GhostUMAP: Measuring Pointwise Instability in Dimensionality Reduction

Myeongwon Jung*
Sungkyunkwan University

Takanori Fujiwara†
Linköping University

Jaemin Jo‡
Sungkyunkwan University

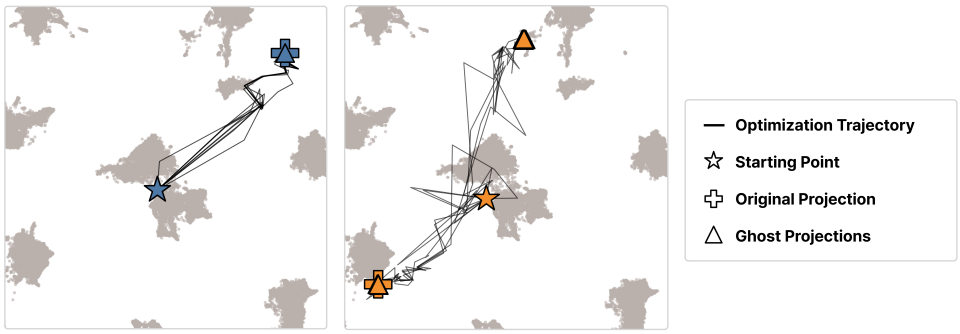

Figure 1: Each projection is part of a GhostUMAP projection generated for the CIFAR-10 dataset. Case (A) depicts the trajectories of a stable point where the original projection (✚) and its ghosts (▲) are projected to a consistent location. In contrast, Case (B) shows the trajectories of an unstable point. The trajectories diverge, implying instability in the final projection of the point (✚).

Abstract

Although many dimensionality reduction (DR) techniques employ stochastic methods for computational efficiency, such as negative sampling or stochastic gradient descent, their impact on the projection has been underexplored. In this work, we investigate how such stochasticity affects the stability of projections and present a novel DR technique, GhostUMAP, to measure the pointwise instability of projections. Our idea is to introduce clones of data points, "ghosts", into UMAP's layout optimization process. Ghosts are designed to be completely passive: they do not affect any others but are influenced by attractive and repulsive forces from the original data points. After a single optimization run, GhostUMAP can capture the projection instability of data points by measuring the variance with the projected positions of their ghosts. We also present a successive halving technique to reduce the computation of GhostUMAP. Our results suggest that GhostUMAP can reveal unstable data points with a reasonable computational overhead.

Index Terms: Dimensionality Reduction

1 Introduction

Many dimensionality reduction (DR) techniques incorporate stochastic components, such as negative sampling [15], by design to enhance computational efficiency [22]. However, these stochastic components can lead to varying results each time DR is executed, casting questions on its stability. This work aims to explore two key research questions: 1) How significantly do these stochastic components affect projection stability? 2) How can we identify data points whose projections are unstable?

We focus on a widely-used DR technique, Uniform Manifold Approximation and Projection (UMAP) [14]. From a high-level point of view, UMAP introduces stochastic components for two reasons: 1) to generate an initial projection with Spectral Embedding [1] and small random positional perturbation; 2) to simulate the repulsive forces between points during optimization, i.e., negative sampling [22, 23]. While the former could be controlled relatively easily by using a deterministic DR technique, such as Principal Component Analysis (PCA) [5], or by reducing the amount of noise, the latter can significantly affect the final projection of a point depending on which points are chosen as random samples.

As a motivating example, we plot the trajectory of two points during GhostUMAP optimization in Fig. 1. Stars and crosses represent the initial positions of the points and their final projections, respectively. Triangles indicate alternative projections of the points that can be obtained by choosing different sets of negative samples. In Fig. 1A, regardless of the choice of negative samples, the target point (blue cross) and its ghosts (blue triangles) are projected to a consistent location, implying high positional stability. However, in Fig. 1B, the projections of the target point (orange cross) and its ghosts (orange triangles) diverge, implying that the projections are less stable and sensitive to the selection of negative samples.

We introduce GhostUMAP, a novel technique for measuring the pointwise positional instability that results from negative sampling in UMAP. By positional instability, we mean the variance of projections that can be led by selecting different negative samples. Our core idea is to add clones of data points, i.e., *ghosts*, to the optimization process. We optimize the layout for the original data points and ghosts jointly, where the ghosts are affected by the projections of the original points but do not affect others. Fig. 1 is generated by GhostUMAP, where the triangles correspond to ghosts. After the optimization process, the variance between the original projection of a point and its ghosts is used as an instability measure. To reduce the overhead resulting from ghosts, we also present a successive halving technique that prunes the data points identified as stable in the middle of optimization. Our results suggest the effectiveness of GhostUMAP in identifying unstable points due to stochasticity and show that the successive halving technique offers a compelling trade-off between accuracy and computational efficiency.

2 Related Work

2.1 Distortion and Instability of DR

To our knowledge, the problem of measuring pointwise instability due to the stochasticity of DR techniques has not been explored before. However, one relevant research topic is to measure and understand the distortion between the high-dimensional to low-

*e-mail: mw.jung@skku.edu
†e-mail: takanori.fujiwara@liu.se
‡e-mail: jmjo@skku.edu. Jaemin Jo is the corresponding author.

dimensional spaces that DR techniques introduce. Various measures have been proposed to measure distortion at different levels: local [12, 25], cluster level [7, 18, 20], and global levels [4, 10]. These measures are also employed to assess and compare the quality of DR [2, 6, 16]. Researchers further attempted to visually analyze the distortion [13, 21]. In general, these distortion measures are applied to a fixed projection and they do not consider influences of arbitrary changes caused by the stochasticity of DR optimization. Jung et al.'s work [8] is the most closely related to our work, where they compared multiple t-SNE projections and extracted stable *clusters* among the projections. In contrast, we introduce GhostUMAP, which can capture unstable data points (i.e., *local* level) in regards to the stochasticity of UMAP.

2.2 Uniform Manifold Approximation and Projection

UMAP [14] can be delineated into two computation phases: 1) graph construction and 2) layout optimization. In the graph construction phase, UMAP generates a topological representation of the high-dimensional data by connecting local fuzzy simplicial sets, which can be represented as a weighted k-nearest neighbor graph. In the layout optimization phase, UMAP employs a force-directed graph layout algorithm [3] to embed the topological representation into a low-dimensional space by simulating the attractive and repulsive forces between points. GhostUMAP does not add any changes in the projection of the original points but applies the forces to ghosts to generate variations of projections.

The attractive forces of UMAP are exerted between vertices (corresponding to data points) linked by an edge in the k-nearest neighbor graph. The attractive force between vertices i and j, positioned at low-dimensional coordinates o_i and o_j, is defined by:

$$F_{att}(o_i, o_j, w_{ij}) = \frac{-2ab\|o_i - o_j\|_2^{2(b-1)}}{1 + \|o_i - o_j\|_2^2} w_{ij}(o_i - o_j) \quad (1)$$

where a and b are user-specified positive constants and w_{ij} is the weight of the edge, i.e., the similarity between vertices i and j in the high-dimensional space.

The repulsive force between vertices i and j is given by:

$$F_{rep}(o_i, o_j, w_{ij}) = \frac{2b}{(\varepsilon + \|o_i - o_j\|_2^2)(1 + a\|o_i - o_j\|_2^{2b})}(1 - w_{ij})(o_i - o_j) \quad (2)$$

where ε is a small constant to avoid division by zero. Theoretically, the repulsive force should be simulated for each pair of vertices, which is inefficient. Therefore, UMAP employs negative sampling where n_neg points ($n\_neg = 5$ by default) are randomly sampled for each vertex i to only apply the repulsive forces between i and the sampled points. This negative sampling is the root cause of making the layout optimization phase stochastic.

3 GhostUMAP: UMAP with Ghosts

We present GhostUMAP, a novel DR technique for measuring the pointwise instability of UMAP. We aim to introduce clones of the original data points, i.e., *ghosts*, to the optimization phase and jointly optimize the original projections and ghosts while ensuring the ghosts do not affect the original projections, which are detailed in the following sections.

3.1 Layout Optimization with Ghosts

Let x_i and o_i ($i \in [1, ..., N]$) be the i-th data point in the high-dimensional space and lower-dimensional projection space where N is the number of data points. In GhostUMAP, each data point x_i has n_ghosts ghosts in the projection space g_{i_k} ($k \in [1, ..., n\_ghosts]$), in addition to its original projection o_i. We denote the set of the projections of point i, including the original projection and ghosts, as $Y_i = \{o_i\} \cup \{g_{i_k} \mid k \in [1, ..., n\_ghosts]\}$.

Algorithm 1 GhostUMAP Layout Optimization

Input: weighted kNN graph G, halving schedule s, $\mathcal{Y} = \{Y_1, \ldots, Y_N\}$
Output: $\mathcal{Y}$

```
α ← 1.0
for e ← 1, ..., n_epochs do
    if e ∈ s then
        Discard 50% of g_i with the lowest instability INS(x_i)
    end if
    OriginalLayoutOptimization(G, Y, α)
    GhostLayoutOptimization(G, Y, α)
    α ← 1.0 − e/n_epochs
end for
```

Algorithm 2 Layout Optimization for Original Projections

```
1  function OriginalLayoutOptimization(G, Y, α)
2      for all ([i, j], w_ij) ∈ G do
3          f_att ← F_att(o_i, o_j, w_ij)
4          (o_i, o_j) ← (o_i + α · f_att, o_j − α · f_att)
5          for n ← 1, ..., n_neg do
6              l ← random sample from {1, ..., N}\{i}
7              o_i ← o_i + α · F_rep(o_i, o_l, w_il)
8          end for
9      end for
10 end function
```

Algorithm 3 Layout Optimization for Ghost Projections

```
1  function GhostLayoutOptimization(G, Y, α)
2      for all ([i, j], w_ij) ∈ G do
3          for all k ← 1, ..., n_ghosts do
4              g_ik ← g_ik + α · F_att(g_ik, o_j, w_ij)
5              for n ← 1, ..., n_neg do
6                  l ← random sample from {1, ..., N}\{i}
7                  g_ik ← g_ik + α · F_rep(g_ik, o_l, w_il)
8              end for
9          end for
10     end for
11 end function
```

In GhostUMAP, we alternately optimize the layout of the original projections and ghosts, as outlined in Alg. 1. The optimization process consists of two phases: the original layout phase and the ghost layout phase. In the original layout phase, we update the layout of the original projections by simulating the forces given by Eq. 1 and 2 (Alg. 2), which is exactly the same as UMAP. In the ghost layout phase, we simulate the force on a ghost g_{i_k}, assuming it was the original projection of the i-th data point. This means that we simulate the attractive force between g_{i_k} and x_i's neighbors on the k-nearest neighbor graph and the repulsive force between g_{i_k} and randomly selected points, as shown in Alg. 3. Note that the force exerted on the ghost is not the same as the force on the original point. The force on the ghost is recalculated based on its position rather than the position of its original point.

As contrasted by Line 4 in Alg. 2 and Line 4 in Alg. 3, the ghost layout phase applies the attractive force between g_{i_k} and o_j asymmetrically; the ghost is pulled towards the neighboring original projections but not vice versa. This means that the original projections are unaware of ghosts, while the ghosts can be pushed or pulled by the original projections but not by each other.

The positions of the ghosts of a certain point x_i can be seen as

Table 1: Benchmark results comparing execution times and F1-scores for UMAP, GhostUMAP with SH, and GhostUMAP without SH across various datasets. The time required for the weighted kNN graph construction is consistent across all conditions, whereas the time specified under each condition corresponds to the layout optimization. All time measurements are expressed in seconds. F1-scores evaluate the consistency between GhostUMAP with SH and without SH. Each measurement represents means with standard deviations (±).

		Time ($n\_neighbors = 15$ and $n\_ghosts = 8$)				
Dataset	Size ($N \times$dims)	weighted kNN	UMAP	GhostUMAP with SH	GhostUMAP without SH	F1-score
C. elegans [17]	6,188×50	7.42 (±0.17)	3.19 (±0.11)	10.73 (±0.21)	16.10 (±0.55)	0.99 (±0.03)
AG News [26]	7,600×1,024	7.98 (±0.18)	3.54 (±0.08)	12.69 (±0.54)	18.40 (±0.68)	0.88 (±0.03)
20NG [19]	18,846×1,024	8.43 (±0.20)	4.20 (±0.24)	11.36 (±0.71)	19.54 (±1.37)	0.97 (±0.04)
CIFAR-10 [9]	50,000×768	9.94 (±0.22)	7.31 (±0.18)	25.76 (±0.34)	43.02 (±0.49)	0.91 (±0.01)
MNIST [11]	70,000×784	11.19 (±0.56)	8.60 (±0.18)	30.29 (±0.65)	55.23 (±1.83)	0.96 (±0.01)

alternative projection results when different points are sampled by negative sampling. If there is no stochasticity at all, the ghosts g_{i_k} would be projected to the same location as the original projection o_i. However, due to the stochasticity, they become misaligned as the optimization proceeds, eventually arriving at different locations. As a measure to detect unstable points, we define the instability of a data point x_i, $INS(x_i)$, as the variance of Y_i, i.e., o_i and g_{i_k}:

$$\mu_i = \frac{1}{n\_ghosts + 1} \sum_{p \in Y_i} p \tag{3}$$

$$INS(x_i) = \frac{1}{n\_ghosts + 1} \sum_{p \in Y_i} ||p - \mu_i||_2^2 \tag{4}$$

3.2 Successive Halving on Ghosts

Ghosts slow down the optimization phase $(n\_ghosts + 1)$ times as it is equivalent to optimizing $N \cdot (n\_ghosts + 1)$ data points instead of N. To reduce such an overhead, we introduce a successive halving (SH) technique. All data points start with n_ghosts ghosts per each, but after a certain number of epochs, we measure the instability of the points and discard the ghosts of half of the points with the lowest instability, which are not of interest. For example, suppose a scenario with $N = 1,000$ data points, $n\_ghosts = 7$ ghosts per each, and a SH schedule $s = [50, 100, 150]$. Initially, 8,000 data points are subject to optimization (1,000 original points and 7,000 ghosts). At the 50th iteration, we drop the ghosts of the 500 most stable points, resulting in 1,000 original points (original points are always kept) and 3,500 ghosts, summing up to 4,500 points in total. We repeat the halving operation at iterations 100 and 150, which reduces the number of data points to 2,750 and 1,875, respectively. In the following section, we show that this halving operation significantly reduces the running time while keeping the most unstable points. The source code of GhostUMAP is available at https://github.com/jjmmwon/ghostumap.

4 Evaluation

In this section, we validate our technique by conducting a performance benchmark and presenting its use case in data analysis.

4.1 Performance Benchmark

The goals of the performance benchmark were to understand 1) the overhead that ghost computation exhibits and 2) the effectiveness of the SH technique. To this end, we compared three conditions: 1) UMAP, 2) GhostUMAP with SH, and 3) GhostUMAP without SH. For the first goal, we compared the execution time of the three conditions. For the second goal, we investigated to what extent our SH technique can maintain the most unstable points by checking the consistency between the results with SH and without SH.

To check the effectiveness of SH, we computed the instability of each point using GhostUMAP without SH, i.e., $INS(x_i)$. We then defined that a point is unstable if its instability exceeds $mean + 2.5\sigma$ where $mean = \sum INS(x_i)/N$ and $\sigma = (\sum (INS(x_i) - mean)^2/N)^{1/2}$.

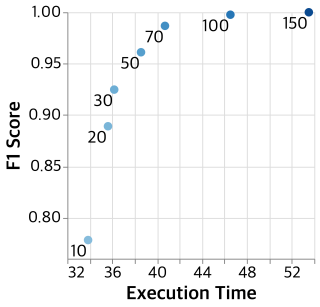

Figure 2: The trade-off between execution times and F1-scores depending on the epoch at which halving is performed. The text label placed next to each dot indicates the halving-performed epoch.

This criterion identified about 0.5% data points of each dataset as unstable. We used these selected unstable points as the ground-truth set. We ran GhostUMAP with SH on the same dataset and chose the same number of points as the ground-truth set with the highest instability, which we defined as the answer set. Finally, we computed the F1-score between the ground-truth and answer sets. As two sets have the same number of points, the precision and recall are equivalent to the F1-score.

Dataset and Setting. We used five datasets that were employed to evaluate DR techniques in previous work [17, 22, 24], encompassing different sizes and data types, such as tables, text, and images (Table 1). The UMAP hyperparameters for all conditions were the same as the default specified in the official UMAP documentation: $n\_neighbors = 15$ and $min\_dist = 0.1$. The number of epochs, n_epochs, is determined by the dataset size: 500 epochs up to 10,000 rows and 200 epochs for larger datasets. For GhostUMAP, n_ghosts was set to 8. We performed the halving operations at the 50th, 100th, and 150th epochs when $n\_epochs = 200$ and at the 200th, 300th, and 400th epochs when $n\_epochs = 500$. We repeated 10 trials for each condition to compute the mean and standard deviation of each measure. We used Apple MacBook M1 Pro 14-inch 2021 for the benchmark.

Result and Discussion. In Table 1, we separately report the time spent for the weighted kNN graph construction and layout optimization to enable a precise comparison of the layout optimization phase. Unlike the layout optimization phase, the graph construction phase is the same across UMAP, GhostUMAP with SH, and GhostUMAP without SH. The total running time of each technique can be computed by summing the times spent on these two phases. For example, for the C. elegans dataset in Table 1, UMAP, GhostUMAP with SH, and GhostUMAP without SH took averages of 10.61, 18.15, and 23.52 seconds, respectively.

We could observe that GhostUMAP exhibited a longer execution time than UMAP, but the difference was smaller than we expected. For example, GhostUMAP without SH took about five times longer (16.10 seconds) than UMAP (3.19 seconds) for the C. elegans dataset, respectively, while theoretically, GhostUMAP opti-

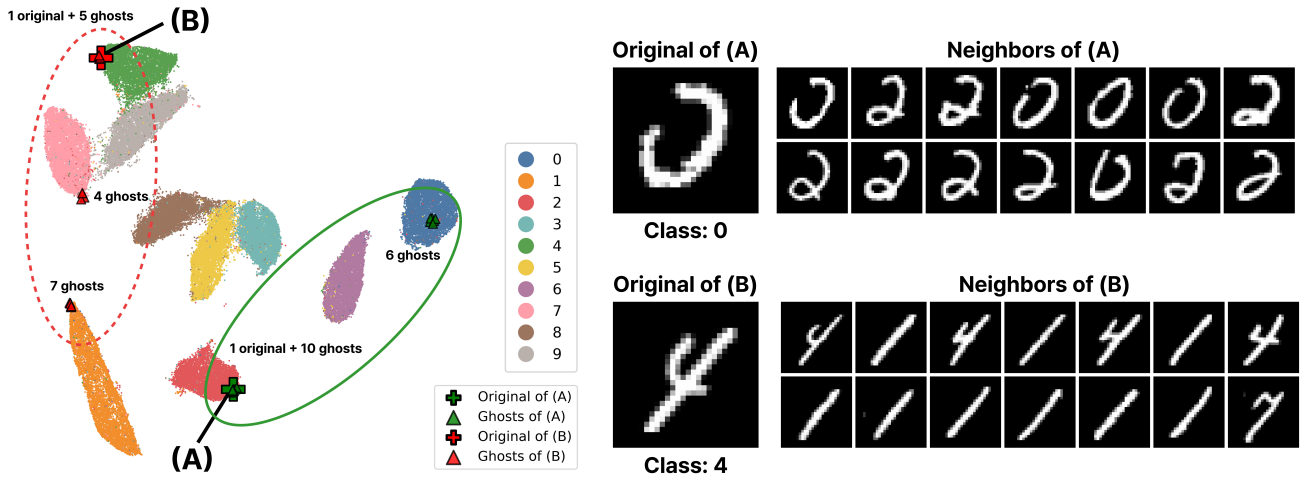

Figure 3: GhostUMAP highlights two unstable cases (A) and (B) in the MNIST dataset. Note that these two cases are representative examples and more unstable points were identified by GhostUMAP. Crosses represent the original projections of points, while triangles indicate the projections of their ghosts. In both cases, one can observe the spatial discrepancy between the original projections and ghosts, which implies positional instability. The right panel displays the images corresponding to the original points and their neighbors, providing visual context to the detected instability.

mizes nine times more ($n\_ghosts + 1$) data points. This may be due to the parallelization implemented by the Python `numba` package, where ghost computations could be efficiently parallelized since they are independent of each other.

Comparing GhostUMAP with and without SH, our SH technique achieved a speed-up of 38.3% on average; for example, in the MNIST dataset, SH reduced the execution time from 55.23 seconds to 30.29 seconds, showing a speed-up of 45.2%. The F1-scores between the GhostUMAP results with and without SH showed remarkable consistency in preserving unstable points, ranging from 0.88 to 0.99. This consistency suggests that our SH technique effectively accelerated the computations related to ghosts while maintaining the most unstable points.

To better understand the trade-off between execution times and F1-scores, we conducted an additional experiment where we halved data points only once at different epochs. We used the MNIST dataset and the same setting described above. The result is shown in Fig. 2. We could observe that as we perform the first halving earlier, the execution time decreases at the cost of the F1-score. This is because if we halve the points at an early epoch, we can optimize fewer points in the rest of the epochs. However, since the instability of points is not fully simulated, an early halving can wrongly discard the points that are actually unstable. Performing the first halving at the 50th or 70th epochs exhibited a reasonable trade-off for the MNIST dataset.

4.2 Use Case

We present a use case to show how GhostUMAP can find and review unstable data points in a projection. Fig. 3 shows a GhostUMAP projection with n_ghosts set to 16 for the MNIST dataset. We highlighted two unstable points (A) and (B) and their ghosts in Fig. 3. We chose these two points among the top 0.5% of the points in terms of the instability score, $INS(x_i)$, and more examples on different datasets can be found in the supplementary material.

In Fig. 3A, we can see that the original projection of point A belonged to the red cluster of digit 2 (green cross). However, among its 16 ghosts (green triangles), only ten were projected near the original projection, but the other six belonged to a different cluster, the blue cluster of digit 0. To understand why this occurred, we analyzed the handwritten image that the point represents (first row of digit images in Fig. 3). The image showed that the digit 0 was written ambiguously. Indeed, its neighbors in the high-dimensional space consisted of classes of digits 0 and 2. These neighbors from different classes could make attractive forces diverged and the projection more sensitive to negative samples.

For another unstable point B, its ghosts were separated into three clusters, as annotated by the dotted red ellipse. The original projection (red cross) and five ghosts (red triangles) belonged to the green cluster of digit 4, four were to the pink cluster of digit 7, and the other seven were to the orange cluster of digit 1. The second row of digit images in Fig. 3 shows the original image and its neighbors. Although the original image was corresponding to digit 4, its neighbors consisted of digits 4, 1, and 7. Note that while there was only one neighbor of digit 7 among the 15 neighbors, four out of 16 (25%) ghosts were projected to the cluster of 7. The projection to the cluster of digit 7 may imply a complex non-linear relationship between the composition of neighbors and projections.

Our qualitative investigation of the GhostUMAP projection of the MNIST dataset suggests two implications. First, one should interpret the projection of an individual point with care, as the projection of a single point can be the result of mere chance. Therefore, it is recommended that pointwise interpretation be validated by checking multiple projections. Second, our investigation highlights the stability limitation of UMAP. The user should be warned of such instability once detected, especially when no external information (e.g., class labels) is available to validate the result.

In sum, the results of the performance benchmark and use case confirm the capability of GhostUMAP to measure the pointwise instability arising from the stochasticity of UMAP and to detect unstable points that should be interpreted with care. We could also confirm that our SH technique can accelerate the instability computation while maintaining the most unstable data points.

5 Conclusion

We present GhostUMAP to measure the pointwise instability of UMAP. Our evaluations show that GhostUMAP can identify unstable data points in projections even with the successive halving technique designed to reduce the computational overhead. For future work, we are interested in rigorously examining how faithful ghosts are by replacing the original points with the ghosts and measuring the projection quality, e.g., the Kullback-Leibler divergence [4]. We also aim to explore methods to dynamically adjust the number of ghosts per instance for better efficiency.

ACKNOWLEDGMENTS

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-00421, AI Graduate School Support Program (Sungkyunkwan University)) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00221186). This work has been supported in part by the Knut and Alice Wallenberg Foundation through Grant KAW 2019.0024.

REFERENCES

[1] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*, 14, 2001. 1

[2] M. Espadoto, R. M. Martins, A. Kerren, N. S. Hirata, and A. C. Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2153–2173, 2019. 2

[3] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. 2

[4] G. E. Hinton and S. Roweis. Stochastic neighbor embedding. *Advances in Neural Information Processing Systems*, 15, 2002. 2, 4

[5] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417, 1933. 1

[6] H. Jeon, A. Cho, J. Jang, S. Lee, J. Hyun, H.-K. Ko, J. Jo, and J. Seo. ZADU: A python library for evaluating the reliability of dimensionality reduction embeddings. In *Proc. VIS*, pp. 196–200. IEEE, 2023. 2

[7] H. Jeon, H.-K. Ko, J. Jo, Y. Kim, and J. Seo. Measuring and explaining the inter-cluster reliability of multidimensional projections. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):551–561, 2021. 2

[8] M. Jung, J. Choi, and J. Jo. Projection Ensemble: Visualizing the robust structures of multidimensional projections. In *Proc. VIS*, pp. 46–50. IEEE, 2023. 2

[9] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 3

[10] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. 2

[11] Y. LeCun, C. Cortes, and C. J.C. Burges. The MNIST database of handwritten digits. https://www.openml.org/search?type=data&id=554, 1999. Accessed: 2024-3-26. 3

[12] J. A. Lee and M. Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72(7-9):1431–1443, 2009. 2

[13] S. Lespinats and M. Aupetit. CheckViz: Sanity check and topological clues for linear and non-linear mappings. *Computer Graphics Forum*, 30(1):113–125, 2011. 2

[14] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv:1802.03426*, 2018. 1, 2

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 2013. 1

[16] L. G. Nonato and M. Aupetit. Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE Transactions on Visualization and Computer Graphics*, 25(8):2650–2673, 2018. 2

[17] J. S. Packer, Q. Zhu, C. Huynh, P. Sivaramakrishnan, E. Preston, H. Dueck, D. Stefanik, K. Tan, C. Trapnell, J. Kim, et al. A lineage-resolved molecular atlas of *C. elegans* embryogenesis at single-cell resolution. *Science*, 365(6459):eaax1971, 2019. 3

[18] 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. 2

[19] J. Rennie. 20 Newsgroups. http://qwone.com/~jason/20Newsgroups/. Accessed: 2024-04-30. 3

[20] M. Sips, B. Neubert, J. P. Lewis, and P. Hanrahan. Selecting good views of high-dimensional data using class consistency. *Computer Graphics Forum*, 28(3):831–838, 2009. 2

[21] 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, 2015. 2

[22] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *Proc. WWW*, pp. 287–297, 2016. 1, 3

[23] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: Large-scale information network embedding. In *Proc. WWW*, pp. 1067–1077, 2015. 1

[24] L. Van der Maaten. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014. 3

[25] J. Venna and S. Kaski. Local multidimensional scaling. *Neural Networks*, 19(6-7):889–899, 2006. 2

[26] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *Advances in Neural Information Processing Systems*, 28, 2015. 3