Cracks and Strain: Visualizing Spatial Distortions in Dimensionality Reduction

Wiley Corning

Fall 2021

1 Introduction

Dimensionality reduction (DR) techniques are an effective tool for enabling the human exploration of high-dimensional datasets. One standard and prevalent technique is to visualize dimensionally-reduced data in the form of a 2D scatterplot. In this visualization pipeline, the designer aims to apply a DR transformation that encodes some structural quality of the high-dimensional dataset into 2D position, allowing a human viewer to observe this quality and perform relevant tasks.

While DR can provide a useful view of the data, it must by definition discard some information. In the case of PCA, a *linear* DR technique, we retain each point's projection only along the two basis vectors of greatest variance in the dataset; the position of each point along every other axis is lost. In t-SNE, a *nonlinear* DR technique, we optimize the 2D representation of the dataset in an attempt to preserve its local topology, thereby approximating the high-dimensional neighborhood structure.

The loss of information in DR can create distortions in the resulting visualization. For example, two points that are distant from one another on a non-principle axis may be flattened together by PCA, creating an impression of similarity that could be misleading. Depending on its parametrization, t-SNE may create spurious clusters or obscure real clusters; it also tends to expand tight clusters and compress loose ones [4]. The basic 2D scatterplot provides no means of determining whether such distortions are present, and to what extent.

Our goal in this project is to augment the 2D scatterplot with visual cues representing the spatial distortions caused by dimensionality reduction. We introduce the visual metaphors of *cracks* and *strain* to indicate regions that have been "pulled apart" or "pushed together", respectively. After an initial design process using hand-drawn prototypes, we implement the crack and strain layers in Python with Matplotlib. We demonstrate the results of our technique on a series of examples, and informally evaluate its successes and drawbacks via critique. We conclude with a discussion of limitations and potential future work.

2 Concept

In designing our solution, we began with the basic concept of rendering a background to the 2D scatterplot that would convey its distortions. This background layer would consist of marks in the negative space of the chart, visually distinct from the marks encoding data points, which would indicate the extent to which nearby points have been "pushed" together or "pulled" apart. The background layer should be empty for an undistorted transformation (e.g., running PCA on a set of coplanar points). Finally, while this is not the main emphasis, the result should add to the decorative appeal of the scatterplot.

We also consider our possible solution in light of the goals a viewer is likely to have when engaging with a DR visualization. Distortion is relevant for tasks that involve the spatial distribution of data points:

- The viewer may be interested in identifying structures, such as clusters, based on the apparent topology of the dataset; our intervention should therefore expose structures that are illusory.
- They may wish to summarize how tightly a given cluster is packed, or to compare the spatial extent between two clusters; our intervention should reveal artificial compression or expansion.
- They may be looking for outliers; we should indicate situations where a non-outlier is distanced or a true outlier is compacted into a cluster.
- They may wish to compare different point-to-point distances; we should provide help in determining whether such a comparison would be valid.

Earlier work has explored the use of interactivity to allow viewers to *probe* the distortions in a DR scatterplot [3]: for example, drawing "motion lines" toward a selected point from every other point to show the extent to which they are inaccurately projected as close or distant. In contrast, our design aims to convey this information statically and globally. It would thus provide at-a-glance information about distortion without requiring additional user engagement. This would also make it suitable for printing and other scenarios with limited interactive capacity.

These design constraints led us to consider two particular ideas: drawing firm lines between some sets of points, and rendering soft halos of color around points and between neighbors. Each idea corresponded to a physical metaphor. The hard lines resembled cracks in ceramic, cuts in paper, or seams in fabric, and could be seen to represent discontinuity; the soft gradients could be interpreted as a measure of strain or pressure. Both of these encodings could naturally fade away in situations with minimal distortion, and neither would place marks directly on top of the data points.

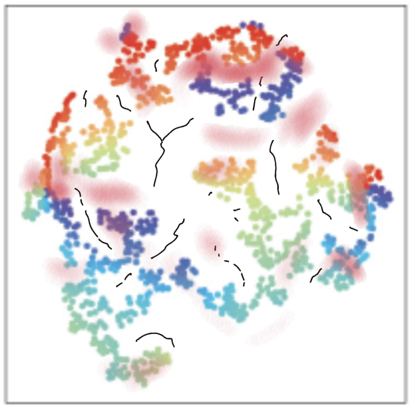

Figure 1: Early sketch (hand-drawn overlay on existing scatterplot).

We chose to implement both ideas simultaneously, using each to represent a different directional quality of distortion. Because strain is suggestive of points being pushed together, we chose to use it to represent areas in which the 2D distance is substantially contracted from 3D distance. This left the cracks to represent the opposite phenomenon, i.e. new gaps being created in the transformation to 2D.

3 Implementation

The initial prototypes for this project took the form of hand-drawn sketches. Our approach in producing these sketches was to take an existing or hypothetical DR scatterplot, manually identify areas of spatial distortion, and draw an additional layer onto the plot which would make these distortions visible. Sketching helped us to refine our concept into a set of concrete ideas, discarding variations that seemed ineffective. It also suggested that the graph of 2D-neighboring points would be the most useful data structure on which to build; this led us to consider the Voronoi diagram as a fundamental building block.

Note on terminology. In our subsequent discussion, we will use the term *data point* to refer to a record in our dataset (before or after DR); *ridge* to refer to one of the line segments bordering a Voronoi cell; and *corner* to refer to a point (not a data point) at which two or more ridges meet.

Once the core concepts of the design were settled, we began prototyping a software implementation in Python using Matplotlib and Jupyter notebooks. Our code uses the SciPy library to compute the Voronoi graph data structure. We use scikit-learn to perform t-SNE and PCA, as well as to source some of our example datasets. We began the our prototyping process by copying code from SciPy's `voronoi_plot_2d` function, which draws a 2D scatterplot overlaid with the Voronoi diagram's ridges and corners; we then iterated on this code and added additional functionality to implement our own visualizations.

3.1 Cracks

The implementation of the crack layer is straightforward. For each pair of neighboring points in the Voronoi diagram, we compute a scalar weight that quantifies their DR-induced separation. This weight is then encoded as the opacity of the ridge separating the two points. Pairs whose distance is undistorted by DR, or that have been brought closer together, will not be visibly separated by a crack.

Let us define the weight formula as follows. Let (a, b) be a pair of neighboring points in 2D, and let (a', b') be their original high-dimensional forms. Then the weight function $w(a, b)$ is given by:

$$w(a,b) = ln(\frac{|b-a|}{|b'-a'|})$$

I.e., it is the logarithm of the ratio of the points' 2D distance to their high-dimensional distance, per the L_2 norm. This measure will be positive iff the points have been separated.

Note that, without further adjustment, this measure could be rendered useless by artifacts of the DR transformation. For example, suppose a set of high-dimensional points happens to be coplanar. Applying a linear DR technique might leave the points' relative relationships intact while scaling the entire plane, which would cause $w(a, b)$ to be a constant nonzero value for all pairs. To correct for this, we take the mean μ of all distance ratios, and divide each pair's distance ratio by μ before applying the logarithm.

3.2 Strain

We experimented with several different approaches for rendering the strain map.

As an initial proof of concept, we drew a line between each pair of neighboring points and set the opacity of the line according to the pair's strain value. This was sufficient for testing the strain function, but otherwise did not achieve our goals. The lines between closely-packed points were difficult to distinguish, and sometimes made the points themselves difficult to see. The lines also created the impression of an underlying network structure in the data, when in reality it was derived from the Voronoi diagram and related only to the positioning of the points.

Our next approach drew on ideas from 3D graphics. We generated a triangular mesh by connecting each Voronoi ridge with edges to its adjacent data points. For each ridge, we computed a weight based on the strain value between its data points; we then assigned a weight to each corner of the Voronoi graph by averaging the weights of its connected ridges. We furthermore assigned each data point a weight of zero. Rendering the triangular mesh with Gouraud interpolation produced a set of gradients between each point and the edges of its Voronoi cell.

Ultimately, we found that this technique did not produce the intended visual result. High strain values tended to accumulate at the Voronoi corners, creating peaks in seemingly arbitrary locations rather than directly between pairs of points. Long ridges, such as those extending out into the border of the plot, would result in oddly stretched gradients that would deposit strain into completely empty areas of the map. Because the gradient within each triangle was linear, the seams between triangles were perceptible in the image.

We also considered a variation of the triangle mesh which placed a vertex at the midpoint of each Voronoi ridge. While this was intended to reduce the irregularity of the strain map, in practice it only exacerbated the existing issues.

In our next (and ultimately final) revision of the strain map, we sought to implement a technique that would be invariant to ridge length and avoid visual artifacts at corners. To that end, we constructed a scalar field mapping each position in $\mathbb{R}^2$ to an associated strain value, and then sampled this field along a grid to produce a *strain image*. This image could then be rendered beneath the scatterplot; we used bicubic interpolation to create a smooth visual result.

Our scalar field is constructed as follows. For each pair of neighboring points a, b, we compute a single strain value $s(a, b)$; this strain value is then assigned to a *point source* situated at the midpoint between a and b. $s(a, b)$ is simply the inverse of the crack weight:

$$s(a,b) = ln(\frac{|b' - a'|}{|b - a|})$$

The strain contributed by (a, b) at a given sample location x is then computed using exponential falloff:

$$\hat{s}(x, (a,b)) = s(a,b) \cdot 2^{-\beta|\frac{a+b}{2} - x|}$$

To create a "halo" effect around each individual point, we also add a zero-weighted point source at each data point.

Finally, an artificial zero-weighted source is considered at a constant distance γ.

The total strain at a given point x is then computed as the average of all point source contributions, weighted by their distance to x:

$$\hat{S}(x) = \frac{\Sigma_{(a,b)} s(a,b) \cdot 2^{-\beta|\frac{a+b}{2} - x|}}{(\Sigma_{(a,b)} 2^{-\beta|\frac{a+b}{2} - x|}) + (\Sigma_a 2^{-\beta|a - x|}) + 2^{-\beta\gamma}}$$

This formula has a few noteworthy features:

- Although the field is generated by point sources, it contains no singularities due to the choice of falloff.
- The formula is a weighted average, not a sum (as you would more typically see in, say, an electric field).

- The parameter β controls the extent to which strain is "diffused" outward from point sources. Higher values of β will correspond to a more closely localized impact from each source.
- The parameter γ causes the total strain to converge toward zero at positions that are far from any point source. Smaller values of γ induce a more rapid falloff.

4 Examples

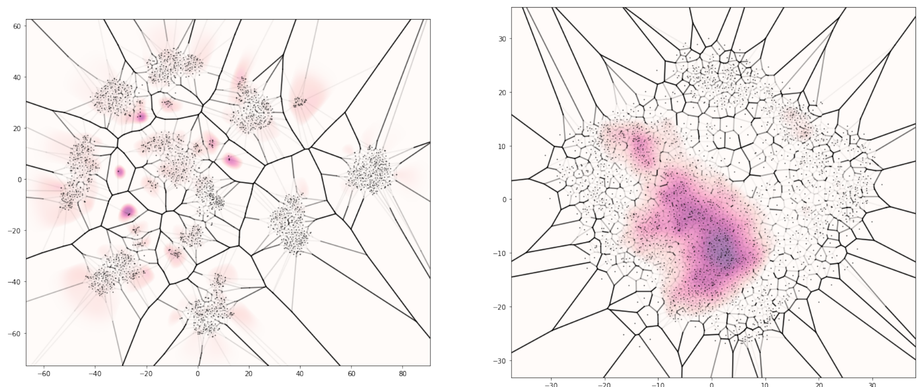
Figure 2: Visualization of the MNIST digit dataset. (left: t-SNE; right: PCA)

MNIST is shown in dramatic form. Note that t-SNE has compacted several outliers into artificial mini-clusters. For PCA, the strain map shows heavy information loss in a specific region.

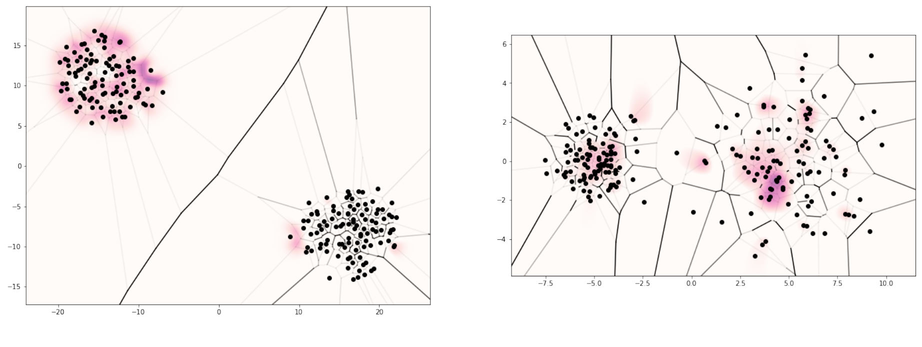
Figure 3: Synthetic data containing two 5D noise clusters with different standard deviations. (left: t-SNE; right: PCA)

The data in Figure 4 contains two clusters of different densities (as represented faithfully by PCA), but t-SNE adjusts them to be the same size. The strain map effectively represents this distortion.

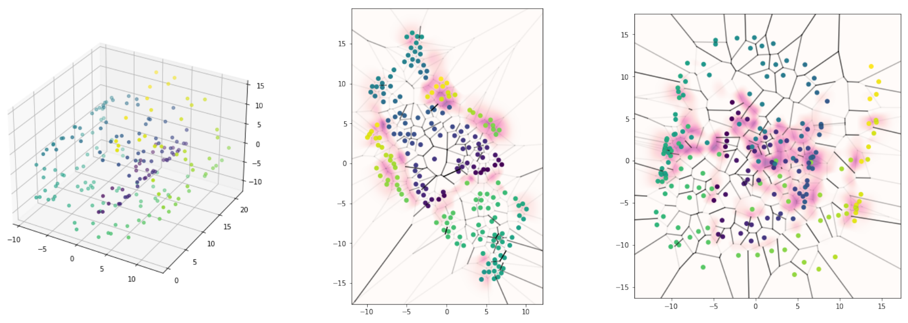

Figure 4: 3D "Swiss Roll" dataset. (left: 3D scatterplot; center: t-SNE; right: PCA)

t-SNE manages to unwrap the Swiss Roll in Figure 4, except for some errant strips along the side; there is a great deal of strain between these strips and the main body.

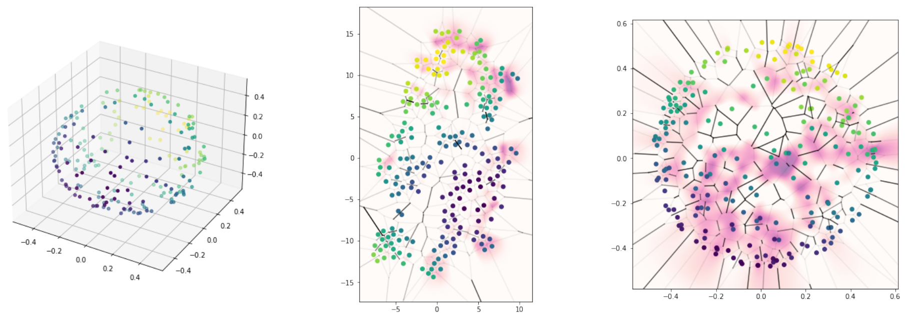

Figure 5: Random points on the surface of a sphere. (left: 3D scatterplot; center: t-SNE; right: PCA)

5 Evaluation

Recalling the tasks with which this solution was intended to help, we believe that it substantially improves on the baseline in each case. The cracks indicate a clear discontinuity in the chart, discouraging comparisons of distance that will not be meaningful. When PCA flattens distant points onto one another, or when t-SNE scales heterogeneous clusters to have the same apparent density (as in Figure 4), these distortions produce a pronounced effect in the strain map. We feel that the crack and strain layers are satisfactory from an aesthetic point of view, as well.

The precise meaning of the cracks may be confusing or non-obvious without deliberate explanation. Our intention was to use cracks to highlight areas that "should be closer together"; however, drawing a thick line between a pair of points has the effect of *increasing* their apparent separation. For some viewers, this could create the misleading impression that the points should instead appear further apart.

On the other hand, the cracks greatly enhance the apparent visual separation of clusters in some datasets. This can be seen in the t-SNE visualization of MNIST (Figure 4): because t-SNE pulls the points into tight 2D clusters compared to their original high-dimensional arrangement, the cracks are highly evident. The viewer must just take care to interpret this as indicating an aggressive clustering operation performed by the DR process, rather than indicating that the clusters were cleanly separated in their raw form.

For larger datasets, the cracks appear to scale better than the strain map in several ways. Within a crowd of many points, both indicators become muddled; however, the cracks remain prominent between clusters. Also, from a purely computational standpoint, our current strain function is expensive: its runtime scales at a roughly linear pace with the number of data points. (This could be improved by implementing more efficient spatial data structures.)

5.1 Limitations

Our techniques are subtly, but meaningfully, limited in that they only reflect the relationships of points that are neighbors in the 2D scatterplot. If two points are closely adjacent in the higher-dimensional space but are moved to opposite ends of the 2D plane (with other points interspersed), we have no way of representing this incongruity. The cracks or strain between two t-SNE clusters will also depend entirely on the points on the periphery of each cluster, which may not accurately represent the points in the interior.

We should also note that the strain map has an *intuitive* meaning, but does not represent any formally-defined *mathemtical* qualities of the DR transformation. This can be advantageous: it allows us to compute a strain field for any arbitrary DR transformation using only the sets of raw and transformed data points. This is particularly useful for an iterative pointwise techniques like t-SNE, which does not provide an analytic model of its applied transformation. However, because strain is blind to the structure of the transformation, it may ignore distortions that are not well characterized in the dataset.

In order to compensate for possible affine scaling of the data during DR, we apply a statistical correction to our crack and strain functions. However, we have not fully characterized the impact of the number of dimensions in a source dataset on the distribution of distance ratios after DR. It may be the case that differently dimensional datasets will be best handled with different crack/strain "curves". Our current implementation also includes multiple parameters that must be hand-tuned to produce a useful result; to improve the usability of this tool, it would be desirable to fit these parameters automatically based on the data.

5.2 Future work

5.2.1 Vector-valued strain

We currently compute the strain as a scalar value that aggregates contributions from all directions. However, because each contribution to the strain arises from a pairwise relationship between two points, these contributions have an associated directionality and therefore be treated as vectors. We can then consider formulas for aggregating vector-valued contributions to compute the overall strain as a vector field.

Given a vector field of strain, we could apply any of the standard vector field visualization idioms. It would be straightforward to draw a grid of arrows onto the plot, although studies have found that this is not the most effective encoding [2]. A more interesting approach, although more performance-intensive, would be to use line integral convolution [1] to trace the paths of least strain within the field. With an appropriate choice of shading, the resulting image might resemble a piece of fabric that has been stretched and squished between the data points. A pronounced ridge separating two points would indicate that they have been "squished" together, while multiple ridges along line between two points would indicate that they ave been "stretched" apart. This approach might obviate the need for cracks entirely.

5.2.2 Non-point strain sources

Our implementation places a point source of strain at the midpoint between each pair of neighboring data points. This can produce disproportionate visual results for pairs that have the same strain but different internal distances. To resolve this inconsistency, we could explore ways of distributing each pair's "quantity" of strain along a line segment between its two points.

This technique would create new mathematical difficulties. Let a and b be neighboring points with strain value s, and let σ be a function such that $s = \int_a^b \sigma(x)dx$. To compute the strain contribution from this pair at a given point v, we would then need to evaluate the the following line integral:

$$\oint_{\bar{ab}} \frac{\sigma(x)}{|x-v|^2} dx$$

$$= \int_a^b \frac{\sigma(x)}{|x-v|^2} dx$$

This has a closed-form solution if σ is constant, but is generally intractable otherwise. A reasonable approximation could be computed using the Riemann sum at some level of granularity. Doing so would naturally entail a multiplicative cost to performance over the current point-source solution, as we would essentially be generating many point sources.

6 Conclusion

Dimensionality reduction algorithms are an essential tool for visualizing high-dimensional data. Nonetheless, they unavoidably bear the potential for misleading distortions. In this work, we have proposed crack and strain layers as a universal and interaction-free design for visualizing distortions in DR scatterplots. Our software implementation constitutes a ready-to-use prototype of this technique, and we have demonstrated its application on a collection of real and synthetic datasets. While further work is needed to validate and extend the design, we believe it shows great promise in helping users confidently navigate high-dimensional data.

References

[1] Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270, 1993.

[2] David H Laidlaw, Robert M Kirby, Cullen D Jackson, J Scott Davidson, Timothy S Miller, Marco Da Silva, William H Warren, and Michael J Tarr. Comparing 2d vector field visualization methods: A user study. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):59–70, 2005.

[3] Julian Stahnke, Marian Dörk, Boris Müller, and Andreas Thom. Probing projections: Interaction techniques for interpreting arrangements and errors of dimensionality reductions. *IEEE transactions on visualization and computer graphics*, 22(1):629–638, 2015.

[4] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 1(10):e2, 2016.