DimReader: Using auto-differentiation to explain non-linear projections

Rebecca Faust and Carlos Scheidegger
Computer Science Department, University of Arizona

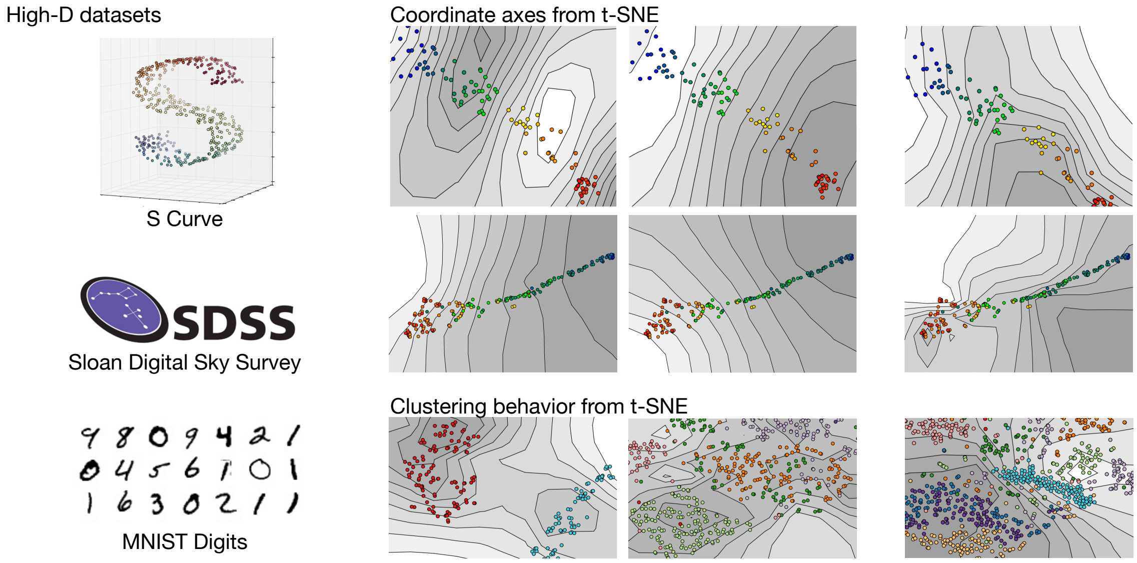

Figure 1: DimReader explains non-linear dimensionality reduction methods by illustrating the effects of user-designed perturbations of the input dataset. It provides answers to the question "if the input data had been slightly different in a particular way, how would the plot have changed?". In the case of traditional scatterplots, it recovers exactly the axis lines being displayed. In the case of non-linear methods, DimReader recovers *generalized axes*, which indicate how dimensions of interest behave (as shown in the S-curve dataset and observed star colors from the SDSS) to how clusters of data behave in the plot (as shown with the MNIST dataset).. By differentiating perturbations which generate readable axes (middle and left columns) from perturbations which generate less well-behaved axes (right column), DimReader can help explain which variables are effectively represented in the plot.

Abstract- Non-linear dimensionality reduction (NDR) methods such as LLE and t-SNE are popular with visualization researchers and experienced data analysts, but present serious problems of interpretation. In this paper, we present DimReader, a technique that recovers readable axes from such techniques. DimReader is based on analyzing infinitesimal perturbations of the dataset with respect to variables of interest. The recovered axes are in direct analogy with positional legends of traditional scatterplots, and show how to solve the computational challenges presented by the generalization to non-linear methods. We show how automatic differentiation makes the calculation of such perturbations efficient and can easily be integrated into programs written in modern programming languages. We present results of DimReader on a variety of NDR methods and datasets both synthetic and real-life, and show how it can be used to compare different NDR methods and hyperparameter choices. Finally, we discuss limitations of our proposal and situations where further research is needed.

1 Introduction

One of the central promises of data visualization is that its techniques will help users and analysts make sense of large, complicated datasets. Data visualization, and specifically techniques in dimensionality reduction, are routinely used in practice during exploratory data analysis of challenging datasets.

Classical linear methods such as Principal Components Analysis have existed for more than a century, but recent advances from *non-linear* methods that started with Tenenbaum et al's Isomap [41] have revolutionized the practice of dimensionality reduction. The potential to understand high dimensional data via low-dimensional representations is clearly attractive. But just what, exactly, are these non-linear dimensionality reduction (NDR) methods showing? This is the fundamental question that drives the work we report here.

Consider van der Maaten and Hinton's t-SNE, arguably the most powerful and currently most popular method for NDR [30]. Although practical experience attests to t-SNE's power to uncover cluster relationships in very challenging datasets, its sensitivity to the hyper-parameters is remarkable [46]. If small changes in parameter settings produce plots that are fundamentally different, we must ask ourselves: are some results generated by NDR methods just bad? Do different parameter settings show different features of the data? More importantly, how do we even answer these ques-

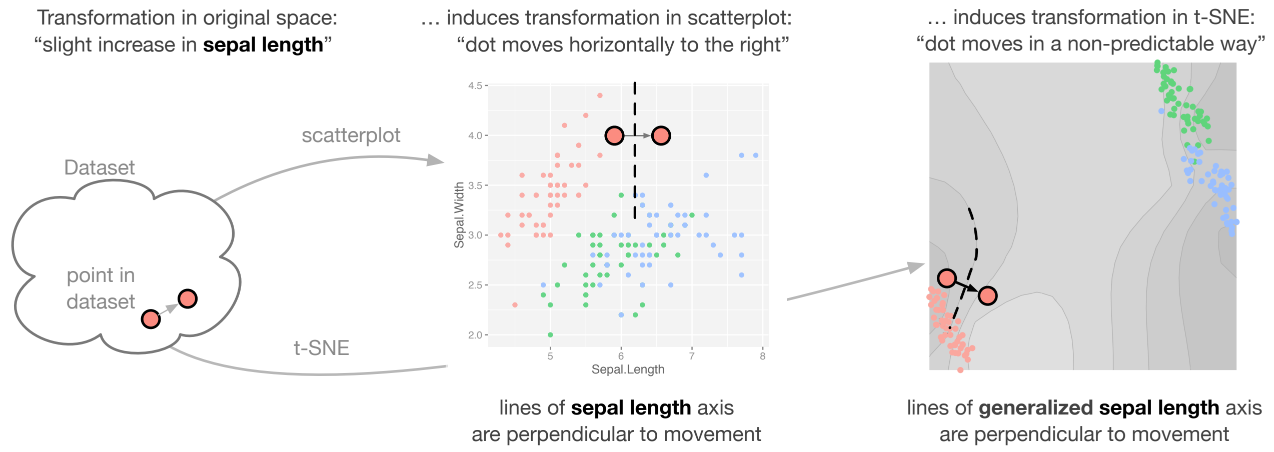
Transformation in original space:
"slight increase in **sepal length**"
… induces transformation in scatterplot:
"dot moves horizontally to the right"
… induces transformation in t-SNE:
"dot moves in a non-predictable way"
scatterplot
Dataset
point in dataset
t-SNE
4.5
4.0
3.5
3.0
2.5
2.0
Sepal.Width
5
6
7
8
Sepal.Length
lines of **sepal length** axis are perpendicular to movement
lines of **generalized sepal length** axis are perpendicular to movement

Figure 2: In traditional scatterplots, the grid lines (or axes lines) exist to explain what the plot is showing. Equivalently, they capture *infinitesimal perturbations* of the dataset in specific directions, because they are always perpendicular to the directions of movement. DimReader extends the same principle to non-linear dimensionality reduction (NDR) methods, and recovers *generalized axis lines*, which help explain NDR methods in terms of interpretable data transformations.

tion?

In this work, we employ the perspective introduced by Kindlmann and Scheidegger's algebraic design process [27]. In a nutshell, this process involves the construction of data transformations, which then induce visualization transformations. Specifically, we use infinitesimal perturbations — small changes of the data in its original space — to produce infinitesimal changes of the visualization. We then show how these visualization changes can be interpreted as producing *effective, non-linear axis legends*. In this way, our non-linear axes *explain* the NDR plot in the same way that axis legends explain the positional encoding in scatterplots. As a result, analysts can understand and evaluate dimensionality reduction plots similarly to how they evaluate *linear* methods. In fact, we show in Section 3 that our methods *exactly* recovers the axes of typical scatterplots. DimReader is quite general, and can be applied to many different NDR techniques, only requiring access to the source code of its implementation. Specifically, we use a method known as *automatic differentiation* to produce the necessary gradients [22]. An overview of the process is given in Figure 2.

In summary, our contributions are:

- A general framework to explain plots generated by non-linear dimensionality reduction, using infinitesimal perturbations
- A practical implementation of the framework using automatic differentiation
- An experimental study of the effectiveness and efficiency of DimReader using three well-known NDR methods: Isomap [41], LLE [34], and t-SNE [30].

2 Related Work

Projection methods have received a considerable amount of attention in information visualization. In this section, we review the work that is most directly related to our research, but cannot hope to cover the entirety of the field. For a comprehensive view on multidimensional scaling and dimensionality reduction, we recommend Born and Grönen's textbook [6], and Fodor's survey [18].

Projection methods in information visualization The observation that pairwise similarities (or distances) can be converted into low-dimensional representations by a mathematical formulation is due to Torgerson and his now-classical theory of multidimensional scaling [42]. In information visualization, force-directed methods have long been used as a dimensionality reduction technique, from fully-automatic methods [12, 24, 31], to methods which take some amount of interaction, either through placement of exemplar points [16, 26, 32] or through direct interaction with projection parameters [25]. Although interactive methods offer a better hope for understandability because the perturbation analysis we discuss can happen "in the analyst's head" during interaction, we argue that the visual encoding these techniques provide can still be unclear. The technique we propose in this paper can be applied to essentially all of the methods above, and offers an attractive complement to both automated and interactive projection methods.

Perturbation Analysis for data science The idea of understanding a system by examining its behavior under perturbations is well-established in the engineering and statistics literature. In the 1970's, Cook introduced the notion we now know as Cook's distance [13], which measures the influence of a point on the parameters of linear regression models. In the context of visualization, Bergner et al. point to sensitivity analysis as one of the requirements in understanding computer simulations [5]. In this paper, we use perturbation analysis as a central tool to recover readable axes from NDR methods, in a sense incorporating sensitivity analyses into familiar visual metaphors.

Automatic Differentiation Perturbation analysis is clearly an important tool for understanding systems, but the issue of how to implement it in existing computer systems is crucial. Automatic differentiation (which we explain in detail in Section 3) provides a way to compute derivatives

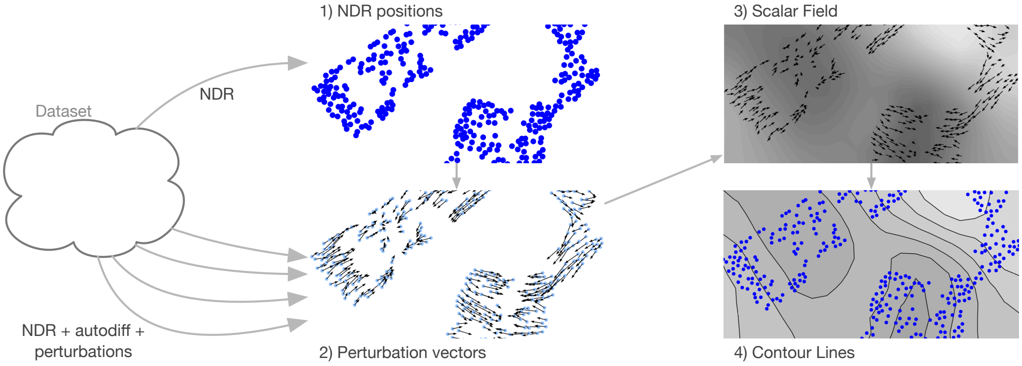

Figure 3: An overview of DimReader. For a given NDR method, we 1) compute its position using the original implementation; 2) compute perturbation directions for the input points with the transformed version of the implementation which uses dual numbers (We discuss how to choose appropriate perturbations in Section 3.3); 3) compute the scalar field whose gradient best matches the perturbation vectors in a least-squares sense; and finally 4) compute its isocontours. Section 3 explains these steps in detail.

of arbitrary functions in a computer program, provided access to the source code (or similar structural information about the computation) is available [22]. To the best of our knowledge, the most mature software library employing automatic differentiation is Ceres, written in C++ and employing template metaprogramming [2]. DimReader is implemented in Python for simplicity and terseness, but could easily be redesigned in C++.

Guidance and validation of projection results One of the issues with NDR is that it's hard to know what a plot is actually showing. This has resulted in a variety of papers which offer guidance on how to interpret projections, based on a combination of real-world experience, synthetic examples, and theoretical arguments [10, 36, 37]. This work is essential to the current practice, we argue, because current NDR methods do not offer explanations of their own results — there are much fewer research papers offering guidance for understanding and interpreting traditional scatterplots. As we show in Section 5, our technique provides a way for a projection method to explain itself. Although analyst guidance and validation will always be a part of a well-designed analysis infrastructure, our technique could mitigate some of the problems that have been observed in deployed systems, where projection methods are ultimately discarded because of readability issues [8, 23].

Augmented visual representations There is another avenue of attack on the readability problem of NDR methods. Often, researchers will *augment* the results of the projections with visual diagnostics that poinpoint potential problems. Here we highlight two papers: Seifert et al. augment the projection by showing how the projection's stress (roughly the discrepancy between source-space distances and target-space distances) varies spatially in the NDR plot [38]. More recently, Stahnke and co-authors described methods to *probe* a projection, through carefully designed user interactions and custom visual encodings [40]. Our method for extracting effective axes can be seen as a way to allow *any* NDR method to augment itself with metaphors that have a well-defined analogy in the linear case, as can be seen in Section 5, and Figure 10 specifically. In Section 6, we provide a more direct comparison to some of the methods used in Stahnke et al.'s work.

Explainable visualizations Tufte's classic book explores the idea that visual metaphors can *explain* [43]. Still, every plot assumes an audience capable of reading it, and the issues around visualization literacy remain an active area of research [7]. Often, novel visualization metaphors are necessary because of the complexity of the data or the tasks [9]. Nevertheless, we argue that generalizing well-established techniques such as axis legends to NDR provides a way to *explain* those techniques. Gleicher's Explainers take user interaction to design specific projections that help explain input data [19]. In contrast, our technique extracts axes inherent in the non-linear projections being employed.

3 Technique

In principle, all that DimReader requires is the ability to compute derivatives of the projection coordinates with respect to each of the input points. For extremely simple techniques (such as scatterplots and other fixed linear projections), these derivatives can easily be evaluated in closed form. However, more sophisticated methods such as Isomap, LLE, and t-SNE involve long computation chains, for which the evaluation of the derivative would introduce significant development overhead. Instead of trying to solve them in closed form, we take central advantage of automatic differentiation, or *autodifferentiation* [22].

3.1 Autodifferentiation

In this paper, we use a particular form of automatic differentiation known as forward-mode autodifferentation. In what follows, we will refer to it as "automatic differentiation" or "autodiff". Before we explain what autodiff is, it is illustrative to explain what it is *not*.

Autodiff is not finite differencing By replacing the limit operation in the definition of the derivative with a *finite* difference, one can arrive at an expression similar to the derivative which requires only two function evaluations. Unfortunately, determining the right step-size for finite differencing is far from trivial. Picking a value that is too large will mean that higher-order terms of the Taylor can dominate, and the expression diverges from the derivative. Too small a value, on the other hand, can cause catastrophic cancellation [20].

Autodiff is not symbolic differencing Many derivatives can be computed by a combination of syntactic rules for function composition and base cases. As a result, it is possible to derive *symbolic formulas* for the derivatives of many functions. Unfortunately, such symbolic expressions for function derivatives either require sophisticated optimization passes (as evidenced by Theano, a state-of-the-art symbolic differentiation library for machine learning [3]) or can take significantly longer to evaluate. Both of these limit the computation of derivatives over arbitrarily long programs.

In contrast, the use of automatic differentiation offers a good balance between computational efficiency, programmatic convenience, and generality. In forward-mode automatic differentiation, the program's derivative with respect to any one variable is computed alongside the function value, by using an *extended number system*. In this system, we represent values by *two* numbers. In other words, a number x has the form $x = a + b\varepsilon$. The value a holds the real part of the value, and the value b represents the derivative of x with respect to our variable of interest. Because it is represented by two values, x is called a dual number. When we initialize a variable v represented with dual numbers, we set b to zero in the majority of the cases, but we set b to one if that is the variable we want to differentiate with respect to (since $dv/dv = 1$). The operations on dual numbers behave like polynomials in ε, expect with an added algebraic rule. We define $\varepsilon^2 = 0$, which simplifies all polynomial terms with degree greater than one.

We give a simple example of using automatic differentiation to compute derivatives in Figure 4. Note that, like finite differencing, automatic differentiation is always performed at a specific value, and with respect to a specific variable. It produces two numbers as a result: the function value and the partial derivative with respect to the chosen variable. This has two important consequences for our design. First, we need to decide over exactly which variables we will take derivatives. Second, we need to execute the program many times in order to evaluate many different derivatives. This will become important in Section 5.5.

The prototype we use to generate the results in this paper is implemented in Python, and in our setting, automatic differentiation is achieved through custom classes and operator overloading. A simplified snippet of the class responsible for automatic differentation is shown in Figure 5.

$$\begin{array}{r|ccc}
x_1 & a_1 + b_1\varepsilon & = & 1.5 + 1\varepsilon \\
x_2 & a_2 + b_2\varepsilon & = & 2 + 0\varepsilon \\
x_3 = x_1 + x_2 & (a_1 + b_1\varepsilon) + (a_2 + b_2\varepsilon) & = & 3.5 + 1\varepsilon \\
y = x_1 * x_3 & (a_1 + b_1\varepsilon) \times (a_3 + b_3\varepsilon) & & \\
 & = a_1a_3 + a_1b_3\varepsilon + a_3b_1\varepsilon + b_1b_3\varepsilon^2 & & \\
 & = a_1(a_1 + a_2) + (2a_1 + a_2)\varepsilon & = & 5.25 + 5\varepsilon
\end{array}$$

Figure 4: Automatic differentiation of the equation $y = x_1(x_1 + x_2)$ with respect to x_1, evaluated at $x_1 = 1.5, x_2 = 2$. We know that $\frac{dy}{dx_1} = 2x_1 + x_2$, and so $\frac{dy}{dx_1} = 5$, which is the value associated with ε in the last expression.

```
class DualNum:
    def __init__(self,val,dot):
        self.val = val
        self.dot = dot
    def __add__(self, o):
        return DualNum(self.val + o.val, self.dot + o.dot)
    def __mul__(self, o):
        return DualNum(self.val * o.val,
                       self.val * o.dot + self.dot * o.val)
    def sin(self):
        return DualNum(sin(self.val),
                       cos(self.val)*self.dot)
    def log(self):
        return DualNum(log(self.val),
                       self.dot/self.val)
    # ...
```

Figure 5: A bare-bones implementation of a class for automatic differentiation. By representing data in terms of dual numbers instead of built-in `float` values, the implementations of dimensionality reduction compute derivatives alongside the function with a manageable performance overhead. Note the direct relationship between the expressions for values in the `dot` field and the chain rule for derivatives.

3.2 Overview of the process

To apply DimReader to an NDR method, there are four steps. Each of these steps is discussed in a subsection below.

- A user chooses a perturbation of interest, which defines an infinitesimal change for each data point (possibly in different directions).
- The NDR method is executed many times using dual numbers, from which we obtain the perturbation vectors, one for each input point.
- From the perturbation vectors, a scalar field whose gradient matches the perturbation vectors is computed.
- The isolines of this scalar field, which perpendicular to the gradient, are extracted using Marching Squares. They form the effective axes.

3.3 Choosing which perturbation to use

The first step of our method involves a choice of the perturbation of the dataset. This choice corresponds, effectively, to an analyst answering the following question: "if each data point were slightly different in this specific way, what would happen to the visualization?" In order to recover different features of the NDR method and its effect on the dataset of interest, different perturbations can be designed. In Section 6, we offer preliminary guidance for which perturbations are interesting, but a full investigation of the matter remains necessary, and is the subject of future work.

Datasets with interpretable dimensions Some datasets have interpretable columns. Take the iris dataset, for example, which is used in Figure 2. In that case, a perturbation that changes each of the input points in the direction of a given dimension will reconstruct, for an NDR method, curved axes lines that correspond, roughly to the linear grid lines in scatterplots. Concretely speaking, we evaluate each input point p_i as $p_i + \varepsilon(0, \cdots, 0, 1, 0, \cdots, 0)$, where the value 1 is positioned at the dimension of interest.

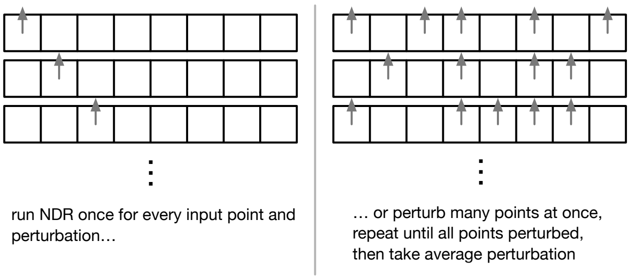
⋮
run NDR once for every input point and perturbation...
⋮
... or perturb many points at once, repeat until all points perturbed, then take average perturbation

Figure 6: The naive way to extract perturbation vectors — illustrated in the left subfigure — is simple to understand, but takes as many NDR runs as there are input points. This can sometimes be impractical. Instead, we can choose many perturbations to be performed at once. This alternative approach (right subfigure) takes as many NDR runs as the base-2 logarithm of the number of input points. The input dataset provided to the autodiff version of NDR is represented here by the square vectors, and the input points being perturbed are represented by the presence of the arrows.

```
# Basic method, O(numPoints) runs
for i in range(0, numPoints):
    points = copy(inputPoints)
    points[i] = perturb(points[i], perturbation)
    projection = project(points) # project uses autodiff
    dx, dy = projection.derivative[i]
    projectionVectors[i] = vector(dx, dy)
return projectionVectors

# Improved method, O(log(numPoints)) runs
counts = zero_array(numPoints)
projectionVectors = zero_matrix(numPoints, 2)
while any(counts < 1):
    points = copy(inputPoints)
    for i in range(numPoints):
        if random() < 0.5: # perturb each point with probability 0.5
            perturbed[i] = true
            points[i] = perturb(inputPoints[i], perturbation)
    projection = project(points) # project uses autodiff
    for i in range(numPoints):
        if perturbed[i]: # only store vectors of perturbed points
            dx, dy = projection.derivative[i]
            projectionVectors[i] += vector(dx, dy)
            counts[i] += 1

for i in range(numPoints): # average all perturbations performed
    projectionVectors[i] /= counts[i]
return projectionVectors
```

Figure 7: Although a basic implementation of DimReader is easy to understand (top), it only extracts one perturbation vector at a time. A more efficient implementation (bottom) extracts half of the perturbation vectors from the input at once. To remove possible correlations between the outputs, we choose which points to include at random, and iterate until all points have been included. The expected time in this case is logarithmic on the size of the input point dataset.

Datasets with categorical columns On the other hand, some datasets lack directly-interpretable dimensions. Take the mnist database of handwritten digits [29], where each input point in a vector in 784-dimensional Euclidean space, corresponding to a 28×28 image. Perturbing any one pixel in each image will not give much meaningful information. For datasets like this, some other perturbation needs to be chosen. We offer two kinds of perturbations to investigate the behavior of the NDR method with respect to categorical columns.

The first perturbation moves each point in the direction of the centroid of the class it belongs to. This perturbation is designed to investigate the extent to which the NDR method captures a meaningful clustering behavior corresponding to the given categorical column. The isolines, in this case, will tend to circle the "projected center" of the clusters. In situations where there is significant cluttering of the NDR plot, this can provide added readability, as we show in Figure . The second perturbation moves all points in the direction that takes a centroid of some class to a centroid of some *other* class. This perturbation is designed to elucidate the effective boundary between two classes in the projection. We show examples of this in Figure 14.

The design of these perturbations is central to the understanding of the NDR methods. We think of them as concrete realizations of some of Brehmer and Munzner's task typology in the context of NDR methods. Each perturbation, then, is appropriate for a specific task, and should be designed with that in mind.

3.4 Extracting derivatives from NDR methods

In this section, we describe two techniques used in DimReader to extract the perturbation vectors for a given projection. The first technique is simple, straightforward, and provides a good intuition for the overall strategy. Unfortunately, this technique requires as many executions of the NDR method as there are input points in the dataset, which often means the overall performance can suffer. The second technique, on the other hand, only requires as many runs as the *logarithm* of the number of input points. We illustrate the difference between the two approaches in Figure 6, and give pseudo-code for the two approaches in Figure 7.

DimReader needs access to the source code for the NDR method at this step so the method can be executed with dual numbers. In principle, the source code can be executed without any modifications. In practice, some issues arise because of efficiency concerns and library limitations. We discuss these issues at length in Section 4. We note, in addition, that our choice of automatic differentiation in DimReader is not necessary. There are other methods to evaluate function derivatives, including manual derivation of the expressions. When using DimReader in other settings, these alternative techniques might be more appropriate.

3.4.1 Perturbing one point at a time

After a perturbation is chosen, the NDR technique is executed with automatic differentiation, for every point in the dataset. On run i, the point p_i is perturbed (that is, we replace p_i with $p_i + b\varepsilon$). The NDR technique will return the projection coordinates, v, for all points, along with the derivative of each reduced point with respect to the perturbation of p_i ($\frac{dv}{dp_i}$). We use the derivative of each coordinate in the reduced point v_i as the vector that describes the change in the coordinate, and discard the rest of the information of the run. In some cases, perturbing p_i has an effect on points other than v_i, but since we are only interested in how v_i changes when p_i is perturbed, we can safely ignore them. The pseudocode for this is given on the top half of Figure 7.

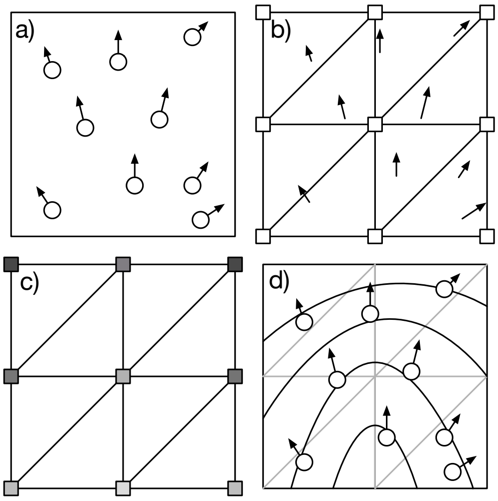

Figure 8: An illustration of the process to recover generalized axes. Given the point positions and perturbation vectors (a), we construct a triangular mesh and interpret each vector as a linear constraint on the gradient of a function (b), which gives values on each of the vertices (c). From these values, we can extract lines perpendicular to the perturbation vectors using marching squares.

3.4.2 Perturbing many points at a time

The method described above is clearly inefficient. In principle, we could evaluate the projection derivatives with respect to all of the points at once, and only run the autodiff version of the code once. Unfortunately, this does not work for many perturbations. This happens because many dimensionality reduction methods are *invariant to dataset translations*. Even though the perturbation of only one input point at a time offers interesting insight into the NDR method, if we move all of the points at once in the same direction, NDR methods such as Isomap, LLE, and t-SNE will produce exactly the same projection.

Surprisingly, adding a small amount of randomization mostly solves this problem. Instead of perturbing one point at a time, we can choose half of the points at random to perturb, while the other half does not change. We then store the perturbation vectors for the points we chose to perturb, and repeat the process until we have actually perturbed all of the input points. After each round, we expect to halve the number of unperturbed points, which gives a number of repeated runs which is logarithmic on the number of input points. The pseudocode for this is given on the bottom half of Figure 7.

3.5 Reconstructing the direction field

Once we have the projected points and their derivatives (that is, the perturbation vectors), we need to reconstruct the direction field, in order to extract perpendicular lines. We achieve this by computing a scalar field whose gradient best matches the vectors. We use a simple least-squares reconstruction technique, adapted from Ferreira et al.'s vector-field clustering work [17], which we illustrate in Figure 8. We first decompose the output plane in a rectangular grid, and split the grid into two triangles, giving a triangular mesh of the output space. The resolution of this grid needs to be decided ahead of time, and we use a 10x10 grid in our examples for this paper. We model a scalar field on the output plane as a piecewise-linear function on the grid values, and then interpret each point and its perturbation vector as a linear constraint on the vertices of its corresponding triangle. To find the best-fitting scalar field, we solve it in a least-squares sense, regularizing the system to ensure a unique solution [17].

3.6 Extracting perpendicular lines

The final step is quite simple. With the scalar field expressed as values in a triangular mesh, we can use marching squares to extract isocontours [4]. By construction, the gradient of this scalar field matches the perturbations. Since isolines are perpendicular to a function's gradient [35], the resulting curves will tend to be perpendicular to the perturbations. As we show in Figure 2, these isoline can be thought of as *generalized axes lines*.

4 Implementation

Our current prototype for DimReader is implemented in Python and numpy [45]. Our t-SNE implementation is closely based on van der Maaten's Python code [44], while the LLE and Isomap implementations are from-scratch. The entire method takes about 3,500 lines of Python, including implementations of Marching Squares, the classes for dual numbers, and the linear solvers described below.

4.1 Isomap

Isomap was one of the first NDR techniques to recover curved manifolds well in practice [41]. Isomap builds a weighted graph which approximates the manifold, where edges have weight equal to the distance between points, but edges connect only small neighborhoods of each point. The global distance is defined to be the shortest-path metric on the graph. The low-dimensional projection is constructed from the shortest-path metric using classical MDS [6].

We implemented Isomap not only because of its historical significance and relatively high-quality results, but also because it highlights an interesting property of automatic differentiation: it works over code bases that we tend to not think of as differentiable. Specifically, the operations in Dijkstra's algorithm for shortest paths are all well defined for dual numbers, and so we naturally can extract the sensitivity of shortest-path distances with respect to changes in the input points [14].

As a matter of efficiency, we performed a few optimizations. First, we precompute the distance matrix (in the high-dimensional space) since much of it remains unchanged in each run. For each run of Isomap, we recalculate the distance from each perturbed point to all other points to update the derivative value, and reuse the constant portion of the distance matrix. After finding the nearest neighbors, we use Dijkstra's algorithm to find the shortest paths between all pairs of points. In Dijkstra's algorithm, if two paths have the same length we choose the one with the smaller derivative because it would be the shortest path of the perturbed point.

Interaction with numerical linear algebra routines The final step of Isomap is Classical MDS, and this presents

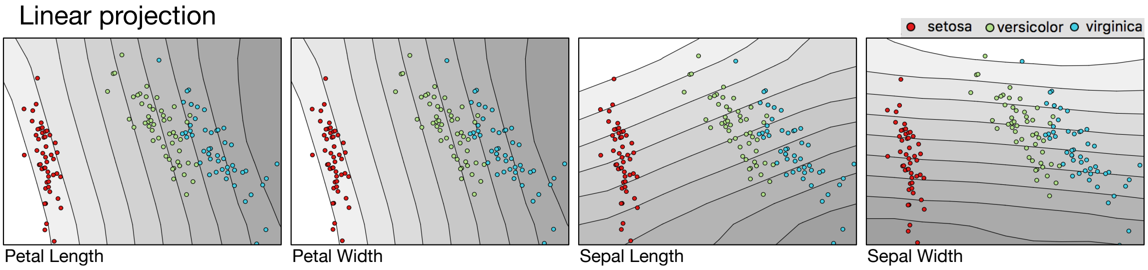

Figure 9: When linear projections are examined with DimReader, we recover axis lines that are equivalent to oblique axes (up to minor reconstruction errors from boundary conditions of the least-squares solver). The projection we use in this plot corresponds to the first two principal components of the iris dataset.

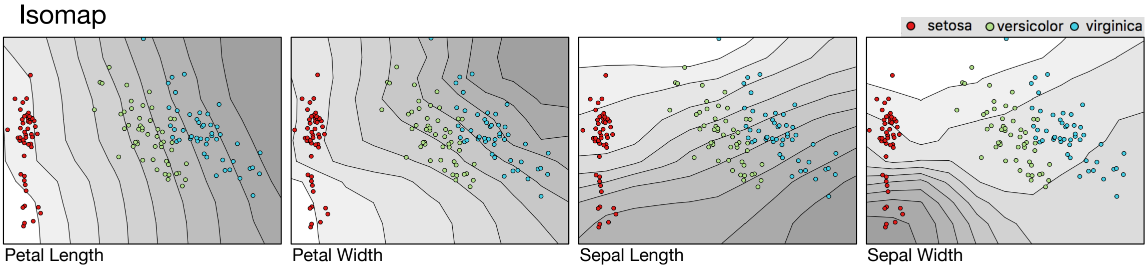

Figure 10: Extracting axes of Isomap. Compare this with Figure 9. Notice that even though the projected points look very similar, the recovered axes can be quite different, as is the case with the sepal width of the flowers (rightmost column). Thus, DimReader can *augment* existing dimensionality-reduction plots in a non-intrusive way.

unique challenges for our autodiff implementation based on operator overloading. Specifically, Classical MDS requires the computation of eigenvectors, and since Python libraries for numerical linear algebra are implemented through high-performance libraries like Lapack, the operator-overloading functionality is not present. To solve this issue, we implement the eigenvalue computation through power iterations [21], since matrix-vector multiplication of dual numbers has efficient dual-number implementations in terms of matrices of values and ε terms.

We show recovered axes for Isomap in Figure 10.

4.2 Locally Linear Embedding

The next algorithm we highlight is Roweis and Saul's Locally Linear Embedding [34] (LLE). Like Isomap, LLE uses a nearest-neighbor graph to recover a global view of the dataset. Unlike Isomap, LLE does not need shortest-path graph distances, which can be costly to compute. Instead, LLE computes edge weights for the nearest neighbor graph, such that each vertex can be best reconstructed by a linear combination of its neighbors using those weights. On a second step, the projection coordinates are recovered by finding positions on the plane that respect the weights.

Interaction with numerical linear algebra routines Similarly to Isomap, our autodiff implementation of LLE involves a small degree of adaptation. In the case of Isomap, we required the computation of the largest eigenvalues of a matrix. In the case of LLE, we need to compute the *smallest non-zero* eigenvalues. Our implementation uses *inverse power iteration* [21]. Inverse power iteration, in turn, requires a linear system solver, which presents similar issues for dual number implementations. Our solution is to implement a black-box linear system solver using conjugate gradients [39].

In order to increase the performance of our method with LLE, we perform the following optimizations. Before running LLE with dual numbers, we execute an initial run with floating-point numbers to cache two intermediate values: the nearest neighbor graph and the last guess for each eigenvector in inverse power iteration. Perturbing a point infinitesimally does not change the nearest neighbor graph, and so recomputing them repeatedly is unnecessary. Saving the last guess before inverse iteration converges for each eigenvector allows us to use them as the initial guesses for future runs of inverse iteration which causes it to converge in a single iteration.

We show recovered axes for LLE in Figure 11.

4.3 t-SNE

t-SNE is generally considered to be among the most powerful techniques for dimensionality reduction, and also one of the hardest to interpret appropriately [30, 46]. As such, it is a natural target for DimReader. In addition, t-SNE is significantly different from Isomap and LLE in both formulation and implementation. This provides us with an opportunity to explore practical issues of using DimReader to explain its results.

We highlight two separate issues to discuss: the presence of multiple local minima, and its formulation in terms of the gradient of an energy function. While the first issue presents challenges for implementations that depend on repeated executions, the second issue allows us to achieve a significant speedup.

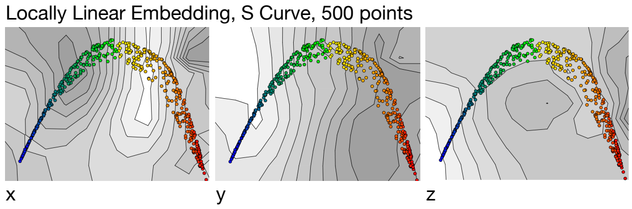

Figure 11: Extracting axes of LLE, using the S Curve as an example. In this plot, we can see the typical arc-like shape that LLE tends to generate, as well as the pinching behavior in the boundaries. In this situation, DimReader can help analysts understand whether this shape arises from the data itself, or from distortions in the reconstruction. Note how the generalized axes do not tend to cross the dataset perpendicularly; this appears to indicate that LLE is failing to recognize the structure of the dataset. Compare this with the t-SNE axes in Figures and 12.

Multiple minima The energy function that t-SNE minimizes has more than one local minimum. This means that any source of randomness in the implementation will cause multiple runs to possibly diverge, presenting a challenge for our approach. Most implementations of t-SNE require an initial guess for the projection, and we take central advantage of this. Specifically, the first execution of t-SNE is given a random initial guess, and in this run, we use regular floating-point numbers. In this run, we capture variables that are unchanged in multiple runs, similarly to what we do with Isomap.

Gradient descent t-SNE is implemented as an explicit gradient descent formulation through an additive update of the parameters. Specifically, the main loop of t-SNE is roughly as follows:

```
pos = initial_guess
g = gradient(energy(pos), pos)
while mag(g) > epsilon:
    pos = pos - rate * g
    g = gradient(energy(pos), pos)
```

As a result, when the loop exits, we know that the gradient of the energy with respect to the position will be close to zero. This means that to recover any one perturbation of the t-SNE formulation with respect to an input point, all that is required is to run one single iteration of t-SNE with dual numbers. By providing the dual-number implementation the result of the execution of the floating-point implementation (as explained in the previous paragraph), the loop will execute at most once before exiting – in fact, in order for the sensitivity of the positions with respect to the input to be recorded in the `pos` variable, we must force the loop to execute at least once. Still, since t-SNE typically executes between 100 and 1000 iterations in this loop, this simple optimization achieves a significant speedup.

We show the results of the t-SNE axes recovered with DimReader in Figures , 2, and 12.

5 Experiments

In this section, we discuss a suite of experiments designed to explore the capabilities, performance, and limitations of DimReader. We start with a sequence of examples using synthetic datasets and simple projection algorithms, in order to better understand the behavior of the technique [28].

5.1 Linear projections

We start with showing results of linear projections as a basic sanity check on the behavior of DimReader. Figure 9 shows a typical example of the axes reconstructed by DimReader when using linear projections. Since linear projections can be exactly represented by a matrix multiplication, the derivatives of input points position with respect to one direction will always be constant vectors. As a result, the reconstructed scalar field is almost (except for the influence of the regularization terms) a linear ramp, and so the contour lines are evenly spaced and parallel, which indicate that changes in the input variable will behave identically across the entire field. Despite their limited power, this property is one of the main advantages of linear projections.

5.2 Isomap

The first non-linear projection algorithm we examine is Isomap [41]. Because Isomap uses classical MDS (which is essentially a linear projection), we should expect that, to some degree, Isomap would behave much like linear projections. This is indeed the case with simpler datasets, such as the Iris dataset, shown in Figure 9. However, there are some interesting differences. Consider the generalized axis for the “sepal width” variable which DimReader recovers. Even though the point positions generated by Isomap are quite similar to that of PCA, the sensitivity of the projection differs dramatically from the cluster of Setosa samples to that of Virginica and Versicolor samples. Even more interestingly, it seems that the sensitivity is caused by only some of the Setosa samples. This differentiation is not present in the linear projection, and would not be clear from the Isomap plot alone.

5.3 LLE

Locally Linear Embedding is often used because of its favorable performance characteristics, but is known to produce distorted projections [15]. In this section, we illustrate how DimReader might help pinpoint such problems. Consider the embedding of the S curve shown in Figure 11. Note that none of the recovered axes show a strong gradient across the central portion of the arc (that is, an axis that would explain the “thickness” of the projection along the center portion with green and yellow points). This means that regardless of the details of the projection, no change in data will produce a vertical variation in projected coordinates, and so we should not try to interpret that variation. Instead, it shows that such variation must be an artifact of the algorithm rather than the data. Contrast this with more successful projections, such as the result with t-SNE shown in Figure .

5.4 t-SNE

Although t-SNE is possibly the state of the art in NDR methods, one of the main objections to its use in practice is the opaque nature of its optimization criteria. DimReader can be used to show that, with appropriate hyperparameter choices,

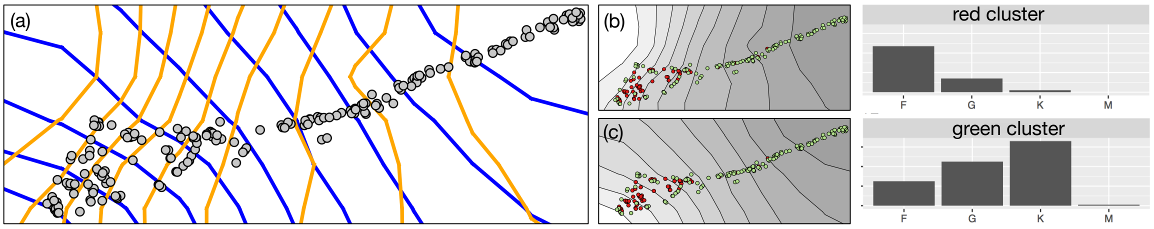

Figure 12: Two different t-SNE axes recovered by DimReader from a sample of the Sloan Digital Sky Survey, Data Release 7 [1]. While each axis provides interesting information on its own, additional insight into the data and the relationship with the NDR method can be optained by combining the information of more than one axis. On the left (a), we show the reconstructed axes of two variables from the SDSS dataset: $g-r$ and $u-g$ respectively (these variables measure the observed "color" of a star). Along the middle (b and c), we show both axes separately. The histograms show the distribution of stellar classes across the two clusters (see the main text for a full explanation).

t-SNE projections do in fact recover high-dimensional information effectively. In contrast with LLE, successful t-SNE plots tend to have explanatory axes for most projection features, as can be evidenced in Figures 12.

5.5 Performance

In this section, we report performance figures for the prototype implementation of DimReader. Although we were reasonably careful with algorithmic and high-level design decisions that impact performance, we did not make a significant effort to increase the performance. We expect that carefully-implemented versions of our proposal in high-performance languages such as C++ or Java would be significantly faster, possibly by an order of magnitude (this is typically the performance difference between Python and aggressively optimized, compiled languages).

A table showcasing typical results is included in Figure 13. The performance of DimReaderfor a given NDR method is dependent on two main factors: the number of input points and the overhead incurred by dual numbers. We need to execute a number of repeated runs proportional to the base-2 logarithm of the number of input points, and that is essentially unavoidable. We note that for the case of LLE and t-SNE, the optimizations we described in the previous section make the execution of the dual-number version of the projection much faster than that of the regular numbers. As a result, DimReader can extract axes with a relatively small performance overhead.

For cases such as Isomap, on the other hand, where we performed no such optimizations, the performance of our method suffers a bit. We argue that this is an acceptable tradeoff: DimReader still works in an acceptable amount of time in the general case, but more careful implementations can be significantly more efficient.

6 Discussion

DimReader can recover more than coordinate axes If the dataset has interpretable columns, then it is natural to try and recover axes corresponding to those columns. But often, high-dimensional data does not have interpretable columns. For example, individual pixels of the MNIST dataset are hardly interpretable. In these cases, other perturbations are possible. A thorough study of this use case deserves further research, but in Figure 14 we show one possible application, where we move each input point towards its centroid — in high dimensions, and then use DimReader's axes to investigate where these cluster centers tend to be in the projected coordinates.

	MNIST	100	200	500	1000
Regular	tSNE	2.9	6.4	22.1	78.4
Dual, per input point		0.4	1.4	9.0	41.4
Dual, per axis, 4 cores		5.8	18.8	111.8	532.1
Regular	Isomap	0.6	2.3	16.6	87.8
Dual, per input point		1.1	5.3	51.6	335.3
Dual, per axis, 4 cores		8.6	47.8	530.4	3764.9
Regular	LLE	4.6	18.9	115.8	411.3
Dual, per input point		0.7	1.5	3.2	5.7
Dual, per axis, 4 cores		10.0	32.1	147.9	473.6
	SDSS	100	200	500	1000
Regular	tSNE	2.9	6.3	22.6	88.4
Dual, per input point		0.4	1.5	9.0	43.3
Dual, per axis, 4 cores		5.9	19.1	112.0	563.7
Regular	Isomap	0.2	0.8	7.7	51.1
Dual, per input point		0.9	4.8	51.0	343.2
Dual, per axis, 4 cores		7.3	42.0	515.8	3814.4
Regular	LLE	0.1	0.2	0.9	4.3
Dual, per input point		0.0	0.2	0.3	2.5
Dual, per axis, 4 cores		0.5	1.7	4.2	31.4

Figure 13: Performance figures for the SDSS dataset and the MNIST dataset, for progressively larger samples and three different NDR methods. All figures are reported in seconds.

Interpreting isolines In our plots, the relative density of isocontours can be interpreted similarly to the behavior in scalar fields. Narrowly-spaced isocontours indicate a high sensitivity to changes in the *independent* variable, (in our case, projection coordinates). Widely-spaced isocontours indicate *low spatial sensitivity*: a change in the projection coordinates is not expected to change the outcome variable by much. However, we caution against interpreting the *values* of the scalar field generated, since these are defined only up to translation. This, incidentally, is why we don't provide color legends to the density plot: the only sensible readings are relative assessments, and these do not require a color legend.

Information in perturbation vectors Consider the example from Figure 12. Notice that the axes appear almost parallel to each other on the right side of the plot, and become perpendicular to one another on the left. This coincides with a qualitative change in the behavior of the projection, which

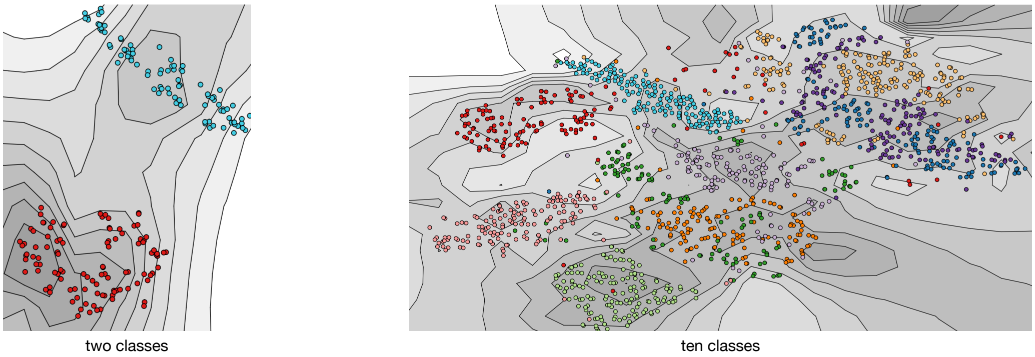

Figure 14: In addition to recovering axes, DimReader supports arbitrary perturbations. For example, it is possible to extract the *implied clustering* of an NDR method by perturbing each point to the high-dimensional centroid of its class assignment, and then look for local maxima of the contour ("islands of dark gray"). On the left, we show this use case on two classes of the MNIST dataset; on the right, we show it on all ten classes. Note how the implied some of the implied centers are away from the projected centroid, and how some classes cluster more tightly than others.

switches from essentially one-dimensional on the right to essentially two-dimensional on the left. This observation is supported by performing a simple k-means clustering of the projected points according to their perturbation vectors. The two histograms on the right side of the figure show the distribution of stellar classes across these clusters. This information was not provided to t-SNE, and yet the distributions are quite different from one another, suggesting that 1) t-SNE successfully identified two different regimes in the dataset, and 2) our clustering procedure (and hence, DimReader's perturbation vectors) captured the relationship.

DimReader axes are not heatmaps It is important to contrast the contours generated by DimReader with heatmaps (and specifically heatmap contours), an existing popular choice to augment dimensionality reduction plots. Take, for example, Stahnke et al.'s Probing Projections system [40], which provides, along other very useful features, such heatmap functionality. While heatmaps are clearly useful for some projections, they depict fundamentally different phenomena, as we show in Figure 15. A heatmap helps with *locating dense regions*, but crucially does *not* provide "hypothetical readings" such as: "if this input point were to change, what would happen to the plot?" This distinction can become important in scenarios where users choose projections explicitly [16,25]. A full investigation of the complementary strengths and weakness of these two approaches remains necessary, but is beyond the scope of this work.

Inverse readings DimReader enables interpretation of forward transformations: given a perturbation of an input and a visualization, DimReader provides an answer. But we envision a setting in which the natural reading is the *inverse*: given a projected point and a direction of movement in the projection, what changes in the data could generate such movement? In principle, the derivative information obtained by autodiff also captures this inverse relationship [11], but the fact that we are dealing with *projections* appears to make the problem fundamentally harder. A full investigation is beyond the scope of this work, but remains necessary.

More algorithms, better infrastructure While DimReader shows that it is possible to adapt a large number of existing NDR methods to run within an autodiff framework, one potential goal is to provide DimReader axes to as much existing visualization infrastructure as practically possible. In such scenarios, reducing the implementation effort even further would be desirable. The majority of our difficulties in porting algorithms to autodifferentiation arose due to difficulties in evaluating derivatives of linear-algebraic concepts, such as solutions of a linear system and eigenvectors. Some of these have explicit formulas [33], but incorporating them in an autodiff system appears to be a fundamental challenge beyond the scope of our work.

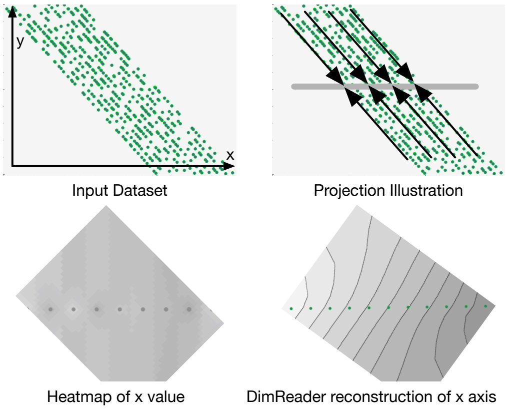

Figure 15: An example projection in which DimReader shows fundamentally different information than value heatmaps. The dataset and projections are synthetic examples specifically designed to highlight the difference. Clutter in projections will generally cause value heatmaps to show averages of the points, and oblique projections will generally cause input values to cancel each other out in value heatmaps. DimReader directly measures the perturbation outcomes, and so will show a different result. We rotate the screenshots simply to clarify the presentation.

User evaluation A proper, user-centric evaluation of the settings in which DimReader's axes are more informative than naked NDR plots is clearly necessary, and will be the subject of future work. We note, nevertheless, that DimReader directly addresses many of the following gaps identified in Sedlmair et al.'s interview study about gaps between theory and practice in dimensionality reduction (DR). These include the *conceptual gap* ("what is DR doing?"), *interpretation gap*: "what do the results mean?"; *guidance gap*, "what algorithm to use?", *non-linear unmapping gap*: "how do projection dimensions relate to input dimensions?"; and the *categorical dimension assumption gap*: analysts look for point clusters [36]. At the same time, DimReader has nontrivial performance overhead, and so it exacerbates the observed *scalability gap*. Further research remains necessary.

7 Conclusion and Future Work

In this paper, we identified *infinitesimal perturbations* as a central tool to enable interpretation of non-linear dimensionality reduction plots, and presented DimReader, a technique that produces generalized axes for studying such perturbations. DimReader allows us to highlight strengths and weaknesses of specific NDR methods, and provides insight into what these methods are actually showing.

One of the most exciting avenues for future work is the design of NDR methods that take user-specified perturbations into account. We have shown that such perturbations can be interpreted as generalized axes, and so we envision future methods where these drive the projections more directly.

Acknowledgements We acknowledge fruitful discussions over with Sean Stephens, Luiz Gustavo Nonato, and Stephen Ingram about the energy formulation of t-SNE, and projection methods in general.

References

[1] K. N. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, S. S. Allam, C. A. Prieto, D. An, K. S. Anderson, S. F. Anderson, J. Annis, N. A. Bahcall, et al. The seventh data release of the sloan digital sky survey. *The Astrophysical Journal Supplement Series*, 182(2):543, 2009.

[2] S. Agarwal, K. Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[3] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[4] E. Angel. Interactive computer graphics: A top-down approach using opengl, 2003.

[5] S. Bergner, M. Sedlmair, T. Moller, S. N. Abdolyousefi, and A. Saad. Paraglide: Interactive parameter space partitioning for computer simulations. *IEEE transactions on visualization and computer graphics*, 19(9):1499–1512, 2013.

[6] I. Borg and P. J. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[7] J. Boy, R. A. Rensink, E. Bertini, and J.-D. Fekete. A principled way of assessing visualization literacy. *IEEE transactions on visualization and computer graphics*, 20(12):1963–1972, 2014.

[8] M. Brehmer, S. Ingram, J. Stray, and T. Munzner. Overview: The design, adoption, and analysis of a visual document mining tool for investigative journalists. *IEEE transactions on visualization and computer graphics*, 20(12):2271–2280, 2014.

[9] M. Brehmer and T. Munzner. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2376–2385, 2013.

[10] A. Buja and D. F. Swayne. Visualization methodology for multidimensional scaling. *Journal of Classification*, 19(1):7–43, 2002.

[11] M. P. Carmo. Differential geometry of surfaces. *Differential Forms and Applications*, pp. 77–98, 1994.

[12] M. Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *Proceedings of the 7th conference on Visualization'96*, pp. 127–ff. IEEE Computer Society Press, 1996.

[13] R. D. Cook. Detection of influential observation in linear regression. *Technometrics*, 19(1):15–18, 1977.

[14] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[15] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[16] A. Endert, C. Han, D. Maiti, L. House, and C. North. Observation-level interaction with statistical models for visual analytics. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pp. 121–130. IEEE, 2011.

[17] N. Ferreira, J. T. Klosowski, C. E. Scheidegger, and C. T. Silva. Vector field k-means: Clustering trajectories by fitting multiple vector fields. *Computer Graphics Forum*, 32(3pt2):201–210, 2013.

[18] I. K. Fodor. A survey of dimension reduction techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, 9:1–18, 2002.

[19] M. Gleicher. Explainers: Expert explorations with crafted projections. *IEEE transactions on visualization and computer graphics*, 19(12):2042–2051, 2013.

[20] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

[21] G. H. Golub and C. F. Van Loan. *Matrix computations*, vol. 3. JHU Press, 2012.

[22] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[23] S. Ingram. Personal communication, 2017.

[24] S. Ingram, T. Munzner, and M. Olano. Glimmer: Multilevel mds on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):249–261, 2009.

[25] D. H. Jeong, C. Ziemkiewicz, B. Fisher, W. Ribarsky, and R. Chang. ipca: An interactive system for pca-based visual analytics. *Computer Graphics Forum*, 28(3):767–774, 2009.

[26] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2563–2571, 2011.

[27] G. Kindlmann and C. Scheidegger. An algebraic process for visualization design. *IEEE transactions on visualization and computer graphics*, 20(12):2181–2190, 2014.

[28] R. M. Kirby and C. T. Silva. The need for verifiable visualization. *IEEE Computer Graphics and Applications*, 28(5), 2008.

[29] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, last accessed 2017/03/17.

[30] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[31] A. Morrison, G. Ross, and M. Chalmers. A hybrid layout algorithm for sub-quadratic multidimensional scaling. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pp. 152–158. IEEE, 2002.

[32] F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):564–575, 2008.

[33] K. B. Petersen, M. S. Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7:15, 2008.

[34] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[35] H. M. Schey. *Div, grad, curl, and all that*. Norton, 1973.

[36] M. Sedlmair, M. Brehmer, S. Ingram, and T. Munzner. Dimensionality reduction in the wild: Gaps and guidance. *Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, Tech. Rep. TR-2012-03*, 2012.

[37] M. Sedlmair, T. Munzner, and M. Tory. Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2634–2643, 2013.

[38] C. Seifert, V. Sabol, and W. Kienreich. Stress maps: analysing local phenomena in dimensionality reduction based visualisations. In *IEEE Int. symposium on Visual Analytics Science and Technology.*, 2010.

[39] J. R. Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[40] J. Stahnke, M. Dörk, B. Müller, and A. Thom. Probing projections: Interaction techniques for interpreting arrangements and errors of dimensionality reductions. *IEEE transactions on visualization and computer graphics*, 22(1):629–638, 2016.

[41] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[42] W. S. Torgerson. Multidimensional scaling of similarity. *Psychometrika*, 30(4):379–393, 1965.

[43] E. R. Tufte. *Visual explanations: images and quantities, evidence and narrative*, vol. 36. Graphics Press Cheshire, CT, 1997.

[44] L. van der Maaten. A Python implementation of t-SNE. `https://lvdmaaten.github.io/tsne/`.

[45] S. v. d. Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[46] M. Wattenberg, F. Vigas, and I. Johnson. How to use t-sne effectively. *Distill*, 2016. http://distill.pub/2016/misread-tsne.