffice, as shown in [1]. Bounds on the number of directions needed for reconstruction of compact definable sets in $\mathbb{R}^d$ can be found in [5].

Given a continuous piecewise linear function $f : [a, b] \to \mathbb{R}$ with finitely many vertices, the graph $G(f) = \{(x, y) \in \mathbb{R}^2 \mid y = f(x)\}$ can be viewed as a finite geometric simplicial complex of dimension 1. Our goal is to determine the precise location of the local maxima and minima of $G(f)$ assuming knowledge of a persistence diagram for each direction $v \in S^1$. We label a direction v with the angle θ that the vector v forms with the positive direction of the x-axis, and consider only the upper open hemisphere of S^1, so that $0 < \theta < \pi$. Our algorithm requires precisely three *admissible* directions, in a sense that we make precise.

We also present an algorithm that locates the local maxima and minima of a *smooth* function $f\colon [a,b] \to \mathbb{R}$, and prove that the algorithm converges to each critical point of f assuming that the second derivative f'' does not vanish at critical points. Although this assumption can be weakened, it simplifies our presentation of results and is sufficiently plausible in practice —it excludes the occurrence of intervals in which the graph of f is horizontal.

The injectivity of the PHT for graphs of smooth functions on a bounded domain follows from results in [12]. Indeed, the problem of reconstructing a smooth function from the set of its directional persistence diagrams is essentially equivalent to the problem of recovering a smooth function from the set of its tangent lines. From [12] we quote the fact that, given a sequence of distinct tangent lines T_j converging to a tangent line T, the sequence of intersection points of T_j with T converges to a point where T is tangent to the graph of f. Our assumption on the second derivative f'' at critical points replaces the assumption made in [12] that f'' has only finitely many zeros.

Both in the piecewise linear case and in the smooth case, our algorithm produces a piecewise linear approximation of the given function with the same set of critical points.

Motivation for this work came from a study of explainability in the process of classification of signals by a neural network. This study is summarized in Section 3.1 and further details are given in [9]. The possibility of reversing information obtained from persistence landscapes in order to partially reconstruct data allowed us to design a system of importance attribution in signal classification tasks. Therefore, our method contributes to interpretability in machine learning.

1 Critical points and critical lines

A *critical point* of a continuous function $f\colon [a,b] \to \mathbb{R}$ will mean either $(a, f(a))$ or $(b, f(b))$ or any of the local maxima or local minima $(x_i, f(x_i))$ of f in the interior of $[a,b]$. Inflection points are not treated as critical points in this article. We assume that f has only finitely many critical points; in particular, horizontal segments in the graph of f are excluded. If f is smooth, then for every critical point $(x_i, f(x_i))$ with $a < x_i < b$ the derivative $f'(x_i)$ vanishes, and we additionally impose that $f''(x_i) \neq 0$ at each critical point.

We denote by S^1 the unit circle and view each $v \in S^1$ as a complex number $e^{i\theta}$. Given any subset X of $\mathbb{R}^2$, the *sublevel set* of X in the direction $v = e^{i\theta}$ is defined for $t \in \mathbb{R}$ and $0 < \theta < \pi$ as

$$X_t(v) = \{(x, y) \in X \mid x \cos\theta + y \sin\theta \leq t\}.$$

That is, $X_t(v)$ is the subset of X consisting of those points that are placed on or below the line $x \cos\theta + y \sin\theta = t$ along the direction v; see Fig. 1. Sublevel sets are nested, since $X_s(v) \subseteq X_t(v)$ if $s \leq t$, and we denote by $X(v)$ the filtered space with $X_t(v)$ at height t.

Let $f\colon [a,b] \to \mathbb{R}$ be a continuous function and let $v \in S^1$. We define the *directional persistence module* $M(f, v)$ of f in the direction v as

$$M_t(f, v) = H_0(G(f)_t(v); \mathbb{R})$$

for $t \in \mathbb{R}$, where $G(f)$ is the graph of f with the topology induced by $\mathbb{R}^2$ and H_0 is zero-dimensional homology. Hence $M_t(f, v)$ counts the connected components, for each value of t, of the sublevel sets of the graph of f in the direction v. The translation operators $M_s(f, v) \to M_t(f, v)$ for $s \leq t$ are induced by inclusions of sublevel sets; see [4] for details about persistence modules.

Definition 1.1. A *critical line* at a height t for a direction $v = e^{i\theta}$ with $0 < \theta < \pi$ is a line in $\mathbb{R}^2$ orthogonal to v (hence of slope $-1/\tan\theta$) such that the filtered space $G(f)(v)$ changes its number of connected components at height t, where height 0 corresponds to the line passing through $(0, 0)$.

Lines passing through $(a, f(a))$ or $(b, f(b))$ may not be critical. In fact, for the vertical height function these lines will be critical if and only if $(a, f(a))$ respectively $(b, f(b))$ are local minima.

In the smooth case, critical lines are tangent to the graph of f, except perhaps those passing through $(a, f(a))$ or $(b, f(b))$. In the piecewise linear case, each critical line in any direction contains at least one critical point of f.
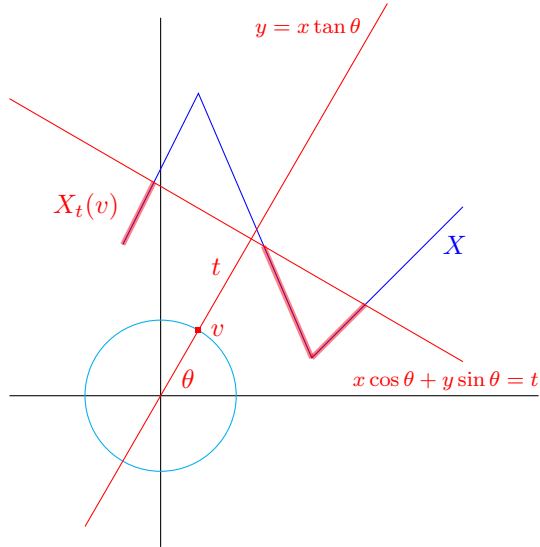
Figure 1: A sublevel set of a graph X at a height t along a direction v.

Each directional persistence module $M(f, v)$ for a continuous function $f \colon [a, b] \to \mathbb{R}$ gives rise to a *barcode* and a *persistence diagram* as in [7, 10]. When two connected components of $G(f)(v)$ merge at some height, the component which was born more recently (i.e., at a greater height) is annihilated, and if several components were born at the same height then the one with smaller x-value is kept. This convention yields a *barcode template* for homological dimension 0 in the sense of [13, Definition 4.3]. Hence, a point (β, δ) in a directional persistence diagram represents a connected component of $G(f)(v)$ which is born at a height β and merges with an older component at a height δ. The possibility that $\delta = \infty$ is not discarded; in fact, since the graph of f is connected, each directional persistence diagram contains precisely one point (β, ∞), where β is the birth height of the whole graph.

A persistence diagram in a direction v tells us precisely which lines orthogonal to v are critical.

Proposition 1.2. *For a direction v, a line orthogonal to v at a height t is critical for a function f if and only if either $t = \beta$ for some point (β, δ) in the persistence diagram for zero-homology H_0 of sublevel sets of f in the direction v, or $t = \delta$ when δ is finite.*

Proof. If there is a point (β, δ) with δ finite in the persistence diagram, then at least one new connected component of $G(f)(v)$ arises at height β (so the orthogonal line at height β is critical), and at least two connected components merge at height δ, which implies that the orthogonal line at height δ is also critical. If $\delta = \infty$, then only the line at height β is critical. Conversely, if a line at height t is critical, then either a new connected component arises or two connected components merge at t. $\square$

Persistence diagrams in this article have been calculated using GUDHI (Geometry Understanding in Higher Dimensions [17]), a generic open source library with a Python interface.

2 Reconstruction algorithms

2.1 The piecewise linear case

In [8] an algorithm was given for reconstructing an embedded simplicial complex of arbitrary dimension using a finite number of directional persistence diagrams. In our case, three directional persistence diagrams suffice to reconstruct a continuous piecewise linear function f by providing an approximation that coincides with f on its critical points.

Assuming that f has a total number of n critical points, the number of critical lines in each direction is less than or equal to n. A critical line can pass through two or more critical points, and a line passing through a critical point can fail to be critical. Indeed, if the graph of f crosses

transversally a line at some critical point P, then the corresponding directional persistence diagram will not include any information relative to P, since there is no change in the number of connected components of the sublevel sets $G(f)_t(v)$ when the line passes through P. Since we need to exclude this possibility from our algorithm, we give the following definition.

Definition 2.1. For a continuous piecewise linear function $f : [a, b] \to \mathbb{R}$, we say that a direction $v = e^{i\theta} \in S^1$ with $0 < \theta < \pi$ is *admissible* if for each critical point $P = (p_1, p_2)$ of f there is an $\varepsilon > 0$ such that all the points $(x, f(x))$ on the graph of f with $p_1 - \varepsilon < x < p_1 + \varepsilon$ and $x \neq p_1$ are on the same open half-plane in $\mathbb{R}^2 \smallsetminus L$ where L is the line orthogonal to v containing P.

If v is admissible then all lines orthogonal to v containing critical points of f are critical lines, except perhaps those at the endpoints a and b.

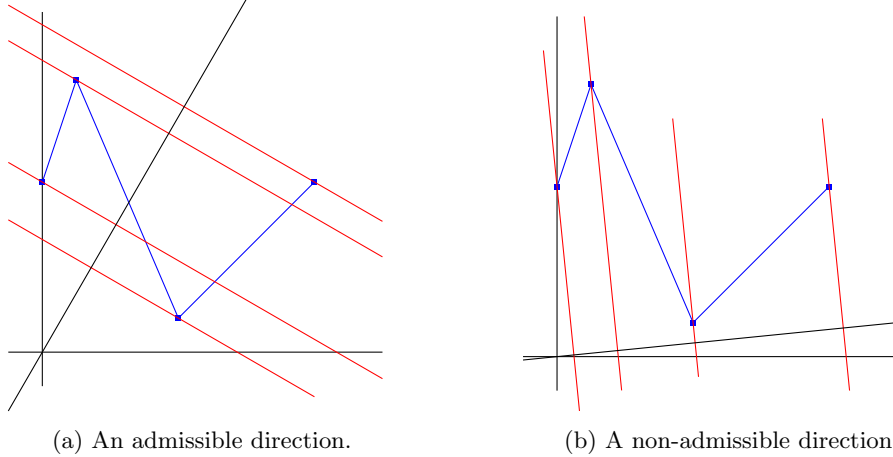(a) An admissible direction.   (b) A non-admissible direction.

Figure 2: A direction v is not admissible if some line orthogonal to v is transverse to the graph at some critical point.

If there are critical points in which the graph of f has very small slopes in absolute value, then the only admissible directions are those with θ close to $\pi/2$. More precisely, the following observation guarantees the existence of admissible directions.

Proposition 2.2. *Let m be the smallest slope in absolute value between two consecutive vertices in the graph of f. Then every direction $e^{i\theta}$ with $0 < \theta < \pi/2$ and $\tan \theta > 1/m$ is admissible.*

Proof. Since, by assumption, the graph of f does not contain any horizontal segments, we have that $m > 0$. Given $v = e^{i\theta}$ with $0 < \theta < \pi/2$, the lines orthogonal to v have slope $-1/\tan\theta$. If one such line L contains a critical point P, then the admissibility condition is fulfilled at P since the slope of L is smaller in absolute value than the slope of each segment of the graph of f at P. $\qquad\square$

The intersection points of two distinct critical lines will be called *double points* and the intersection points of three distinct critical lines will be called *triple points*.

In what follows we fix three directions $v_0 = e^{i\theta_0}$, $v_1 = e^{i\theta_1}$, $v_2 = e^{i\theta_2}$ with $0 < \theta_2 < \theta_1 < \theta_0 \leq \pi/2$, and describe a procedure to recover a continuous piecewise linear function $f : [a, b] \to \mathbb{R}$ from the corresponding directional persistence diagrams. Although the algorithm accepts as input any values of θ_0, θ_1 and θ_2, we normally use $\theta_0 = 90°$, $\theta_1 = 85°$ and $\theta_2 = 80°$ (angles given in degrees). If some of these happens to be non-admissible, then either the angle should be increased or the reconstruction will omit the conflicting critical points.

Theorem 2.3. *Let $f : [a, b] \to \mathbb{R}$ be a continuous piecewise linear function, and let $v_0 = e^{i\theta_0}$, $v_1 = e^{i\theta_1}$ and $v_2 = e^{i\theta_2}$ be admissible directions such that $0 < \theta_2 < \theta_1 < \theta_0 \leq \pi/2$. Let $\mathcal{P}$ be the set of triple points determined by the three directional persistence diagrams. Suppose that no critical line orthogonal to v_0, v_1 or v_2 contains two or more points from $\mathcal{P}$. Then $\mathcal{P}$ is equal to the set of critical points of f, excluding the boundary points $(a, f(a))$ and $(b, f(b))$ if these are local maxima.*

4

Proof. Since the three directions are admissible, every critical point of f (except perhaps the boundary points) is a triple point, since there is a critical line in each direction passing through it.

Conversely, every triple point must be a critical point of f, since if a triple point P is not critical, then each of the three critical lines passing through P contains at least two triple points. □

Consequently, by joining each pair of consecutive points in $\mathcal{P}$ (ordered by their x-coordinate) we obtain a continuous piecewise linear function with the same critical points as f. Since the boundary points $(a, f(a))$ and $(b, f(b))$ remain undetected unless they are local minima, in order to entirely reconstruct the graph of f it will be necessary to give its boundary points as part of the oracle's data.

The assumption that no critical line contains two or more triple points does not restrict significantly the validity of our algorithm, since the probability that it fails is negligible. Indeed, as in [13, § 5.1], the subset of S^1 of those directions for which there are critical lines passing through two or more triple points has measure zero.

Thus the algorithm consists of finding triple intersections within the set of critical lines determined by three chosen directional persistence diagrams. There is a naive algorithm consisting of looking at all the possible intersections between all such critical lines, that would take $O(n^3)$ where n is the number of critical points of the original function. However, since there is a strong geometrical component, we can improve performance by lowering the complexity to at most $2n \log n + 2n^2$ operations, hence $O(n^2)$, as follows.
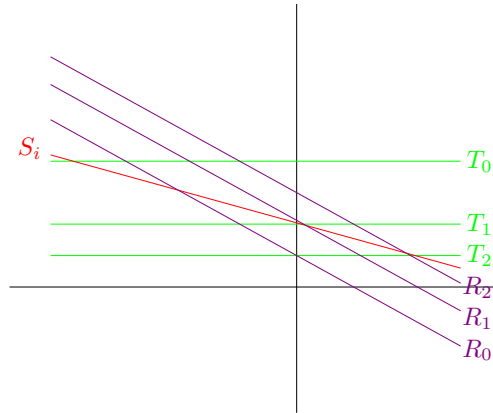

Figure 3: Ordering of critical lines in three directions.

For each critical line S_i, the lines T_j and R_j are moved simultaneously to search for triple intersection points. To optimize the algorithm, each of the sets of lines T and R has to be ordered. Since S_i has negative slope, the T_j lines have to be ordered decreasingly (with T_0 above the others) while the R_j lines must be ordered increasingly (with R_0 below the others).

The rolling ball algorithm. For every S_i, we start by intersecting it with the first lines T_0 and R_0, obtaining $P_t = S_i \cap T_0$ and $P_r = S_i \cap R_0$. If $P_t = P_r$, a triple point has been found and we can move to S_{i+1}. If not, we check which point has a lower value of its x-coordinate (hence the name of the algorithm). If $x(P_t) < x(P_r)$, as depicted in Fig. 3, then the line T_0 can be discarded: since S_i has negative slope and the lines T and R are ordered, a triple intersection cannot occur with T_0.

In general, at each step of the algorithm, given a line S_i we compare, in the prescribed order, T_j with R_k and check for the intersections P_t and P_r. If $P_t \neq P_r$, then we move to compare T_{j+1} and R_k if $x(P_t) < x(P_r)$ and we move to compare T_j and R_{k+1} otherwise.

In order to compare the speed of the naive algorithm with our rolling ball algorithm, we used a bank of functions with a varying number of critical points. For that purpose, we programmed a snippet of code that returns a piecewise linear function with the desired number of points.

We ran both algorithms with functions with the following number of critical points: 5, 10, 25, 50, 100, 150, 200, and recorded the time in seconds each algorithm takes for each function. Table 1 shows the comparison between the two algorithms. The difference in optimization is really noticeable. In fact, when dealing with 200 critical points, the naive reconstructing algorithm took a bit over 300

seconds while the optimized algorithm took only 1.6 seconds. The difference between time is what makes the optimized algorithm a feasible algorithm to be applied to large quantities of data.

	5	10	25	50	100	150	200
Naive	0.008	0.026	0.455	4.431	36.34	127.1	301.7
Optimized	0.002	0.003	0.017	0.115	0.278	0.835	1.635

Table 1: Comparison of a naive reconstruction algorithm with the rolling ball algorithm. Time is recorded in seconds. The upper row indicates the number of critical points of the given piecewise linear functions.

2.2 The smooth case

Suppose given a smooth function $f \colon [a, b] \to \mathbb{R}$, and assume that $f''(x_i) \neq 0$ for all critical points $(x_i, f(x_i))$. Hence we discard the possibility that f is "too flat" at some critical point (similarly as in the piecewise linear case), and only isolated critical points are considered.

Let T be a tangent line to f at a critical point (x_0, y_0). By [12], (x_0, y_0) is a limit point of intersection points of tangent lines T_j close to T (in the sense that both the slope and the intercept of T_j are close to those of T). In our situation, such tangent lines are provided by directional persistence diagrams approaching the vertical height direction as much as needed (so the tangent lines become close to horizontal).

Our algorithm works as follows. If (x_0, y_0) is a critical point of f with $a < x_0 < b$, then $f'(x_0) = 0$ and the horizontal line $y = y_0$ is a critical line in the vertical direction. Hence, in order to obtain x_0 from directional persistence diagrams, it suffices that the algorithm detects the existence of (x_0, y_0) and that it converges to x_0. We treat detection and convergence separately.

Detection of critical points. Let (x_0, y_0) be a critical point of $f \colon [a, b] \to \mathbb{R}$ with $a < x_0 < b$. For a direction $v = e^{i\theta}$ with $0 < \theta < \pi/2$ and a positive real number $\tau > 0$, we say that (x_0, y_0) is τ-detected by the direction v if there are critical lines of slopes $\pm 1/\tan\theta$ intersecting $y = y_0$ at points (x_1, y_0) and (x_2, y_0) with $x_1 < x_0 < x_2$ and $x_2 - x_1 < \tau$ (see Fig. 4), and there is no other critical point of f whose x-coordinate lies between x_1 and x_2.

Proposition 2.4. *Suppose that (x_0, y_0) is a critical point of a smooth function $f \colon [a, b] \to \mathbb{R}$. Suppose that $|f''(x)| > \varepsilon$ if $|x - x_0| < \delta$ for some numbers $\delta > 0$ and $\varepsilon > 0$. If we pick an angle $0 < \theta < \pi/2$ such that $\tan\theta > 1/\delta\varepsilon$ and a number τ with $\tau > 2\delta$, then (x_0, y_0) is τ-detected by $e^{i\theta}$.*

Proof. Suppose first that $f''(x_0) > 0$. Note first that

$$f'(x_0 + \delta) = \int_{x_0}^{x_0+\delta} f''(t)\,dt > \delta\varepsilon > \frac{1}{\tan\theta}.$$

Hence there is a value a with $x_0 < a < x_0 + \delta$ such that the tangent to the graph of f at $(a, f(a))$ has slope $1/\tan\theta$. Now the Lagrange formula for the remainder of the first-order Taylor polynomial of f at x_0 yields

$$f(a) = y_0 + \tfrac{1}{2}f''(c)(a - x_0)^2 > y_0 + \tfrac{1}{2}\varepsilon(a - x_0)^2$$

for some c with $x_0 < c < a$. The tangent to the graph of f at $(a, f(a))$ intersects $y = y_0$ at a point (x_2, y_0) and the parallel line through $(a, y_0 + \tfrac{1}{2}\varepsilon(a - x_0)^2)$ intersects $y = y_0$ at (x_*, y_0) with $x_* = a - \tfrac{1}{2}\varepsilon\tan\theta(a - x_0)^2$. Since $\tau > 2/\varepsilon\tan\theta$ and $a - x_0 < \delta < \tfrac{1}{2}\tau$, we find that

$$x_2 - x_0 < x_* - x_0 = (a - x_0)\left[1 - \tfrac{1}{2}\varepsilon\tan\theta(a - x_0)\right]$$

$$< (a - x_0)\left(1 - \frac{a - x_0}{\tau}\right) < \tfrac{1}{2}(x_0 + \tau - a) < \tfrac{1}{2}\left(\tfrac{1}{2}\tau + \tfrac{1}{2}\tau\right) = \tfrac{1}{2}\tau.$$

By symmetry, there is a tangent with slope $-1/\tan\theta$ at a point $(b, f(b))$ with $x_0 - \delta < b < x_0$ intersecting $y = y_0$ at a point (x_1, y_0) with $x_0 - x_1 < \tfrac{1}{2}\tau$. Consequently, $x_2 - x_1 < \tau$. Moreover,

there is no other critical point whose x-coordinate lies between x_1 and x_2 since f' is monotone in this interval. Hence, (x_0, y_0) is τ-detected by the direction $e^{i\theta}$. Again by symmetry, the conclusion is the same if $f''(x_0) < 0$. □

Convergence. Suppose that (x_0, y_0) is a local *minimum* that has been detected by the algorithm. Let L_0 be the corresponding horizontal critical line $y = y_0$, and let $v_1 = e^{i\theta_1}$ be the direction involved in the successful detection process. Choose another direction $v_2 = e^{i\theta_2}$ with $0 < \theta_1 < \theta_2 < \pi/2$ (in fact, we pick θ_2 sufficiently close to $\pi/2$).

Let us denote $m_1 = 1/\tan\theta_1$ and $m_2 = 1/\tan\theta_2$, and let L_1, L_2, L_3, L_4 be the corresponding critical lines of respective slopes $-m_1, m_1, -m_2$ and m_2. Let (x_i, y_0) denote the intersection point of L_i with L_0 for $i = 1, 2, 3, 4$.

The x-coordinate x_0 of the critical point M is located between x_3 and x_4 (although it need not be the midpoint), and we can approximate it by considering the line L passing through the vertices P and Q of the triangles formed by L_0, L_1, L_2 and by L_0, L_3, L_4 respectively. The intersection point $M_* = L \cap L_0$ is (x_*, y_0), where

$$x_* = \frac{m_2(x_1 + x_2)(x_3 - x_4) - m_1(x_1 - x_2)(x_3 + x_4)}{2m_2(x_3 - x_4) - 2m_1(x_1 - x_2)}. \tag{2.1}$$

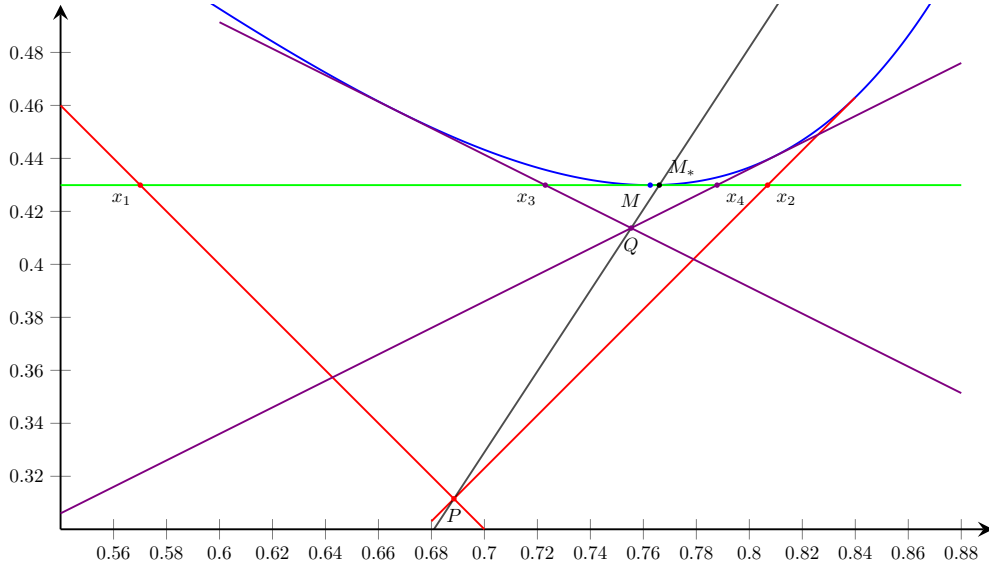This is taken as an approximation to the true x-coordinate of $M = (x_0, y_0)$.

Figure 4: An approximation M_* to the critical point M is obtained by means of five critical lines. The point M_* is the intersection of the line $L = PQ$ with the horizontal tangent L_0. Here P is the intersection of tangents with slopes $\pm 45°$ and Q is obtained using slopes of $\pm 30°$. The function represented in the picture is $f(x) = x^9 - x^4 + x^3 - x + 1$.

This step can be repeated with smaller slopes in order to achieve any desired precision, and the case of a local maximum is treated analogously, by symmetry.

Theorem 2.5. *Let $f: [a, b] \to \mathbb{R}$ be a smooth function, and let (x_0, y_0) be a critical point of f with $a < x_0 < b$ and such that $f''(x_0) \neq 0$. Then x_0 can be approximated with any arbitrary degree of precision by means of critical lines.*

Proof. If $f''(x_0) \neq 0$ then there exist $\varepsilon > 0$ and $\delta > 0$ such that $|f''(x)| > \varepsilon$ if $|x - x_0| < \delta$. Use Proposition 2.4 to find an angle θ with $0 < \theta < \pi/2$ and a number $\tau > 0$ such that (x_0, y_0) is τ-detected by $e^{i\theta}$. Next, choose a sequence of pairs of critical lines L_{2k-1}, L_{2k} tangent to the graph of f with slopes $\pm 1/\tan\theta_k$ tending to zero, and intersecting the line $y = y_0$ at points within $(x_0 - \frac{1}{2}\tau, x_0 + \frac{1}{2}\tau)$. If we calculate (2.1) in each case, the resulting sequence converges to x_0 as in [12]. □

7

The five-line algorithm. The first step of the algorithm aims to detect all critical points, even with a risk of introducing false positives. Since the functions are assumed to be unknown and the algorithm depends on an oracle with information from directional persistence diagrams, we use the same values of θ and τ for all critical points. All triangles with base less than τ formed by critical lines are tentatively assumed to contain a critical point. The algorithm usually starts with a slope of $30°$ and a value $\tau = 0.08$.
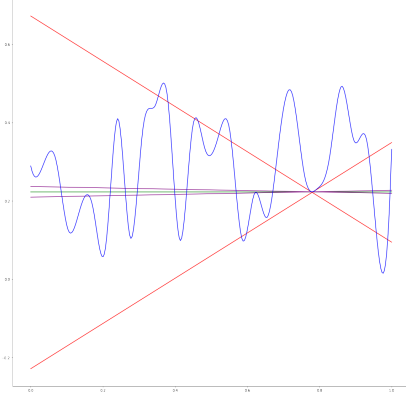
(a) Location of an asymmetric critical point by means of five tangent lines.
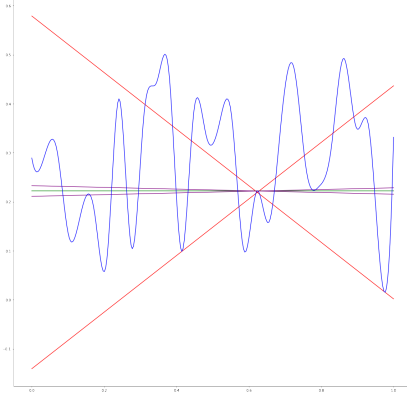
(b) A critical point coincidentally detected by quasi-multiple tangent lines (red).
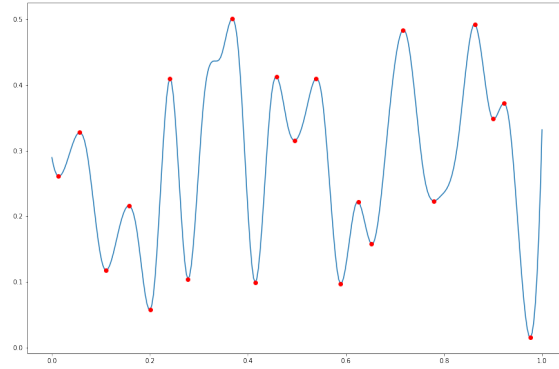
(c) The algorithm finds all critical points except perhaps endpoints (inflection points are not considered).

Figure 5: Instances of the five-line algorithm for smooth functions.

The second step of the algorithm tries to eliminate false positives and accurately locate each critical point for true positives. Our working hypothesis is that critical lines with slope close to 0 will only pass inside a triangle from the first step (that is, x_3 and x_4 are between x_1 and x_2 as depicted in Fig. 4) if that triangle truly contains a critical point. Typically, we use a slope of $0.1°$ for that step, and we disregard candidate triangles where betweenness fails. Finally, we apply the formula (2.1) to every resulting pair of triangles to approximate the enclosed critical point.

The choice of parameters depends on the nature of the signals to which the algorithm is to be applied. In the first step of the algorithm, the choice of τ is crucial. Smaller values of τ may miss some critical points but also reduce the number of candidate triangles, thus diminishing the possibility of a false positive. Bigger values of τ ensure that no critical point is missed, but the chances of false positives increase. Figure 5a shows an example where a critical point could be missed due to asymmetry.

The working hypothesis made in the second step of the algorithm can fail due to the existence of lines that are tangent at a critical point and very close to being tangent at another critical point of the graph. We call the latter *quasi-multiple* tangent lines.

Experimental results show that triangles formed with quasi-multiple tangent lines usually appear next to triangles that truly contain a critical point. In those cases, the same pair of critical lines with small slope appears between both triangles, resulting in closeby duplicate images of the same

critical point. Since lines close to horizontal are the ones really deciding where a critical point is, in the algorithm we impose that each pair of lines is used only once. If the same pair of lines passes inside another triangle, then the resulting approximate critical point will be very close to one that we already have, and thus there is no need to recompute it. Figure 5b shows an example of a critical point detected by a triangle of quasi-multiple tangent lines.

In most cases the algorithm retrieves all critical points correctly, as in Fig. 5c. However, if we suppose that the data functions are unknown, there is no way to decide if a function is correctly approximated or not. For this reason, we normally restrict the algorithm to five tangent lines and do not iterate further. Instead, we have implemented a simple test that checks if local maxima and local minima alternate in the resulting graph.

We first applied the five-line algorithm to 100 functions generated as combinations of sines and cosines, and checked in which cases all the critical points were properly retrieved with at least 4 digits of precision. All functions shared the domain $[0, 1]$ and had between 20 and 30 critical points. This kind of functions may present periodicity, and critical points are usually symmetric. By using slopes of $30°$ and $0.1°$, and $\tau = 0.08$, we achieved 89% of correctly approximated functions. Errors in the reconstruction tended to be due to false negatives, i.e., candidate critical points that had been left out in the first step of the algorithm, rather than false positives.

A second test was performed over 100 functions generated by spline interpolation of third order over 30 points on the interval $[0, 1]$. Such functions are harder to approximate since inflection points, flat critical points and asymmetric critical points appear with more frequency than in the previous case. Again, we obtained around 90% of properly approximated functions. For this kind of functions, errors regarding false positives and false negatives were balanced.

3   Approximating a function from selected landscape levels

Landscapes of persistence diagrams were introduced in [3] as follows. For each point (β, δ) in a given persistence diagram, one denotes $\Lambda_{(\beta,\delta)}(t) = \max\{0, \min\{t - \beta, \delta - t\}\}$. Then a piecewise linear function $\lambda_k \colon \mathbb{R} \to \mathbb{R}$ is defined for each $k \geq 1$ as

$$\lambda_k(t) = \mathrm{kmax}\{\Lambda_{(\beta_i, \delta_i)}(t)\},$$

where $\{(\beta_i, \delta_i)\}$ is the set of all points in the given persistence diagram and kmax returns the k-th largest value of a given set of real numbers whose elements are possibly counted with multiplicities, or zero if there is no k-th largest value. Since the number of points in a persistence diagram is finite, there is an integer n such that $\lambda_k = 0$ for $k \geq n$. The first landscape levels $\lambda_1, \lambda_2 \ldots$ depict the most significant features from the persistence diagram, while the last ones correspond to short-living phenomena or perhaps to noise.
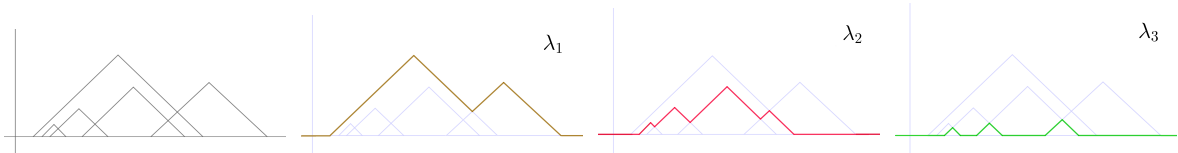
Figure 6: Sequence of nonzero levels of a persistence landscape (left).

Suppose given a continuous function $f \colon [a, b] \to \mathbb{R}$ and consider the persistence diagram of sublevel sets of f in the vertical direction. In this case, for each point (β, δ) in the persistence diagram, the coordinates β and δ correspond to y-values of critical points of f by Proposition 1.2, except for $\delta = \infty$. Hence it is possible to determine which critical points of f are associated with each landscape level λ_k. This is done under the assumption that the function f is known, so our task is to find out which subset of its critical points correspond to λ_k for a given value of k.

Our algorithm to do this is the following. Denote by (t, u) the coordinates of landscape points. A point $(t, 0)$ is called a *take-off point* for a nonzero landscape function λ_k if there is an $\varepsilon > 0$ such that

$\lambda_k(s) = 0$ for $t - \varepsilon < s \leq t$ and $\lambda_k(s) \neq 0$ for $t < s < t + \varepsilon$. Similarly, $(t, 0)$ is a *landing point* for λ_k if there is an $\varepsilon > 0$ such that $\lambda_k(s) \neq 0$ for $t - \varepsilon < s < t$ and $\lambda_k(s) = 0$ for $t \leq s < t + \varepsilon$.

Proposition 3.1. *Let λ_k be a nonzero landscape level of a continuous function $f \colon [a, b] \to \mathbb{R}$. Let $\{t_i\}$, $i \geq 0$, be the ordered set of t-coordinates of vertices in the graph of λ_k, where $t_i < t_{i+1}$ for all i. Then the sequence defined by $y_i = t_i$ if $(t_i, 0)$ is a take-off point and $y_i = 2\left(t_i - \frac{1}{2}y_{i-1}\right)$ otherwise is a sequence of y-values of critical points of f.*

Proof. If $(t_i, 0)$ is a take-off point of λ_k (including the case $i = 0$) then t_i is the birth coordinate β of a point (β, δ) in the persistence diagram of sublevel sets of f, hence a y-value of some critical point of f. If (t_i, u_i) is a local maximum of λ_k, then $t_i = \frac{1}{2}(\beta_p + \delta_q)$ and $u_i = \frac{1}{2}(\delta_q - \beta_p)$ for points (β_p, δ_p) and (β_q, δ_q) in the persistence diagram (that may be equal or different), and $y_{i-1} = \beta_p$. Hence

$$y_i = 2\left(t_i - \tfrac{1}{2}y_{i-1}\right) = 2\left(t_i - \tfrac{1}{2}\beta_p\right) = \delta_q,$$

which is a y-value of a critical point of f. If (t_i, u_i) is a local minimum of λ_k, then $t_i = \frac{1}{2}(\beta_p + \delta_q)$ and $u_i = \frac{1}{2}(\delta_q - \beta_p)$ for some points (β_p, δ_q) and (β_p, δ_q) as well, yet now $y_{i-1} = \delta_q$. In this case,

$$y_i = 2\left(t_i - \tfrac{1}{2}y_{i-1}\right) = 2\left(t_i - \tfrac{1}{2}\delta_q\right) = \beta_p.$$

If $(\delta, 0)$ is a landing point, then $y_{i-1} = \delta$ in the preceding maximum. Therefore $2\left(\delta - \frac{1}{2}y_{i-1}\right) = \delta$, which comes repeated and hence landing points can be omitted. $\square$

This method produces a list of y-values of critical points of f associated with a subset of selected landscape levels. Next, the values in this list are compared with the list of all critical points of f in order to obtain the matching x-values. The precise algorithm is detailed below as Algorithm 8. If two or more critical points of f coincidentally have the same y-value, the algorithm retrieves them simultaneously.

If all nonzero landscape levels λ_k are used, then all the critical points of f are recovered. As shown in the proof of Proposition 3.1, local minima of f are obtained from take-off points or local minima of landscape functions, while local maxima of f are paired with local maxima of the corresponding landscape functions. An example of a full reconstruction is shown in Fig. 7. Since the reconstruction is done sequentially by levels, if only a subset of landscape levels is provided then a partial reconstruction is obtained.

3.1   An application to importance attribution in neural networks

Our work was initially motivated by a practical problem, namely finding out which information from signals is discriminative for classification by a neural network, which is an important study subject in machine learning.

Towards this goal, the aforementioned algorithm was applied to classification of a sample of 87,554 electrocardiogram signals (ECG) taken from the MIT-BIH Arrhythmia Database [14, 15], with the purpose of characterizing the relevant features that define each of five classes of interest: one corresponding to normal beats, three classes corresponding to different arrhythmia types, and one class for unidentifiable heartbeats. The database is accessible from [6]. A neural network consisting of three convolutional layers combined with max pooling layers followed by two dense layers was designed for the classification task. A custom gating layer was placed before the dense layer as described in [9].

Zero-homology of sublevel sets in the vertical direction was computed in order to pre-process each ECG sample and obtain its persistence landscapes. The resulting sequences of functions λ_k were fed into a neural network for classification, and the custom gating layer was used to assign a weight to each landscape level in every sequence. Those landscape levels with the largest weights were considered as the ones conveying the most relevant information for the classification task. Next, Algorithm 8 was used to approximate the shape that contributed the most in characterizing each of the classes. This approximation only leverages information from the selected landscape levels. As a result, the data signals were partially reconstructed using only a subset of their persistence landscapes.

Table 2 shows average accuracy (percentage of correctly classified samples) using a 5-fold cross-validation. The table compares the classification accuracy of a neural network fed with the original
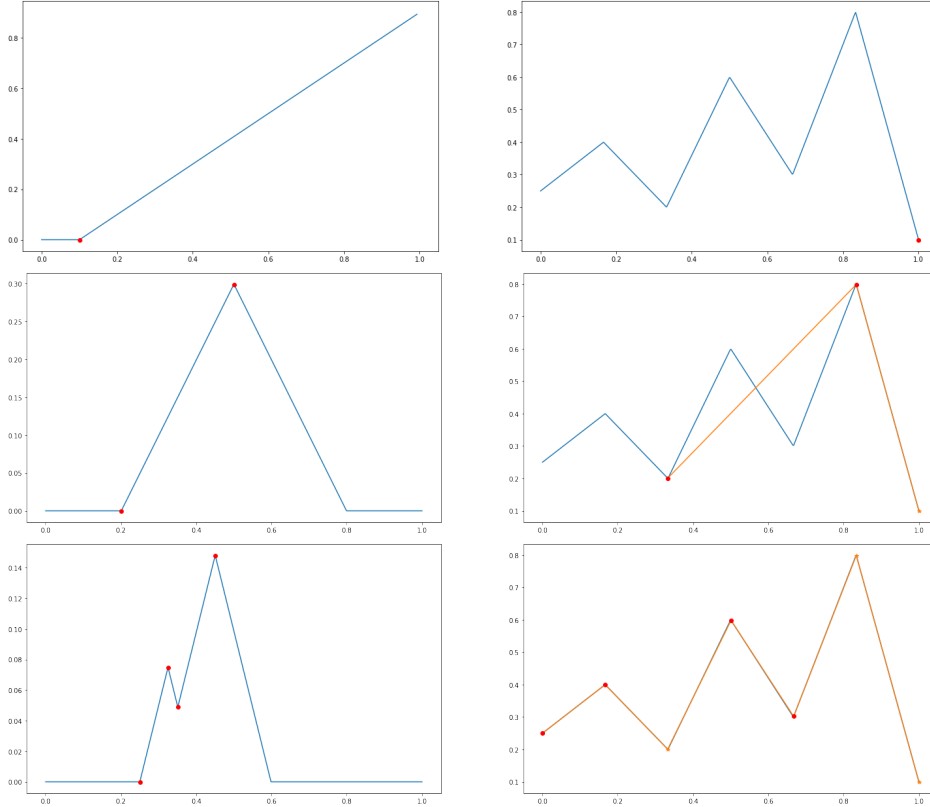
Figure 7: Each landscape level (left) matches a subset of critical points of a given function (right). The pictures illustrate how the vertices of the successive landscape levels are paired with local extrema of the function (marked in red). Landscape heights have been rescaled.

unprocessed signal with the same architecture using a set of ten levels of landscape decomposition, and the accuracy when using only the three most important landscape levels. The results show that the decomposition of the signals using persistence landscapes usually produces a small loss in classification accuracy. This was expected, as the decomposition process loses information in terms of complete signal reconstruction. Despite this fact, the results were close to each other —average accuracy of 98.41% with the original data functions versus 94.55% for the set of the first 10 landscape levels and 94.00% using only the most meaningful 3 landscape levels.

The simplified versions of data signals were subsequently fed into the same neural network in order to find out if the data features emphasized by our reconstruction method were sufficient for the network's classifications task. The results can also be seen in Table 2 and demonstrate that, indeed, the accuracy obtained with the simplified functions was comparable to that of the original data.

	Raw data	10 levels	3 levels	Reconstructed
Accuracy	98.41 ± 0.09	94.55 ± 0.16	94.00 ± 0.13	97.04 ± 0.14

Table 2: Classification accuracy of a neural network fed with raw data versus processed data using ten landscape levels and the most significant three levels, as well as data reconstructed from three landscape levels.

The outcome of persistent homology adds considerable value towards a more comprehensive attribution along the classification problem. A ranking of landscape importance for the network can be seen in Fig. 8. Observe that the first three landscape levels gather most of the attribution power.
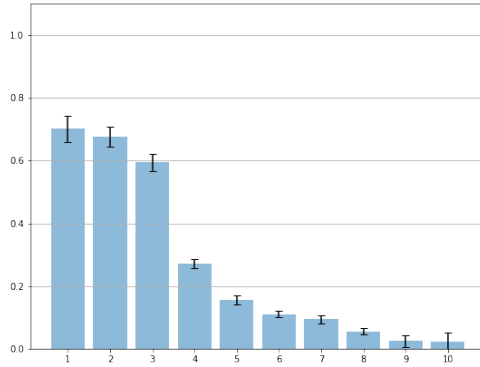
Figure 8: Rank of the importance of each landscape level based on results of a neural network. Higher bars represent greater importance. Confidence intervals correspond to five repetitions of each classification.

4   Discussion

This article is a contribution to the general inverse problem of reconstructing data from persistence diagrams [1, 8, 16]. While previous works focused mainly on simplicial complexes (in particular, on planar graphs), our method applies to both piecewise linear functions and smooth functions, albeit with distinct algorithms. We implemented our techniques and tested it with examples from various sources, namely piecewise linear functions, smooth functions obtained using a harmonic function generator, and spline interpolations of third order. We also used our method with data from a biomedical study [15] involving electrocardiographic signals, which had been classified into five classes. Our motivation was to address a case of the attribution problem in neural network classifiers.

Our algorithm in the piecewise linear case is a deterministic implementation, with only mild assumptions on the data functions, of the general procedure of using three directional persistence diagrams to reconstruct a planar graph. However, in the application to heartbeat registers, we used a simplified form of the algorithm due to the fact that the neural network was fed with sequences of landscape levels, each of which derived from a sole persistence diagram. Since the data functions remained available during the classification process, one persistence diagram was sufficient to find the relevant critical points among the set of all vertices of the given graphs. Our method allowed us to simplify the given graphs by substantially reducing the number of critical points while keeping essential information about what the neural network was focusing on. After the classification had been performed, we carried out a reduction of the number of critical points based on what the network regarded as important, thus highlighting the parts of the original function that were crucial.

In the case of smooth functions, five directional persistence diagrams are generally needed for reconstruction, except possibly in instances where the algorithm encounters a difficulty to locate some critical point, which can be solved by means of additional tangent lines. We stated a general result (Theorem 2.5) with assumptions guaranteeing the convergence of the algorithm, hence yielding an approximation of a given smooth function with a piecewise linear function with the same set of critical points, using persistence diagrams of sublevel sets.

In the smooth case, our algorithm can fail to detect some critical points, or instead converge to false positives if the assumptions of the reconstruction theorem are not fulfilled. In practice a trade-off is required when choosing the τ value in the algorithm in order to minimize the chances of errors of both kinds. Functions with a large number of critical points per length unit may require more careful choices of τ.

Also critical points where the second derivative vanishes or is very close to zero may cause detection difficulties. The nonvanishing assumption on f'' aims to prevent the existence of critical points that may be missed by the algorithm. We excluded graphs with horizontal segments, although our

algorithms could easily be modified in order to treat such cases as well. In the piecewise linear case, the algorithm detects either the beginning or the end of each horizontal segment, and in order to detect the other end it would suffice to repeat the process using critical lines with reverse slope. In the smooth case, the algorithm still works in the case of sufficiently short horizontal segments (for which it finds the midpoint), although there is an increased risk of not detecting them. Indeed, detection of critical points with f'' close to 0 requires larger values of τ.

False occurrences of critical points may be caused by the existence of multiple tangent lines or *quasi-multiple* tangent lines, that is, lines that are tangent at a critical point and very close to being tangent at another critical point. In order to prevent erroneous detection of nonexistent critical points, we perform a global checking, namely that critical points alternate between local maxima and local minima. This allows us to eliminate false positives in some cases.

There are two ways in which the numerical precision of the code can affect the result of our algorithm. On one hand, graphs are stored as vectors, and if the step is too large then flat segments may be created at critical points. On the other hand, if the threshold τ to detect a critical point is small compared with the code precision, then we could lose critical points.

If a given function f is defined on an interval $[a, b]$, then the points $(a, f(a))$ and $(b, f(b))$ can only be detected if they are local minima. Otherwise the sublevel set does not change its topology when passing through them. To deal with this issue, the location of the initial point and the final point of the graph of f should be given as input to the reconstruction algorithm. A similar strategy could be used to deal with graphs starting and ending with horizontal segments, which is sometimes the case in practice.

While it was feasible to optimize our algorithm in the piecewise linear case, reducing the complexity in the smooth case would require a new design of the algorithm. In its current form, the detection step and the approximation step are successive. Detection of critical points is made in $O(n^3)$ time, where n is the number of critical points, and further calculations are needed pointwise in order to determine each precise location. A more efficient algorithm could be produced by mixing the two steps into a simultaneous searching procedure.

This study illustrates another instance of the usefulness of persistence landscapes as descriptors from homological persistence diagrams. Thanks to the fact that landscapes are made of a hierarchical sequence of levels, it is possible to analyze the way in which information from data is distributed along the sequence of levels. As a consequence, landscapes are valuable as a tool for interpretability in classification processes.

Reconstruction of data is also related with privacy in data science [2]. In our case, knowledge of a persistence diagram of sublevel sets of a function does not permit to fully reconstruct the function, since the x-values of critical points cannot be recovered from a persistence diagram. However, as shown in this article, an oracle yielding enough directional persistence diagrams conveys as much information as the original functions. Towards applications, removing critical points that are not essential for classification purposes is a way to erase information stored in the data while preserving usability. Consequently, selective critical point removal could be a convenient method to diminish the chances of deanonymization when the data functions encode confidential content.

References

[1]   Robin Lynne Belton et al. "Reconstructing embedded graphs from persistence diagrams". In: *Computational Geometry: Theory and Applications* 90 (2020), p. 101658.

[2]   Abhishek Bhowmick et al. "Protection against reconstruction and its applications in private federated learning". In: (2018). arXiv: `arXiv:1812.00984v2 [stat.ML]`.

[3]   Peter Bubenik. "Statistical topological data analysis using persistence landscapes". In: *Journal of Machine Learning Research* 16 (2015), pp. 77–102.

[4]   Frédéric Chazal et al. "Persistence modules". In: *The Structure and Stability of Persistence Modules*. SpringerBriefs in Mathematics, Springer, Cham, 2016, pp. 405–433.

[5] Justin Curry, Sayan Mukherjee, and Katharine Turner. "How many directions determine a shape and other sufficiency results for two topological transforms". In: (Sept. 2021). arXiv: `1805.09782v3 [math.AT]`.

[6] *ECG Heartbeat Categorization Dataset. Segmented and Preprocessed ECG Signals for Heartbeat Classification.* URL: `https://www.kaggle.com/datasets/shayanfazeli/heartbeat`.

[7] Herber Edelsbrunner and John Harer. "Persistent homology: A survey". In: *Contemporary Mathematics* 453 (2008), pp. 257–282.

[8] Brittany Terese Fasy et al. "The first algorithm for reconstructing simplicial complexes of arbitrary dimension from persistence diagrams". In: (Jan. 2022). arXiv: `1912.12759v4 [cs.CG]`.

[9] Aina Ferrà, Carles Casacuberta, and Oriol Pujol. "Importance attribution in neural networks by means of persistence landscapes of time series". In: (Feb. 2023). arXiv: `2302.03132 [cs.LG]`.

[10] Robert Ghrist. "The persistent topology of data". In: *Bulletin of the American Mathematical Society* 45 (2008), pp. 61–75.

[11] Robert Ghrist, Rachel Levanger, and Huy Mai. "Persistent homology and Euler integral transforms". In: *Journal of Applied and Computational Topology* 2 (2018), pp. 55–60.

[12] Alan Horwitz. "Reconstructing a function from its set of tangent lines". In: *The American Mathematical Monthly* 96.9 (1989), pp. 807–813.

[13] Jacob Leygonie, Steve Oudot, and Ulrike Tillmann. "A framework for differential calculus on persistence barcodes". In: *Foundations of Computational Mathematics* (2021), pp. 1–63.

[14] George B. Moody and Roger G. Mark. *MIT-BIH Arrhythmia Database.* 2005. URL: `https://physionet.org/content/mitdb/1.0.0/`.

[15] George B. Moody and Roger G. Mark. "The impact of the MIT-BIH arrhythmia database". In: *IEEE Engineering in Medicine and Biology Magazine* 20.3 (2001), pp. 45–50.

[16] Steve Oudot and Elchanan Solomon. "Inverse problems in topological persistence". In: *Topological Data Analysis: The Abel Symposium 2018.* Springer, 2020, pp. 405–433.

[17] The GUDHI Project. *GUDHI User and Reference Manual.* GUDHI Editorial Board, 2015. URL: `http://gudhi.gforge.inria.fr/doc/latest/`.

[18] Katharine Turner, Sayan Mukherjee, and Doug M. Boyer. "Persistent homology transform for modeling shapes and surfaces". In: *Information and Inference: A Journal of the IMA* 3.4 (2014), pp. 310–344.

Algorithm 1: Intersect function

Input: θ_0, θ_1 angles, t_0, t_1 heights of the lines with angles θ_0, θ_1;
Output: A point where the two given lines cross;

1 return $((t_1 \sin\theta_0 - t_0 \sin\theta_1)/\sin(\theta_0 - \theta_1),\ (t_0 \cos\theta_1 - t_1 \cos\theta_0)/\sin(\theta_0 - \theta_1))$

Algorithm 2: Naive reconstruction algorithm

Input: T, S, R persistence diagrams, $\theta_0, \theta_1, \theta_2$ directions, start starting point and end last point of the function to be reconstructed;
Output: List of all critical points of the given function;

1 crits $\leftarrow$ crits $\cup$ {start, end};
2 for $t \in T$ **do**
3 $\quad$ **for** $r \in R$ **do**
4 $\quad\quad$ **for** $s \in S$ **do**
5 $\quad\quad\quad$ $p_{tr} \leftarrow$ intersect$(\theta_0, \theta_1, t, r)$;
6 $\quad\quad\quad$ $p_{ts} \leftarrow$ intersect$(\theta_0, \theta_2, t, s)$;
7 $\quad\quad\quad$ **if** $||p_{tr} - p_{ts}|| < 10^{-6}$ **then**
8 $\quad\quad\quad\quad$ crits $\leftarrow$ crits $\cup$ $\{p_{tr}\}$;
9 $\quad\quad\quad$ **end**
10 $\quad\quad$ **end**
11 $\quad$ **end**
12 **end**
13 **return** crits

Algorithm 3: Rolling ball algorithm for piecewise linear functions

Input: T, S, R persistence diagrams, $\theta_0, \theta_1, \theta_2$ directions, start starting point and end last point of the function to be reconstructed;
Output: List of all critical points of the given function;

1 crits $\leftarrow$ crits $\cup$ {start, end};
2 sort(T, decreasing);
3 sort(S, increasing);
4 for $s \in S$ **do**
5 $\quad$ $i = 0, j = 0$;
6 $\quad$ foundTriple $=$ False;
7 $\quad$ **while** *!foundTriple and $i < |R|$ and $j < |T|$* **do**
8 $\quad\quad$ $p_r \leftarrow$ intersect$(\theta_1, \theta_2, s, R_i)$;
9 $\quad\quad$ $p_t \leftarrow$ intersect$(\theta_1, \theta_0, s, T_j)$;
10 $\quad\quad$ **if** $||p_r - p_t|| < 10^{-6}$ **then**
11 $\quad\quad\quad$ foundTriple $=$ True;
12 $\quad\quad\quad$ crits $\leftarrow$ crits $\cup$ $\{p_r\}$;
13 $\quad\quad$ **else if** $x(p_r) < x(p_t)$ **then**
14 $\quad\quad\quad$ $i = i + 1$;
15 $\quad\quad$ **else**
16 $\quad\quad\quad$ $j = j + 1$;
17 $\quad\quad$ **end**
18 $\quad$ **end**
19 **end**
20 **return** crits

Algorithm 4: Compute the x-coordinate of a critical point

    Input: x_1, x_2, x_3, x_4 x-coordinates near a critical point on a horizontal line,
           m_1, m_2 slopes of critical lines;
    Output: An approximation to the x-coordinate of the critical point;

1 **return** $[m_2(x_1 + x_2)(x_3 - x_4) - m_1(x_1 - x_2)(x_3 + x_4)]/[2m_2(x_3 - x_4) - 2m_1(x_1 - x_2)]$

Algorithm 5: Five-line algorithm for smooth functions

    Input: $\mathsf{T},\mathsf{S},\mathsf{R}$ persistence diagrams corresponding to $\pi/2$, θ_1, $-\theta_1$ respectively,
           $\mathsf{S2}$, $\mathsf{R2}$ persistence diagrams corresponding to θ_2, $-\theta_2$ respectively,
           τ detection parameter;
    Output: List of all critical points of the given function;

1 crits $\leftarrow \{\}$;
2 triangles $\leftarrow \{\}$;
3 **for** $t \in \mathsf{T}$ **do**
4   **for** $r \in \mathsf{R}$ **do**
5   **for** $s \in \mathsf{S}$ **do**
6 $p_{tr} \leftarrow$ intersect$(\pi/2, -\theta_1, t, r)$;
7 $p_{ts} \leftarrow$ intersect$(\pi/2, \theta_1, t, s)$;
8   **if** $|(p_{tr})_x - (p_{ts})_x| < \tau$ **then**
9   triangles $\leftarrow$ triangles $\cup \{(t, s, r)\}$;
10   **end**
11   **end**
12   **end**
13 **end**
14 pass_triangles $\leftarrow \{\}$;
15 used $\leftarrow \{\}$;
16 **for** $(t, r, s) \in$ triangles **do**
17   **for** $ss \in \mathsf{S2}$ **do**
18   **for** $rr \in \mathsf{R2}$ **do**
19   **if** (rr, ss) *between* (t, r, s) *and* $(rr, ss) \notin$ used **then**
20   pass_triangles $\leftarrow$ pass_triangles $\cup \{(t, r, s, rr, ss)\}$;
21   used $\leftarrow$ used $\cup \{(rr, ss)\}$;
22   **end**
23   **end**
24 **end**
25 **for** $(t, r, s, rr, ss) \in$ pass_triangles **do**
26 $x_1 \leftarrow$ intersect$(\pi/2, -\theta_1, t, r)$;
27 $x_2 \leftarrow$ intersect$(\pi/2, \theta_1, t, s)$;
28 $x_3 \leftarrow$ intersect$(\pi/2, -\theta_2, t, rr)$;
29 $x_4 \leftarrow$ intersect$(\pi/2, \theta_2, t, ss)$;
30 $x \leftarrow$ compute_x$((x_1)_x, (x_2)_x, (x_3)_x, (x_4)_x, 1/\tan\theta_1, 1/\tan\theta_2)$;
31 $y \leftarrow (x_1)_y$;
32   crits $\leftarrow$ crits $\cup \{(x, y)\}$;
33 **end**
34 crits $\leftarrow$ sort(crits);
35 **return** crits

Algorithm 6: Get y-values of critical points

Input: l vector of landscape data, t vector of t-coordinates of landscape data;
Output: List of y-values of critical points;

1 crits $\leftarrow \{\}$;
2 for $i \leftarrow 0$ **to** $|l|$ **do**
3 $\quad$ **if** l_i *is a take-off vertex* **then**
4 $\quad\quad$ crits $\leftarrow$ crits $\cup \{t_i\}$;
5 $\quad$ **else if** l_i *is a local minimum or a local maximum* **then**
6 $\quad\quad$ crits $\leftarrow$ crits $\cup \{t_i\}$;
7 $\quad\quad$ crits$_i \leftarrow 2(\text{crits}_i - 0.5\,\text{crits}_{i-1})$;
8 end
9 return crits

Algorithm 7: Get x-values of critical points

Input: crits list of y-values of critical points, *original* vector of original data, x vector of x-coordinates of original data;
Output: List of x-values of critical points;

1 candcrits $\leftarrow \{\}$;
2 for $crit \in$ crits **do**
3 $\quad$ $compare \leftarrow (original - crit)^2$;
4 $\quad$ **for** $i \leftarrow 0$ **to** $|compare|$ **do**
5 $\quad\quad$ **if** $compare_i < 10^{-4}$ **then**
6 $\quad\quad\quad$ candcrits $\leftarrow$ candcrits $\cup \{i\}$;
7 $\quad$ **end**
8 end
9 xcrits $\leftarrow \{\}$;
10 for $i \in$ candcrits **do**
11 $\quad$ **if** $original_i$ *is a local minimum or a local maximum* **then**
12 $\quad\quad$ xcrits $\leftarrow$ xcrits $\cup \{x_i\}$;
13 end
14 return xcrits

Algorithm 8: Subset of critical points of a function associated with a set of landscape levels

Input: landscapes a list of landscape levels, *original* vector of original data;
Output: List of selected critical points of the given function;

1 crits $\leftarrow \{\}$;
2 for $land \in$ landscapes **do**
3 $\quad$ $candcrits \leftarrow \{\texttt{get\_y\_values}(land)\}$;
4 $\quad$ crits $\leftarrow$ crits $\cup \{\texttt{get\_x\_values}(candcrits, original)\}$;
5 end
6 crits $\leftarrow \texttt{sort}(\text{crits})$;
7 return crits